Petr Krysl

Finite Element Modeling with Abaqus and Python for Thermal and Stress Analysis

Contents

1

Introduction

1.1 Goals and Approach

The goal of the present textbook is to introduce the basics of the finite element analysis for thermal and deformation problems that can be analyzed with linear models.

In this book we focus on the following aspects of which we believe the user of finite element analysis must have a basic understanding:

1. Understanding of the mathematical models: what are the underlying assumptions, how is the model properly defined, what are the solutions of elementary models.
2. How is the finite element model related to the mathematical model: what is the effect of the conversion of the continuous mathematical model to the discrete finite element model, what is the accuracy, how to control the accuracy.
3. How to use results from finite element modeling: interpretation of results, estimation of errors, verification and validation, proper formulation of reports.

In this respect the present textbook differs from the current popular engineering textbooks on finite elements. The mathematics is also presented as simply as possible, so that the present textbook is also quite different from state of the art the mathematical treatments of the finite element method.

1.2 Coverage

The finite element models are in this book derived from the method of weighted residuals (the Galerkin form). This has the advantage of being very general and applicable, not only in linear modeling but also in nonlinear modeling.

Only linear models are discussed. The thermal analysis will not consider radiation boundary conditions. These conditions leads to a nonlinear model which is outside of the scope of this book. In stress analysis we will not consider contact conditions, which are again a nonlinearity-generating boundary condition.

The finite element method is developed for the so-called continuum finite elements. Structural finite elements are not considered: beams, shells, and various types of discrete elements (connectors, dampers, ...) are not used in our examples and they are not discussed at all. The structural finite elements are reduced versions of the continuum finite elements, but contrary to expectations, this makes the structural elements much more complicated to describe and use. They are probably best left to an advanced finite element course. (The trivial structural elements— two-node truss elements and two-dimensional beams— are an exception to the rule: they are simple. They are also quite limited in what they can do.)

1.3 Organization of the book

Important bits of information are boxed:

> This is how important information will be set off. For instance, here is an important equation
>
> $$\exp i\phi = \cos \phi + i \sin \phi$$

At times there will be a warning:

> This is a common mistake!

The electronic book has built-in links to additional information and resources. This is likely to prove useful when searching for text or code. The external links lead to a website storing the PDF tutorials, Python scripts, Abaqus model files, and other resources. Use a capable PDF viewer that will allow you to control how to open these files. Some browsers will attempt to display the Abaqus model files, with useless results. In that case attempt to make them save the file to the hard drive.

> The material in the regular sections presents the basic information. At the end of each chapter, there is a special section with detailed background information, additional explanations, tables, and such. This is not required reading at the basic level: if you need it or if you are plain interested, go for it; otherwise you can leave it till it becomes useful.

1.4 Software

The software used in this textbook consists of a finite element program with a graphical user interface, Abaqus/CAE, and the open-source object-oriented language Python.

> The PDF version of the textbook includes links to outside sources of information: model files, tutorials, Python source files, ... **Note: until you reach the box that says otherwise, the links in the margin are only examples of what the linking to external resources looks like later in the book: these few links don't work!**

Abaqus
- CAE file

Abaqus examples are typically accompanied by a model database. The link to such a database (extension ".cae") is provided in the margin. The web browser should open such data bases with Abaqus.

Abaqus
- CAE file
- tutorial

Oftentimes the Abaqus example comes with a tutorial. The tutorial is a PDF file which shows the process step-by-step. The web browser would typically open the PDF file when you click the link.

Abaqus
- CAE file
- INP file
- tutorial

In a few instances the Abaqus model is derived from an input file (extension ".inp"). Then one of the links will point to an INP file. This is a plain-text file which can be edited with any text editor (`notepad++` is a free offering on Windows and Linux). If the file is opened in a web browser instead of saved to your local disk storage (which depends on your particular computing platform), save

the file with the ".inp" extension. As an example, in the Chrome browser, right-click on the text displayed in the browser and select "`Save as...`". In the selection box "`Save as type`" choose "`All Files`", and make sure to delete the ".`txt`" extension, if there is one attached to the name. Then in the folder where the file was saved you have the ".`inp`" file which can be now edited with any text editor. The tutorials will explain how to use the INP file.

Some detailed finite element calculations are carried out with Python. Links to the Python script files (extension `.py`) are available in the margin. Save the file and then load it in development environment of choice (see Section 1.4).

Python
- script

Some points are illustrated with animations. Links to video files are available in the margin, and are typically opened in a web browser. If that is not the case, locate the file in your Downloads folder and drag it to a browser window. (There are also specialized applications that can display video files.)

Animation
- video

Links to outside resources from this box on should all work. If you find a problem with the downloads, please let the author know (`pkrysl@ucsd.edu`).

1.4.1 Abaqus software

Abaqus is a suite of commercial finite element codes. It consists of Abaqus Standard, which is a general purpose finite element software, and Abaqus Explicit for dynamic analysis. It is now owned by Dassault Systèmes and is part of the SIMULIA range of products, see

www.simulia.com/products/unified_fea.html

In this book we will interact with the finite element analysis in Abaqus Standard through the Abaqus/CAE, a graphical user interface. Abaqus® is a registered trade mark of Dassault Systèmes. For product information, please refer to the website http://www.3ds.com. This book has been tested with the Abaqus/CAE Teaching Edition and the Student Edition, as they were current in 2020.

Abaqus is tightly coupled to **Python**, a very popular and widely used scripting language. The Abaqus software can be driven by scripts, which makes it a very flexible and powerful tool. In addition, many exercises in this book will be supported by numerical or symbolic computations with Python in the form of script files.

1.4.2 Cloud computing

The easiest way of running most Python examples from the textbook (except those that require Abaqus/CAE) may well be provided by the cloud computing environment try.jupyter.org. Select from the "`New`" menu to create "`Python 2`" or "`Python 3`" notebook, and then paste blocks of code or in fact the entire code of an example into the first cell of the notebook and select for instance "`Run all`" or "`Run Cells`". This will create an output cell, with printouts and graphics.

The programming environment at repl.it is also very easy to use. Simply copy the text of the Python program into the `main.py` window and click on Run. Registration is free (as of 2020).

1.4.3 Local Python environment

The examples in this book were written with modules from the so-called SciPy stack (SciPy = Scientific Python). For those interested in having a local Python environment, a good option is to visit www.scipy.org/install.html. At the top of the page there is a list of comprehensive packages for Linux, Windows and Mac ("Scientific Python Distributions"). All come with free versions, and as of 2020 there were these:

1. Anaconda: A free distribution for the SciPy stack. Supports Linux, Windows and Mac.
2. Enthought Canopy: The free and commercial versions include the core SciPy stack packages. Supports Linux, Windows and Mac.
3. WinPython: A free distribution including the SciPy stack. Windows only.

The two at the top require registration, the one at the end of the list is really the most straightforward to get. Download, install, and run the provided Integrated Development Environment (IDE) (called `Spyder`). Then download the script file by clicking one of the links in this textbook, open the file in the IDE, and run it.

1.4.4 Running Python in Abaqus

The examples in this book are worked with the Abaqus/CAE program. This program can be driven by a Python script. When the user operates the Graphical User Interface (GUI), for instance to partition a face, Abaqus/CAE writes the commands to accomplish this to a few files: the main ones being the replay file, `abaqus.rpy`, the journal file (with extension `.jnl`), and the recovery file (with extension `.rec`). These are all Python scripts that can be executed from the "File" menu.

> 🐾 Abaqus/CAE Student Edition does not write the replay file and the journal file. Also, the recovery file is erased once the user quits the GUI.

These Python scripts can be edited by the user to accomplish slightly modified tasks, for instance the dimensions of the part may be changed, or the material properties can be defined to work with user inputs, etc.

The Abaqus Scripting Interface is an application programming interface (API) to the modeling and model data. The scripting interface is an extension of the Python object-oriented programming language: the interface scripts *are* Python scripts. One can (a) create and modify the components of an Abaqus model (including, but not limited to, parts, materials, loads, and steps); (b) manage analysis jobs; (c) manage output databases; (d) postprocess the results of an analysis.

An even nicer facility is part of Abaqus/CAE: the "`Macro Manager`". Invoke the "`Macro Manager`" from the "`File`" menu, and start recording the macro. Then execute a few Abaqus commands, such as create a sketch and extrude it into a part. Then stop the recording of the macro, and voilà: the file called `abaqusMacros.py` appears in the working folder (typically `c:\temp`). The macro is named, and its name should now appear on the list of the "`Macro Manager`". Should it not appear automatically, feel free to click "`Reload`". The contents of the `abaqusMacros.py` script file may now be inspected with an editor of your choice. It consists essentially of definitions of Python functions. Each function is a macro.

As an example, here's a link to a sample macros file. When this file is copied to the working folder, and the "`Macro Manager`" is opened, two macros should be listed:

Python
- script

- `MakeCylinderPart` to create a cylindrical solid part; and
- `MakeUsefulMaterials` to create several material models.

1.4.5 Standing assumptions for Python code

Working with arrays and linear algebra in Python is best done with the array module `numpy`. The functions `array`, `dot`, and so on are part of this module, as is the submodule `linalg` for linear-algebra operations. When we present Python code we usually don't state that explicitly, but these objects need to be imported, for instance as

```
from numpy import array, dot
from numpy import linalg
```

Similarly, when we work with the symbolic-math module `sympy`, we import the objects we need from this module

```
from sympy import symbols, simplify, Matrix, diff
```

or, alternatively, we can import the module and then refer to the objects as for instance

```
import sympy
A, B = sympy.symbols('A, B')
```

> Some modules are not installed for the Abaqus Python environment. Code referenced in this book which relies on these modules will not run in the Abaqus command line. For instance neither `sympy`, nor `matplotlib` are available. It is easiest to use these modules in the cloud (in a Jupyter notebook), or with a local Python installation.

1.5 Units

Abaqus puts the onus of inputting the data in consistent units on the user. It is not alone in this, as typical finite element programs do not allow for explicit provision of physical units. This means that input data is only typed in as numbers, without any indication of the physical units. The program will assume that the numbers that were input by the user were all in consistent units.

In thermomechanical problems one must choose four units, for example for length, mass, time, and temperature. For instance, in SI units for these quantities, we use m for length, kg for mass, s for time and ^oK for temperature; then the units for forces will be in N, the stresses and the pressure will be in Pa, the mass density will be in $kg \cdot m^{-3}$, the thermal conductivity in $W \cdot m^{-1} \cdot {^oK}^{-1}$, and the specific heat in $J \cdot kg^{-1} \cdot {^oK}^{-1}$.

One should usually choose physical units that make the numerical values of the quantities with which the program calculates (typically the stiffness coefficients, the forces, the displacements) not too small and not too large. In analyses of systems of small spatial extent, a possible choice of units is mm for length, N for force, s for time and ^oK for temperature. Then mass density would be supplied in the surprising units of metric tons (tonnes) per millimeter cubed, so that the mass density of steel (let us say 7850 $kg.m^{-3}$) would need to be input as 7.85×10^{-9} tonne $\cdot mm^{-3}$ (so actually type in `7.85e-9` for the material Density property). However, elastic moduli, stresses, and pressure would be in the convenient units of MPa. Table 1.1 lists four sets of consistent units.

Many unit conversion utilities are available on the web, for instance at `www.translatorscafe.com`.

- URL

Quantity	SI	SI (mm)	US Unit (ft)	US Unit (inch)
Length	m	mm	ft	in
Force	N	N	lbf	lbf
Mass	kg	tonne (10^3 kg)	slug	lbf s^2/in
Time	s	s	s	s
Stress	Pa (N/m^2)	MPa (N/mm^2)	lbf/ft^2	psi (lbf/in^2)
Energy	J (N $\times$ m)	mJ (10^{-3} J)	ft lbf	in lbf
Density	kg/m^3	tonne/mm^3	slug/ft^3	lbf s^2/in^4

Table 1.1. Consistent sets of units (after the Abaqus Analysis User's Guide)

1.6 Abbreviations

Abbreviation	Meaning
CAE	Complete Abaqus Environment
CTE	Coefficient of Thermal Expansion
DOF	degree of freedom
FE	Finite Element
FEA	Finite Element Analysis
FEM	Finite Element Method
H8	Finite element with eight nodes (brick, solid)
H20	Finite element with 20 nodes (brick, solid)
L2	Finite element with two nodes (curve)
L3	Finite element with three nodes (curve)
Q4	Finite element with four nodes (quadrilateral, surface)
Q8	Finite element with eight nodes (quadrilateral, surface)
T3	Finite element with three nodes (triangle, surface)
T6	Finite element with six nodes (triangle, surface)
T10	Finite element with 10 nodes (tetrahedron, solid)
WR	Weighted Residuals
WRM	Weighted Residuals Method

Table 1.2. Abbreviations

Acknowledgments

2

Getting acquainted with the FEM

2.1 High-level explanation of the principles of FEM

Consider the mechanical part of the heavy-duty ball bearing housing of Figure 2.1. Without doubt the geometry of the part is fairly complex. If we wish to find for instance the natural frequencies, in order to judge how stiff the housing really is, we are unlikely to be able to find an analytical solution. Even approximate methods such as Rayleigh-Ritz where we have to guess the response of the structure in some way will not be successful because guessing the shape of vibration of this complicated mechanical part is not easy. This is where the finite element method (FEM) may step in to make the solution not only possible, but also automateable (and hence easily accomplished through computer modeling).
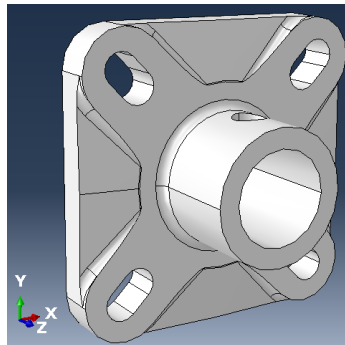
Fig. 2.1. Rexnord Heavy Duty Ball Bearing housing. The dimensions of the flange are approximately 100 mm on the side.

Figure 2.2 shows a snapshot from the animation of the mode shape corresponding to the 24th natural frequency at approximately 24.8 kHz. (The animation is provided as a video: follow the link on the right.) The shape of the vibration mode is complicated. It would be a nontrivial matter (impossible?) to express this shape analytically (i.e. as a function of the three space coordinates that satisfies all the balance equations exactly). The vibration mode shape is color coded: dark blue color corresponds to places with very little displacement, while large displacements are marked with dark red color. So we can see for instance that the corners of the flange moved by a large amount (out of the plane of the flange), while there are multiple dark-blue spots (corresponding to the so-called nodes of the vibration mode shape).

Figure 2.3 shows the close-up of the top right corner of the flange. The color indicates that the neighborhood of the bolthole experiences displacements of both the lowest (dark blue) and the highest magnitude (dark red). The lines that separate colors (level curves of the displacement magnitude) are decidedly curved. That speaks to the complexity of the displacement pattern. The
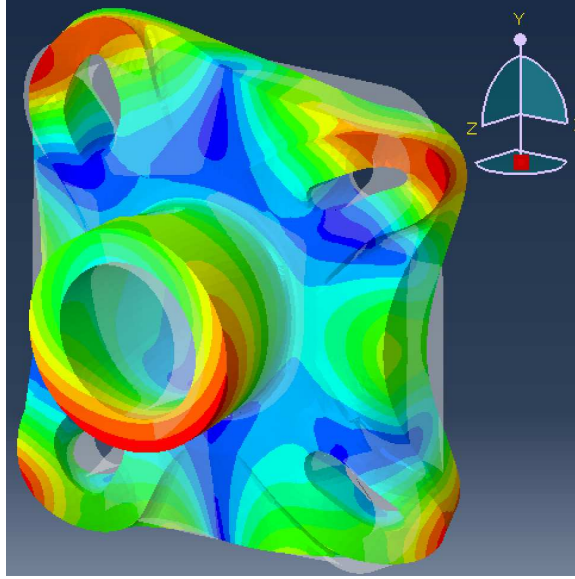
Fig. 2.2. Rexnord Heavy Duty Ball Bearing housing. Shape of vibration corresponding to the 24th natural frequency at approximately 24.8 kHz. The grey transparent surface is the undeformed shape, which is overlaid with the solid surface where magnitude of displacement is encoded as color: from blue (little to no motion) to red (large displacement).

hole is sheared, while at the same time the flange bends out of plane. Such a motion would be difficult to describe as some function of X, Y, Z over the entire neighborhood of the hole. For this reason the finite element method does not attempt to do this, but approaches the solution differently.
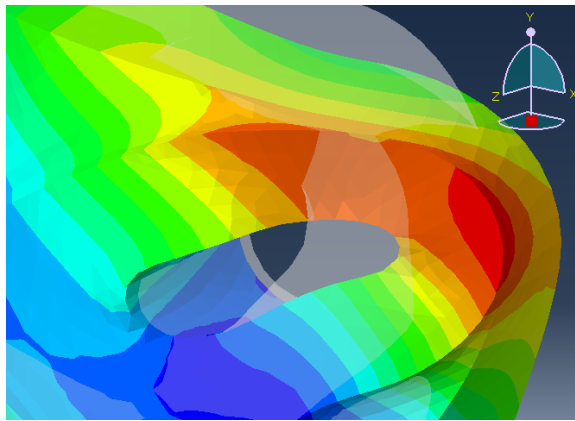
Fig. 2.3. Rexnord Heavy Duty Ball Bearing housing. Close-up of Figure 2.2, top right corner.

Animation
- video

    The first step is to cover the geometry with suitable simple shapes. In the present case they are solids with four triangular faces, the so-called **tetrahedra** (tetra= 4, hedron = base). The covering must be without gaps or overlaps and such that two tetrahedra can share either a face, an edge, a vertex, or nothing (when the tetrahedra are not next to each other). Figure 2.4 shows such a covering: this is the **finite element mesh**, where each tetrahedron is one finite element. Of course the figure actually shows only those triangular faces of the tetrahedra that are part of the outer surface.

    Of the tetrahedra shown in Figure 2.4 we select one, which is shown in Figure 2.5 by highlighting the selected tetrahedron in thick red line. If we focus on that single tetrahedron, we notice that
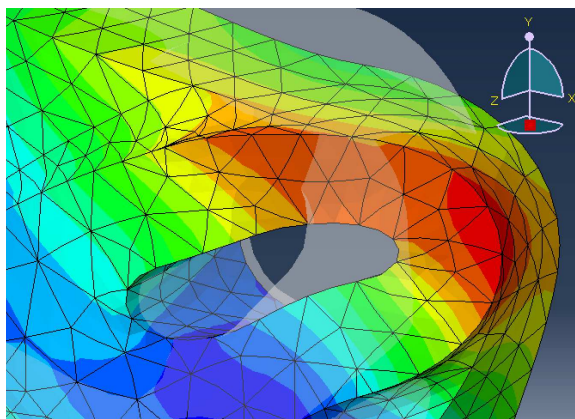
Fig. 2.4. Rexnord Heavy Duty Ball Bearing housing. As in Figure 2.3, but with the tetrahedral finite element mesh shown.

the displacement across that single element is apparently much simpler than across the hole in the flange or even across the entire part. This is easily seen from the level curves of the displacement magnitude: within the extent of the single triangle that we can see facing us we have a few level curves between darker green color, lighter green color, and yellow. That is a much smaller range of variation of displacement than from blue to red. Also the level curves are nearly parallel, which also indicates a relatively simple relationship for the displacements. The animation of the movement of the selected tetrahedron shows this quite well: the tetrahedron stretches, shears, rotates, and translates, but those are relatively simple motions. Compared to the motion shown for instance in Figure 2.3 it seems quite plausible that we would be able to describe the displacements across the single element in relatively simple terms.

Animation
- video
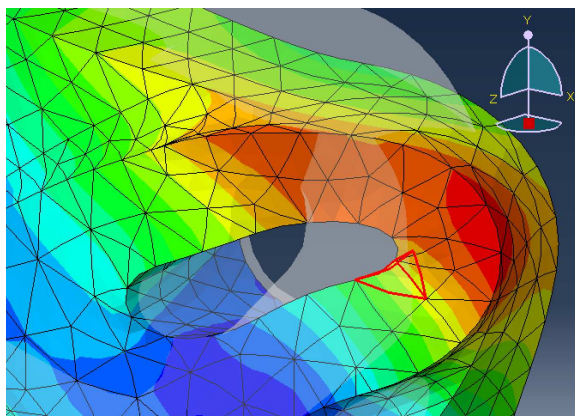
Fig. 2.5. Rexnord Heavy Duty Ball Bearing housing. As in Figure 2.4, but with the one tetrahedral finite element highlighted in thick red line.

That is indeed the case. The motion within any single tetrahedron is entirely described by giving the displacements of the so-called finite element **nodes** (shown with dots in Figure 2.6), with the displacements of the material at any point within the element being determined from the displacements of the nodes using interpolation with very simple, polynomial, functions: the so-called **basis functions**. So for the tetrahedron element, the task of describing *any* deformation of this element boils down to giving the displacements of ten points (nodes), and interpolating with known (given) functions of very simple form to obtain the displacement at an arbitrary point within the element. That is the essence of the finite element method.
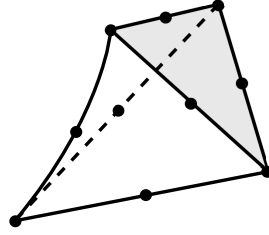
Fig. 2.6. A single tetrahedral finite element. The dots indicate the locations of the finite element nodes.

The finite elements also come in handy for a couple of other tasks. Firstly, we may wish to describe how the properties of the materials of the considered structure are distributed. In the present case, the bearing was made of a single material, but built-up structures are common (for instance reinforced-concrete/steel decks, or sandwich plates), and then we need to describe how the properties of the materials vary within the volume. The tiling with the simple shapes of the finite elements enables the description of how the material properties are distributed throughout the volume. In this context we may also mention composite materials where in addition to the distribution of the material we may also need to describe the orientations of the reinforcing fibers throughout the volume.

Secondly, the finite element method relies on the ability to evaluate volume and surface integrals. Also for this task the finite elements are perfect. They are very simple shapes for which simple integration rules exist, and therefore evaluating arbitrarily complex integrals becomes possible.

In the next section we will begin looking at some of the details of this process, but to simplify the setting somewhat we will consider the problem of heat conduction instead of stress analysis.

2.2 Thermally driven micro-actuator

Understanding the phenomena of heat transfer in solid structures is of considerable interest. The effects of the temperature that varies from point to point across the structure are twofold:

1. The change of the temperature from the reference value leads to the so-called thermal strains, which cause deformation and generate stresses.
2. The properties of the material may depend on temperature. As an example we may mention high-temperature softening of metals. Therefore also the deformation and the stress in the structure will change as a result of the change of the properties of its material.

Introduction to the simulation of the processes of heat transfer is the subject of the present chapter.

Let us start with a motivating example: simulation of the function of a thermally driven micro-actuator. The micro-actuator is an example of a structure of geometrical dimensions that are tiny compared to human scale. The entire actuator would fit into the space of 1 mm cubed (Figure 2.7). The actuator is etched from crystalline silicon, produced layer-by-layer. The termini are attached to contact plates which are part of the substrate, and the actuator is cantilevered from the termini. It is actuated by thermally-generated strains. The heat is produced by running electric current through the structure, either through the loop that consists of the inner legs, or through the loop that consists of the outer legs. When the voltage to generate the current is applied on the termini of the inner legs, the inner legs warm up more than the rest of the structure, and since the inner legs are on a lower level then the outer legs and since they get longer, the actuator bends upwards. If the voltage is applied to the termini of the outer legs, the outer legs warm up more than the inner legs, and since they get longer and since they are on a higher level than the inner legs the actuator bends downwards. Given the tiny size, the thermal inertia is very small, and the actuation can be performed at the rate of hundreds of cycles per second. Mechanical inertia can also be ignored, at least in the first approximation. Finally, we may assume that the silicon material properties do not change very much when the silicon is heated. Thus, the task of simulating the response of the actuator under various conditions, may be broken down into
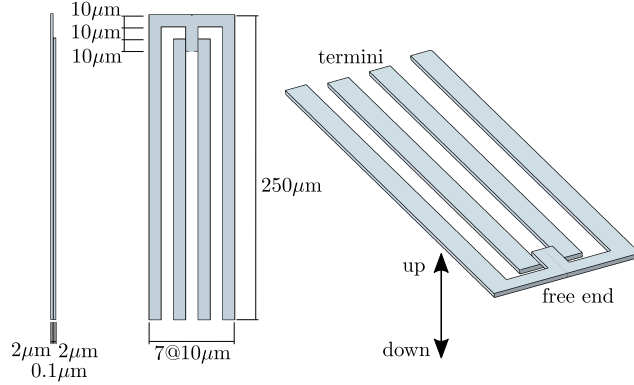
Fig. 2.7. Geometry of the micro-actuator

1. Find the distribution of temperature within the structure.
2. Find the deformation of the structure and the associated stresses.

The continuum mechanics models for these two tasks are well-understood. However, it is very unlikely that we will be able to find a closed-form solution. The shape of the structure is much too complicated for that. This is where the finite element method comes in. It does not find the exact solution to the mathematical problems posed by the two models, but it does find good approximations. And, it can do that for an arbitrarily complicated geometry. Also, it can do it even when the mathematical models become more complex still by inclusion of further mechanical and thermal effects: geometrical and material nonlinearities, temperature dependence of the material properties, and so on.

Figure 2.8 shows the relevant results for the micro-actuator for the loading produced by the heating of the inner leg loop. The temperature is shown in Figure 2.8(a), and the significant rise in temperature may be appreciated (roughly 1000^oC above reference temperature). Significant thermal strains are generated as a result, and the actuator deflects upwards by slightly less than 8μm (around 1/30 of the cantilevered length), as shown in Figure 2.8(b). Finally, in Figure 2.8(c) the maximum principal stress in absolute value is displayed. Checking that the maximum tensile stress remains below the fatigue limit of the material is a common design check.

Before we are done with this book, you will be able to set up the solution process for similarly complicated analysis tasks. But first we will start with the heat transfer model alone, as it is the easier of the two mathematical models.

Abaqus
- CAE file

2.3 The heat transfer problem

In the following sections we will discuss the basics of the finite element modeling in the context of steady-state heat conduction. The mathematical model of heat conduction is both simple and important: it is much simpler than the model for stress analysis, and at the same time thermal loads on structures are the second most important kind of loads (after "mechanical" loads). The mathematical model of heat conduction is developed in many textbooks, for instance [5]. The PDF version of this book also includes a description of what is needed to discuss the finite element solutions. See Box 1

In the examples in this chapter, the heat conduction model is described by the partial differential equation for the temperature T

$$-\kappa \operatorname{div}\left[(\operatorname{grad}T)^T\right] - Q = 0 \ , \tag{2.1}$$

that is expected to be satisfied at all points of the volume V. Here T is a function of the Cartesian coordinates x, y, z and represents the distribution of temperature across the volume of the part; Q is the rate of heat generation in units of power per unit volume; and κ is the so-called thermal
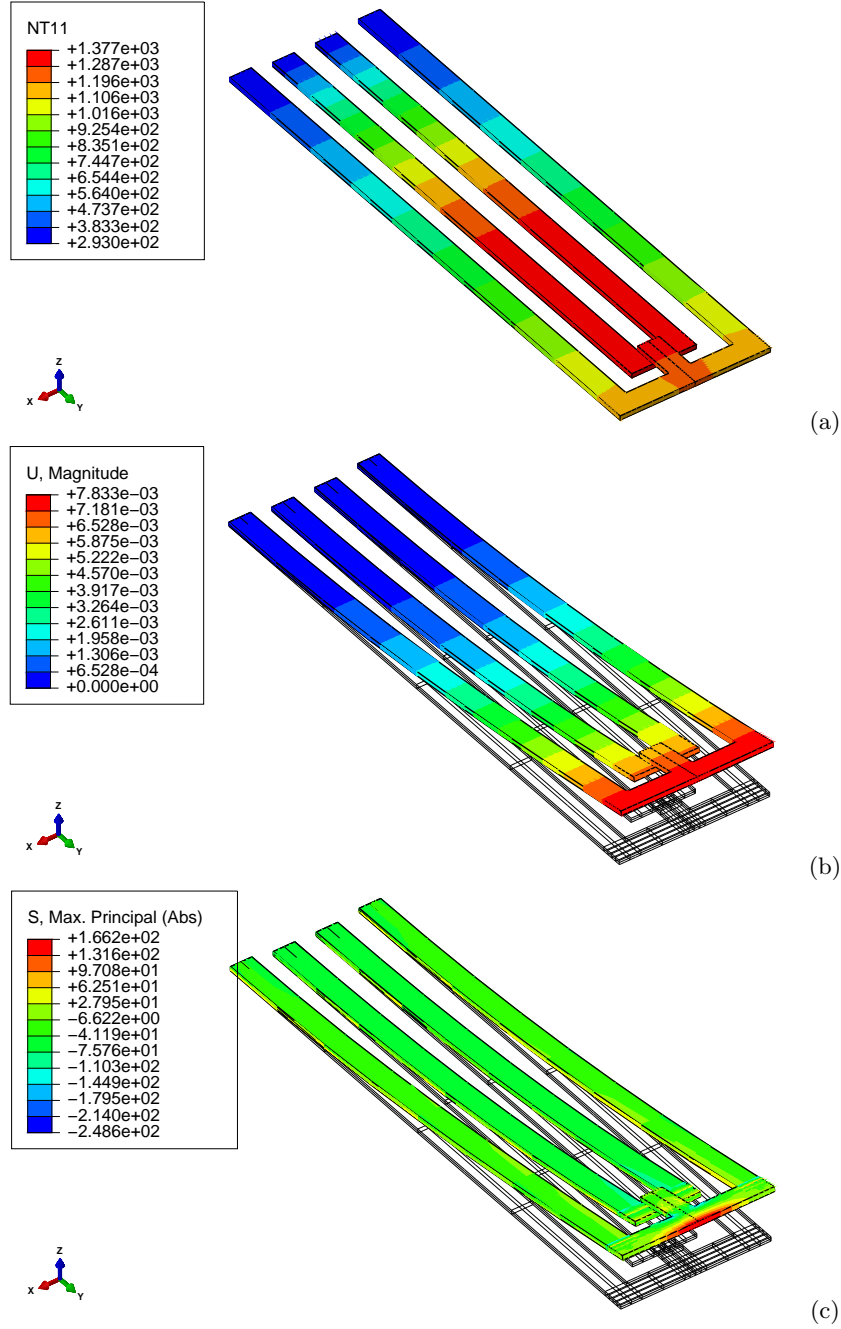
Fig. 2.8. Thermally driven micro-actuator. (a) Temperature in degrees Kelvin. (b) Deflection magnitude in millimeters (shape amplified by approximately a factor of three). (c) Maximum principal stress in absolute value in MPa.

conductivity, which is a property of the concrete material; the material is assumed to be the same everywhere within V (i.e. homogeneous). The div and grad are the divergence and gradient operators. Explicitly we write

$$-\kappa \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right) - Q = 0 \ . \tag{2.2}$$

This equation expresses the balance of heat energy, and if we find a solution, it will hold at every single point in the volume V.

The boundary conditions are the decisive part of the mathematical model: they can make the solution unique, and they are crucial in deciding whether or not the solution exists in the first place. Physically the boundary conditions are an expression of the interaction of the world with a subset of it that we wish to model. We always limit ourselves to a simple three-dimensional body. The interaction of this modeled three-dimensional body with the environment in which it exists (other three-dimensional bodies, the ambient media, and so on) is expressed through the interactions that the modeled body has with the not-modeled rest of the world. This interaction occurs across the boundaries.

In the first example we will consider the two simplest kinds of boundary condition, namely the temperature being known and the zero heat flux condition.(For a thorough discussion of boundary conditions see See Box 2 .)

2.4 Concrete column

We shall start by inspecting a simple finite element model. Figure 2.9 shows the distribution of temperature in a section of a concrete column. What is shown is not the geometry of the column itself, but rather an approximation of the geometry by a finite element mesh. The volume is actually a perfect truncated cylinder as shown in Figure 2.10 , where on the left is the volume, in the middle the top circular cross-section is highlighted, and on the right the side (cylindrical) surface is highlighted. (The volume naturally also has a bottom circular cross-section, but that is not highlighted in this figure.)

Abaqus
- CAE file
- tutorial
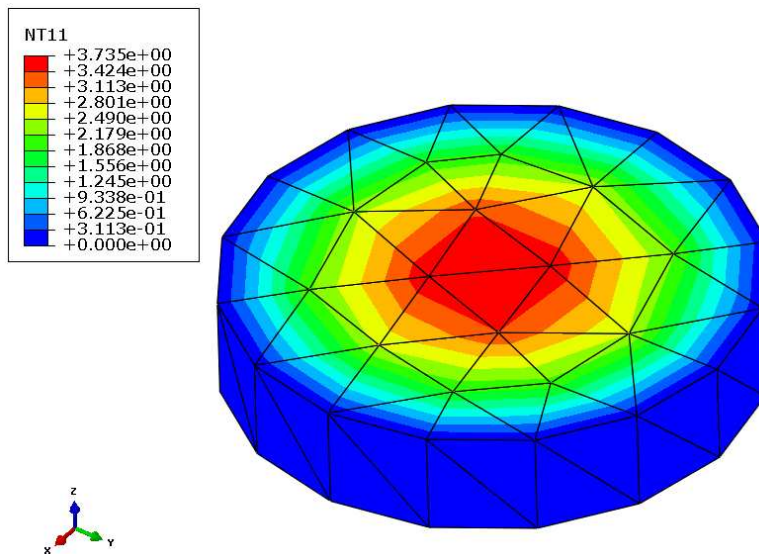
Fig. 2.9. Distribution of temperature in a piece of concrete column

For simplicity, we will call the volume of the actual geometrical shape V, and the volume of the approximate geometrical shape as represented by the mesh V^h.
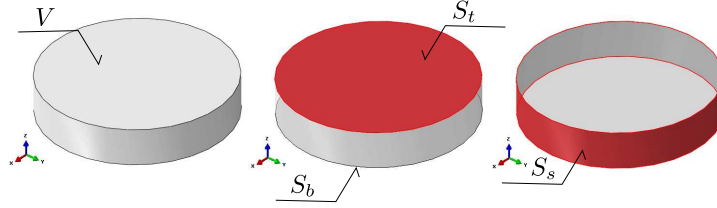
Fig. 2.10. The geometry of the concrete column

The temperature visualized with color coding in Figure 2.9 is also an approximation: the finite element software found an approximate solution of the mathematical model that describes the distribution of temperature at steady state (steady-state: when the temperature distribution does not change in time).

In addition to an equation that holds within the volume, the mathematical model also needs equations that describe how the objects in the space outside of the volume V influence what is happening inside V. The volume V is influenced by the material outside of it through its boundary. In particular, in this example we consider the short section shown in Figure 2.10 to be a fraction of a very long column that is submerged in water. The volume V is therefore influenced along the surface S_t by the material of the column above the surface and along the surface S_b by the material of the column below. Water is adjacent to the volume V along the surface S_s. The adjacent material can influence the response of the material of the volume V by bringing in or draining off heat energy, or it could in fact prevent exchange of energy (insulation). Only careful observation of the physical situation to be modeled can serve as a guide as to how to express the interaction of the volume V with its surroundings. Since the interaction of the volume with what is around it is described along its boundary surface, such conditions are called boundary conditions.

> Boundary conditions are an expression of our knowledge of the physical situation.

In the present example, the boundary conditions are described as follows: along the surface S_t and the surface S_b the derivative of the temperature along the direction of the normal to the surface is set to zero

$$\left.\frac{\partial T}{\partial n}\right|_{S_t} = 0 \quad \text{and} \quad \left.\frac{\partial T}{\partial n}\right|_{S_b} = 0 . \tag{2.3}$$

Along the surface S_s the temperature is assumed to be known to be equal to a constant, the temperature of the water, T_w,

$$T|_{S_s} = T_w . \tag{2.4}$$

These two mathematical statements are used to express the following conditions: It is deduced by observation that the temperature of the columns above surface S_t and below the surface S_b is the same as the temperature within the corresponding locations of volume V. Therefore the volume V exchanges no heat energy with the rest of the column and the conditions (2.3) are saying that the heat flow through the two surfaces is zero. Further, as the temperature of the water can be reasonably assumed to be constant along the surface of the entire column, and as the volume of water is assumed to be rather large so that the interaction with the column does not change the temperature of the water, we can assume that also the wet boundary of the volume V, that is S_s, will be at the temperature of water, T_w.

2.4.1 Boundary value problem solved

Equations (2.1), (2.3) , and (2.4) are a package that together defines the mathematical model. The mathematical model is known as a boundary value problem. The significance of the boundaries is clearly indicated in the name!

Some of these boundary value problems can be solved exactly by clever tricks and sophisticated mathematical techniques. The point of finite element methods is that they are applicable when no analytical techniques exist to solve these problems.

The analytical solution can be derived in cylindrical coordinates $\boxed{\text{See Box 3}}$ as $T(r) = \frac{Q}{4\kappa}\left(a^2 - r^2\right)$. For the temperature distribution in a concrete column of circular cross-section of radius $a = 2.5$m, immersed in water at $T_w = 0^o$ C, where the concrete gives off hydration heat at the rate of $Q = 4.5$ W/m^3, and the thermal conductivity of concrete is taken at $\kappa = 1.8$W/m/^{o}K, the temperature at the axis of the cylinder is 3.906^oC. The temperature varies only in the radial direction, not in the circumferential direction and not in the axial direction. Therefore the gradient of the temperature points directly to the axis of the cylinder (the temperature increases from the circumference towards the center). The maximum temperature gradient value obtains at the circumference, $\partial T(r = a)/\partial r = \frac{Q}{2\kappa}a = 3.125^o$C/m. The quantity that is used in thermal analyses to measure the flow of energy per unit area is the heat flux. By definition, the radial component of the heat flux would in this case be

$$q_r(r) = -\kappa\frac{\partial T(r)}{\partial r} \ . \tag{2.5}$$

The radial component of the heat flux at the circumference of the column would be $q_r(r = a) = 5.625$W/m^2.
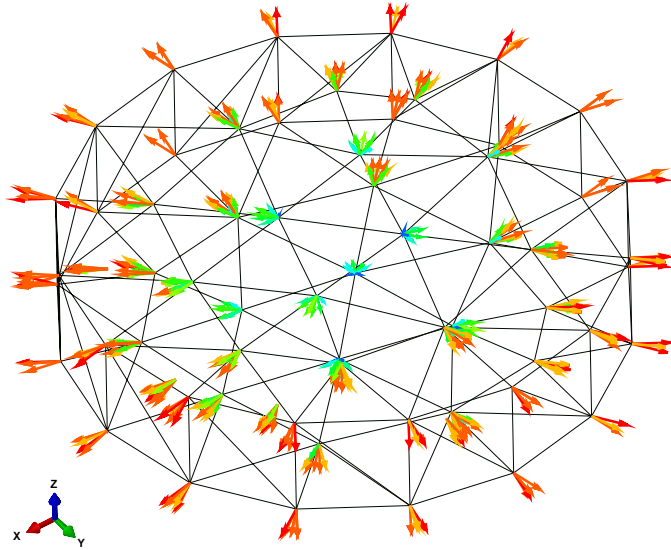
Fig. 2.11. The arrows corresponding to the calculated heat flux at the nodes, contributions from all elements connected to a node.

We can see immediately from Figures 2.9 and 2.11 that the finite element solution is not exact. Nevertheless, it is a fairly reasonable approximation. For instance, the maximum temperature computed in the FE solution is reported as 3.735^oC, and a query of the magnitude of the heat flux reported at the bottom-most node in Figure 2.11 gives the following values

```
4.57564, 5.05033, 5.23379, 4.66911, 4.63599, 5.23379, 4.71777
```

These values are not too far off from the analytical solution. At the same time, the approximate nature of the finite element solution is very clear:

1. The analytical solution is axially symmetric: the temperature does not change in any other direction but the radial. The finite element solution is not axially symmetric.
2. The radial variation of the temperature is a parabolic arc analytically, but it is a piecewise broken line in the finite element solution (refer to Figure 2.12(a)).
3. The analytical radial variation of the heat flux should be a straight line, increasing from zero at the axis of the cylinder to the maximum value at the circumference. The finite element solution is apparently piecewise constant (refer to Figure 2.12(b)).

(a)   (b)
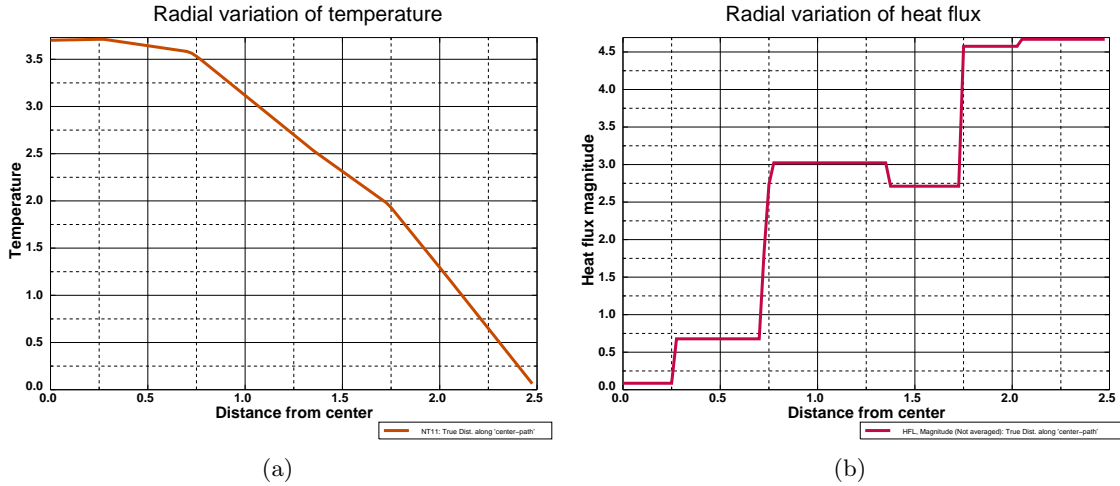
Fig. 2.12. The finite element solution for the temperature (left), and the heat flux magnitude (right). The solution is shown along a line at 45° to the x and y axis at the middle of the height of the cylinder.

2.4.2 What are finite elements?

Let us now focus our attention on Figure 2.13. It shows the finite element mesh. The mesh consists of 150 tetrahedral elements (tetra= four, suffix -hedron= used in geometry to form the names of solid figures bounded by a certain number of planes). The corners of the tetrahedra all come together at the so-called nodes – they are the vertices of the tetrahedra. Tetrahedra have four bounding triangular faces, six edges in-between the faces, and four nodes (vertices). One tetrahedron is shown with its nodes labeled in Figure 2.13. The elements are specified by listing the nodes that the element connects together in the proper order (more about the proper order later). Thus the tetrahedron shown in Figure 2.13 could be defined by listing the nodes 58, 31, 5, and 30.

The temperature is assumed to vary linearly across each element. Consequently we can write for the temperature variation within any element

$$T(x, y, z) = \alpha + \beta x + \gamma y + \delta z \tag{2.6}$$

where α, β, γ, and δ are coefficients which are particular to each and every element. In other words, each tetrahedron has its own set of these coefficients. Not all of these coefficients are independent. The temperature needs to be continuous across the entire mesh (as it is continuous in the analytical solution), so there must be some relationships between the sets of coefficient in one tetrahedron and its neighbors. Wherever two tetrahedra meet (at the vertex, along an edge, or along a face), the temperature described on one tetrahedron must match that described on the other.

This becomes much easier if instead of the general expression (2.6), the temperature distribution is described the "finite element way":
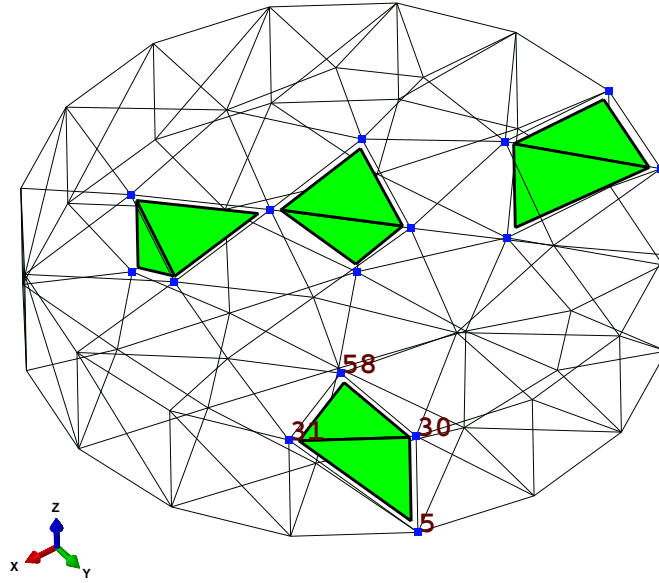
Fig. 2.13. The mesh. Edges of all the faces of the tetrahedra on the boundary of the domain are shown in thin solid line. Four tetrahedra are shown shrunken in solid color, and their nodes are shown with blue squares. One tetrahedron has its nodes labeled with numbers 58, 31, 5, and 30.
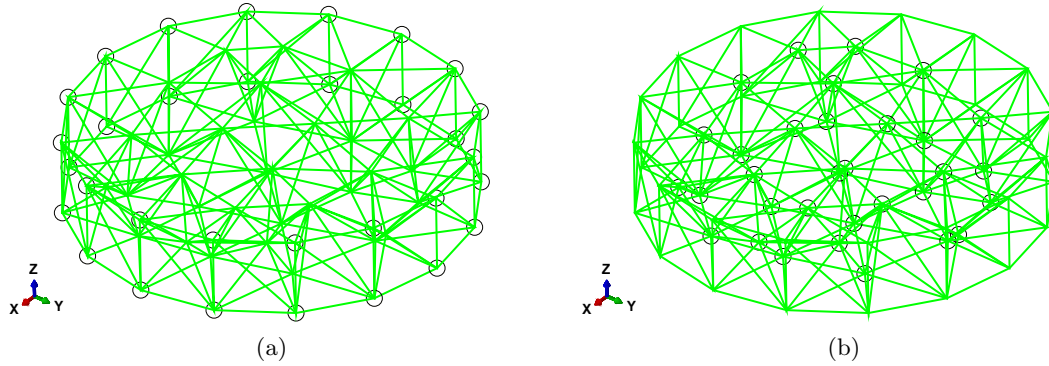
Fig. 2.14. The finite element nodes (the circles). (a) Nodes with known temperature values. (b) Nodes with unknown temperature values.

$$T(x, y, z) = \sum_i T_i N_i(x, y, z) \tag{2.7}$$

where T_i is the temperature degree of freedom i (we can think of this as attached to a node, but these details will be sorted out below), and $N_i(x, y, z)$ is the so-called **_basis function_** (a.k.a. shape function). The basis function is a linear function of x, y, z, and its main properties are that it is either zero or one at each of the nodes of the element.

The main advantages of the form (2.7) are that (A.) the value of the temperature at the node that carries the degree of freedom i is the coefficient T_i, (B.) to calculate the temperature at any other point is a matter of linear interpolation of the values at the nodes, and (C.) the resulting temperature function is naturally continuous from one tetrahedron to the next.

> For the four node tetrahedron the basis function is linear in x, y, z. The value of a basis function at a node is either zero or one— this is referred to as having the Kronecker delta property. The derivatives of the basis function with the respect to x, y, z are constant within the element.

For the tetrahedron it is relatively straightforward to calculate the form of such linear basis functions from the Kronecker delta property $\boxed{\text{See Box 4}}$, but there is a more natural way which will be discussed later.

Figure 2.14 shows two sets of nodes. (a) The temperature is known for nodes that are located on the boundary where the temperature value is prescribed (the cylindrical side surface S_s); and (b) the temperature is unknown at all the other nodes. The quantities that are associated with nodes are called the ***degrees of freedom***. The degrees of freedom at nodes that are at the boundary are called fixed, because their value is determined, before the calculation even starts, from the boundary conditions. The degrees of freedom that are unknown until the calculation finishes are the free degrees of freedom. Here "free" is used as the opposite of "fixed".

> The degrees of freedom (DOFs) are the values of the temperature at the nodes. Some DOFs are *fixed* by the boundary conditions, others needs to be calculated from the finite element equations: these are *free* (as opposed to fixed).

The expressions for the basis functions can be also easily differentiated to obtain the gradient of temperature at any point. For instance, the temperature values at the nodes 30, 31, and 5 are known to be zero because of the boundary condition (see Figure 2.13). The temperature at node 58 is 2.07121^o C as we find by querying the temperature in the Abaqus model. $\boxed{\text{See Box 5}}$ Furthermore we can query the coordinates of these four nodes, and therefore we can calculate the basis functions and their gradients. The magnitude of the heat flux can be readily calculated within the element 58, 31, 5, 30 to be 5.0503W/m^2. $\boxed{\text{See Box 6}}$ Importantly, the heat flux is the same everywhere within the element: it is uniform. That actually fits together with our earlier discussion of the temperature varying linearly across the element: a linear function has constant gradient, and the heat flux is proportional to the gradient. This explains why in Figure 2.12 there are multiple arrows of heat flux per node: as there are several elements connected to each node, and as each of the elements has its own value of the heat flux which is constant across the entire element, including at its nodes, when the elements are connected to the node, each element contributes an arrow which in general will be different from that of its neighbors.

The heat flux is proportional to the gradient of the temperature. Taking the gradient means taking derivatives. It is well known that in computing whenever a derivative is taken, accuracy is reduced. Compare Figures 2.9 (for the temperature) and 2.15 (for the heat flux magnitude): clearly the temperature is much better represented than the heat flux: the color coding shows that the temperature varies across each element, whereas the heat flux is constant within each element.

> The more derivatives we need to take, the less accurate the results will be. Temperature is more accurate than the gradient of temperature, because to get the gradient we need to differentiate the temperature. Consequently also the heat flux is less accurate than the temperature.

2.5 Hexahedral elements

The bewildering variety of finite elements available in typical finite element programs is due to a number of factors: there are no silver bullets[1], some elements are better than others in one situation, and the position is reversed in another situation. The 8-node hexahedron is a valuable element to

---

[1]A simple, seemingly magical, solution to a difficult problem.
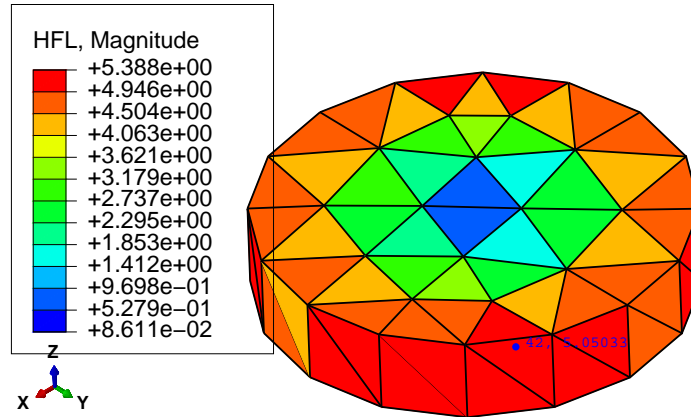
Fig. 2.15. The heat flux magnitude displayed with color coding for the entire mesh, and with annotation label for the element with nodes [58, 31, 5, 30] (heat flux magnitude 5.0503).

(a)   (b)
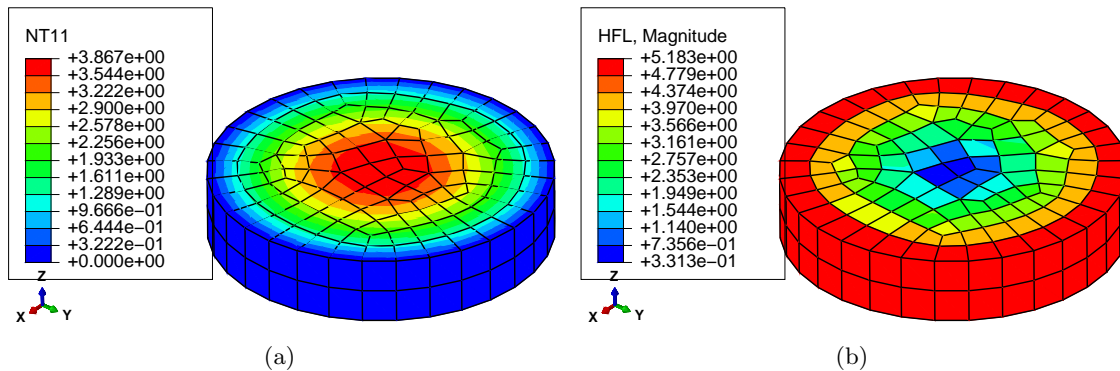
Fig. 2.16. The temperature and heat flux computed on a hexahedral mesh with elements of roughly the size of half the thickness of the slice.

have in our toolbox.

Figure 2.16 shows the temperature distribution and the heat flux magnitude for the finite element simulation using hexahedral elements of roughly half the size of the tetrahedra in Figure 2.9. Two important observations: Firstly, the result quality is better with the hexahedra, and that is a general observation that one can make with respect to the relative merits of these elements. Secondly, the smaller the elements (the more of them that we use), the more accurate the results become. Accuracy is measured in terms of how close we can get with the finite element solution to the true solution. The process of getting closer to the true solution is called convergence, and without convergence there would be no point in using the finite element method, because we couldn't rely on the results being relevant to our investigations.

We will consider the hexahedral (and their 2-d counterpart, quadrilateral) elements later. To start with tetrahedral (triangular, in 2-d) elements is preferable, as they are so much simpler.

Abaqus
- CAE file

2.6 Using symmetry

The geometry and the heat-transfer conditions (the material, the body load, the boundary conditions) possess two kinds of symmetry. Firstly, axial symmetry: the axis of the cylinder is an axis of symmetry. Secondly, translational symmetry: the solution is independent of the z coordinate.

Since the column has an axis of symmetry, any plane passing through the axis of symmetry is a plane of symmetry. This is important: the temperature distribution being symmetric with respect

to the plane of symmetry means that the gradient of temperature through the plane of symmetry is zero. This means that the heat flux through the plane of symmetry is known to be zero. Thus we get a natural boundary condition that we can use, we only have to be able to identify a plane of symmetry.

Since any plane passing through the axis is a symmetry plane, which one of the infinitely many planes do we pick? We can pick a single plane, effectively slicing the column in half. This is nice, the size of the problem is immediately reduced with a factor of two. But we can do better, we can pick two planes. The pie slice generated in this way could be of any interior angle. The smaller the angle the smaller the model (the fewer the elements in the mesh). However, we should not make the angle too small: small angle means poorly shaped elements, and poorly shaped elements lead to errors. So let us make it 15°.

Figure 2.17 illustrates the temperature distribution in two ways: (a) as a banded surface plot, and (b) as a collection of isosurfaces. Both plots highlight that the results from the FE calculations will converge towards the analytical representation. The level surfaces of temperature should be concentric cylinders of various radii. The level surface at temperature $T = 0^o$C is the outer surface of the concrete column (i.e.the interface between the column and the water).
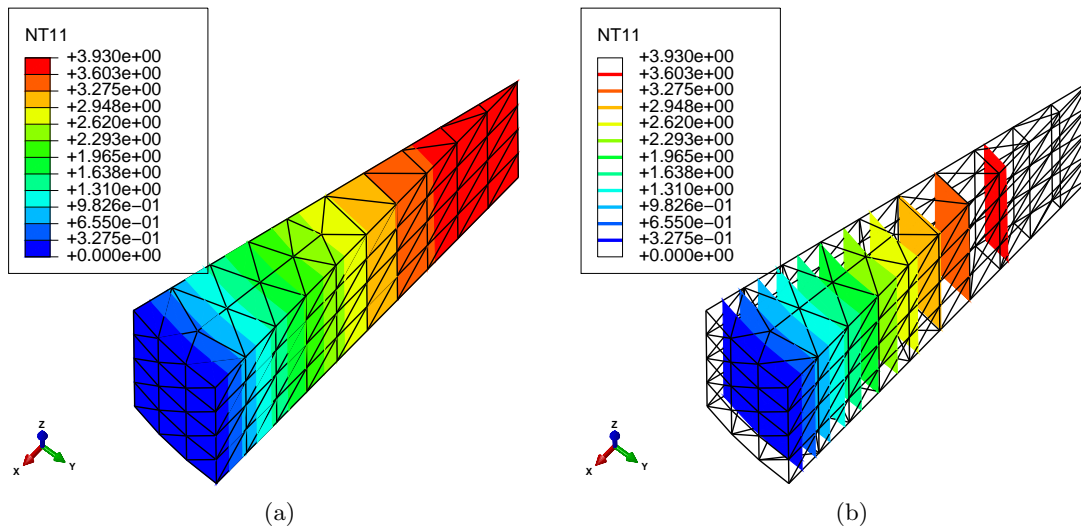
Fig. 2.17. The temperature for the 3-D model with two planes of symmetry used, element size 0.25 (maximum temperature 3.930).

Importantly, using the symmetry in this way leads to considerable savings. The number of elements is reduced by a factor of $360/15 = 24$.

> Using symmetry can lead to significant efficiencies. Later we will consider other uses for symmetry: it will allow us to apply boundary conditions. Refer to later chapters on stress analysis.

2.7 More symmetry: axisymmetric model

We can think of an axially symmetric geometry as being generated by rotating the so-called generating section through 360°. The generating section for the column part is a simple rectangle.

Since the temperature in the column example does not vary in the circumferential direction, we could formulate the finite element model in the two coordinates in the plane of the generating

section, radial r and axial z. (Actually, the temperature does not vary in the z direction either in the present example, but we shall ignore that for the moment.)

The finite element used in the axially symmetric model is based on the quadrilateral shape, as can be seen in Figure 2.18.
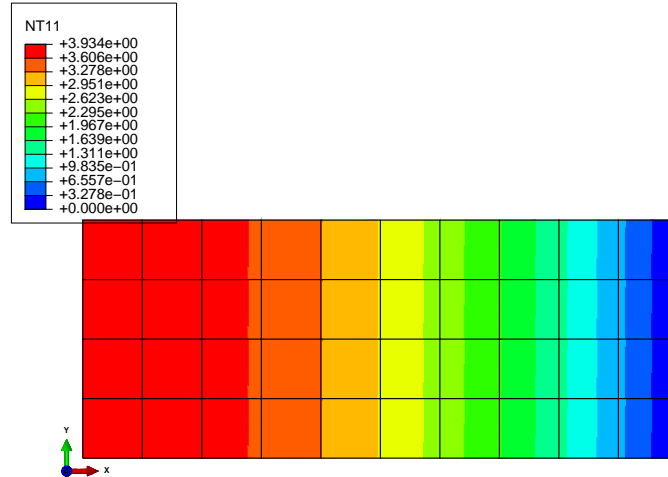
Fig. 2.18. The temperature for the axially symmetric model, element size 0.25 (maximum temperature 3.934).

When we calculate the heat flux at the middle of the height of the rectangle of the generating section, and plot the heat flux as a function of the radial distance, and we do that for elements of two different sizes, we get Figure 2.19. The analytical solution for the heat flux should be a straight line between zero at the axis of symmetry and the maximum at the circumference. Clearly the zigzag line is an approximation of such a straight line. This simple visualization allows us to say something about convergence: How far off is the finite element solution for the heat flux from the true value? Within an element the heat flux is a constant value (horizontal line in the figure). The departure of an inclined line of the true solution from the horizontal is the error, and we can see that the error is proportional to the size of the element (we can use the order-of notation: the error is approximately proportional to the element size h, or in mathematical notation $O(h)$). The smaller the element, the smaller the error. It is thus that we get convergence: by using smaller elements, the error is reduced. We will see later that the error of the temperature itself is proportional to the size of the element squared. We will understand that the temperature error would therefore decrease much quicker than the error of the heat flux: The square of a small quantity (element size) is much smaller than the quantity itself.

2.8 Even more symmetry: 2-D model in the cross-section

The axially symmetric model of the previous section managed to reduce the number of model coordinates from three to two: the radial distance and the axial coordinate. The present approach will also manage such a reduction, but in this case the two coordinates will be the in-plane Cartesian coordinates of points in the cross-section of the column. In a way this is natural, since we already discussed that the temperature field was independent of the z coordinate.

Figure 2.20 shows the Abaqus solution on a 30° pie-slice shaped model of the cross-section. Three-node triangles are used with the mesh size of 0.25 (10 element edges along the radius). The improvement of accuracy with respect to the coarser tetrahedral mesh discussed previously is obvious.
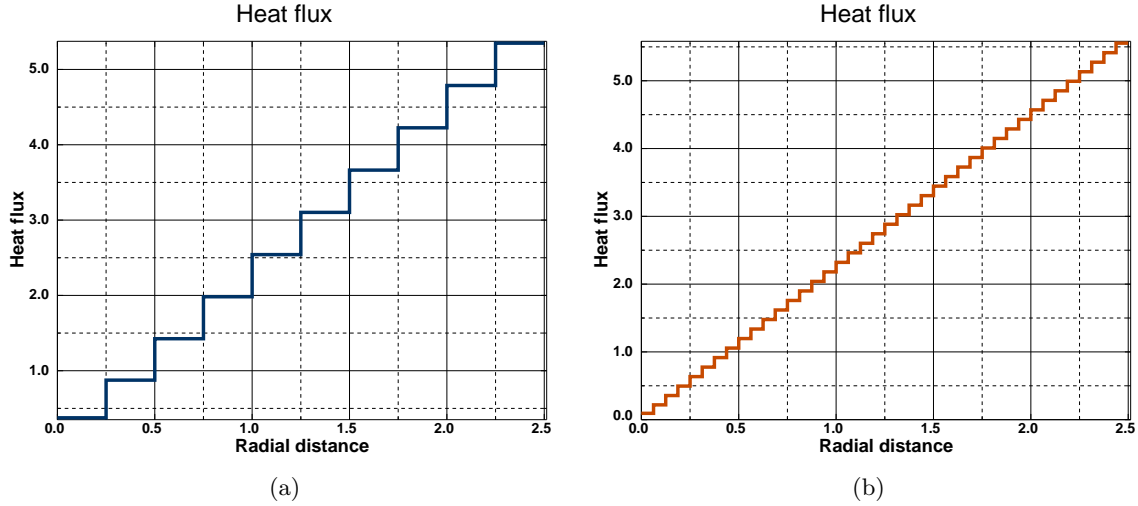
Fig. 2.19. The heat flux for the axially symmetric model along the middle of the height, element size 0.25 and 0.0625.

The three-node triangle will be the first finite element discussed in the next chapter. It is really the simplest element in more than one dimension, and therefore conducive to hand calculations. Even so, the technique becomes quickly impossible to use without a computer. Fortunately, the finite element methodology is ideally suited for computer execution.
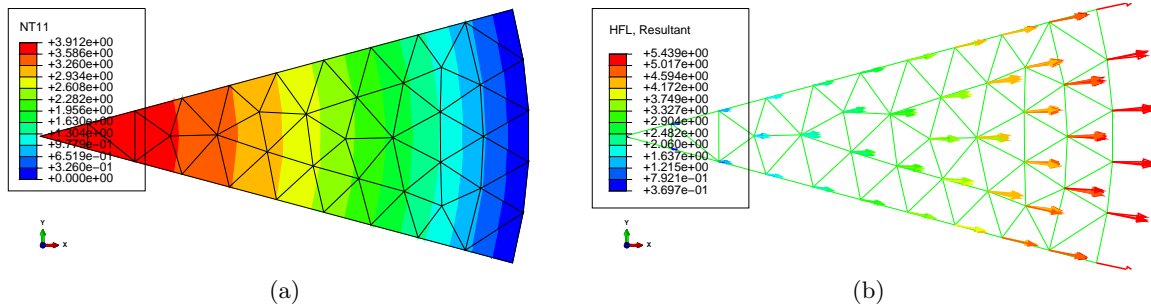
Fig. 2.20. Temperature and heat flux for the 2d pie-slice model, element size 0.25. (a) The temperature, (maximum temperature 3.912); (b) Heat flux resultant vector.

2.9 Modeling with film condition

Consider again the concrete column from Section 2.4. It is often the case that a solid exposed to a fluid medium along its surface does not have precisely the same surface temperature as the temperature of the fluid medium. There is a difference, a temperature jump, and the jump is often related to how much heat can flow through a layer of fluid right next to the solid surface through a coefficient that can be determined by experiment, the surface heat transfer coefficient (in Abaqus called the film coefficient).

This observation leads to a formulation of the convection boundary condition

$$q_n = h_o(T - T_w) \,, \tag{2.8}$$

where T is the temperature of the surface of the solid, and the normal component of the heat flux is therefore proportional to the jump of the temperatures between the solid and the surrounding

medium. ⬛ See Box 2 ⬛ In this example the convection boundary condition is applied on the side cylindrical surface. The ambient temperature is again taken as 0^oC, and the surface heat transfer coefficient (film coefficient) is $h_o = 5.0 \text{W/m}^2/^o\text{K}$.

Figure 2.21 shows the distribution of the temperature. Comparison should be drawn with Figure 2.9 which shows results for the prescribed-temperature boundary condition. The difference is really only a shift of the entire temperature field by approximately 1.06^oC. The heat flux picture is essentially the same as for the boundary conditions with fixed temperature.

It is of interest to compare the magnitude of the largest heat flux in the volume of the column (5.37) with the heat flux through the film condition (i.e. the convection boundary layer): the temperature drop is 1.06^oC, which when multiplied with the film coefficient h yields $1.06 \times 5.0 = 5.3$. The close agreement of these two numbers is no coincidence: what comes into the boundary layer must also come out, there is no accumulation of heat possible.
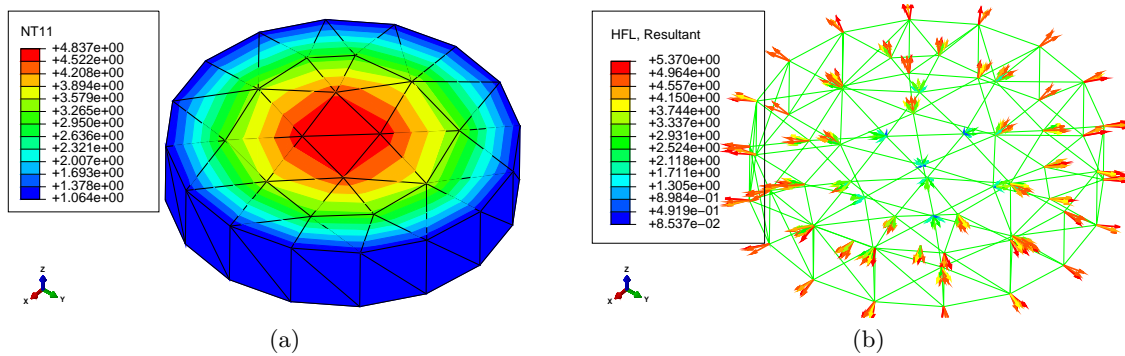
Fig. 2.21. Concrete column with convection boundary condition. (a) Distribution of the temperature. Note the nonzero temperature on the boundary. (b) Heat flux displayed as symbols (arrows).

2.10 Bird's-eye view of FEA

Finite element analysis may be viewed as part of several phases of a process. The phases may be summarized as:

1. Formulate objectives, scope, deliverables.
2. Choose mathematical model and idealizations.
3. Set up FE model(s).
4. Verification and validation.
5. Error control.
6. Interpret results, make predictions.

2.10.1 Phase 1: Formulate objectives, scope, deliverables

The goal here is to crisply state what needs to be done and why. Nothing is worse than a mushy and ambiguous request "to do FEA". It is really very hard or impossible to reach such movable goalposts, and the scope has a tendency to balloon as the work progresses: a never ending job.

2.10.2 Phase 2: Choose mathematical model and idealizations

Engineering models of mechanical processes (which we will loosely interpret here to include heat conduction, and other types of coupling effects to connect mechanical response of the structure to other loads) are typically expressed in terms of rate of change. A really simple example may be an

axial member in a truss structure: the rate of change of the displacement (strain) produces stress. The balance of the structure is expressed in terms of the rate of change of the stress. Other models have similar characteristics. For instance in heat conduction the balance of power is expressed in terms of the rate of the flow of power (heat flux). And so on.

Mathematical models are chosen as compromises. They need to be simple enough to allow us to compute solutions, yet sufficiently complicated to capture the physics of the response of the structure. An example of a mathematical model in common use is linear elasticity: In this simple couple of words we can express the basic parameters of the model. We are saying that the response of the material is elastic (reversible), and the displacements and strains are small. Other types of models in common use include, for instance, linear thermoelasticity (linear elasticity with effects of thermal strains), or linear elastodynamics (linear elasticity of dynamic response). These verbal statements can also be provided by listing the mathematical equations that describe the models. Evidently, we can be as precise as necessary in this way.

Since the models need to be sufficiently simple, one usually needs to adopt idealizations. These may include choosing the dimension of the model (beam, shell, solid), material response (reversible or irreversible, such as plasticity), simplification of geometry, and, importantly, choice of the so-called boundary and initial conditions.

2.10.3 Phase 3: Set up FE model(s)

Once the mathematical models and the idealizations have been selected, we need to concern ourselves with how to obtain the approximate solutions of these models (exact, analytical solutions are not available, in general). The finite element method (FEM) is one of the tools that can be used, and there are others. However, FEM *is* a quite *common* approach: in analyses of structures it is at least an order of magnitude more likely to be used than anything else.

The finite element model needs to respect the peculiarities of the mathematical model and the structure to be modeled. We need to consider geometry, the expected response of the structure, which features of the structure and the loading are important versus unimportant, the response of the material, connections to adjacent parts, and other aspects.

2.10.4 Phase 4: Verification and validation

It is important to realize that the finite element model is solved with a computer program or programs. The solution relies on multiple layers of software: from the bottom where basic instructions for manipulating binary data is executed, through layers of linear algebra and other libraries, to the finite element program and to layers of dialogues and graphical displays exposed to the user. The task of ascertaining that the finite element model was solved correctly is called verification. Any reputable finite element program would come with extensive verification suites, which would compare the solution obtained with the software to other independent solutions.

The task of validation is then to try to address the question whether the physical situation at hand was solved in a way that provides relevant and reliable answers. In other words, was the correct model used?

2.10.5 Phase 5: Error control

In the process of obtaining some quantitative answers in this way there will always be errors. The errors we need to concern ourselves with are the discretization error, the solver (a.k.a. computer arithmetic) error, the modeling error, the error due to the inputs, and the experimental error.

In this course we will be mostly concerned with the discretization error: error due to the use of the finite element method to solve the mathematical model. However, in general all sources of error needs to be considered when entering the last phase of the process, the interpretation of results.

2.10.6 Phase 6: Interpret results, make predictions

Provided we have gained some control over the errors, we can try to undertake interpretations of the quantitative data resulting from the solution process. We can make predictions of performance of a structure which doesn't exist yet, explain the performance of an existing structure, and so on.

Iteration

It is important to realize that we may need to cycle back to previous phases. For instance, in order to control the modeling error, we may have to choose a better mathematical model. We will find out whether or not the model is good enough in Phase 4, so in order to change the mathematical model we need to come back to Phase 2. Multiple such iterative changes may be required, including refining the statement of purpose in Phase 1. At some point we will accept the limitations of the current models, and the report and deliverables will be written and submitted as the result of the modeling process. It is necessary to realize that the result will never be perfect, but it needs to be good enough.

How to do *good* FEA?

- Success is conditioned on an in-depth knowledge of the principles.
- It is necessary to have a feel (understanding) of the physical artifact or process in order to model it. FEA requires creativity and imagination.
- Modeling is by necessity an iteration.
- FEA is just one part of the entire process, and understanding the whole process is essential.
- Bad FEA is easy; to do FEA well requires know-how, time, and effort.

2.11 Background, explanations, details

Box 1. Heat conduction PDE

The excellent thermal analysis textbook by Lienhard and Lienhard [5] has all the details one might require to supplement the treatment that follows.

Our first goal is to derive the balance equation that describes heat conduction in solids as a partial differential equation (PDE). To begin, we pick a control volume, and we keep track of the heat energy within that volume. The control volume may be the whole structure, part of the structure, or just a very small chunk of material surrounding a given point in space (Figure Box 1(a)). The amount of heat energy in the control volume U is expressed as an integral of the volume density of heat energy, u

$$U = \int_V u \, \mathrm{d}V \tag{2.9}$$

The amount of heat energy within the control volume may change by *outflow* (inflow) of heat energy via the boundaries, and *heat generation* (or loss) within the volume. These quantities will be expressed in terms of rates. Therefore, the amount of energy flowing out of the control volume through its bounding surface S per unit time is

$$\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \, , \tag{2.10}$$

where $\boldsymbol{n}$ is the outer normal to the surface S, and $\boldsymbol{q}$ is the heat flux (amount of heat flowing through a unit area per unit time). The amount of energy generated within the control volume per unit time is
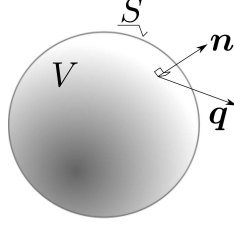
Fig. 2.22. The domain for the heat conduction problem

$$\int_V Q \, \mathrm{d}V \,, \tag{2.11}$$

where Q is the rate of heat generation per unit volume; for example, heat is released or consumed by various deformation and chemical processes (as work of viscous stresses, reaction product of curing concrete or polymer resins, and so on).

Collecting the terms, we can write for the change of the heat energy within the control volume the rate equation

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \,. \tag{2.12}$$

Finally, differentiating U with respect to time will be possible if we assume that $U = U(T)$, i.e. if U is a function of the absolute temperature T. Holding the control volume fixed in time, the time differentiation may be taken inside the integral over the volume

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t} \int_V u \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V \,, \tag{2.13}$$

and with the application of the chain rule, the relationship (2.13) is expressed as

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}T} \frac{\partial T}{\partial t} \, \mathrm{d}V \,. \tag{2.14}$$

The quantity $c_V = \mathrm{d}u/\mathrm{d}T$ is a characteristic property of a solid material (called specific heat at constant volume). It is typically dependent on temperature, but we will assume that it is a constant; otherwise it leads to nonlinear models.

Substituting, we write

$$\int_V c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \,. \tag{2.15}$$

This equation consists of volume integrals and a surface integral. If all the integrals were volume integrals, over the same volume of course, we could proclaim that the integral statement (sometimes called a global balance equation) would hold provided the integrands satisfied a so-called local balance equation (recall that to get the local balance equation is our goal). For instance, from the integral statement

$$\int_V \alpha \frac{\partial M}{\partial t} \, \mathrm{d}V = \int_V \mu \, \mathrm{d}V \,, \tag{2.16}$$

where α, M, and μ are some functions, one could conclude that

$$\alpha \frac{\partial M}{\partial t} = \mu \,, \tag{2.17}$$

which is a local version of (2.16). An argument along these lines could for instance invoke the assumption that the volume V was arbitrary, and that it could be shrunk around a given point, which in the limit would allow the volume to be canceled on both sides of the equation.

To execute this program for equation (2.15), we have to convert the surface integral to a volume integral. We have the needed tool in the celebrated ***divergence theorem*** (also known as the Gauss theorem)

$$\int_V \text{div}\boldsymbol{q} \, \mathrm{d}V = \int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \, , \tag{2.18}$$

where the divergence of the flux vector is defined in Cartesian coordinates as

$$\text{div}\boldsymbol{q} = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \, .$$

Consequently, Eq. (2.15) may be rewritten

$$\int_V c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = -\int_V \text{div}\boldsymbol{q} \, \mathrm{d}V + \int_V Q \, \mathrm{d}V \, , \tag{2.19}$$

and grouping the terms as

$$\int_V \left[c_V \frac{\partial T}{\partial t} + \text{div}\boldsymbol{q} - Q \right] \, \mathrm{d}V = 0 \, . \tag{2.20}$$

we may conclude that the inside of the bracket has to vanish since the volume could be entirely arbitrary. Therefore, we arrive at the ***local balance equation***

$$c_V \frac{\partial T}{\partial t} + \text{div}\boldsymbol{q} - Q = 0 \, . \tag{2.21}$$

Equation (2.21) contains too many variables: both temperature and heat flux. Since it is a scalar equation, the logical step is to express the heat flux in terms of temperature. That is the content of the Fourier model: heat flows opposite to the gradient of the temperature (downhill). In matrix form

$$\boldsymbol{q} = -\boldsymbol{\kappa}(\text{grad}T)^T \, . \tag{2.22}$$

The matrix $\boldsymbol{\kappa}$ is the conductivity matrix of the material. The most common forms of $\boldsymbol{\kappa}$ are

$$\boldsymbol{\kappa} = \kappa \mathbf{1} \tag{2.23}$$

for the so-called thermally isotropic material, and

$$\boldsymbol{\kappa} = \begin{pmatrix} \kappa_x & 0 & 0 \\ 0 & \kappa_y & 0 \\ 0 & 0 & \kappa_z \end{pmatrix} \, , \tag{2.24}$$

for materials that have three orthogonal directions of different thermal conductivities (orthotropic material); κ is the isotropic thermal conductivity coefficient, $\mathbf{1}$ is the identity matrix, and κ_x, κ_y, and κ_z are the orthotropic thermal conductivities. To explain the orthotropic conductivity model we note that some materials have preferred directions in which heat would like to flow, for instance along the fibers in a composite. Visually, we can imagine a corrugated steel roof, with the channels running not directly downhill, but tilted away from the slope – the water would run preferentially in the channels, but generally downhill.

The funny looking transpose of the temperature gradient follows from the definition: the gradient of the scalar is a row matrix

$$\text{grad}T = \left[\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}, \frac{\partial T}{\partial z} \right] \, . \tag{2.25}$$

With the constitutive equation, the balance equation (2.21) is now expressed purely in terms of the absolute temperature,

$$c_V \frac{\partial T}{\partial t} - \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] - Q = 0 \, . \tag{2.26}$$

Box 2. Heat conduction boundary conditions

Here we consider V to be the volume of the whole solid domain. The most important fact about the boundary conditions is that we need to have a boundary condition *at each point* of the surface S which bounds this volume (see Figure Box 1). The heat conduction model is all about temperature. Correspondingly, the boundary conditions are an expression of our *a priori* knowledge of the temperature distribution in the solid and on its surface.
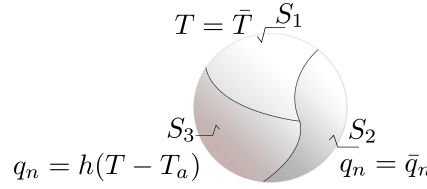
Fig. 2.23. The partitioning of the boundary surface used in the description of the boundary conditions. Note that the boundary surface of the volume V may consist of multiple patches of either type S_1, or type S_2, or type S_3. Together these patches cover the entire surface.

The simplest boundary condition results if we know the surface temperature along one part of S at all times. This part of the surface will be called S_1 (see Figure Box 2(a)). Therefore,

$$T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \quad \boldsymbol{x} \text{ on } S_1 . \tag{2.27}$$

This type of condition is known as the primary, or **essential**, boundary condition. Sometimes it is also referred to as the boundary condition of the first kind, or the Dirichlet boundary condition.

The heat flux entering or leaving the solid may also be known (measured by a heat flux gauge, for instance). Generally, we do not know the heat flux *along* the surface, only the *normal* component, which is available from the normal to the surface and the heat flux vector as $q_n = \boldsymbol{n} \cdot \boldsymbol{q}$. Therefore, along the S_2 part of the surface the normal component of the heat flux may be prescribed

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = \overline{q}_n, \quad \boldsymbol{x} \text{ on } S_2 . \tag{2.28}$$

All quantities are given at a particular point on the boundary as functions of time, similarly to the first boundary condition. This type of condition is known as the **natural** (or flux) boundary condition. Sometimes it is also referred to as the boundary condition of the second kind, or the Neumann boundary condition.

As the last example of a boundary condition, we will mention heat transfer driven by a temperature difference across a surface. The normal component of the heat flux is given as

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = h(T - T_a), \quad \boldsymbol{x} \text{ on } S_3 , \tag{2.29}$$

where T_a is the known temperature of the surrounding medium (ambient temperature), and h is the surface heat transfer coefficient. This boundary condition is often used in the so-called forced-convection situations where a solid body is exposed to a forced flow of some fluid medium at a given temperature. As a result of the sticking of the fluid to the solid a flow boundary layer develops. In general yet another boundary layer develops, a thermal boundary layer, over which there will be a temperature gradient between the fluid outside of the boundary layer and the surface of the wetted solid body. The inverse of the thermal resistance of the boundary layer is expressed through the surface heat transfer coefficient which depends on the many parameters involved (the properties

of the surface of the body, the properties of the fluid, the properties of the convection flow, etc.). Sometimes it is also referred to as the boundary condition of the third kind, or the Newton boundary condition. In Abaqus this boundary condition is known as the convective film condition. Since this kind of boundary condition has application also in other situations such as modeling of the thermal resistance of contact between two solid bodies, or modeling of thin insulation layers, as surfaces with thermal gradients across them, we will refer to this condition as the **surface heat transfer** boundary condition.

End Box 2

Box 3. Analytical solution for the cylindrical domain

Consider a cylinder of radius a. In cylindrical coordinates, where r is the radial distance from the axis of symmetry, ϕ is the circumferential angle, and z is the coordinate along the axis of the cylinder, the differential equation

$$\frac{1}{r}\frac{\partial}{\partial r}\left(\kappa r \frac{\partial T}{\partial r}\right) + Q = 0 \tag{2.30}$$

describes the variation of temperature in the radial direction. The solution for uniform heat generation rate Q and homogeneous material κ, with the two boundary conditions

$$\frac{\partial T(r=0)}{\partial r} = 0, \quad T(r=a) = T_w = 0 \ , \tag{2.31}$$

is

$$T(r) = \frac{Q}{4\kappa}\left(a^2 - r^2\right) \tag{2.32}$$

End Box 3

Box 4. Basis function for the tetrahedron

The basis functions for the T4 tetrahedron $KLMP$ are linear functions with four coefficients, for instance

$$N_K(x,y,z) = a_K x + b_K y + c_K z + d_K$$

The interpolation conditions are written

$$N_K(x_J, y_J, z_J) = a_K x_J + b_K y_J + c_K z_J + d_K = \delta_{KJ} \ ,$$

where δ_{KJ} is the Kronecker Delta,

$$\delta_{ik} = \begin{cases} 1, \text{ if } i = k; \\ 0, \text{ otherwise.} \end{cases}$$

The relationship written for all four nodes results in

$$\underbrace{\begin{bmatrix} x_K, \ y_K, \ z_K, \ 1 \\ x_L, \ y_L, \ z_L, \ 1 \\ x_M, y_M, z_M, 1 \\ x_P, \ y_P, \ z_P, \ 1 \end{bmatrix}}_{\boldsymbol{X}} \underbrace{\begin{bmatrix} a_K, \ a_L, \ a_M, \ a_P \\ b_K, \ b_L, \ b_M, \ b_P \\ c_K, \ c_L, \ c_M, \ c_P \\ d_K, \ d_L, \ d_M, \ d_P \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} 1, 0, 0, 0 \\ 0, 1, 0, 0 \\ 0, 0, 1, 0 \\ 0, 0, 0, 1 \end{bmatrix}}_{\boldsymbol{1}}$$

Consequently, the coefficients of the basis functions can be computed as

$$\boldsymbol{A} = \boldsymbol{X}^{-1}$$

and in particular the matrix of basis function gradients with respect to x, y, z are the first three columns of $\boldsymbol{A}^T$.

End Box 4

Box 5. Query the coordinates of the nodes 58, 31, 5, 30

Abaqus prints out:

```
Node: PART-1-1.58
                          1              2              3         Magnitude
Base coordinates:  1.00788e+000,  1.38768e+000,  1.00000e+000,       -
No deformed coordinates for current plot.

Node: PART-1-1.31
                          1              2              3         Magnitude
Base coordinates:  1.76777e+000,  1.76777e+000,  1.00000e+000,       -
No deformed coordinates for current plot.

Node: PART-1-1.5
                          1              2              3         Magnitude
Base coordinates:  9.56709e-001,  2.30970e+000,  0.00000e+000,       -
No deformed coordinates for current plot.

Node: PART-1-1.30
                          1              2              3         Magnitude
Base coordinates:  9.56709e-001,  2.30970e+000,  1.00000e+000,       -
No deformed coordinates for current plot.
```

End Box 5

Box 6. Calculation of the temperature gradient in element 58, 31, 5, 30

Here is a bit of Python code to calculate the coefficients of the basis functions of element 58, 31, 5, 30 from the known coordinates given in Box 5. This matrix lists those coordinates in the first three columns.

```
X = array([[1.00788e+000,  1.38768e+000,  1.00000e+000,     1.0],
       [1.76777e+000,  1.76777e+000,  1.00000e+000,     1.0],
       [9.56709e-001,  2.30970e+000,  0.00000e+000,     1.0],
       [9.56709e-001,  2.30970e+000,  1.00000e+000,     1.0]])
```

This matrix then will hold the coefficients of the four basis functions as described in Box 4

```
A = linalg.inv(X)
print(A)
```

i.e.

```
[[-0.75259342  1.28043508  0.          -0.52784166]
 [-1.1263432   0.07106261  0.           1.0552806 ]
 [ 0.          0.         -1.          1.          ]
 [ 3.32152779 -1.38913707  1.         -1.93239072]]
```

The gradients of the basis functions then follow from

```
gradN = A[0:3,:].T
print(gradN)
```

as

```
[[-0.75259342 -1.1263432   0.          ]
 [ 1.28043508  0.07106261  0.          ]
 [ 0.          0.         -1.          ]
 [-0.52784166  1.0552806   1.          ]]
```

where the first row holds the gradient of the basis function N_{58} and so on. Given that the temperatures at the nodes are $T_{58} = 2.07121$, and $T_{31} = T_5 = T_{30} = 0$, the gradient of temperature results as

```
gradT = 2.07121*gradN[0, :]
print(gradT)
```

i.e.

```
[-1.55877901 -2.3328933   0.          ]
```

and the heat flux magnitude 5.05033399533 as obtained from

```
print(linalg.norm(-1.8*gradT))
```

which may be readily compared with Figure 2.11.

End Box 6

3

FEA for 2-D Heat conduction with Triangles

3.1 Model in two coordinates

In this section we show how the originally three-dimensional model can be reduced to just two active coordinates. The reduced model will still describe the heat conduction through a *three-dimensional domain*; the function describing the temperature distribution will depend only on *two* spatial coordinate variables though.
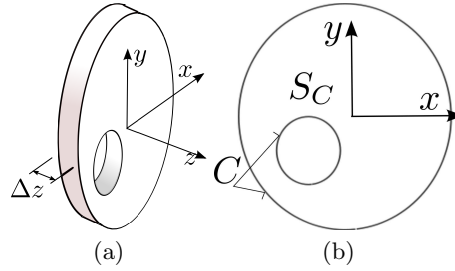
Fig. 3.1. The geometry of a domain in which the temperature does not change with the z coordinate. (a) Slice of a long cylindrical structure. (b) The cross-section: S_c=the interior, C=the boundary curves.

Under certain conditions, the temperature can be assumed to be independent of the z coordinate. See Box 8 Referring to Figure 3.1: the heat flux *through* the cross section surfaces is zero, and the temperature throughout the thickness of the disk is uniform (i.e. the temperature does not vary with z). The surface of the three-dimensional solid consists of the two cross sections, and of the cylindrical surfaces, the inner and the outer. The two cylindrical surfaces may be associated with boundary condition of any type. The two cross sections are associated with the boundary condition of zero normal component of the heat flux, $\overline{q}_n = 0$ (See Box 8 , where the boundary is referred to as type S_2, Equation (2.28))

$$\boldsymbol{n} \cdot \boldsymbol{q} = \pm q_z = 0, \quad \text{on the cross sections .} \tag{3.1}$$

In this section we will refer to a simplified version of the heat conduction model: we will only consider isotropic materials, steady-state heat conduction, and the boundary conditions of two types: insulating boundary condition (heat flux zero) and prescribed temperature.

Under these conditions, the BVP is defined by the differential equation (which expresses thermal power balance, and hence is called the balance equation)

$$-\mathrm{div}\left[\kappa \left(\mathrm{grad}T\right)^T\right] - Q = 0 \ , \tag{3.2}$$

with the boundary conditions on the part S_1 of the boundary surface

$$T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \quad \boldsymbol{x} \text{ on } S_1 ,\tag{3.3}$$

and, along the S_2 part of the surface the normal component of the heat flux may be prescribed

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = 0, \quad \boldsymbol{x} \text{ on } S_2 .\tag{3.4}$$

Note that the boundary condition with zero normal component of the heat flux is usually referred to as the "insulated" boundary.

3.2 The discrete model: Method of Weighted Residuals

The partial-differential-equation model described in the previous section is usually called "continuous": the solution is expressed in terms of functions that satisfies the balance equation (3.2) and the boundary conditions when differentiated and substituted. Analytical solutions are unfortunately rare: only for a very special geometries and fairly simple heat loads can we hope to obtain one.

An alternative is to transition to a discrete model: instead of finding the solution to partial differential equations we solve systems of linear algebraic equations. This is not only much easier, but also, thanks to computers, not only tractable but also very efficient.

The methodology for converting the continuous boundary value problem models to discrete models that is commonly adopted in engineering practice is the weighted residual method (WRM).

To explain the basics we will discuss in this section a really simple BVP model that describes the deflection of a wire prestressed with force P, which is loaded by a uniformly distributed transverse load q. For simplicity we shall consider these two quantities chosen to give $q/P = 1$. The length of the wire is taken as $L = 1$, and the wire has a pin support at $x = 0$ and a roller support at $x = 1$. The balance (equilibrium) equation reads

$$Pw'' + q = 0 ,\tag{3.5}$$

or, with the force and the transverse loading given as above

$$w'' + 1 = 0 .\tag{3.6}$$

The solution is really easy: On the interval $0 \leq x \leq 1$ the shape of the wire is described by the function

$$w(x) = x - \frac{1}{2}x^2 .\tag{3.7}$$

When we differentiate this function twice and substitute into the differential equation (3.6) we can see that the function satisfies it as well as the two boundary conditions

$$w(0) = 0 , \quad w'(1) = 0 .\tag{3.8}$$

Forget that the BVP is so simple, and let us pretend that we need to guess the solution in the form of the function

$$w_h(x) = \sin\left(\frac{\pi}{2}x\right) ,\tag{3.9}$$

which we will call the **_trial function_**. Substituting the endpoints of the interval, we can see that the boundary conditions are satisfied. However, we obtain by differentiation

$$w_h''(x) = -\left(\frac{\pi}{2}\right)^2 \sin\left(\frac{\pi}{2}x\right)\tag{3.10}$$

so that the differential balance equation is seen *not* to be satisfied

$$w_h''(x) + 1 = -\left(\frac{\pi}{2}\right)^2 \sin\left(\frac{\pi}{2}x\right) + 1 \neq 0 .\tag{3.11}$$

The expression

$$r(x) = w_h''(x) + 1 = -\left(\frac{\pi}{2}\right)^2 \sin\left(\frac{\pi}{2}x\right) + 1 \tag{3.12}$$

is called the **residual**, and because the residual is not identically zero, we can see that w_h is *not* the exact solution. Figure 3.2 shows the exact and approximate solutions. Evidently, the approximate solution is quite off.
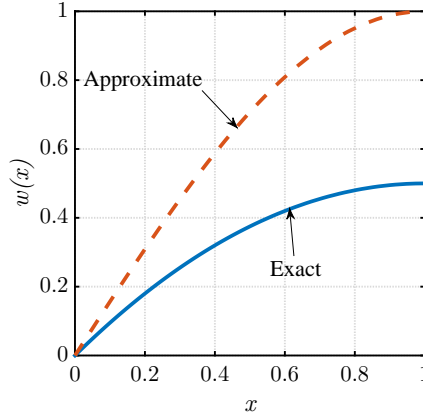
Fig. 3.2. The exact and approximate solution of the BVP (3.6), (3.8). The trial function is (3.9)

In addition to the substitution of the function into the differential equation to check whether the residual (3.9) is identically zero, we can also use the so-called *weighted residual test* for the same purpose:

$$\int_0^1 \vartheta(x)r(x)\,\mathrm{d}x \;=\; ? \tag{3.13}$$

where the arbitrary function $\vartheta(x)$ is usually called the **test function**. (What do we mean by arbitrary? Any function $\vartheta(x)$ such that the integral can be evaluated.)

What is the value of the expression on the left telling us?

- If for any arbitrary function $\vartheta(x)$ we get as a result zero, the residual must be identically zero, and consequently w_h must be the exact solution;
- otherwise, when we don't get a zero for some test function $\vartheta(x)$, the guess of the solution w_h is not the exact solution, only an approximation.

The attempted solution (3.9) is in fact not very good. Not only the balance equation is not satisfied (the residual is nonzero), but also the curve of the exact solution is not matched in magnitude, the approximate solution is about twice as big. We can improve upon our initial guess by adding a scalar multiplier

$$w_h(x) = \alpha_1 \sin\left(\frac{\pi}{2}x\right) \tag{3.14}$$

where α_1 is called the degree of freedom (DOF), and it is to be determined from a condition that the approximate solution should be somehow close to the exact one. The function $\sin\left(\frac{\pi}{2}x\right)$ is called the **basis function**.

The weighted residual statement (3.13) is now brought in again, but this time we *demand* (rather than check) the right-hand side to be zero

$$\int_0^1 \vartheta(x)r(x)\,\mathrm{d}x \;=\; 0\,. \tag{3.15}$$

If we now substitute our guess (trial) (3.14), equation (3.15) can be solved for α_1, and voilà, there is our approximate solution.

One more tweak: In order to improve our chances of finding a good trial function, the weighted residual statement (3.15) is replaced with a simple modification. If we substitute the trial function (3.14) into (3.15) we see that the trial function must be twice differentiable

$$\int_0^1 \vartheta(x)r(x)\,\mathrm{d}x \;=\; \int_0^1 \vartheta(x)\left[w_h''(x)+1\right]\,\mathrm{d}x \;=\; \int_0^1 \vartheta(x)w_h''(x)\,\mathrm{d}x \;+\; \int_0^1 \vartheta(x)\,\mathrm{d}x \;=\; 0 \qquad (3.16)$$

which is easy here, but considerably more difficult for real situations in two or three dimensions. Therefore, we use integration by parts to reduce the number of derivatives needed for the weighted residual statement as

$$\int_0^1 \vartheta(x)w_h''(x)\,\mathrm{d}x = -\int_0^1 \vartheta'(x)w_h'(x)\,\mathrm{d}x + [\vartheta(x)w_h'(x)]_0^1 \qquad (3.17)$$

So the weighted residual statement (3.16) will be rewritten as

$$\int_0^1 \vartheta(x)r(x)\,\mathrm{d}x \;=\; -\int_0^1 \vartheta'(x)w_h'(x)\,\mathrm{d}x + [\vartheta(x)w_h'(x)]_0^1 + \int_0^1 \vartheta(x)\,\mathrm{d}x = 0 \qquad (3.18)$$

Notice that we haven't changed the rules of the game: this is still the original weighted residual statement, but now we are allowed to pick a trial function with only one derivative, and in exchange we now also require the test function to be differentiable. The test function is typically taken in the same form as the trial function, meaning we would take the test function here to be the basis function $\sin\left(\frac{\pi}{2}x\right)$. When we do that, the weighted residual statement (3.18) yields

$$-\int_0^1 \left[\sin\left(\frac{\pi}{2}x\right)\right]'\left[\alpha_1\sin\left(\frac{\pi}{2}x\right)\right]'\,\mathrm{d}x$$

$$+\sin\left(\frac{\pi}{2}\times 1\right)\left[\alpha_1\frac{\pi}{2}\underbrace{\cos\left(\frac{\pi}{2}\times 1\right)}_{0}\right] - \underbrace{\sin\left(\frac{\pi}{2}\times 0\right)}_{0}\left[\alpha_1\frac{\pi}{2}\cos\left(\frac{\pi}{2}\times 0\right)\right] \qquad (3.19)$$

$$+\int_0^1 \sin\left(\frac{\pi}{2}x\right)\,\mathrm{d}x = 0$$

and in numbers

$$-\alpha_1\frac{\pi^2}{8} + \frac{2}{\pi} = 0\,, \quad \Rightarrow \quad \alpha_1 = \frac{16}{\pi^3} \approx 0.516 \qquad (3.20)$$

Figure 3.3 shows the approximate solution (3.14) for the calculated value of α_1: clearly the approximate solution now hugs the exact solution fairly closely.

If we wanted to improve the approximate solution further, we could use for instance this function

$$w_h(x) = \alpha_1\sin\left(\frac{\pi}{2}x\right) + \alpha_2\sin\left(\frac{3\pi}{2}x\right) \qquad (3.21)$$

with two degrees of freedom, which we can write using the notation $\phi_1(x) = \sin\left(\frac{\pi}{2}x\right)$ and $\phi_2(x) = \sin\left(\frac{3\pi}{2}x\right)$ as

$$w_h(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) \qquad (3.22)$$

Now we need two equations, because we have two unknown degrees of freedom. So we write the weighted residual equation once, for $\vartheta = \phi_1$,
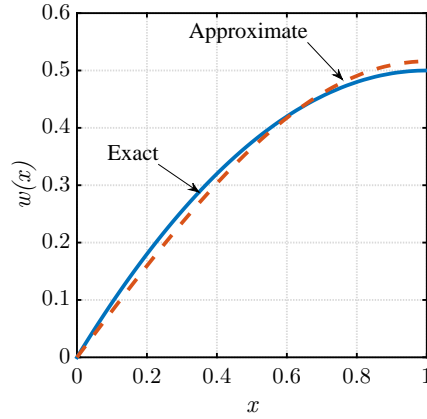
Fig. 3.3. The exact and approximate solution of the BVP (3.6), (3.8) for the trial function (3.14)

$$-\int_0^1 \phi_1'(x)\left[\alpha_1\phi_1(x) + \alpha_2\phi_2(x)\right]' \, \mathrm{d}x + \left[\phi_1(x)\left[\alpha_1\phi_1(x) + \alpha_2\phi_2(x)\right]'\right]_0^1 + \int_0^1 \phi_1(x)\,\mathrm{d}x = 0 \quad (3.23)$$

and twice, for the test function $\vartheta = \phi_2$,

$$-\int_0^1 \phi_2'(x)\left[\alpha_1\phi_1(x) + \alpha_2\phi_2(x)\right]' \, \mathrm{d}x + \left[\phi_2(x)\left[\alpha_1\phi_1(x) + \alpha_2\phi_2(x)\right]'\right]_0^1 + \int_0^1 \phi_2(x)\,\mathrm{d}x = 0 \quad (3.24)$$

Since the basis functions are known, the above two equations are simply linear algebraic equations for two unknowns, α_1 and α_2. Solving for the unknown degrees of freedom we obtain the approximate solution shown in Figure 3.4: a very considerable improvement of the original guess!
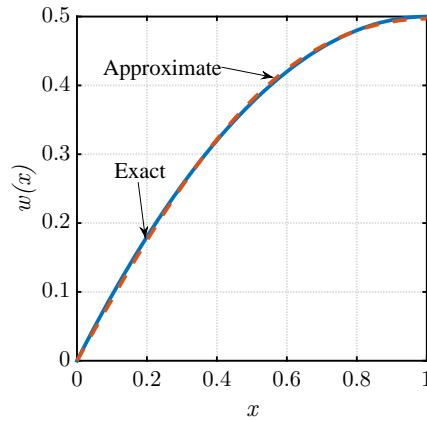
Fig. 3.4. The exact and approximate solution of the BVP (3.6), (3.8) for the trial function (3.22)

Figure 3.5 shows the residuals for the two trial functions (3.14) and (3.22) and for the solution with three terms (whose details we have not shown here). We can see how the residual gets "molded" by each additional term in the trial function and by each additional test function. (By the way, the residual does not diminish at $x = 0$ because each of the sine terms has zero curvature at that point. Using only sines will not produce a very good approximation for a function that is supposed to have constant curvature.)

Let us contemplate a tangible analogy of what we're trying to do here with the test functions. Consider the exciting game of Whac-A-Mole. We use the mallet whenever the mole pokes its head
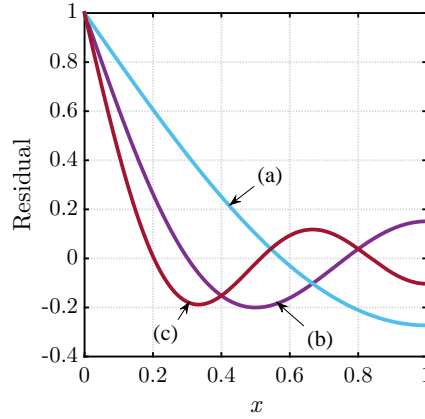
Fig. 3.5. The residual of the approximate solutions of the BVP (3.6), (3.8). (a) for the trial function (3.14), (b) for the trial function (3.22), and (c) for the trial function $w_h(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \alpha_3\phi_3(x)$

out of the table to push it back in. If we had five mallets (and five hands to operate them), we could keep all five moles inside. The residual behaves in places like the mole poking its head out. Each of the mallets may be thought of as a single test function η that pushes the residual (a mole) down in some spots.

Evidently, the residual bulges out a little bit in between the mallets. However, we have the option of add more mallets, and distribute the locations of the mallets wisely, and we will manage to do a better and better job of keeping the moles underground. Indeed, with an infinite number of mallets, we can control the critters no matter where they wish to stick their head out. Note that we have to distribute the mallets in some sense **densely** and **uniformly**– no parts of the interval $0 \leq x \leq 1$ may be left out, since the residual could stay nonzero there.

> Recall that this book is about finite element methods, and all the test functions will be the finite element basis functions, and all the trial functions will be constructed as linear combinations of the finite element basis functions. The finite element mesh provides us with an efficient systematic way of constructing the test and trial functions.

3.3 Weighted residual method for the heat conduction problem

The finite element method proceeds from the so-called weighted residual statement instead of the BVP equations (3.2–3.4) See Box 7

$$\int_V (\mathrm{grad}\vartheta)\, \kappa(\mathrm{grad}T)^T \, \mathrm{d}V - \int_V \vartheta Q \, \mathrm{d}V = 0, \quad \vartheta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in S_1 \, . \tag{3.25}$$

We can compare the weighted residual statement (3.18) with the weighted residual equation for the heat conduction (3.25): It isn't difficult to identify terms which belong together, and we may begin to divine how that equation came about: We formed the residual from the heat balance equation (3.2) and used integration by parts to shift a derivative from the trial function to the test function. In the end it is still the same simple principle that was outlined in the previous section.

In (3.25) $T(x, y, z)$ is the function that describes how the temperature varies within the region V (known as the **trial function**); κ and Q are given thermal conductivity and the body heat load (the internal heat generation rate); and finally $\vartheta(x, y, z)$ is the so-called **test function** (also called weight function). The essence of the FEM is the ability to systematically define the trial function and the test function in a way that leads to an efficient calculation of the unknown coefficients that define the trial function (the free DOFs).

First, let us simplify equation (3.25) to accomplish the elimination of the z coordinate. Since the temperature does not vary with z, the integrals (3.25) may be simplified by pre-integrating in the thickness direction, $dV = \Delta z \, dS$. The volume integrals are then evaluated over the cross-sectional area, S_c, (see Figure 3.1(b))

$$\int_{S_c} (\text{grad}\vartheta) \, \kappa(\text{grad}T)^T \, \Delta z \, dS - \int_{S_c} \vartheta Q \, \Delta z \, dS = 0, \tag{3.26}$$

$$\vartheta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in C_{c,1} \,,$$

where $C_{c,1}$ is the part of the boundary curve(s) C where the temperature is prescribed (See Box 8 , boundary of type 1). Note that the thickness Δz is a constant and could be canceled without any effect on the solution. Nevertheless, Equation (3.26) still applies to a fully three-dimensional body. To maintain this notion throughout the book, we shall not cancel the thickness.

In (3.26) the trial function $T(x,y)$ and the test function $\vartheta(x,y)$ no longer depend on z. So the gradients are now simplified to

$$\text{grad}T = \left[\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y} \right] \,, \tag{3.27}$$

for the trial function and similarly for the test function.

3.4 Test and trial functions and finite elements

Both the test and trial function are functions of x and y only, $\vartheta = \vartheta(x,y)$ and $T = T(x,y)$. On the part of the boundary where the temperature is being prescribed, $C_{c,1}$, they assume the values

Trial function: $T(x,y) = \overline{T}(x,y)$,   Test function: $\vartheta(x,y) = 0$, $[x,y]$ on $C_{c,1}$.

Let us consider first the test function. It needs to be defined as a function of x and y over arbitrarily shaped domains. Finite elements have been thought up to make the definition of such functions easy. The triangle with three nodes is the simplest finite element in more than one coordinate. In this book we will call this element T3 (Triangle with 3 nodes). In Abaqus it has different names that change according to the problem being solved. This is the first finite element that we will study.

The domain of the disk with a hole (shown in Figure 3.1) is approximated as a collection of triangles (in other words, it is *tiled* with triangles, or *triangulated*), see Figure 3.6. The mesh consisting of triangles is typically called **triangulation**, even though sometimes *any* mesh is called that. The vertices of the triangulation are called **nodes**, while the line segments connecting the nodes are called **edges**. Evidently, the triangles are the finite **elements**.
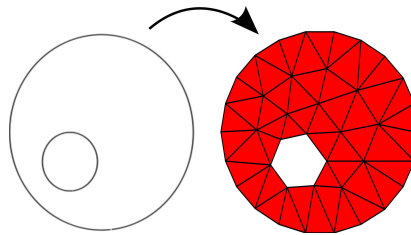
Fig. 3.6. Mesh of the disk domain

Any function on the triangle mesh will be expressed as a linear combination of "tent" functions. Each individual tent is formed by grabbing one of the nodes (say J) and raising it out of the plane of the triangulation (traditionally to a unit height). The tent canvas is stretched over the edges that

connect at the node J, and are clamped down by the ring of the edges that surround node J. The cartoon of one particular tent is shown in Figure 3.7. The term *hat function* or shape function is also commonly used. We will prefer the term basis function: similarly to the space of vectors where one can pick a set of vectors as a basis, these tent (hat, shape) functions form the basis of some space.

> Basis functions are at the foundations of the finite element method.
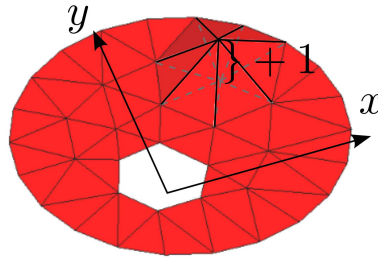
Fig. 3.7. Visual representation of one basis function on the mesh of the disk

All the triangles that are connected in the node J **support** the function N_J, which is another way of saying that the function N_J is nonzero in these triangles; it is defined to be zero everywhere else. (If we are inside the "tent", we are standing on the support of the function.) Mathematically, the support of the basis function N_J is

$$\text{supp}N_J = \{\boldsymbol{x} : N_J(\boldsymbol{x}) \neq 0\} \ .$$

Since the set $\text{supp}N_J$ is a finite piece of the computational domain, it is also called a **compact support**.

> The compact supports of the finite element basis functions make the finite element matrices sparse, and hence are crucial for the efficiency of these methods.

It remains to write down the equations that define the function N_J at any point within its support. That means writing an expression for each triangle within the support separately. Referring to Figure 3.7, there are only three such functions: the three basis functions associated with the nodes at the corners of the element; all the other basis functions in the mesh are identically zero over this element. Thus, our task here is to write down the expressions for the three basis functions over the domain of a single triangle. In contrast, to express the form of a single basis function, we would put together pieces of this basis function over each of the triangles in the mesh.

3.5 The standard triangle

On any triangle there are three non-zero basis functions, and each of the three basis functions is zero along one edge of the triangle: again, refer to Figure 3.7. We can conclude from this that we could describe such functions as triangular pieces of planes. For a general triangle we would have to solve some equations, but for some triangles, with their nodes in special positions, the basis functions can be determined simply by looking at a picture.

This is the idea behind the isoparametric formulation: the expressions for the basis functions are derived for elements of particularly simple shapes, such as a right triangle or a square. These are called **standard shapes**, hence we will be talking about the standard triangle here. The standard triangle is described in coordinates ξ, η, and these are not the physical coordinates in which the BVP is described. These are coordinates used only to describe the standard shape, and they will be referred to as the parametric coordinates. Refer to Figure 3.8.

> Nodes of a single element will be shown with an underline, and are always between $\underline{1}$ and $\underline{n}$, where $\underline{n}$ is the number of nodes of the element (three for the three-node triangle). Global node numbers (that is numbers within the entire mesh) will always be shown without an underline (like this: K).

The basis functions associated with nodes $\underline{2}$ and $\underline{3}$ are simply

$$N_{\underline{2}}(\xi, \eta) = \xi \ , \tag{3.28}$$

and

$$N_{\underline{3}}(\xi, \eta) = \eta \ . \tag{3.29}$$

As is easily verified, $N_{\underline{2}}$ is zero along the edge $[\underline{3}, \underline{1}]$, and assumes value $+1$ at node $\underline{2}$; analogous properties hold for $N_{\underline{3}}$. If $N_{\underline{1}}$ should be equal to $+1$ at the origin, it must be written as

$$N_{\underline{1}}(\xi, \eta) = 1 - \xi - \eta \ . \tag{3.30}$$

Clearly, $N_{\underline{1}}$ vanishes at the edge $[2,3]$ opposite node 1. Thus, we see that the three functions we just formulated satisfy the Kronecker delta property,

$$N_{\underline{i}}(\xi_{\underline{k}}, \eta_{\underline{k}}) = \delta_{\underline{ik}} \ , \tag{3.31}$$

where the symbol $\delta_{\underline{ik}}$ is known as the **Kronecker delta**

$$\delta_{\underline{ik}} = \begin{cases} 1, \text{ if } \underline{i} = \underline{k}; \\ 0, \text{ otherwise.} \end{cases}$$

In words: basis function $N_{\underline{i}}$ associated with node $\underline{i}$ assumes the value of either 1.0 or 0.0 at node $\underline{k}$, depending on whether it is the same node ($\underline{i} = \underline{k}$) or not.
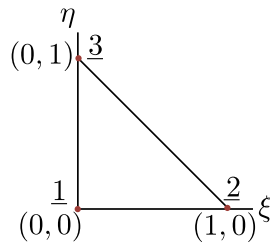
Fig. 3.8. Standard triangle

Now that we have the three basis functions we can use them to describe functions within the standard triangle. We simply write

$$a(\xi, \eta) = \sum_{\underline{i}=1}^{3} N_{\underline{i}}(\xi, \eta) a_{\underline{i}} \ , \tag{3.32}$$

to generate a function $a(\xi, \eta)$ where $a_{\underline{i}}$ are arbitrary coefficients. So, the function $a(\xi, \eta)$ is a linear combination of the basis functions. What are the coefficients? That is what equation (3.31) is for: write

$$a(\xi_{\underline{k}}, \eta_{\underline{k}}) = \sum_{\underline{i}=1}^{3} N_{\underline{i}}(\xi_{\underline{k}}, \eta_{\underline{k}})a_{\underline{i}} = \sum_{\underline{i}=1}^{3} \delta_{\underline{i}\underline{k}}a_{\underline{i}} = a_{\underline{k}} \; , \tag{3.33}$$

and we can see that the coefficients $a_{\underline{k}}$ are simply the values of the function at the nodes, $a(\xi_{\underline{k}}, \eta_{\underline{k}}) = a_{\underline{k}}$. Therefore, we may make the observation that data that sit at the nodes (the corners of the triangle) are **_naturally interpolated_** across the triangle using (3.32). The coefficients $a_{\underline{k}}$ are called the **_degrees of freedom_**.

Let us now take a triangle in the physical Cartesian space, with corners at locations $[x_{\underline{1}}, y_{\underline{1}}]^T$, $[x_{\underline{2}}, y_{\underline{2}}]^T$, and $[x_{\underline{3}}, y_{\underline{3}}]^T$. Consider what happens when we interpolate the values of the x- and y-coordinates of the corners:

$$x(\xi, \eta) = \sum_{\underline{i}=1}^{3} N_{\underline{i}}(\xi, \eta)x_{\underline{i}} \; , \quad y(\xi, \eta) = \sum_{\underline{i}=1}^{3} N_{\underline{i}}(\xi, \eta)y_{\underline{i}} \; . \tag{3.34}$$

This is a mapping from the pair of coordinates (point in the standard triangle) ξ, η to the point x, y in the physical space. Substituting for the basis functions from (3.28– 3.30), it may be written explicitly as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (x_{\underline{2}} - x_{\underline{1}}) , (x_{\underline{3}} - x_{\underline{1}}) \\ (y_{\underline{2}} - y_{\underline{1}}) , (y_{\underline{3}} - y_{\underline{1}}) \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_{\underline{1}} \\ y_{\underline{1}} \end{bmatrix} \; . \tag{3.35}$$

This matrix equation is accompanied by the picture in Figure 3.9. The two vectors, $\boldsymbol{v}$ and $\boldsymbol{w}$, are
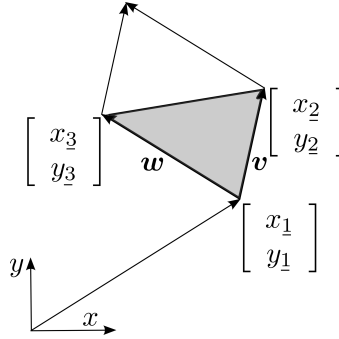
Fig. 3.9. Interpolating Cartesian coordinates on the standard triangle

the two columns of the square matrix in (3.35):

$$\boldsymbol{v} = \begin{bmatrix} (x_{\underline{2}} - x_{\underline{1}}) \\ (y_{\underline{2}} - y_{\underline{1}}) \end{bmatrix} \; , \quad \boldsymbol{w} = \begin{bmatrix} (x_{\underline{3}} - x_{\underline{1}}) \\ (y_{\underline{3}} - y_{\underline{1}}) \end{bmatrix} \; . \tag{3.36}$$

If both ξ and η vary between zero and one, equation (3.35) adds the two vectors, $\xi\boldsymbol{v}$ and $\eta\boldsymbol{w}$ to the vector $[x_{\underline{1}}, y_{\underline{1}}]^T$, and the result then covers the entire parallelogram; on the other hand, if ξ and η are confined to the interior of the standard triangle, Equation (3.35) produces points to cover the area of the filled triangle. To summarize, Equation (3.35) is a **_map_** from the standard triangle to a triangle in the physical Cartesian coordinates with corners in given locations. In the next section and the rest of the book we will also use them to describe any quantities that occur within the BVP that we wish to solve (heat conduction: temperature, stress analysis: displacement).

> Finite elements on which the geometrical coordinates $\boldsymbol{x}$ are interpolated in exactly the same way as the variable(s) of the PDE (temperature, displacements, and so on) are called *isoparametric elements*.

It is important to make clear that a single standard triangle is mapped to all the triangles in the mesh (in x, y). Figure 3.10 shows the mapping of the nodes $\underline{1}$, $\underline{2}$, and $\underline{3}$ to four different triangles.

The x, y triangles are defined by listing their three nodes in counterclockwise order, starting with the first.

> An element is defined by listing the global node numbers that are connected by it. This array of node numbers is called connectivity.

Thus for the triangle $[B, C, A]$ the nodes in the standard triangle correspond to the nodes in the mesh as

$$\underline{1} \to B, \quad \underline{2} \to C, \quad \underline{3} \to A \tag{3.37}$$

as also indicated by the arrows in the figure. A node may be the first in one triangle, the third in another. The only thing that matters is the counterclockwise order. The important consequence is that for the counterclockwise order of nodes the area computed from the cross product of the vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ will not be negative.

> Order in which nodes are listed matters! It determines the orientation of the element (and hence also its volume, area, length).
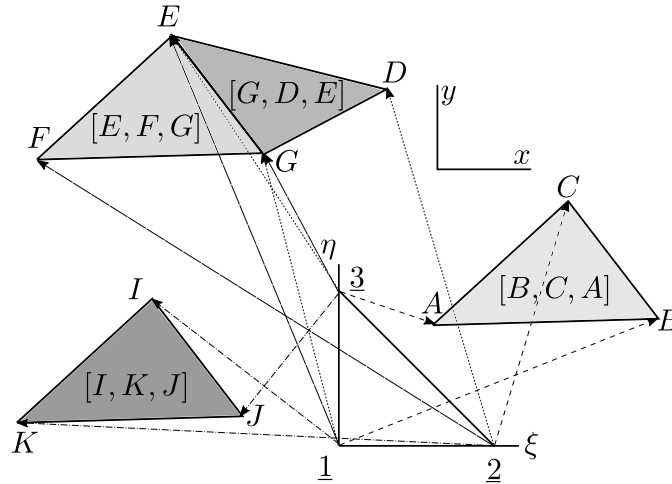
Fig. 3.10. Mapping the standard triangle to four different x, y triangles. Triangle $[B, C, A]$, the two adjacent triangles $[G, D, E]$ and $[E, F, G]$, and triangle $[I, K, J]$. The labels $A, B, \ldots, J$ are node numbers. The triangles are defined by listing three nodes in counterclockwise order, starting with the first.

3.6 Interpolation over the x, y triangle

Let us talk about the trial function: $T(x, y)$. This represents the possible (hence "trial") distribution of the temperature over the entire domain. Using the finite element basis functions, we can write

$$T(x,y) = \sum_{i=1}^{N} N_i(x,y)T_i \tag{3.38}$$

provided we knew the expressions for the basis functions in the coordinates x, y, i.e. $N_i(x,y)$. At this point we don't, we know $N_j(\xi, \eta)$, $j = 1, 2, 3$ instead. But let us pretend we do. How would we actually use it to describe $T(x,y)$? Have a look at Figure 3.7: each basis function has a different description over each triangle in its support (i.e. over each triangle where it is nonzero). So to describe $T(x,y)$ we must do it triangle-by-triangle. The function $T(x,y)$ looks like triangular facets on the surface of the diamond! It is piecewise linear.

What are the is in (3.38)? They are degree of freedom numbers. Simply put, a degree of freedom is a number (coefficient) that we use to describe the temperature, as opposed to the basis function which is not a number but, as its name implies, a function. The basis functions are determined by the mesh, the degrees of freedom will determine the solution. The basis functions N_i are labeled with i as the degree of freedom number, and so are the T_i degrees of freedom.

> The basis function N_i is to be understood as the basis function associated with the node that carries the degree of freedom i.

At this point we know how to write the basis functions over the standard triangle (3.28– 3.30), and we know that any triangle in the mesh in the coordinates x, y is an image of the standard triangle obtained from (3.34). And now we also realize that the describe $T(x,y)$ we have to do it one triangle at a time. Therefore a perfectly reasonable approach to the expression of the trial function (3.38) is to express

$$T(x,y) = \sum_{i=1}^{N} N_i(x(\xi, \eta), y(\xi, \eta))T_i = \sum_{i=1}^{N} N_i(\xi, \eta)T_i \tag{3.39}$$

That is, for any point x, y we can say what the value $T(x,y)$ is obtained by finding the corresponding point ξ, η in the standard triangle, calculating the values of the basis functions at that point, and plugging them into (3.39). If it seems awkward, it is because it is awkward. But we never actually need to go through with this to express $T(x,y)$ for a given x, y. Rather, all operations that we will need will be expressed on the standard triangle, so that we actually *know* ξ, η (which allows us to calculate x, y, should we be interested).

> In any given triangle of the mesh, the trial functions are expressed for any point x, y within that triangle by writing
>
> $$T(x,y) = \sum_{i=1}^{N} N_i(\xi, \eta)T_i$$
>
> where it is understood that ξ, η is such a point in the standard triangle that is mapped to x, y by (3.34).

The test function is actually a lot simpler than the trial function. Our choice for a test function will be a single basis function

$$\vartheta(x,y) = N_j(\xi, \eta) \tag{3.40}$$

We will explain how to arrive at the degree of freedom number j below. In brief, each degree of freedom that is free (i. e. an actual unknown) will be associated with one test function. The reason is that for each unknown in the problem we need one equation: N_f unknowns leads to N_f coupled equations.

3.7 Elementwise calculations

Here is a reminder: we wish to solve (3.26) using the finite element method. The finite elements are useful in two ways:

1. We *describe* the test and trial function with the help of the basis functions that are defined on the finite elements, and
2. to calculate the required integrals we *integrate* over the triangles, one by one.

We have seen how to construct the basis functions for triangle meshes in the previous section. In equation (3.26) we need to take derivatives of the basis functions, and we don't know how to do that yet (but we will discuss this soon). At this point we can consider the overall structure of the evaluation of those integrals, however, and we will work on the details later.

We will explain how things work on an example. Figure 3.11 shows a mesh of a pie-shaped piece of the cross-section of the concrete column. We consider the boundary conditions of (2.3–2.4). The mesh consists of three triangles, [1,2,3], [2,4,5], and [2,5,3]. The temperatures along the boundary with water, i.e. along the edge [4,5], is known to be T_w and that determines the temperature values at the nodes 4 and 5. Temperatures at the nodes 1, 2, and 3 are unknown. Note that we will take the degrees of freedom numbers to be identical to the node numbers. So that temperature degree of freedom at node 2 is T_2 and so on. The total number of degrees of freedom is $N = 5$, and the number of free (unknown) degrees of freedom is $N_\mathrm{f} = 3$.
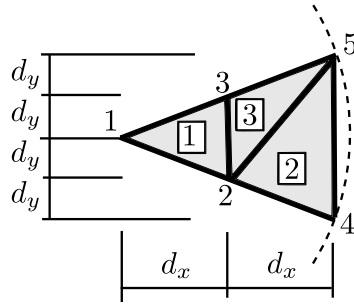
Fig. 3.11. Mesh of three triangles for the concrete column with hydration heat load and zero temperature on the circumference

The trial function is given by equation (3.39) so that its gradient follows as

$$\mathrm{grad}T = \mathrm{grad}\left(\sum_{i=1}^{N} N_i T_i\right) = \sum_{i=1}^{N} T_i \,\mathrm{grad}N_i \;. \tag{3.41}$$

Furthermore we will take as the test function

$$\vartheta = N_j \;, \quad \text{for } j = 1, \ldots, N_\mathrm{f} \tag{3.42}$$

so that we adopt $\mathrm{grad}\vartheta = \mathrm{grad}N_j$.

Now we take the weighted residual statement (3.26) and substitute the above expressions for the trial function gradient and the test function gradient

$$\int_{S_c} \mathrm{grad}N_j \,\kappa \sum_{i=1}^{N} T_i \mathrm{grad}N_i^T \,\Delta z \,\mathrm{d}S - \int_{S_c} N_j Q \,\Delta z \,\mathrm{d}S = 0, \quad \text{for } j = 1, \ldots, N_\mathrm{f} \;. \tag{3.43}$$

This can be further simplified by taking the degree of freedom T_i from underneath the integral sign

$$\sum_{i=1}^{N} T_i \int_{S_c} \mathrm{grad}N_j \,\kappa \mathrm{grad}N_i^T \,\Delta z \,\mathrm{d}S - \int_{S_c} N_j Q \,\Delta z \,\mathrm{d}S = 0, \quad \text{for } j = 1, \ldots, N_\mathrm{f} \;. \tag{3.44}$$

Note that our choice of the test function satisfies the condition $\vartheta(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in C_{c,1}$. In the present example $C_{c,1}$ is the edge $[4,5]$, and all test functions used here, $N_1, N_2,$ and N_3, all vanish along this edge.

Next we must consider the integrals. The cross-section (or rather a piece of it) S_c is represented by the mesh. It is therefore natural to carry out the integration over S_c by splitting the integral into three pieces, one over each triangle. Therefore, equation (3.44) needs to be rewritten by summing over all the triangles e in the mesh

$$\sum_{i=1}^{N} T_i \sum_{e} \int_{e} \mathrm{grad} N_j \, \kappa \mathrm{grad} N_i^T \, \Delta z \, \mathrm{d}S - \sum_{e} \int_{e} N_j Q \, \Delta z \, \mathrm{d}S = 0, \quad \text{for } j = 1, \dots, N_{\mathrm{f}} \, . \tag{3.45}$$

where by $\int_{e}$ we mean integrate over the area of the triangle e. Using these shorthands

$$\boldsymbol{g}_j = \mathrm{grad} N_j, \quad \int_{e1} \boldsymbol{g}_j \kappa \boldsymbol{g}_i^T = \int_{e=1} \mathrm{grad} N_j \, \kappa \mathrm{grad} N_i^T \, \Delta z \, \mathrm{d}S, \quad \int_{e1} N_j Q = \int_{e=1} N_j Q \, \Delta z \, \mathrm{d}S \tag{3.46}$$

in order to fit everything onto the page, we explicitly rewrite (3.45) with the three equations

$$T_1 \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_1^T \quad +T_2 \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_2^T \quad +T_3 \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_3^T \quad +0 \quad +0 \quad = \int_{e1} N_1 Q$$

$$T_1 \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_1^T +T_2 \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \end{array} \right) +T_3 \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \end{array} \right) +T_4 \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_4^T +T_5 \left(\begin{array}{c} \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \end{array} \right) = \left(\begin{array}{c} \int_{e1} N_2 Q \\ + \int_{e2} N_2 Q \\ + \int_{e3} N_2 Q \end{array} \right)$$

$$T_1 \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_1^T +T_2 \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \end{array} \right) +T_3 \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \end{array} \right) \quad +0 \quad +T_5 \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_5^T \quad = \left(\begin{array}{c} \int_{e1} N_3 Q \\ + \int_{e3} N_3 Q \end{array} \right) \tag{3.47}$$

We need to realize that the triangles over which one of the gradients is identically zero ($\boldsymbol{g}_j$ or $\boldsymbol{g}_i$) are not carried over from (3.45), since those integrals are obviously zero. For instance, $\boldsymbol{g}_1$ is identically zero over the triangles 2 and 3, which is why the first equation in (3.47) does not refer to triangles 2 and 3.

We can organize these three equations using the matrix form

$$\begin{bmatrix} \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_3^T, & 0, & 0 \\ \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_1^T, & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \end{array} \right), & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \end{array} \right), & \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_4^T, & \left(\begin{array}{c} \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \end{array} \right) \\ \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_1^T, & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \end{array} \right), & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \end{array} \right), & 0, & \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_5^T \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} \int_{e1} N_1 Q \\ \left(\begin{array}{c} \int_{e1} N_2 Q \\ + \int_{e2} N_2 Q \\ + \int_{e3} N_2 Q \end{array} \right) \\ \left(\begin{array}{c} \int_{e1} N_3 Q \\ + \int_{e3} N_3 Q \end{array} \right) \end{bmatrix} \tag{3.48}$$

Now realize that all the integrals will evaluate to a real number. Hence, what we have in (3.48) can be described as linear algebraic equations for the unknowns T_1, T_2 and T_3. Recall that the boundary

conditions fixed the temperatures $T_4 = T_5 = 0$: We can use it to eliminate the fourth and fifth column on the left-hand side and so the system of linear algebraic equations can be put as

$$
\begin{bmatrix}
\int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_3^T \\[2ex]
\int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_1^T, & \left(\begin{aligned} &\int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ +&\int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ +&\int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \end{aligned} \right), & \left(\begin{aligned} &\int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \\ +&\int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \end{aligned} \right) \\[4ex]
\int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_1^T, & \left(\begin{aligned} &\int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \\ +&\int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \end{aligned} \right) & \left(\begin{aligned} &\int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \\ +&\int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \end{aligned} \right)
\end{bmatrix}
\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix}
=
\begin{bmatrix}
\int_{e1} N_1 Q \\[2ex]
\left(\begin{aligned} &\int_{e1} N_2 Q \\ +&\int_{e2} N_2 Q \\ +&\int_{e3} N_2 Q \end{aligned} \right) \\[4ex]
\left(\begin{aligned} &\int_{e1} N_3 Q \\ +&\int_{e3} N_3 Q \end{aligned} \right)
\end{bmatrix}
\tag{3.49}
$$

The abbreviated form is rendered neatly with bold symbols for the matrices and vectors as

$$
\boldsymbol{K} \qquad\qquad \boldsymbol{T} \quad = \quad \boldsymbol{L} \tag{3.50}
$$

where the matrix $\boldsymbol{K}$ is known as the conductivity matrix, the vector $\boldsymbol{T}$ collects the degrees of freedom (temperatures at the nodes), and $\boldsymbol{L}$ is the vector of heat loads.

Now we will consider the left-hand side matrix in equation (3.48). We will split it into three matrices, each one will hold only the contributions from a single element from the mesh:

$$
\begin{bmatrix}
\int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_3^T, & 0, & 0 \\[2em]
\int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_1^T, & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T \end{array} \right), & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T \end{array} \right), & \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_4^T, & \left(\begin{array}{c} \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \\ + \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \end{array} \right) \\[3em]
\int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_1^T, & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T \end{array} \right), & \left(\begin{array}{c} \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \\ + \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T \end{array} \right), & 0, & \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_5^T
\end{bmatrix} = \tag{3.51}
$$

$$
\left. \begin{bmatrix}
\int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_1 \kappa \boldsymbol{g}_3^T, & 0, 0 \\[1.5em]
\int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T, & 0, 0 \\[1.5em]
\int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_1^T, & \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T, & \int_{e1} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T, & 0, 0
\end{bmatrix} \right\} \text{from element 1} \tag{3.52}
$$

$$
+ \left. \begin{bmatrix}
0, & 0, & 0, & 0, & 0 \\[1.5em]
0, & \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T, & 0, & \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_4^T, & \int_{e2} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \\[1.5em]
0, & 0, & 0, & 0, & 0
\end{bmatrix} \right\} \text{from element 2} \tag{3.53}
$$

$$
+ \left. \begin{bmatrix}
0, & 0, & 0, & 0, & 0 \\[1.5em]
0, & \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_2^T, & \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_3^T, & 0, & \int_{e3} \boldsymbol{g}_2 \kappa \boldsymbol{g}_5^T \\[1.5em]
0, & \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_2^T, & \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_3^T, & 0, & \int_{e3} \boldsymbol{g}_3 \kappa \boldsymbol{g}_5^T
\end{bmatrix} \right\} \text{from element 3} \tag{3.54}
$$

The meaning of this decomposition is: we can easily calculate the overall coefficient matrix (the conductivity matrix) by going through the mesh element-by-element, calculating the contributions, and adding them into the overall matrix in appropriate rows and columns. This process is called **finite element assembly**.

Similarly the heat load vector can be also composed of the contributions from the individual finite elements.

$$
\begin{bmatrix}
\int_{e1} N_1 Q \\[1em]
\left(\begin{array}{c} \int_{e1} N_2 Q \\ + \int_{e2} N_2 Q \\ + \int_{e3} N_2 Q \end{array} \right) \\[3em]
\left(\begin{array}{c} \int_{e1} N_3 Q \\ + \int_{e3} N_3 Q \end{array} \right)
\end{bmatrix}
= \underbrace{\begin{bmatrix} \int_{e1} N_1 Q \\[1em] \int_{e1} N_2 Q \\[1em] \int_{e1} N_3 Q \end{bmatrix}}_{\text{from element 1}}
+ \underbrace{\begin{bmatrix} 0 \\[1em] \int_{e2} N_2 Q \\[1em] 0 \end{bmatrix}}_{\text{from element 2}}
+ \underbrace{\begin{bmatrix} 0 \\[1em] \int_{e3} N_2 Q \\[1em] \int_{e3} N_3 Q \end{bmatrix}}_{\text{from element 3}} \tag{3.55}
$$

Again, the overall heat load vector would be computed by adding contributions that are calculated for each element separately.

3.8 Derivatives of the basis functions; Jacobian

The results of this section are much more general than may be expected. While the formulas for the derivatives of basis functions are derived for the linear triangles, the same formulas (and implementation) is used for all the so-called *isoparametric* elements in common finite element software.

Using the chain derivatives we find that the derivative of $N_i(\xi, \eta)$ with respect to x, y is expressed using the Jacobian matrix $\boxed{\text{See Box 10}}$

$$\left[\frac{\partial N_i(\xi, \eta)}{\partial x}, \ \frac{\partial N_i(\xi, \eta)}{\partial y} \right] = \left[\frac{\partial N_i(\xi, \eta)}{\partial \xi}, \ \frac{\partial N_i(\xi, \eta)}{\partial \eta} \right] [J]^{-1} \ . \tag{3.56}$$

The elements of $[J]$ are

$$[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi}, & \dfrac{\partial x}{\partial \eta} \\[2ex] \dfrac{\partial y}{\partial \xi}, & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \displaystyle\sum_{i=1}^{3} \dfrac{\partial N_i}{\partial \xi} x_i, & \displaystyle\sum_{i=1}^{3} \dfrac{\partial N_i}{\partial \eta} x_i \\[3ex] \displaystyle\sum_{i=1}^{3} \dfrac{\partial N_i}{\partial \xi} y_i, & \displaystyle\sum_{i=1}^{3} \dfrac{\partial N_i}{\partial \eta} y_i \end{bmatrix}, \tag{3.57}$$

as follows from (3.34). The triangle finite element has three basis functions associated with its three nodes. Thus for a single triangle we will define the matrix of gradients of the basis functions with respect to ξ, η

$$\left[\mathrm{grad}_{(\xi,\eta)} N \right] = \begin{bmatrix} \mathrm{grad}_{(\xi,\eta)} N_1 \\ \mathrm{grad}_{(\xi,\eta)} N_2 \\ \mathrm{grad}_{(\xi,\eta)} N_3 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi}, & \dfrac{\partial N_1}{\partial \eta} \\[2ex] \dfrac{\partial N_2}{\partial \xi}, & \dfrac{\partial N_2}{\partial \eta} \\[2ex] \dfrac{\partial N_3}{\partial \xi}, & \dfrac{\partial N_3}{\partial \eta} \end{bmatrix} = \begin{bmatrix} -1, & -1 \\[1ex] +1, & 0 \\[1ex] 0, & +1 \end{bmatrix}, \tag{3.58}$$

and the matrix $[x]$ to collect the coordinates of the nodes, one pair of x and y per row. For the triangle with three nodes such a matrix reads

$$[x] = \begin{bmatrix} x_1, & y_1 \\ x_2, & y_2 \\ x_3, & y_3 \end{bmatrix} \ . \tag{3.59}$$

Then the easiest way of computing the Jacobian matrix for the three node triangle is to employ matrix multiplications to carry out the summations in (3.57):

$$[J] = [x]^T \left[\mathrm{grad}_{(\xi,\eta)} N \right] \ . \tag{3.60}$$

and so, finally, we express the matrix of the ***gradients of the basis functions*** with respect to x, y on the three node triangle

$$\left[\mathrm{grad}_{(x,y)} N \right] = \begin{bmatrix} \mathrm{grad}_{(x,y)} N_1 \\ \mathrm{grad}_{(x,y)} N_2 \\ \mathrm{grad}_{(x,y)} N_3 \end{bmatrix} = \begin{bmatrix} \mathrm{grad} N_1 \\ \mathrm{grad} N_2 \\ \mathrm{grad} N_3 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_1}{\partial x}, & \dfrac{\partial N_1}{\partial y} \\[2ex] \dfrac{\partial N_2}{\partial x}, & \dfrac{\partial N_2}{\partial y} \\[2ex] \dfrac{\partial N_3}{\partial x}, & \dfrac{\partial N_3}{\partial y} \end{bmatrix}, \tag{3.61}$$

from (3.56) using (3.58)

$$\left[\text{grad}_{(x,y)}N\right] = \left[\text{grad}_{(\xi,\eta)}N\right][J]^{-1} . \tag{3.62}$$

The right-hand side is readily evaluated: the matrix $\left[\text{grad}_{(\xi,\eta)}N\right]$ is easily computed from the defini- tion of the basis functions on the standard element (for the triangle from (3.30), (3.28), and (3.29)), and the Jacobian matrix follows from the definition (3.60). When there is no possibility of confusion we could use the simplified notation $\text{grad}N_{\underline{j}} = \text{grad}_{(x,y)}N_{\underline{j}}$ and $[\text{grad}N] = \left[\text{grad}_{(x,y)}N\right]$.

3.9 Jacobian matrix for the triangle

The computation of the gradient of the basis functions critically depends on the Jacobian matrix. The Jacobian (determinant of the Jacobian matrix) should be positive. The Jacobian matrix will then be invertible. The Jacobian matrix for the three-node triangle can be directly obtained from (3.35)

$$[J] = \begin{bmatrix} (x_{\underline{2}} - x_{\underline{1}}) , & (x_{\underline{3}} - x_{\underline{1}}) \\ (y_{\underline{2}} - y_{\underline{1}}) , & (y_{\underline{3}} - y_{\underline{1}}) \end{bmatrix} . \tag{3.63}$$

It really consists of the two vectors shown in Figure 3.9. This can be readily used to see which conditions guarantee the determinant of the Jacobian matrix (i.e. the Jacobian) to be positive. Thinking about the geometrical meaning of the cross product in two dimensions, we can see that the cross product of $\boldsymbol{v}$ and $\boldsymbol{w}$ is in fact equal to the Jacobian, and so as long as these two vectors are not co-linear the Jacobian will be nonzero.

Another way of thinking about the Jacobian is in terms of area. By inspection of Figure 3.9 we can conclude that the Jacobian is twice the area of the triangle, to which we will refer to in the following as S_e,

$$\det[J] = 2S_e . \tag{3.64}$$

3.10 Bookkeeping

The following rules will apply in all hand calculations done in this book (such rules do not necessarily need to apply in all finite element programs: the developers are free to do whatever they want, as long as the result is correct).

The nodes of the mesh may be numbered arbitrarily. Each node will be associated with a degree of freedom. This information will be given in a table. For our example of Section 3.7 such a table would be trivial:

Node numbers	1	2	3	4	5
DOF numbers	1	2	3	4	5

It simply says that the degree of freedom numbers are the same as the node numbers. In general, they will not be.

We will assume that all degrees of freedom are numbered sequentially from 1 to N. First the **free degrees of freedom** will be numbered $1, 2, ..., N_{\text{f}}$ (those degrees of freedom that are actually unknown and need to be solved for), and only then the degrees of freedom that are **given as data** (known from the boundary conditions) will be numbered $N_{\text{f}} + 1, ..., N$. In our example the free DOFs reside at nodes 1, 2, and 3. That's why these three nodes have the first 3 degrees of freedom.

All elements connect some nodes. In a three-node triangle mesh each element connects three nodes. The so-called element **connectivity** are the numbers of the nodes, given in a counterclockwise order. Again, the connectivity of the triangles will be given by a table. For our example of Section 3.7

Element	Node 1	Node 2	Node 3
1	1	2	3
2	2	4	5
3	2	5	3

For each element we can therefore find out the degrees of freedom at its three nodes. We look at the row of the element connectivity table, and we pick out the degrees of freedom from the node-DOF table. The so-called element degree of freedom array (we will be abbreviating it as `edof` array) is the result. For instance, for element 3 the `edof` array reads $[2, 5, 3]$ (which is the same as the third row in the connectivity table, because in this particular example the node numbers are the same as the degree of freedom numbers).

There is another way we can account for the degrees of freedom. In the context of a single element, we can consider the degrees of freedom at the three nodes of the triangle. They are always numbered the same way as the three nodes of the triangle: $\underline{1}$, $\underline{2}$, and $\underline{3}$. We can call these "local" numbers as opposed to the global numbers that are counted in the entire mesh. The global degree of freedom numbers change from triangle to triangle (the `edof` array is different for each triangle), the local numbers are always the same.

3.11 Concrete-column example continued

Now that we know how to calculate the derivatives of the basis functions, we can continue the example from Section 3.7. The radius of the circular cross-section is $a = 2.5$m, and we take the internal angle of the pie-shaped domain as 15°. The dimensions in Figure 3.11 are then $d_x = 1.2074$m and $d_y = 0.3235$m. We consider the boundary conditions of (2.3–2.4). The mesh consists of three triangles, [1,2,3], [2,4,5], and [2,5,3]. The temperature along the boundary with water, i.e. along the edge [4,5], is known and that determines the temperature at the nodes 4 and 5. Temperatures at the nodes 1, 2, and 3 are unknown. We will associate the degrees of freedom with the nodes as

Node numbers	1	2	3	4	5
DOF numbers	3	1	2	5	4

The total number of degrees of freedom is $N = 5$, and the number of free (unknown) degrees of freedom is $N_\mathrm{f} = 3$.

As shown in Section 3.7, we will proceed by computing contributions to the algebraic equations element-by-element. The goal is to evaluate integrals such as

$$\int_e \mathrm{grad} N_j \; \kappa \mathrm{grad} N_i^T \; \Delta z \; \mathrm{d}S \tag{3.65}$$

But rather than iterate $i = 1, \ldots, N$ and $j = 1, \ldots, N_\mathrm{f}$ and compute the integral for all such pairs, we will calculate the *elementwise conductivity matrix* for the local degrees of freedom $\underline{k}$ and $\underline{m}$, where $\underline{k} = \underline{1}, \underline{2}, \underline{3}$, $\underline{m} = \underline{1}, \underline{2}, \underline{3}$, and when the matrix is calculated we will decide to which DOFs i and j it should contribute and *assemble* the appropriate numbers to the global (systemwide) quantities.

How do we evaluate the integral in the first place? Let us inspect one of the terms, for instance

$$\int_{e=1} \mathrm{grad} N_{\underline{k}} \; \kappa \mathrm{grad} N_{\underline{m}}^T \; \Delta z \; \mathrm{d}S \tag{3.66}$$

By inspecting the individual terms in the expression for the gradients of the basis functions (3.62), we can see that the gradients of the basis functions with respect to x, y are constant on each three-node triangle, since they are the product of two position-independent matrices, the gradient matrix (3.58), and the inverse of the Jacobian matrix (3.63). But that means that the integrand in (3.66) is a constant number since everything else, κ and Δz are also uniform on each triangle. That makes the integral trivial

$$\int_{e=1} \mathrm{grad} N_{\underline{k}} \; \kappa \mathrm{grad} N_{\underline{m}}^T \; \Delta z \; \mathrm{d}S = \mathrm{grad} N_{\underline{k}} \; \kappa \mathrm{grad} N_{\underline{m}}^T \; \Delta z \int_{e=1} \mathrm{d}S = \mathrm{grad} N_{\underline{k}} \; \kappa \mathrm{grad} N_{\underline{m}}^T \; \Delta z S_{e1}$$

$$(3.67)$$

where $S_{e1} = \int_{e=1} \mathrm{d}S$ is the area of the triangle $e = 1$. Also recall (3.64) so that

$$\int_{e=1} \mathrm{grad}N_{\underline{k}} \; \kappa \mathrm{grad}N_{\underline{m}}^T \; \Delta z \; \mathrm{d}S = (\det[J]/2)\mathrm{grad}N_{\underline{k}} \; \kappa \mathrm{grad}N_{\underline{m}}^T \; \Delta z \qquad (3.68)$$

Python - script

The actual calculating will be performed here with the help of Python. Evidently, it can also be done with a calculator or by hand. To begin, some required functions are imported from the modules `math` and `numpy`. We define some input quantities with obvious meanings (page 88):

```
20 a = 2.5 # radius of the column
21 dy = a/2*math.sin(15./180*math.pi)
22 dx = a/2*math.cos(15./180*math.pi)
23 Q = 4.5 # internal heat generation rate
24 k = 1.8 # thermal conductivity
25 Dz = 1.0 # thickness of the slice
```

The matrix of the gradients of the basis functions with respect to the parametric coordinates is defined as

```
28 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
```

Note that an array is defined row-by-row, for instance `[-1, -1]` is the first row of `gradNpar`, and `[1, 0]` and `[0, 1]` are the second and third row.

The locations of the nodes are defined with an array of coordinates of the nodes, one node per row.

```
30 xall = array([[0, 0], [dx, -dy], [dx, dy], [2*dx, -2*dy], [2*dx, 2*dy]])
```

and the definition of the degrees of freedom

```
31 # Numbers of the degrees of freedom
32 dof = array([3, 1, 2, 5, 4])
33 # Number of free degrees of freedom
34 N_f = 3
35 # Number of all degrees of freedom
36 N = 5
```

3.11.1 Element 1

And now we are ready to evaluate the expressions we need for the first element. The element is defined by listing its three nodes.

```
43 conn = array([1, 2, 3]) # The definition of the element, listing its nodes
```

We can then use this list to extract the coordinates of the three nodes that are connected by element 1. But we need to take into account the Python zero-based indexing: arrays rows and columns are pointed to by numbers `0` to `n-1` (where `n` is the number of rows or columns). Figure 3.12 illustrates the indexing of the entries of a matrix (an array), both when working with the matrix in hand calculations and when indexing the matrix in Python code. To recap: Python works with *offsets* of the rows and columns, whereas in hand calculation we refer to the *index* of the row or column.

So the node number 3 is stored in row 2 of the array `xall` and so on. Therefore, we can create an appropriate array of Python indexes by subtracting 1 from `conn` (page 88).

```
44 zconn =conn - 1 # zero-based node indexes
```

Then the coordinates of the three nodes of element 1 can be retrieved as

```
45 x = xall[zconn, :]# The coordinates  of the three nodes
```
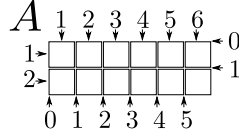
to give

Fig. 3.12. Indexing a 2×6 matrix A. Numbers at the top and along the left side: one-based indexing, of the kind that would be used when working with the matrix by hand. Numbers along the bottom and the right-hand side: zero-based indexing. Python uses this way of referring to the matrix entries by specifying the **offset** of the left-hand side or the top boundary of the entry's box from the top-left corner of the matrix. So the offset of the box of column 3 of the matrix from the top-left corner is 2. Similar scheme is used to refer to the rows of the matrix: the offset of the second row is 1. So for instance the entry in row 2 and column 3 of this matrix is referred to as A_{23} in hand calculations, and A[1, 2] in Python code.

```
x= [[ 0.          0.        ]
 [ 1.20740728 -0.32352381]
 [ 1.20740728  0.32352381]]
```

Now we calculate the Jacobian matrix from (3.60)

```
47 J = dot(x.T, gradNpar) # Compute the Jacobian matrix
48 Se = linalg.det(J)/2 # The area of the triangle
49 print('Se = ', Se)
```

The area of the triangle prints as Se= 0.390625.

Note the use of the dot function to compute the dot product of two matrices, and the use of the .T attribute to transpose a matrix. In the above we have also used the numpy.linalg linear algebra module, namely the function det to compute the determinant of the Jacobian matrix. Elementary geometry confirms that the area is correct: we calculate 0.390625 from the dimensions as dx*dy.

Elementwise conductivity matrix

Next we calculate the gradients of the basis functions on the first triangle with the respect to the x, y coordinates from (3.62). We will again use the linear algebra module, numpy.linalg, for the inverse of the Jacobian matrix (page 88).

```
51 gradN = dot(gradNpar, linalg.inv(J))
52 print('gradN=', gradN)
```

```
gradN =  [[-0.82822094  0.        ]
 [ 0.41411047 -1.54548132]
 [ 0.41411047  1.54548132]]
```

The first row of this matrix is the gradient $\mathrm{grad}N_{\underline{1}}$ and so on. Now we can calculate for instance

$$\int_{e=1} \mathrm{grad}N_{\underline{1}} \, \kappa \mathrm{grad}N_{\underline{1}}^T \, \Delta z \, \mathrm{d}S \tag{3.69}$$

as

```
54 print(Se*dot(gradN[0,:], gradN[0,:].T)*k*Dz)
```

which yields 0.482308546376, or

$$\int_{e=1} \mathrm{grad}N_{\underline{1}} \, \kappa \mathrm{grad}N_{\underline{2}}^T \, \Delta z \, \mathrm{d}S \tag{3.70}$$

as

```
55 print(Se*dot(gradN[0,:], gradN[1,:].T)*k*Dz)
```

which gives -0.241154273188. Note well the use of zero-based indexing of the Python arrays.

In this way we could calculate all nine terms for the various $\underline{k}$s and $\underline{m}$s one by one. Or, we could calculate a 3×3 matrix with a single line of code:

```
57 Ke1= (Se*dot(gradN, gradN.T)*k*Dz)
```

which gives the entire matrix as the array

```
Ke1= [[ 0.48230855 -0.24115427 -0.24115427]
 [-0.24115427  1.8         -1.55884573]
 [-0.24115427 -1.55884573  1.8         ]]
```

In other words, we can calculate the entire 3×3 elementwise conductivity matrix by

$$[K^{(e)}] = \int_{e=1} [\text{grad}N]\, \kappa[\text{grad}N]^T\, \Delta z\, \mathrm{d}S = (\det[J]/2)[\text{grad}N]\, \kappa[\text{grad}N]^T\, \Delta z \tag{3.71}$$

where $[\text{grad}N]$ was already defined in (3.61).

This elementwise matrix will now be assembled into the overall conductivity matrix: refer to the matrix (3.48). The $N_\mathrm{f} \times N$ matrix is initially full of zeros (page 88):

```
39 K = zeros((N_f, N))
```

The assembly is directed by the numbers of the degrees of freedom. They can be obtained from the array `dof`: don't forget that we need to use zero-based indexing and hence the array `zconn`. The code `print(dof[zconn])` prints `array([3, 1, 2])`. These are the degrees of freedom at the nodes 1, 2, 3, which also need to be adjusted for zero-based indexing, and so the element degree-of-freedom array needs to read

```
60 zedof = array(dof[zconn]) - 1
```

which gives `zedof` as `array([2, 0, 1])`.

The elementwise conductivity matrix will be assembled as shown:

$$
\begin{array}{r}
\text{row offset 2} \quad \{ \\
\text{row offset 0} \quad \{ \\
\text{row offset 1} \quad \{
\end{array}
\overbrace{\begin{bmatrix} 0.4823, \\ -0.2412, \\ -0.2412, \end{bmatrix}}^{\text{col. offset 2}}
\overbrace{\begin{matrix} -0.2412, \\ 1.8, \\ -1.559, \end{matrix}}^{\text{col. offset 0}}
\overbrace{\begin{matrix} -0.2412 \\ -1.559 \\ 1.8 \end{matrix}}^{\text{col. offset 1}}
\tag{3.72}
$$

Note well that the above uses the zero-based indexing suitable for Python coding. In hand calculations the assembly would be done with the numbers

$$
\begin{array}{r}
\text{to row 3} \{ \\
\text{to row 1} \{ \\
\text{to row 2} \{
\end{array}
\overbrace{\begin{bmatrix} 0.4823, \\ -0.2412, \\ -0.2412, \end{bmatrix}}^{\text{to col. 3}}
\overbrace{\begin{matrix} -0.2412, \\ 1.8, \\ -1.559, \end{matrix}}^{\text{to col. 1}}
\overbrace{\begin{matrix} -0.2412 \\ -1.559 \\ 1.8 \end{matrix}}^{\text{to col. 2}}
\tag{3.73}
$$

> The translation between one-based indexes and zero-based indexes only needs to be performed in the Python code, not in paper-and-pencil calculations. Here we show both ways. Later we only show the computer indexing: when solving by hand keep the original degree of freedom numbers.

The assembly into the two-dimensional global matrix is performed in our Python code using a double loop. Notice that all three indexes in `zedof` correspond to the free degrees of freedom, and hence all the entries of the elementwise conductivity matrix are assembled into the global matrix as follows (page 88):

```
62 for ro  in arange(len(zedof)):
63     for co  in arange(len(zedof)):
64         K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke1[ro, co]
65 print('K=', K)
```

The function `arange` generates the indexes 0, 1, and 2, all the way up to (but not including) the length of `zedof` (i.e. 3).

The intermediate stage of the global conductivity matrix looks like this

```
K =   [[ 1.8        -1.55884573 -0.24115427  0.         0.        ]
      [-1.55884573  1.8        -0.24115427  0.         0.        ]
      [-0.24115427 -0.24115427  0.48230855  0.         0.        ]]
```

after element 1 was assembled.

Elementwise heat load

To calculate the elementwise heat load vector due to the body-load Q, we need to calculate quantities such as (refer to equation (3.45))

$$\int_{e=1} N_{\underline{k}} Q \, \Delta z \, \mathrm{d}S \tag{3.74}$$

This is an exercise in elementary geometry. Consider for instance the basis function N_1 over the first triangle: the function is a plane triangular surface that is clamped down at value zero along the edge [2,3] and pinned to height one at node 1. The integral $\int_{e=1} N_1 Q \, \Delta z \, \mathrm{d}S$ is thus seen to be nothing else but the volume of the pyramid of height 1.0 and base of the triangular shape of element 1. Evidently

$$\int_{e=1} N_j Q \, \Delta z \, \mathrm{d}S = \frac{S_{e1} \Delta z Q}{3}, \quad j = 1, 2, 3 \tag{3.75}$$

It is instructive to consider the physical meaning of this formula: $S_{e1}\Delta z$ is the total volume represented by the element (S_{e1} is the area of the triangle and the depth in the other dimension is Δz). Multiply this by the rate of heat generation in units of power per unit volume, Q, and the meaning of $S_{e1}\Delta z Q$ is the total power generated inside the volume represented by the triangle. When we divide by three, we are assigning to each node of the triangle one third of the total power.

For element 1, the contribution to each node turns out to be numerically 0.5859, so that to the right-hand side load vector we assemble

$$
\begin{matrix}
\text{row offset 2} \\
\text{row offset 0} \\
\text{row offset 1}
\end{matrix}
\quad
\begin{Bmatrix}
0.5859 \\
0.5859 \\
0.5859
\end{Bmatrix}
\tag{3.76}
$$

The following code fragment assembles the contribution from element 1 to the global load vector (page 88):

```
68 LQe1 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
69 for ro  in arange(len(zedof)):
70     L[zedof[ro]] = L[zedof[ro]] + LQe1[ro]
```

3.11.2 Element 2

The element is defined by its three nodes [2,4,5] (page 88)

```
73 conn = array([2, 4, 5])
```

and therefore its nodes are located at

```
74 zconn = conn - 1 # zero-based node indexes
75 x = xall[zconn, :]# The coordinates  of the three nodes
```

which gives

```
array([[ 1.20740728, -0.32352381],
       [ 2.41481457, -0.64704761],
       [ 2.41481457,  0.64704761]])
```

The Jacobian matrix reads

```
array([[ 1.20740728,  1.20740728],
       [-0.32352381,  0.97057142]])
```

and the triangle area is 0.78125.

Elementwise conductivity matrix

For element 2 the procedure proceeds precisely as for element 1. This is why the finite element method is so well suited to the computer age: it is very easy to program, the procedure is just repeated for one more element, and the computer doesn't care whether it needs to compute the same thing 1000 or 1001 times. The elementwise conductivity matrix for element 2 results as

```
array([[ 0.96461709, -0.72346282, -0.24115427],
       [-0.72346282,  1.38230855, -0.65884573],
       [-0.24115427, -0.65884573,  0.9        ]])
```

The element degree of freedom array of element 2 obtains from the dof map as

```
84 zedof = array(dof[zconn]) - 1
```

i.e. `array([0, 4, 3])`. Notice that the global conductivity matrix is of dimension $N_f \times N = 3 \times 5$, and therefore in zero-based indexing we ignore any row offsets with value ≥ 3. Hence from the conductivity matrix of element 2

$$
\begin{array}{c}
\begin{array}{ccc}
\overbrace{\text{col. offset 0}} & \overbrace{\text{col. offset 4}} & \overbrace{\text{col. offset 3}}
\end{array} \\
\begin{array}{l}
\text{row offset 0} \quad \{ \\
\text{row offset 4} \quad \{ \\
\text{row offset 3} \quad \{
\end{array}
\left[
\begin{array}{rrr}
0.9646, & -0.7235, & -0.2412 \\
-0.7235, & 1.382, & -0.6588 \\
-0.2412, & -0.6588, & 0.9
\end{array}
\right]
\end{array}
\tag{3.77}
$$

the only assembled values will go to row 0.

In order to implement the assembly in Python code, we need to test the contents of the element degree-of-freedom array: when both the row index and the column index fall within the boundaries of the conductivity matrix, the corresponding entries from the elementwise matrix are added to the global matrix; otherwise they are simply ignored (page 88).

```
86 for ro  in arange(len(zedof)):
87     for co  in arange(len(zedof)):
88         if (zedof[ro] < N_f):
89             K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke2[ro, co]
```

For easy reference we also give here the variant of the above algorithm that could be used in hand calculations. In pseudocode:

```
for r  in 1, 2, 3
    for c  in 1, 2, 3
        if (edof[r] <= N_f)
            K[edof[r], edof[c]] = K[edof[r], edof[c]] + Ke2[r, c]
```

for `edof=[1, 5, 4]`, where `edof[2]` is 5 and so on.

Elementwise heat load

To calculate the elementwise heat load vector due to the body-load Q for element 2, equation (3.75) is applied to yield

$$
\begin{array}{ll}
\text{row offset 0} & \{ \\
\text{row offset 4} & \{ \\
\text{row offset 3} & \{
\end{array}
\begin{bmatrix}
1.1719 \\
1.1719 \\
1.1719
\end{bmatrix}
\tag{3.78}
$$

Again, as for the conductivity matrix, the second and third rows will be ignored; only the first row will be assembled into row offset 0 of the right-hand side vector. In Python code (page 88):

```
91 LQe2 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
92 for ro  in arange(len(zedof)):
93     if (zedof[ro] < N_f):
94         L[zedof[ro]] = L[zedof[ro]] + LQe2[ro]
```

3.11.3 Element 3

The third element is dispatched in exactly the same way as the first two. The element is defined by its three nodes

```
99 conn = array([2, 5, 3])
```

The Jacobian matrix is

```
array([[ 1.20740728  0.          ]
       [ 0.97057142  0.64704761]])
```

and the area is 0.390625.

Elementwise conductivity matrix

The gradients of the basis functions are calculated and the conductivity matrix of element 3 is readily evaluated as

```
array([[ 1.8        ,  0.24115427, -2.04115427],
       [ 0.24115427,  0.48230855, -0.72346282],
       [-2.04115427, -0.72346282,  2.76461709]])
```

The `zedof` array is now `array([0, 3, 1])`, which means that the assembled values will be those from the first and third row of the elementwise conductivity matrix.

$$
\begin{array}{l}
\text{row offset 0} \quad \{ \\
\text{row offset 3} \quad \{ \\
\text{row offset 1} \quad \{
\end{array}
\overbrace{
\begin{bmatrix}
1.8, \\
0.2412, \\
-2.041,
\end{bmatrix}}^{\text{col. offset 0}}
\overbrace{
\begin{matrix}
0.2412, \\
0.4823, \\
-0.7235,
\end{matrix}}^{\text{col. offset 3}}
\overbrace{
\begin{matrix}
-2.041 \\
-0.7235 \\
2.765
\end{matrix}}^{\text{col. offset 1}}
\tag{3.79}
$$

Elementwise heat load

To calculate the elementwise heat load vector due to the body-load Q for element 3, equation (3.75) is applied again to yield

$$
\begin{array}{ll}
\text{row index 0} & \{ \\
\text{row index 3} & \{ \\
\text{row index 1} & \{
\end{array}
\begin{bmatrix}
0.5859 \\
0.5859 \\
0.5859
\end{bmatrix}
\tag{3.80}
$$

Again, as for the conductivity matrix, the second row will be ignored; only the first and third rows will be assembled to rows index 0 and 1 of the right-hand side vector.

3.11.4 Assembled equations

The assembled equations for the free unknown nodal temperatures therefore result as

$$
\begin{bmatrix}
4.565, & -3.6, & -0.241, & 0, & -0.723 \\
-3.6, & 4.565, & -0.241, & -0.723, & 0 \\
-0.241, & -0.241, & 0.482, & 0, & 0
\end{bmatrix}
\begin{bmatrix}
T_1 \\ T_2 \\ T_3 \\ T_4 = 0 \\ T_5 = 0
\end{bmatrix}
=
\begin{bmatrix}
2.344 \\ 1.172 \\ 0.586
\end{bmatrix}
\tag{3.81}
$$

where we show all the individual assembled values from the elementwise quantities. The solution follows as

$$
\begin{bmatrix}
T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5
\end{bmatrix}
=
\begin{bmatrix}
2.906 \\ 2.763 \\ 4.049 \\ 0 \\ 0
\end{bmatrix}
\tag{3.82}
$$

This example has also been solved with Abaqus. The results agree to all reported digits. The elementwise matrices can be checked by the method outlined below: the input file for the execution of the job can be supplemented with print directives to generate an output for all elementwise matrices. [See Box 13]

3.12 Concrete-column example: nonzero boundary temperature

We will modify slightly the conditions of the example of Section 3.11. Instead of the water being at 0^oC we will take the temperature along the boundary edge [4,5] to be 10^oC. Therefore, as before, the temperatures at the nodes 1, 2, and 3 will be unknown, but the temperatures at nodes 4 and 5 will be $T_5 = 10$ and $T_4 = 10$.

When all the temperatures on the boundary were zero, we could see in the previous developments that we could totally ignore the known temperatures on the boundary. The zero temperatures had no effect on the solution. But, what if the temperatures at the nodes on the boundary were not zero? How do we handle that situation? The next section explains one possible approach.

3.12.1 Treatment of nonzero boundary conditions

To introduce the idea, we can profitably consider a really simple mechanical model as a stand-in for the heat conduction problem. Figure 3.13 shows four particles which can displace horizontally (but not vertically), so that each particle contributes one degree of freedom. One of the particles is "grounded" : connected to a support. This support can move by a prescribed amount, U_4. The other three degrees of freedom are unknown.

The balance equations of the four particles can be written down in terms of the forces acting on the particles, L, and the displacements of the particles, U as
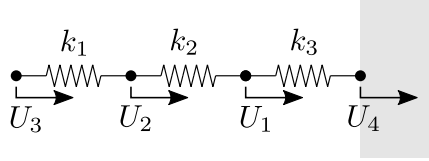
$$
L = KU
\tag{3.83}
$$

Here K is the stiffness matrix

Fig. 3.13. Four particles connected by three springs. The fourth particle is grounded as indicated with the light gray rectangle.

$$
\boldsymbol{K} = \begin{bmatrix} k_2 + k_3 & -k_2 & 0 & -k_3 \\ -k_2 & k_1 + k_2 & -k_1 & 0 \\ 0 & -k_1 & k_1 & 0 \\ -k_3 & 0 & 0 & k_3 \end{bmatrix} \tag{3.84}
$$

where k_j are the coefficients of the springs. The vector $\boldsymbol{U}$ collects the degrees of freedom

$$
\boldsymbol{U} = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3.85}
$$

where we grouped the unknown degrees of freedom in the top three rows. The bottom row holds the known displacement of the support, U_4.

The load vector consists of the applied forces, $\boldsymbol{F}$, and the reactions, $\boldsymbol{R}$

$$
\boldsymbol{L} = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ R_4 \end{bmatrix} = \boldsymbol{F} + \boldsymbol{R} \tag{3.86}
$$

For our purpose we can assume at the moment that all forces applied externally are zero, $F_j = 0$.

Given the numbering of the degrees of freedom, a natural partitioning of the balance equation follows. We can write (3.83) in the equivalent partitioned form as

$$
\begin{aligned}
\boldsymbol{K}_{ff} \boldsymbol{U}_f + \boldsymbol{K}_{fd} \boldsymbol{U}_d &= \boldsymbol{0} \\
\boldsymbol{K}_{df} \boldsymbol{U}_f + \boldsymbol{K}_{dd} \boldsymbol{U}_d &= \boldsymbol{R}_d
\end{aligned} \tag{3.87}
$$

Here we partitioned the stiffness matrix like so

$$
\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{ff} & \boldsymbol{K}_{fd} \\ \boldsymbol{K}_{df} & \boldsymbol{K}_{dd} \end{bmatrix} \tag{3.88}
$$

where

$$
\boldsymbol{K}_{ff} = \begin{bmatrix} k_2 + k_3 & -k_2 & 0 \\ -k_2 & k_1 + k_2 & -k_1 \\ 0 & -k_1 & k_1 \end{bmatrix}, \quad \boldsymbol{K}_{fd} = \begin{bmatrix} -k_3 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{K}_{df} = \begin{bmatrix} -k_3 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{K}_{dd} = \begin{bmatrix} k_3 \end{bmatrix} \tag{3.89}
$$

The vector of the degrees of freedom was partitioned into two pieces: the first for the the unknown (free) degrees of freedom, and the second for the given (datum) degrees of freedom. This will explain the notation: subscript f refers to free, subscript d refers to data.

$$
\boldsymbol{U}_f = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}, \quad \boldsymbol{U}_d = \begin{bmatrix} U_4 \end{bmatrix} \tag{3.90}
$$

The reaction vector was also partitioned, and the first part is omitted since it is identically zero.

$$\boldsymbol{R}_d = \begin{bmatrix} R_4 \end{bmatrix} \tag{3.91}$$

We note that the first of the equations (3.87) can be written in the usual form where the coefficient matrix and the vector of unknowns are on the left-hand side, and everything on the right-hand side is known.

$$\boldsymbol{K}_{ff}\boldsymbol{U}_f = -\boldsymbol{K}_{fd}\boldsymbol{U}_d \tag{3.92}$$

The coefficient matrix $\boldsymbol{K}_{ff}$ is square and invertible (its determinant is $(k_2+k_3)k_1^2 + ((k_2+k_3)(k_2+k_1)+k_2^2)k_1 \neq 0$). Therefore we can solve from (3.92) for the unknown displacements $\boldsymbol{U}_f$. If we wish to find the reaction, we use the second equation (3.87) to solve for the reactions

$$\boldsymbol{R}_d = \boldsymbol{K}_{df}\boldsymbol{U}_f + \boldsymbol{K}_{dd}\boldsymbol{U}_d \tag{3.93}$$

We call the term $-\boldsymbol{K}_{fd}\boldsymbol{U}_d$ the load due to support settlement. More generally, this is the load vector produced by nonzero essential boundary conditions. In the present case this is explicitly

$$-\boldsymbol{K}_{fd}\boldsymbol{U}_d = \begin{bmatrix} -k_3 U_4 \\ 0 \\ 0 \end{bmatrix} \tag{3.94}$$

Note that if we are not interested in the reactions, which is typically the case in heat conduction analysis, since the reactions are difficult to interpret in that case, we can simply work with only the matrices in (3.92), $\boldsymbol{K}_{ff}$, and $\boldsymbol{K}_{fd}$. In other words, the second row in (3.87) corresponding to the known displacement does not need to be calculated.

3.12.2 Application of the partitioning method to the heat conduction problem

Our method still works with the same equation, the weighted residual statement (3.45), but the terms multiplied with T_5 and T_4 do not drop out anymore. We will indicate this by splitting the sum over the degrees of freedom in (3.45) (which is for your convenience reproduced again here)

$$\sum_{i=1}^{N} T_i \sum_e \int_e \mathrm{grad}N_j \, \kappa \mathrm{grad}N_i^T \, \Delta z \, \mathrm{d}S - \sum_e \int_e N_j Q \, \Delta z \, \mathrm{d}S = 0, \quad \text{for } j = 1, \dots, N_f. \tag{3.95}$$

into

$$\underbrace{\sum_{i=1}^{N_f} T_i \sum_e \int_e \mathrm{grad}N_j \, \kappa \mathrm{grad}N_i^T \, \Delta z \, \mathrm{d}S}_{\text{free degrees of freedom } i = 1, 2, 3}$$

$$+ \underbrace{\sum_{i=N_f+1}^{N} T_i \sum_e \int_e \mathrm{grad}N_j \, \kappa \mathrm{grad}N_i^T \, \Delta z \, \mathrm{d}S}_{\text{datum degrees of freedom } i = 4, 5} - \sum_e \int_e N_j Q \, \Delta z \, \mathrm{d}S = 0, \quad \text{for } j = 1, \dots, N_f. \tag{3.96}$$

The first line of this equation are the unknown degrees of freedom T_i multiplied with the entries of the conductivity matrix (which are the integrals). The second line of this equation contains only known quantities. The first line is the left-hand side, and the second line is the right-hand side of the algebraic equations from which we need to solve for the free degrees of freedom. So a good way of rewriting the above is

$$\underbrace{\sum_{i=1}^{N_{\mathrm{f}}} T_i \sum_e \int_e \mathrm{grad}N_j \ \kappa \mathrm{grad}N_i^T \ \Delta z \ \mathrm{d}S =}_{\text{free degrees of freedom } i=1,2,3}$$

$$\underbrace{- \sum_{i=N_{\mathrm{f}}+1}^{N} T_i \sum_e \int_e \mathrm{grad}N_j \ \kappa \mathrm{grad}N_i^T \ \Delta z \ \mathrm{d}S}_{\text{fixed degrees of freedom } i=4,5} + \sum_e \int_e N_j Q \ \Delta z \ \mathrm{d}S, \quad \text{for } j=1,\dots,N_{\mathrm{f}} \ . \tag{3.97}$$

The first term in the second line is the loading produced by the nonzero fixed degrees of freedom. We can also call it "loading due to prescribed nonzero temperatures". This was discussed in the previous section for the spring-particle system as the support-settlement load.

We have dealt with the term on the first line and the second term on the second line in the previous section. Here we will include the loading due to prescribed nonzero temperature.

The actual calculating will be performed here with the help of a simple enhancement of the Python code used in the previous section. In addition to the previously described variables we also define `Tfix = 10` as the prescribed temperatures at the nodes 4 and 5 (page 91). For all the elements in the mesh the elementwise conductivity matrix and the heat load due to the hydration heat are precisely the same as above: no need to show them again. They are also assembled in the same way. In other words, the left-hand side coefficient matrix and the load vector due to the heat generation rate in the volume are the same as in (3.81). We will simply add the "support-settlement" heat loads due to the nonzero temperatures on the boundary.

Python
- script

In the beginning we set up the vector of the temperature degrees of freedom to consist of zeros where the unknowns are, and we set the degrees of freedom that are known to `Tfix = 10`.

```
39 # Vector of degrees of freedom (temperatures at the nodes)
40 T = zeros((N,)).reshape(N, 1)
41 # Set the temperatures at the data (given) degrees of freedom
42 T[N_f:N] = Tfix
```

Then the rectangular conductivity matrix is calculated and assembled as is the heat load vector due to the heat generation rate as shown in (3.81). Finally, the heat load vector due to the nonzero temperatures on the boundary (corresponding to $-\boldsymbol{K}_{fd}\boldsymbol{U}_d$ in (3.92) is calculated by

```
134 # Compute loading from the prescribed temperatures
135 LT = -dot(K[0:N_f, N_f:N], T[N_f:N])
```

This additional load is then used to calculate the 3 free degrees of freedom

```
139 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], L[0:N_f] + LT[0:N_f])
```

The assembled equations for the free unknown nodal temperatures therefore result as

$$\begin{bmatrix} 4.565, & -3.6, -0.241, & 0, -0.723 \\ -3.6, & 4.565, -0.241, & -0.723, & 0 \\ -0.241, & -0.241, & 0.482, & 0, & 0 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 = 10 \\ T_5 = 10 \end{bmatrix} = \begin{bmatrix} 2.344 \\ 1.172 \\ 0.586 \end{bmatrix} \tag{3.98}$$

and rearranging all known quantities onto the right hand side we get

$$\begin{bmatrix} 4.565, & -3.6, -0.241 \\ -3.6, & 4.565, -0.241 \\ -0.241, & -0.241, & 0.482 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = (-1) \times \begin{bmatrix} -10 \times 0.723 \\ -10 \times 0.723 \\ 0 \end{bmatrix} + \begin{bmatrix} 2.344 \\ 1.172 \\ 0.586 \end{bmatrix} \tag{3.99}$$

where we show all the individual assembled values from the elementwise quantities. The solution follows as

$$
\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{bmatrix} = \begin{bmatrix} 12.906 \\ 12.763 \\ 14.049 \\ 10 \\ 10 \end{bmatrix}
\tag{3.100}
$$

Notice that all the temperatures are just shifted upwards by 10^oC with respect to the case of the zero-temperature boundary conditions.

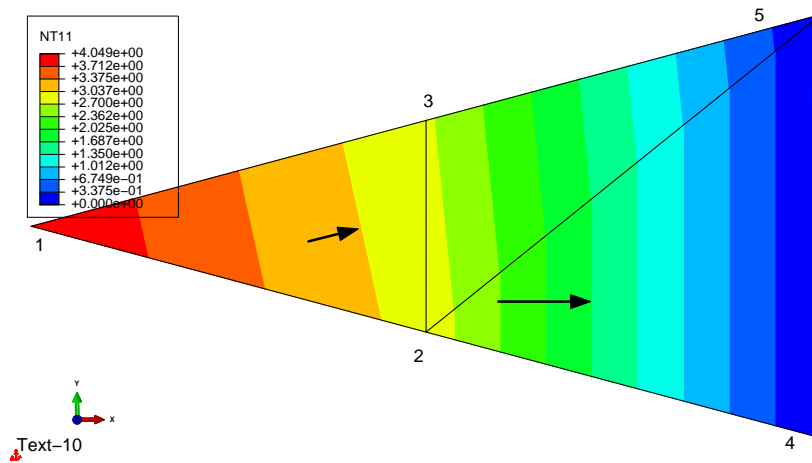### 3.12.3 Natural boundary conditions

Fig. 3.14. Mesh of three triangles for the concrete column with hydration heat load and zero temperature on the circumference. Abaqus solution. Heat flux indicated with arrows at two points.

The temperature varies linearly across each element, which means that the gradient of temperature is uniform. The heat flux in each element is consequently also uniform, as indicated for two elements in Figure 3.14 by the black arrows. These arrows represent the heat flux at any and all points of the triangle in which they are situated. That makes it unlikely that we will satisfy heat flux (natural) boundary conditions. For instance for triangle 123, we are trying to enforce zero-heat-flux boundary condition across the edges 12 and 31. While the black arrow gives hope that very little heat flows across the edge 31, clearly heat flows across the edge 12. Similarly for the edge 24, the heat flux component normal to that edge is nonzero. Therefore we must conclude that the natural boundary conditions are in the finite element method satisfied only *approximately*.

3.13 Layered wall example

In heat conduction problems one commonly encounters layered structures, for instance walls composed of several different materials. The thermal conductivity would be different in each layer, and the finite element mesh constructed along the thickness would need to reflect this by assigning different material properties to the elements.

As an example we will consider the distribution of temperature in the layered wall in Figure 3.15. Consider the following data: Polyurethane cladding $t_1 = 0.07$m, $\kappa_1 = 0.05$W/m/$^\circ$K, concrete $t_2 = 0.23$m, $\kappa_2 = 1.8$W/m/$^\circ$K. A triangle mesh of the width $w = 0.1$m is to be used. Also take $\Delta z = 1.0$m. The triangle connectivity is to be taken as (page 93)

```
28 # Connectivity of the left-region triangles (polyurethane)
29 connL = array([[4, 1, 5], [2, 5, 1]])
30 # Connectivity of the right-region triangles (concrete)
31 connR = array([[6, 5, 2], [2, 3, 6]])
```

The temperatures at the outer faces of the wall are prescribed as shown in the figure.
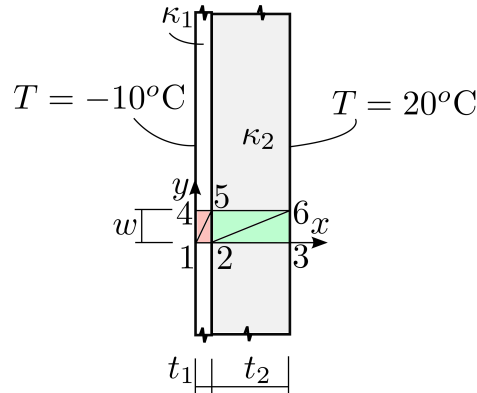

Fig. 3.15. Layered wall exposed to a temperature difference

The unknowns are temperatures T_1, T_2 at nodes 2 and 5. The fixed degrees of freedom are $T_3 = -10^\circ$C at node 1, $T_4 = 20^\circ$C at node 3, $T_5 = -10^\circ$C at node 4, and $T_6 = 20^\circ$C at node 6.

We will refer to the following variables below (page 93):

```
18 Dz = 1.0 # Thickness of the slice (it cancels out  in the end)
19 kappaL = 0.05  # thermal conductivity, layer on the left
20 kappaR = 1.8  # thermal conductivity, layer on the right
```

We also define the map from the nodes to the degrees of freedom as

```
23 N = 6 # total number of nodes
24 N_f = 2 # total number of free degrees of freedom
25 # Mapping from nodes to  degrees of freedom
26 node2dof = array([3, 1, 4, 5, 2, 6])
```

As advertised, the computation will be organized in loops over all the elements of first the polyurethane region, and then the concrete region.

First we set up a variable for the thermal conductivity (the one defined above for the polyurethane region), and the array of zero-based connectivities (page 93).

```
50 kappa = kappaL
51 zconn = array(connL) - 1
```

With these two variables set, the following loop is generic: it works both for the polyurethane region and the concrete region. All we have to do is to change the above two variables.

```
52 for j  in arange(zconn.shape[0]):
53     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
54     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
55     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
56     # Element degree-of-freedom array,  converted to zero base
57     zedof = array(node2dof[zconn[j, :]])-1
```

```
58      # Assemble elementwise conductivity matrix
59      for ro  in arange(len(zedof)):
60          for co  in arange(len(zedof)):
61              if (zedof[ro] < N_f):
62                  K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
```

The index variable `j` runs through all the triangles of the region as defined in the array `zconn`. First we compute the elementwise conductivity matrix, then we assemble it into the global conductivity matrix.

Thus we process the **polyurethane layer elements**: **Element 1**. The matrix of the basis functions gradients with respect to x, y reads

```
array([[-14.28571429,  10.         ],
       [  0.         , -10.         ],
       [ 14.28571429,   0.         ]])
```

The conductivity matrix is obtained from the expression

```
55      Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
```

as

```
array([[ 0.05321429, -0.0175    , -0.03571429],
       [-0.0175    ,  0.0175    ,  0.         ],
       [-0.03571429,  0.        ,  0.03571429]])
```

and assembled using the degree of freedom numbers `array([4, 2, 1])`.

Element 2: Importantly, triangle 2 has the same dimensions as triangle 1, and its nodes are numbered so that the first nodes can be made to coincide by translating and rotating triangle 2 to overlap triangle 1. When that is the case, the elementwise conductivity matrices *are the same*, which the code reproduces.

Next the procedure described above is repeated for the **concrete layer elements**: **Element 3**. Precisely the same code will still do the job, the only difference is the setup of the variables used within the main loop:

```
65 kappa = kappaR
66 zconn = array(connR)-1
```

Element 4: We were able to take advantage of the conductivity matrices of two elements being the same above. That is also available here: triangle 4 can be overlaid by triangle 3 by rotation and translation, and their first nodes then coincide. The conductivity matrix of elements 4 is then the same as the conductivity matrix of element 3. (Strictly speaking, the computer program doesn't care, but it *would* save valuable time in hand calculations.)

Importantly, the Python code to compute and assemble the elementwise quantities is precisely the same as that shown for the polyurethane region elements.

The global matrices and solution: The rectangular conductivity matrix of the structure is assembled from the elementwise matrices above as

$$[K] = \begin{bmatrix} 2.515, & -2.088, & -0.0357, & -0.391, & 0., & 0. \\ -2.088, & 2.515, & 0., & 0., & -0.0357, & -0.391 \end{bmatrix} . \tag{3.101}$$

Since degrees of freedom 5 and 3 (offsets 4 and 2, in zero-based indexing) are prescribed (data), we must calculate the heat load due to the prescribed temperatures. For this purpose we set up an array of the prescribed temperatures for each degree of freedom in the system, and the free degrees of freedom are initialized to 0.0:

```
32 pTe = -10  # Boundary conditions, exterior, interior, deg Celsius
33 pTi = +20
34 T = zeros(N).reshape(N, 1)
35 # Apply fixed temperatures at the left-hand side face
36 for index  in [1, 4]:
```

```
37    T[node2dof[index-1]-1] = pTe # left face, nodes 1 and 4
38
39 # Apply fixed temperatures at the right-hand side face
40 for index   in [3, 6]:
41    T[node2dof[index-1]-1] = pTi # right face, nodes 3 and 6
```

so that initially the array `T` contains

```
array([[  0.],
       [  0.],
       [-10.],
       [ 20.],
       [-10.],
       [ 20.]])
```

The heat load vector generated by the essential boundary conditions is computed as

```
81 LT = -dot(K[0:N_f, N_f:N], T[N_f:N])
```

resulting in

```
[[7.4689441]
 [7.4689441]]
```

The solution of the discrete equations with

```
84 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], LQ + LT)
```

then yields

$$
[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \begin{bmatrix} 17.49 \\ 17.49 \\ -10.00 \\ 20.00 \\ -10.00 \\ 20.00 \end{bmatrix} . \tag{3.102}
$$

Represented graphically the temperature variation shows the characteristic significant drop of the temperature through the insulation layer (polyurethane), see Figure 3.16. It is worth noting that the finite element solution is for the present model exact since the T3 elements can exactly represent linear variation of temperature.

> Put slightly differently: if the exact solution is a linearly varying temperature (with constant gradient, and hence constant heat flux), the finite element method *will* find this exact solution.

The problem was solved with Abaqus, and the results are displayed in Figure 3.16. The boundaries between the individual colors of the color bar are the level curves (isosurfaces). We would expect the temperature gradient to point left-to-right (the temperature on the left-hand side surface is 30^oC lower than on the right-hand side surface). Visually we can confirm that the temperature fits in with our common-sense understanding of the solution: checking whether the solution makes sense in these seat-of-the-pants ways is always a good thing to do.

Abaqus
- CAE file
- INP file
- tutorial

> Plotting with Python will not work in the command line window of Abaqus. To get the code below working requires one of the other options of running Python.

This is a good place to introduce Python plotting. The graphics will be produced with the well-known `matplotlib` module. To make the code a bit more legible, we import the module under the name `plt`.
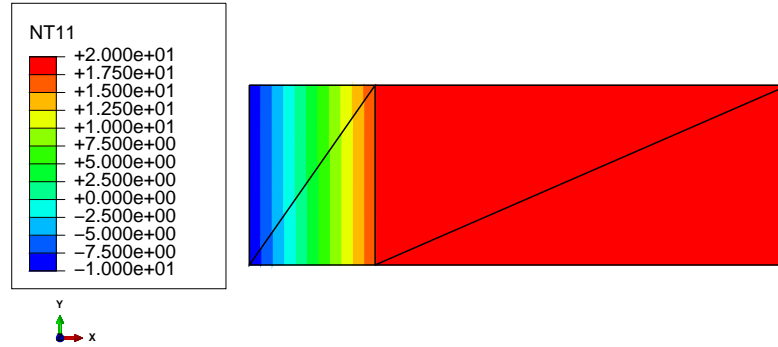
Fig. 3.16. Layered wall: variation of temperature

```
16 import matplotlib.pyplot as plt
```

Then we open a figure, create the axes, and set the aspect ratio.

```
87 # Plotting
88 plt.figure()
89 plt.gca().set_aspect('equal')
```

We set up two arrays for the two coordinates, one value per node.

```
90 # setup three 1-d arrays for the x-coord, the y-coord, and the z-coord
91 xs = x[:, 0].reshape(N,)# one value per node
92 ys = x[:, 1].reshape(N,)# one value per node
```

We also need a third array, to represent the temperature in the third coordinate direction. For this we need to reorder the `T` array so that it lists the temperatures one row per node, instead of one row per degree of freedom.

```
93 ix = node2dof[arange(N)]-1
94 zs = (T[ix]).reshape(N,)# one value per node
```

The mesh consists of the left-hand-side region and the right-hand-side region. Here we don't need to distinguish between the regions, and we will create a connectivity array for the entire mesh. We do need to adjust the connectivity arrays to be zero-based, since they are used to point into the `xs`, `ys`, and `zs` arrays.

```
95 triangles = vstack((connL-1, connR-1))# triangles are defined by the conn arrays
```

Finally we plot the filled contour surface, add the color bar and the title, and show the figure.

```
96 plt.tricontourf(xs, ys, triangles, zs)
97 plt.colorbar()
98 plt.title('Contour plot of temperature')
99 plt.xlabel('x (m)')
100 plt.ylabel('y (m)')
101 plt.show()
```

The result is presented in Figure 3.17.

3.14 Prescribed heat flux on the boundary

When we wish to incorporate as one kind of boundary condition a prescribed normal component of the heat flux, the weighted residual statement (3.45) needs to be augmented with one more term:
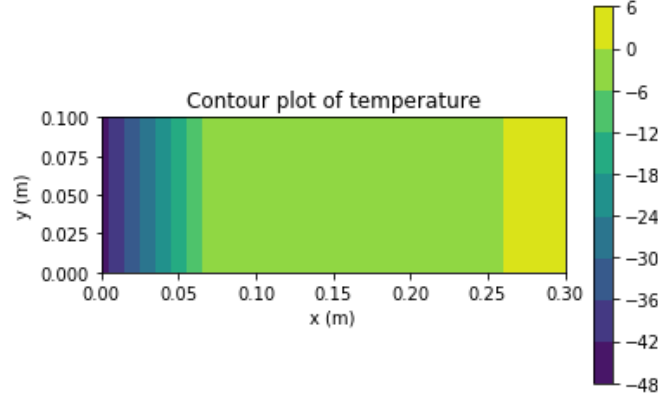
Fig. 3.17. Layered wall: variation of temperature displayed from the Python program

$$\sum_{i=1}^{N} T_i \int_{S_c} \text{grad} N_j \, \kappa \text{grad} N_i^T \, \Delta z \, \mathrm{d}S$$

$$- \int_{S_c} N_j Q \, \Delta z \, \mathrm{d}S + \int_{C_{c,2}} N_j \, \bar{q}_n \, \Delta z \, \mathrm{d}C = 0, \quad \text{for } j = 1, \dots, N_\mathrm{f} \, . \tag{3.103}$$

The last term on the left-hand side is of a different nature than those that we have seen so far: it is a surface integral. Refer to Figure 3.1: the "volume" integrals are evaluated over the cross-sectional area S_c (i.e. over the individual triangles of the mesh in the interior). The "surface" is represented in the figure by the curve C, and we call the part where heat flux is prescribed $C_{c,2}$.

At this point you should say "Wait a minute, we've already had surfaces with prescribed heat flux in our calculations. How come we haven't needed this surface term?" It is true that the prescribed heat flux boundary condition was already included before, for instance for the concrete column, the boundary edges [1,2], [2,4], [5,3], and [3,1] (Figure 3.11) were insulated, i.e. the boundary heat flux was known to be zero through these edges. The keyword is "zero", because then $\bar{q}_n = 0$ along these edges and the last term in (3.103) drops out as identically zero. That is why the heat-flux boundary condition was not mentioned before.

> The boundary condition with zero normal component of the heat flux does not need to be specified in any finite element program: if the boundary condition is not explicitly defined along any piece of the boundary surface, this condition is assumed (implied). It does not contribute to the discrete equations that the program needs to solve, and the effect of such a boundary condition is nil.

In this section we will however consider the case of nonzero prescribed heat flux. As indicated above, we will need to evaluate a surface integral. We needed a mesh of triangles to calculate the integrals that we call volume integrals (the first and the second in (3.103)), and we will also need a surface mesh to evaluate the surface integral. We will write

$$\sum_{i=1}^{N} T_i \sum_{e} \int_{e} \text{grad} N_j \, \kappa \text{grad} N_i^T \, \Delta z \, \mathrm{d}S$$

$$- \sum_{e} \int_{e} N_j Q \, \Delta z \, \mathrm{d}S + \sum_{e' \in C_{c,2}} \int_{e'} N_j \, \bar{q}_n \, \Delta z \, \mathrm{d}C = 0, \quad \text{for } j = 1, \dots, N_\mathrm{f} \, . \tag{3.104}$$

where e' is the number of an element from this surface mesh, and by

$$e' \in C_{c,2} \tag{3.105}$$

we mean that the heat-flux term needs to be evaluated only for surface elements of the boundary where nonzero heat flux is prescribed.

3.14.1 The L2 element

The most important requirement we have for the surface finite elements is that the temperature represented on the surface must match the temperature represented in the interior: the temperature must be continuous when we move from the interior to the surface or vice versa.

We can see from Figure 3.11 that the boundary of the mesh is formed by the triangle edges [1,2], [2,4], [5,3], [3,1], and [4,5]. The mesh of the boundary then must be formed from two-node elements in order to be **compatible** (to match up with) the mesh of the interior. Here we will introduce such a two-node element, called L2.

The basis functions of the L2 finite elements are expressed in terms of the parametric coordinate on the standard interval $-1 \le \xi \le +1$. Node $\underline{1}$ is located at $\xi = -1$ and node $\underline{2}$ is at $\xi = +1$. The basis functions are linear between the values zero and one at the two nodes (also see Figure 3.18):

$$N_{\underline{1}}(\xi) = \frac{\xi - 1}{-1 - 1} = \frac{\xi - 1}{-2} , \quad N_{\underline{2}}(\xi) = \frac{\xi - (-1)}{+1 - (-1)} = \frac{\xi + 1}{+2} . \tag{3.106}$$

The element can be used in any number of dimensions, but here we will think of two coordinates, x, y. The geometry of the element in the coordinates x, y will be described as

$$x(\xi) = N_{\underline{1}}(\xi)x_{\underline{1}} + N_{\underline{2}}(\xi)x_{\underline{2}} , \quad y(\xi) = N_{\underline{1}}(\xi)y_{\underline{1}} + N_{\underline{2}}(\xi)y_{\underline{2}} \tag{3.107}$$

where $x_{\underline{1}}, y_{\underline{1}}$ are the coordinates of the first node and so on. We can see that this entirely matches the process described earlier for the triangle (3.34). This element is another example of the *isoparametric* formulation.

> Note that the parametric coordinate ξ for the L2 element has nothing to do with the parametric coordinate of the same name for the element T3. The standard shapes of these elements are different and separate.

It can be shown that the basis functions on each triangle are linear in the coordinates x, y both within the triangle and on its boundary (meaning: along each of its edges). Similarly, for the L2 element it can also be shown that the basis functions expressed in terms of x, y along the element are linear in each of the arguments. See Box 11 These functions are also uniquely determined by the values at the nodes. Consequently, the T3 element and the L2 element are **compatible**.
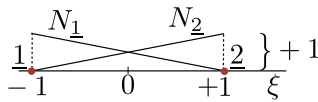
Fig. 3.18. L2 finite element: basis functions in the parametric coordinates

3.14.2 Using the L2 element

The goal is to compute

$$\int_{e'} N_j \, \bar{q}_n \, \Delta z \, \mathrm{d}C \tag{3.108}$$

along an arbitrary L2 element e', for the two degrees of freedom $j = 1, 2$. Refer to Figure 3.19 for an illustration. For simplicity we shall assume the practically important case of uniform prescribed normal component of heat flux $\bar{q}_n$.

The elementwise quantity will be evaluated on the element $[\underline{1}, \underline{2}]$ (see Figure 3.19). The mesh of the interior of the domain is also shown as filled triangles, but only for illustrative purposes, it is not used. The task is to evaluate

Fig. 3.19. L2 finite element of the boundary of the domain with prescribed heat flux

$$L_{q2,j} = -\int_{\boldsymbol{x}_{\underline{1}}}^{\boldsymbol{x}_{\underline{2}}} N_j \, \bar{q}_n \, \Delta z \, \mathrm{d}C \,, \quad j = 1, 2$$

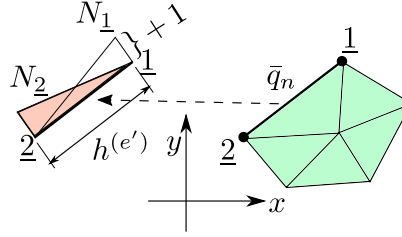where $\boldsymbol{x}_{\underline{k}}$ is the location of the kth node, and the curve C is the straight line that connects the nodes $\underline{1}$ and $\underline{2}$. This expression may be simplified as

$$L_{q2,j} = -\bar{q}_n \, \Delta z \int_{\boldsymbol{x}_{\underline{1}}}^{\boldsymbol{x}_{\underline{2}}} N_j \, \mathrm{d}C \,, \quad j = 1, 2$$

given that $\bar{q}_n \, \Delta z$ does not vary along the length of the element. By observation of the figure, where we especially pay attention to the filled triangle corresponding to $N_{\underline{1}}$, we can readily conclude that the integral $\int_{\boldsymbol{x}_{\underline{1}}}^{\boldsymbol{x}_{\underline{2}}} N_j \, \mathrm{d}C$ refers to the area of the triangle of height 1.0 and base equal to the length of the element $h^{(e')} = \|\boldsymbol{x}_{\underline{2}} - \boldsymbol{x}_{\underline{1}}\|$ (the filled triangle on the left side of the figure). Thus for uniform prescribed normal component of the heat flux we get the elementwise heat load vector

$$[L_{q2}]^{(e')} = -\frac{\bar{q}_n \, \Delta z h^{(e')}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.109}$$

Recall that the direction of the heat flux through the boundary is accounted for by the sign of $\bar{q}_n$: positive means heat flowing out from the domain, negative means heat flowing into the domain. More details about integrations of various quantities along curves can be found in the Background section $\boxed{\text{See Box 12}}$.

> The equations derived in this book assume that positive heat flux leaves *out* of the domain, which is consistent with it being the normal component of the heat flux on the outer normal vector. Abaqus, on the other hand, defines heat flux as positive when it flows *into* the domain. One must *always consult* the documentation on which particular convention is chosen in the software package at hand.

It is instructive to consider the physical meaning of the elementwise heat load vector: $\Delta z h^{(e')}$ is the area of the boundary surface along the edge of the element (depth Δz times the length $h^{(e')}$). Multiply that with the heat flux value $\bar{q}_n$, in units of power per unit area, and we get $\bar{q}_n \, \Delta z h^{(e')}$ as the total power passing through the boundary surface part that is represented by the element. Divide by two: each node gets one half of the total power passing through the element.

3.14.3 Layered wall: Heat flux boundary condition

Here we shall consider again the layered wall from Section 3.13, but the boundary condition at the right-hand side will be changed to assume known heat flux being input into the wall. Figure 3.15. The input data and the mesh are the same as before, except that on the right-hand face of the wall we prescribe the normal component of the heat flux $\bar{q}_n = -30 \text{ W/m}^2$ instead of the fixed temperature boundary condition.

Python
- script

> In the convention of our model negative value means the heat flux enters into the domain (the component on the outer normal is negative because the heat flux points opposite the normal); in Abaqus a negative value of the heat flux means the heat flux leaves the domain.

Importantly, we have to renumber the degrees of freedom, since now the degrees of freedom at the nodes 3 and 6 are also unknown. So in addition to the unknowns T_1, T_2 at nodes 2, 5, we also include unknowns T_3, T_4 at nodes 3 and 6. The fixed degrees of freedom are $T_5 = -10°\text{C}$ at node 1, and $T_6 = -10°\text{C}$ at node 4. We define (page 95)

```
25 N = 6 # total number of nodes
26 N_f = 4 # total number of free degrees of freedom
27 # Mapping from nodes to  degrees of freedom
28 node2dof = array([5, 1, 3, 6, 2, 4])
```

The elementwise conductivity matrices of the four elements are precisely the same as for the previous example. They just need to be reassembled into a new overall conductivity matrix (four rows this time). The same code that computed the conductivity matrix above still works and yields

$$[K] = \begin{bmatrix} 2.515, & -2.088, & -0.391, & 0., & -0.0357, & 0. \\ -2.088, & 2.515, & 0., & -0.391, & 0., & -0.0357 \\ -0.391, & 0., & 2.461, & -2.07, & 0., & 0. \\ 0., & -0.391, & -2.07, & 2.461, & 0., & 0. \end{bmatrix} \tag{3.110}$$

For the heat load vector contribution from the prescribed heat flux we need a surface mesh. In the present example the surface mesh consists of a single L2 element, connecting the nodes $[3, 6]$. We define

```
34 # Connectivity  of the boundary L2 element  on the right
35 connRbdry = array([[3, 6]])
```

The elementwise heat load vector due to the prescribed heat flux follows from (3.109)

$$[L_{q2}]^{(e')} = -\frac{\overline{q}_n \, \Delta z h^{(e')}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -\frac{-30 \times 1.0 \times 0.1}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 1.5 \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.111}$$

to be assembled to degrees of freedom 3 and 4 (at the nodes 3 and 6). The code to compute the elementwise heat load vector due to the prescribed heat flux reads (page 95)

```
79 qnbar = qnbarR
80 zconn = connRbdry - 1
81 for j  in arange(zconn.shape[0]):
82     he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
83     # Element degree-of-freedom array,  converted to zero base
84     zedof = array(node2dof[zconn[j, :]])-1
85     Leq = -qnbar*he*Dz/2*array([[1], [1]])
```

The first two lines again define variables used inside the loop. The loop itself is generic: it will work with any number of boundary finite elements with any numbering.

Here the `diff` function computes the difference of the locations of the two endpoints of the element (row-wise difference, note the `axis=0`): we have `x[zconn[j, :], :]`,

```
array([[ 0.3,  0. ],
       [ 0.3,  0.1]])
```

and therefore `diff(x[zconn[j, :], :], axis=0)` computes

```
array([[ 0. ,  0.1]])
```

and `linalg.norm` evaluates the norm (Euclidean length) of this vector as 0.1.

The elementwise load vector due to the applied heat flux is assembled in standard way as (page 95)

```
86      # Assemble elementwise heat load vector
87      for ro  in arange(len(zedof)):
88          if (zedof[ro] < N_f):
89              Lq[zedof[ro]] = Lq[zedof[ro]] + Leq[ro]
```

So the overall heat load vector including the heat load due to the prescribed temperature at the left-hand side reads

```
[[-0.35714286]
 [-0.35714286]
 [ 1.5       ]
 [ 1.5       ]]
```

The solution is obtained as (in degrees Celsius)

```
T= [[ 32.        ]
 [ 32.        ]
 [ 35.83333333]
 [ 35.83333333]
 [-10.        ]
 [-10.        ]]
```

Figure 3.20 was obtained with Abaqus. Figure 3.20(a) shows the variation of temperature across the domain, and (b) displays the arrows representing the heat flux at the six nodes. It is noteworthy that in this case the exact solution has again been found by the finite element procedure. For instance, observe that all six heat flux arrows are of the magnitude $\bar{q}_n = -30$ W/m^2. Another way of putting it is: what goes in must come out. At the right-hand side we prescribe the heat flux of this magnitude entering the wall. At steady state the heat energy cannot accumulate inside the wall, and therefore the same heat flux must again leave the wall at the left-hand side.
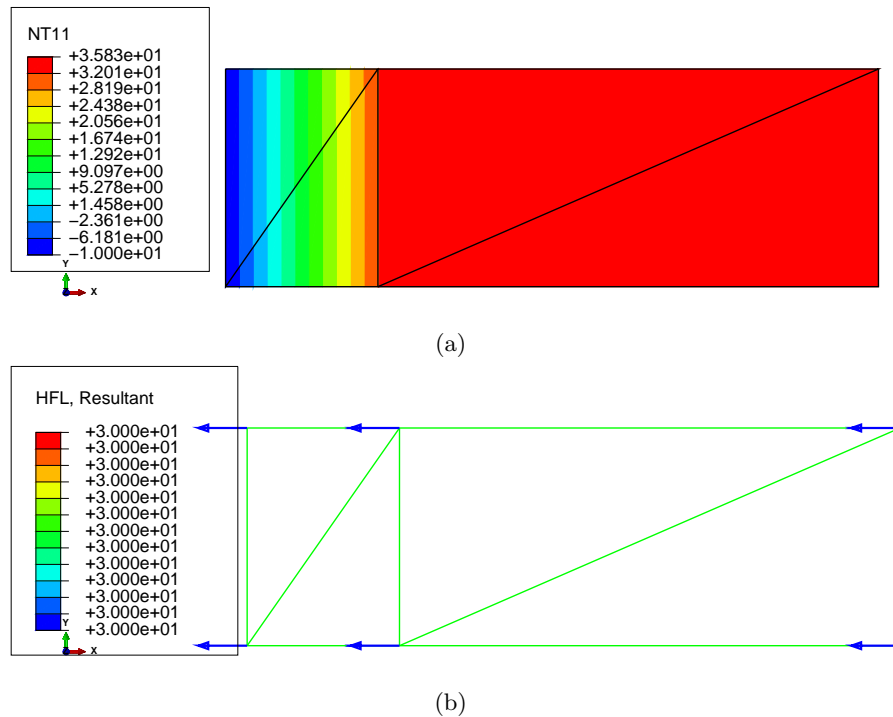
Abaqus
- CAE file
- INP file
- tutorial

(a)

(b)

Fig. 3.20. Layered wall with heat flux boundary condition on the right. (a) Temperature distribution. (b) Resultant heat flux at the nodes.

3.14.4 Layered wall: Heat flux boundary conditions only

Here we shall consider again the layered wall from Section 3.13, but the boundary condition on both sides will be changed to assume known heat flux. The heat flux will be assumed to go in at the right-hand side of the wall and to come out at the left-hand side. The input data and the mesh are the same as before (Figure 3.15), except that on the right-hand face of the wall we prescribe the normal component of the heat flux $\bar{q}_n = -30$ W/m^2 instead of the fixed temperature boundary condition, and on the left-hand side of the wall we prescribe the normal component of the heat flux $\bar{q}_n = +30$ W/m^2 (take note of the changed sign).

All nodes now have unknown degrees of freedom, and we take unknowns T_1, T_2 at nodes 2, 5, unknowns T_3, T_4 at nodes 3 and 6, unknowns T_5, T_6 at nodes 4 and 1 (page 97).

```
24 N = 6 # total number of nodes
25 N_f = 6 # total number of free degrees of freedom
26 # Mapping from nodes to  degrees of freedom
27 node2dof=array([6, 1, 3, 5, 2, 4])
```

As in the previous section, the elementwise conductivity matrices of the four elements do not need to be recalculated. They just need to be reassembled into a new overall 6×6 conductivity matrix:

$$[K] = \begin{bmatrix} 2.515, & -2.087, & -0.3913, & 0, & 0, & -0.03571 \\ -2.087, & 2.515, & 0, & -0.3913, & -0.03571, & 0 \\ -0.3913, & 0, & 2.461, & -2.07, & 0, & 0 \\ 0, & -0.3913, & -2.07, & 2.461, & 0, & 0 \\ 0, & -0.03571, & 0, & 0, & 0.05321, & -0.0175 \\ -0.03571, & 0, & 0, & 0, & -0.0175, & 0.05321 \end{bmatrix} \tag{3.112}$$

The global heat load vector is assembled from the heat-flux contributions from the two surface elements: In the present example the surface mesh consists of two L2 elements, the first on the right-hand side surface, connecting the nodes $[3, 6]$ (degrees of freedom $[3,4]$), and the second on the left-hand side surface, connecting nodes $[4, 1]$ (degrees of freedom $[5,6]$). The global heat load vector then reads

$$[L] = \begin{bmatrix} 0 \\ 0 \\ 1.5 \\ 1.5 \\ -1.5 \\ -1.5 \end{bmatrix} \tag{3.113}$$

We can clearly appreciate that each degree of freedom at the boundary node gets half of the power passing through the surface.

The solution in Python is produced as

```
Tg= [[ -3.17647059]
 [ -3.17647059]
 [  0.65686275]
 [  0.65686275]
 [-45.17647059]
 [-45.17647059]]
```

which means temperatures at the nodes $(1, 2, ..., 6)$ are -45.17647059, -3.17647059, 0.65686275, -45.17647059, -3.17647059, 0.65686275.

The solution was also calculated with Abaqus, with the results shown in Figure 3.21. This time the calculated temperatures at the nodes do not match our solution. The temperatures at the nodes $(1, 2, ..., 6)$ are reported as -42, 0, 3.83, -42, 0, 3.83.

To locate the source of this discrepancy, we must pay attention to the properties of the resulting system of algebraic equations. In fact, Abaqus complained about the system conductivity matrix: in the file with the extension `.msg` we find

```
***WARNING: Solver problem. Zero pivot when processing D.O.F. 11 of 1 nodes.
```
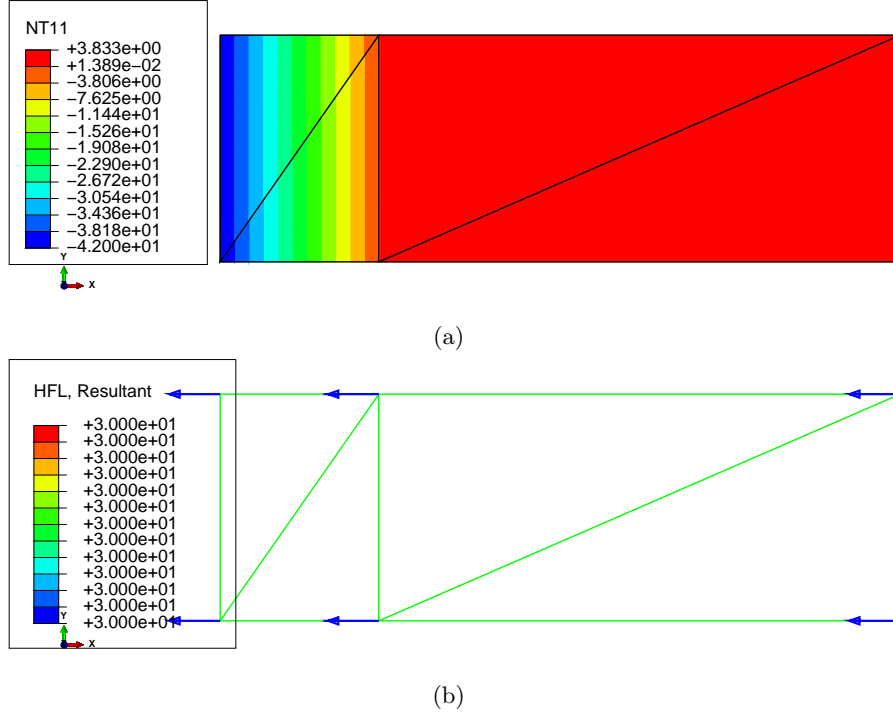

(a)

(b)

Fig. 3.21. Layered wall with heat flux boundary condition on both faces of the wall. (a) Temperature distribution. (b) Resultant heat flux at the nodes.

Here is an explanation: For any triangular element with three nodes we can write the temperature within the extent of this element as

$$T(x,y) = \sum_{i=1}^{3} N_i(x,y)T_i$$

From this we can compute the gradient of the temperature as

$$\text{grad}T(x,y) = \left[\frac{\partial T(x,y)}{\partial x} \; , \; \frac{\partial T(x,y)}{\partial y} \right] = \left[\frac{\partial \sum_{i=1}^{3} N_i(x,y)T_i}{\partial x} \; , \; \frac{\partial \sum_{i=1}^{3} N_i(x,y)T_i}{\partial y} \right]$$

which can be rewritten by taking into account that T_i are not subject to differentiation and by taking the differentiation into the sum

$$\text{grad}T(x,y) = \sum_{i=1}^{3} \left[\frac{\partial N_i(x,y)}{\partial x} \; , \; \frac{\partial N_i(x,y)}{\partial y} \right] T_i$$

At this point we realize that each of the bracket holds a basis function gradient (see (3.61))

$$\text{grad}N_i(x,y) = \left[\frac{\partial N_i(x,y)}{\partial x} \; , \; \frac{\partial N_i(x,y)}{\partial y} \right]$$

so that we can finally write

$$\text{grad}T(x,y) = \sum_{i=1}^{3} \text{grad}N_{\underline{i}}(x,y)T_{\underline{i}} = \sum_{i=1}^{3} T_{\underline{i}}\,\text{grad}N_{\underline{i}}(x,y)$$

This can be also written as a handy matrix expression

$$\text{grad}T(x,y) = \begin{bmatrix} T_{\underline{1}}, T_{\underline{2}}, T_{\underline{3}} \end{bmatrix} \begin{bmatrix} [\text{grad}N_{\underline{1}}(x,y)] \\ [\text{grad}N_{\underline{2}}(x,y)] \\ [\text{grad}N_{\underline{3}}(x,y)] \end{bmatrix}$$

or, even more succinctly

$$\text{grad}T(x,y) = [T^{(e)}]^T \left[\text{grad}N^{(e)}(x,y) \right]$$

where $[T^{(e)}]$ is the vector of the three nodal temperatures, and $\left[\text{grad}N^{(e)}(x,y)\right]$ is the matrix of the gradients of the three basis functions of the triangle (3.61). Now we recall the conductivity matrix for the single-element mesh (3.71) and we write the product of the elementwise conductivity matrix and the elementwise vector of temperatures

$$[K^{(e)}][T^{(e)}] = \left((\det[J]/2)[\text{grad}N]\,\kappa[\text{grad}N]^T\,\Delta z \right) [T^{(e)}] \tag{3.114}$$

Reshuffling results in

$$[K^{(e)}][T^{(e)}] = \Delta z(\det[J]/2)[\text{grad}N]\,\kappa \left([\text{grad}N]^T\,[T^{(e)}] \right)$$

and we realize

$$[\text{grad}N]^T\,[T^{(e)}] = (\text{grad}T)^T$$

so that the above may be rewritten

$$[K^{(e)}][T^{(e)}] = \Delta z(\det[J]/2)[\text{grad}N]\,\kappa(\text{grad}T)^T \tag{3.115}$$

Now, if all the nodal temperatures are the same, $T_{\underline{1}} = T_{\underline{2}} = T_{\underline{3}}$, the gradient of the temperature is identically zero, $\text{grad}T = [0,0]$, and we have

$$[K^{(e)}][T^{(e)}] = \Delta z(\det[J]/2)[\text{grad}N]\,\kappa \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.116}$$

> The elementwise conductivity matrix produces zero output when all the nodes have the same temperature.

Equation (3.116) is stating that the vector $[T^{(e)}]$ is a solution of a system of linear algebraic equations where the right-hand side consists of zeros. Such a matrix $[K^{(e)}]$ is singular.

> The elementwise conductivity matrix will always have **rank** of $(n-1)$ where n is the number of nodes of the element. That is, it will be **singular**. This is true not only for the three node triangle, but for any finite element applied to heat conduction.

So now we take the vector of the temperatures at the nodes to consist of the same numbers in each entry, for simplicity we pick 1.0

$$
\begin{bmatrix} T_1^{(e)} \\ T_2^{(e)} \\ T_3^{(e)} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix} \tag{3.117}
$$

Then the product of the elementwise conductivity matrix and the vector of temperatures will give

$$
\begin{bmatrix} K_{11}^{(e)}, K_{12}^{(e)}, K_{13}^{(e)} \\ K_{21}^{(e)}, K_{22}^{(e)}, K_{23}^{(e)} \\ K_{31}^{(e)}, K_{32}^{(e)}, K_{33}^{(e)} \end{bmatrix} \begin{bmatrix} T_1^{(e)} \\ T_2^{(e)} \\ T_3^{(e)} \end{bmatrix} = \begin{bmatrix} K_{11}^{(e)}, K_{12}^{(e)}, K_{13}^{(e)} \\ K_{21}^{(e)}, K_{22}^{(e)}, K_{23}^{(e)} \\ K_{31}^{(e)}, K_{32}^{(e)}, K_{33}^{(e)} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} K_{11}^{(e)} + K_{12}^{(e)} + K_{13}^{(e)} \\ K_{21}^{(e)} + K_{22}^{(e)} + K_{23}^{(e)} \\ K_{31}^{(e)} + K_{32}^{(e)} + K_{33}^{(e)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
$$
$$
\tag{3.118}
$$

In words, the values of the entries in each row of the elementwise conductivity matrix sum to zero! Now comes the crucial observation: if we take any number of finite elements and assemble their *entire* elementwise conductivity matrices into the global matrix, the entries in each row of the global conductivity matrix will also sum to zero. That means that the global conductivity matrix will also be *singular*. And that is why we are having trouble inverting it. The software does its best, probably by setting some entry of the solution to some arbitrary number.

If we take as the solution the numbers reported by Python and add $+3.17647059$, we will obtain the solution reported by Abaqus. Both of those solutions satisfy all the conditions (in particular the boundary conditions), and so since both of them are solutions, we must conclude that the solution is not unique. We can arbitrarily add some constant temperature and get another solution. The heat flux in the solution will not feel this added constant value of temperature, because as we pointed out earlier, constant temperature produces zero gradient of temperature and therefore zero heat flux.

This argument can be firmed up with just a few equations. We know that for a given element we can write the balance equation as

$$
[K^{(e)}][T^{(e)}] = [L^{(e)}] \tag{3.119}
$$

where $[L^{(e)}]$ is a vector of heat loads at the nodes. At the same time, we know that the conductivity matrix of the element is singular and therefore there is some nonzero vector of temperatures $[\widehat{T}^{(e)}]$ that follows as solution to the system of equations

$$
[K^{(e)}][\widehat{T}^{(e)}] = [0] \tag{3.120}
$$

with a zero heat load vector on the right-hand side. In fact, the solution is an arbitrary multiple of the vector $[\widehat{T}^{(e)}]$

$$
[K^{(e)}][\alpha\widehat{T}^{(e)}] = [0] \tag{3.121}
$$

where $\alpha \neq 0$. Adding together the left-hand sides and the right-hand sides of equations (3.119) and (3.121) yields

$$
[K^{(e)}][T^{(e)}] + [K^{(e)}][\alpha\widehat{T}^{(e)}] = [L^{(e)}] + [0] \ . \tag{3.122}
$$

As a result we have

$$
[K^{(e)}] \left([T^{(e)}] + \alpha[\widehat{T}^{(e)}] \right) = [L^{(e)}] \tag{3.123}
$$

which states that since the conductivity matrix of the element is singular, the solution to its balance equation for an arbitrary load vector $[L^{(e)}]$ is not unique: it is $[T^{(e)}]$ plus an arbitrary multiple of the eigenvector corresponding to the zero eigenvalue $[\widehat{T}^{(e)}]$.

> Heat conduction problems where all the boundary conditions are only of the heat-flux kind do not have unique solutions. The global conductivity matrix is singular.

In fact, it is possible that when the conductivity matrix is singular, there *is no solution*. In order to understand this, think of it this way: imagine that for the wall we would require for the heat flux to go in on either side of the wall. Would a steady-state solution be possible then? The answer must be no: it is not possible for heat to be continuously pumped into the wall without the total content of heat energy inside the wall continually increasing, and hence the temperature changing as a result. (There is no heat sink inside the wall to swallow that added heat.) The applied heat fluxes along the boundary must balance, what goes in must come out.

> Unless the heat fluxes balance (possibly with sources of heat and heat sinks), the solution to a problem with heat-flux boundary conditions only will not exist.

If we change both boundary conditions in Abaqus to list the normal heat flux as the same number (either +30 or -30), the solver will report an error and the solution will fail.

```
Job Job-1: Abaqus/Standard aborted due to errors.
Error in job Job-1: Abaqus/Standard Analysis exited with an error - Please
see the  message file for possible error messages if the file exists.
```

For these loads the solution could not be found (it does not exist!).

3.15 Concrete column with film boundary condition

In Section 3.14 the weighted residual statement to use in formulating a finite element model incorporated both a volume integral (for the conductivity matrix and the heat source loading) and a surface integral (for the flux boundary condition). In order to incorporate the third kind of boundary condition, the convection boundary condition (or the "surface film" boundary condition, as it is called in Abaqus), we need to include one more surface integral. (For the derivation please consult the Background material. $\boxed{\text{See Box 7}}$) Equation (3.104) is augmented to read

$$
\begin{aligned}
\sum_{i=1}^{N} T_i \sum_e & \int_e \mathrm{grad} N_j \ \kappa \mathrm{grad} N_i^T \ \Delta z \ \mathrm{d}S \\
& - \sum_e \int_e N_j Q \ \Delta z \ \mathrm{d}S + \sum_{e' \in C_{c,2}} \int_{e'} N_j \ \bar{q}_n \ \Delta z \ \mathrm{d}C \\
& - \sum_{e' \in C_{c,3}} \int_{e'} N_j \ h T_a \ \Delta z \ \mathrm{d}C \\
& + \sum_{i=1}^{N} T_i \sum_{e' \in C_{c,3}} \int_{e'} N_j \ h \ N_i \ \Delta z \ \mathrm{d}C = 0, \quad \text{for } j = 1, \ldots, N_{\mathrm{f}} \ .
\end{aligned}
\tag{3.124}
$$

> Equation (3.124) is the most general form of the steady-state heat-conduction model considered in this book. All the terms are included: conduction and internal heat generation rate in the volume, and prescribed temperatures, prescribed heat flux, and the surface film condition on the boundary.

We shall assume for simplicity that along each element on the boundary the heat surface transfer coefficient h (the surface film coefficient), and the ambient temperature T_a are constant. This is

not a major drawback, as the two quantities being constant along each boundary element is the practically important case, and anything more complicated can be approximated this way.

The term in line 3, on the $C_{c,3}$ part of the boundary, contributes to the heat load vector. Compare this integral to the integral above it on the part of the boundary $C_{c,2}$ where the normal component of the heat flux is prescribed: the difference between

$$\int_{e'} N_j \, \bar{q}_n \, \Delta z \, \mathrm{d}C \tag{3.125}$$

and

$$\int_{e'} N_j \, hT_a \, \Delta z \, \mathrm{d}C \tag{3.126}$$

is only the constant to be integrated, hT_a instead of $\bar{q}_n$. Similarly to the heat-flux load vector derived in Section 3.14.2, the load vector proportional to the ambient temperature heat flux hT_a follows as

$$[L_{q3}]^{(e')} = \frac{hT_a \, \Delta z h^{(e')}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.127}$$

The line 4 term in (3.124) is a little bit more challenging. It is a contribution to the left-hand side coefficient matrix, which we will call the surface heat transfer matrix. The elementwise matrix for the L2 element will be of dimension 2×2:

$$H_{ji} = \int_{C_{c,3}} N_j \, hN_i \, \Delta z \, \mathrm{d}C \, , \quad i, j = 1, 2. \tag{3.128}$$

Figure 3.22 shows graphically the product of a basis function with itself on the left, and the product of two distinct basis functions on the right. At the bottom of the figure are the parabolic triangles whose areas are the integrals of the products of the basis functions. The elementwise matrix therefore follows as

$$[H]^{(e')} = \frac{h\Delta z h^{(e')}}{6} \begin{bmatrix} 2, 1 \\ 1, 2 \end{bmatrix} \tag{3.129}$$

where $h^{(e')} = ||\boldsymbol{x}_1 - \boldsymbol{x}_2||$ is the length of the element.
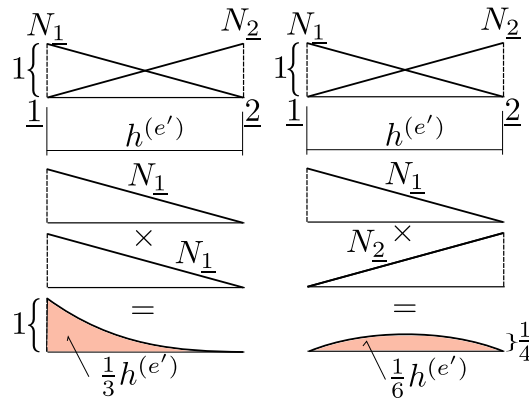

Fig. 3.22. Integral of the product of two basis functions along an L2 element

We will now slightly modify the example from Section 3.7 by replacing the prescribed temperature boundary condition with the convection boundary condition. We will consider the temperature of water $T_a = 0^o\mathrm{C}$ and the film coefficient $h = 5.0\,\mathrm{W} \cdot \mathrm{m}^{-2} \cdot {}^o\mathrm{K}^{-1}$. Temperatures at all the nodes are unknown, and we will associate the degrees of freedom with the nodes as

Node numbers	1	2	3	4	5
DOF numbers	3	1	2	5	4

The total number of degrees of freedom is $N = 5$, and the number of free (unknown) degrees of freedom is $N_f = 5$.

In addition to the triangles in the interior of the domain, we also have to consider the boundary mesh: a single L2 element along the edge [4,5]. The elementwise conductivity matrices of the three triangles are assembled fully into the global conductivity matrix (all degrees of freedom are free). Recall Section 3.14.4 — the global conductivity matrix will be singular. There it is, produced by `print(K)`:

```
array([[  4.56462e+00, -3.60000e+00, -2.41154e-01,  8.32667e-17, -7.23463e-01],
       [ -3.60000e+00,  4.56462e+00, -2.41154e-01, -7.23463e-01,  0.00000e+00],
       [ -2.41154e-01, -2.41154e-01,  4.82309e-01,  0.00000e+00,  0.00000e+00],
       [  8.32667e-17, -7.23463e-01,  0.00000e+00,  1.38231e+00, -6.58846e-01],
       [ -7.23463e-01,  0.00000e+00,  0.00000e+00, -6.58846e-01,  1.38231e+00]])
```

As the sums of the rows and columns we don't get precisely zeros, given the limited number of digits, but in double precision we get very small numbers indeed: the matrix is practically singular. Using `numpy.sum(Kg, axis=1)` (which means sum across the columns), we obtain

```
array([ -6.66133815e-16,  -8.88178420e-16,   0.00000000e+00,
         1.11022302e-16,   0.00000000e+00])
```

However, we are not done yet, there remain the surface terms. There is no prescribed heat flux: the second term on the second line of (3.124) may be ignored. The term on the third line is also zero, since T_a is zero. The surface heat transfer matrix, however, needs to be computed and assembled (page 99). The L2 element on the boundary has connectivity

```
42 connbdry = array([[4, 5]])
```

With the definition of the zero-based connectivity

```
67 zconn = connbdry - 1
```

we can compute the length of the finite element as

```
69     he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
```

This yields 1.2941. In terms of the local degrees of freedom, the surface heat transfer elementwise matrix for the L2 element [4,5] is computed as

```
72     He = h * Dz * he / 6 * array([[2, 1], [1, 2]])
```

The matrix reads

```
array([[ 2.15683,  1.07841],
       [ 1.07841,  2.15683]])
```

It will get assembled to the degrees of freedom 5 and 4 and therefore the assembled surface heat transfer matrix `Hg` (refer to the code below)

```
74     for ro in arange(len(zedof)):
75         for co in arange(len(zedof)):
76             if (zedof[ro] < N_f) and (zedof[co] < N_f):
77                 Hg[zedof[ro], zedof[co]] = Hg[zedof[ro], zedof[co]] + He[ro, co]
```

reads

```
array([[ 0.     ,  0.     ,  0.     ,  0.     ,  0.      ],
       [ 0.     ,  0.     ,  0.     ,  0.     ,  0.      ],
       [ 0.     ,  0.     ,  0.     ,  0.     ,  0.      ],
       [ 0.     ,  0.     ,  0.     ,  2.15683,  1.07841],
       [ 0.     ,  0.     ,  0.     ,  1.07841,  2.15683]])
```

Together, the global conductivity matrix in the global surface heat transfer matrix form the non-singular matrix `Kg + Hg`

```
array([[  4.56462e+00, -3.60000e+00, -2.41154e-01,  8.32667e-17, -7.23463e-01],
       [ -3.60000e+00,  4.56462e+00, -2.41154e-01, -7.23463e-01,  0.00000e+00],
       [ -2.41154e-01, -2.41154e-01,  4.82309e-01,  0.00000e+00,  0.00000e+00],
       [  8.32667e-17, -7.23463e-01,  0.00000e+00,  3.53913e+00,  4.19567e-01],
       [ -7.23463e-01,  0.00000e+00,  0.00000e+00,  4.19567e-01,  3.53913e+00]])
```

as can be readily verified by summing the rows and columns: the fourth and fifth row and column will not add up to zero anymore. So now the solution exists and it is also unique (page 99).

```
85 Tg = linalg.solve((Kg + Hg), Lg)
```

The solution for the 5 degrees of freedom is

```
array([[ 3.98612],
       [ 3.85657],
       [ 5.1362 ],
       [ 1.16556],
       [ 1.00778]])
```

Figure 3.23 shows the Abaqus solution with the node temperature values labeled using the Query tool. The first three temperatures match very well, but the last two temperatures, and those on the wet boundary are slightly different from ours. The explanation is the quadrature rule: Abaqus uses the trapezoidal rule on the boundary instead of the exact integration with a Gauss two-point rule. It is easy to check that the surface heat transfer matrix for the element turns out to be diagonal when calculated with the trapezoidal rule, and when that is reproduced in the Python code, the match is perfect at all five nodes.

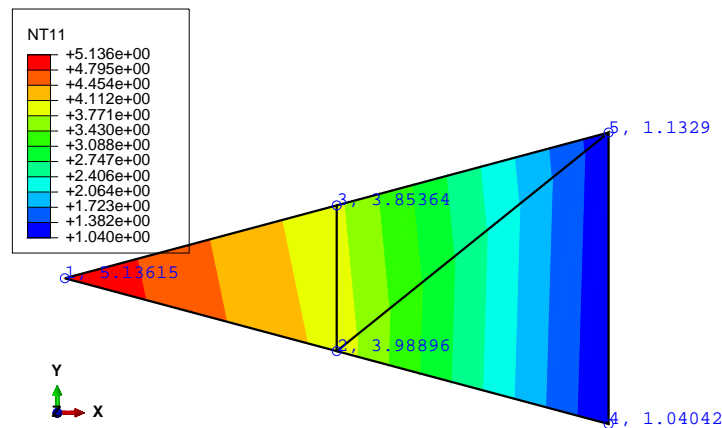Abaqus
- CAE file

Fig. 3.23. Concrete column with hydration heat load and convection boundary condition on the wet surface. Abaqus solution.

3.16 Background, explanations, details

Box 7. Weighted residual statement for heat conduction

The goal is to formulate an equation that will allow the finite element method to calculate an approximate solution to the heat conduction boundary value problem (BVP). The PDE was derived in Box 1 and the boundary conditions are discussed in Box 2.

The method chosen here is the weighted residual statement. The statement is derived in the form of an integral, so that the BVP is satisfied in an "average" sense.

In what follows, T is the so-called trial function, and ϑ is a weight function. The major steps are: satisfy the essential boundary conditions by designing the trial function, shift the derivatives in the balance equation residual, and combine the balanced equation residual with the natural boundary condition into one residual equation.

The essential boundary condition is satisfied by restricting possible trial functions to only those that conform to the essential boundary conditions a priori

$$T(\boldsymbol{x}, t) - \overline{T}(\boldsymbol{x}, t) = 0, \quad \boldsymbol{x} \text{ on } S_1 , \tag{3.130}$$

The next step equalizes the number of derivatives on the test and trial functions. The balance equation (2.1) yields the balance residual as

$$r_B = c_V \frac{\partial T}{\partial t} - \operatorname{div} \left[\boldsymbol{\kappa} (\operatorname{grad} T)^T \right] - Q . \tag{3.131}$$

so that the weighted balance residual reads

$$\int_V \vartheta(\boldsymbol{x}) r_B(\boldsymbol{x}, t) \, \mathrm{d}V = \int_V \vartheta \left(c_V \frac{\partial T}{\partial t} - \operatorname{div} \left[\boldsymbol{\kappa} (\operatorname{grad} T)^T \right] - Q \right) \, \mathrm{d}V . \tag{3.132}$$

The first term $(\vartheta c_V \frac{\partial T}{\partial t})$ and the third term (ϑQ) are kept without change, but in the second term

$$-\vartheta \operatorname{div} \left[\boldsymbol{\kappa} (\operatorname{grad} T)^T \right]$$

the test function ϑ multiplies an expression that contains the second derivatives of temperature (the div $\left[\boldsymbol{\kappa} (\operatorname{grad} T)^T \right]$ term). Balancing the order of differentiation by shifting one derivative from the temperature to the test function ϑ will be beneficial: we will be able to use basis functions that need to be less smooth since the second derivatives would not need to be taken, and also we will be able to satisfy the natural boundary conditions without having to include them as a separate residual (naturally!). As before, the price to pay is the need to place some restrictions on the test function.

For the moment, it will be convenient to work with the expression

$$-\vartheta \operatorname{div} \left[\boldsymbol{\kappa} (\operatorname{grad} T)^T \right] = \vartheta \operatorname{div} \boldsymbol{q} ,$$

that is, we work with the flux variable $\boldsymbol{q} = -\boldsymbol{\kappa} (\operatorname{grad} T)^T$.

The integration by parts in the case of a multidimensional integral is generalized in the divergence theorem (2.18). We may anticipate that $\vartheta \operatorname{div} \boldsymbol{q}$ is the result of the chain rule applied to the vector $\vartheta \boldsymbol{q}$. That is indeed the case, as we have

$$\operatorname{div} (\vartheta \boldsymbol{q}) = \vartheta \operatorname{div} \boldsymbol{q} + (\operatorname{grad} \vartheta) \boldsymbol{q} , \tag{3.133}$$

which is easily verified by working out the equality in components. Therefore, we may start by inspecting the integral

$$\int_V \vartheta \operatorname{div} \boldsymbol{q} \, \mathrm{d}V$$

where we substitute from (3.133)

$$\int_V \vartheta \operatorname{div} \boldsymbol{q} \, \mathrm{d}V = \int_V \operatorname{div} (\vartheta \boldsymbol{q}) \, \mathrm{d}V - \int_V (\operatorname{grad} \vartheta) \boldsymbol{q} \, \mathrm{d}V . \tag{3.134}$$

The divergence theorem may be applied to the first integral on the right to give the identity

$$\int_V \vartheta \operatorname{div} \boldsymbol{q} \, dV = \int_S \vartheta \, \boldsymbol{q} \cdot \boldsymbol{n} \, dS - \int_V (\operatorname{grad}\vartheta) \, \boldsymbol{q} \, dV \;. \tag{3.135}$$

Since $\boldsymbol{q} \cdot \boldsymbol{n}$ is known on some parts of the boundary, but unknown on the others – see Eqs. (2.28) and (2.29), we will split the surface integral into one for each sub-surface,

$$\int_V \vartheta \operatorname{div} \boldsymbol{q} \, dV =$$
$$\int_{S_1} \vartheta \, \boldsymbol{q} \cdot \boldsymbol{n} \, dS + \int_{S_2} \vartheta \, \boldsymbol{q} \cdot \boldsymbol{n} \, dS + \int_{S_3} \vartheta \, \boldsymbol{q} \cdot \boldsymbol{n} \, dS - \int_V (\operatorname{grad}\vartheta) \, \boldsymbol{q} \, dV \;. \tag{3.136}$$

The integral over the part of the surface S_1 is troublesome, because $\boldsymbol{q} \cdot \boldsymbol{n}$ is unknown there. However, we have the option of making ϑ vanish along S_1, making

$$\int_{S_1} \vartheta \, \boldsymbol{q} \cdot \boldsymbol{n} \, dS = 0 \;,$$

where the test function now must satisfy $\vartheta(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in S_1$. Now we switch back from $\boldsymbol{q}$ to $-\boldsymbol{\kappa}(\operatorname{grad}T)^T$ to obtain from (3.136)

$$\int_V \vartheta \operatorname{div} \boldsymbol{q} \, dV = -\int_V \vartheta \operatorname{div} \left[\boldsymbol{\kappa}(\operatorname{grad}T)^T \right] \, dV =$$
$$\int_{S_2} \vartheta \, (\boldsymbol{q} \cdot \boldsymbol{n}) \, dS + \int_{S_3} \vartheta \, (\boldsymbol{q} \cdot \boldsymbol{n}) \, dS + \int_V (\operatorname{grad}\vartheta) \, \boldsymbol{\kappa}(\operatorname{grad}T)^T \, dV \;. \tag{3.137}$$

The heat flux passing through the surface S_2 is known: see the boundary condition (2.28). Therefore, if we attempt to satisfy this boundary condition in a weighted residual sense we write

$$\int_{S_2} \vartheta \left[\bar{q}_n - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS = 0 \tag{3.138}$$

where we test with the function ϑ because we anticipate a possible cancellation with a term in (3.137). Note that the underlined term is present with the opposite sign in (3.136). Similarly, on the boundary surface S_3 we might attempt to satisfy the boundary condition with a weighted residual equation

$$\int_{S_3} \vartheta \left[h(T - T_a) - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS = 0 \tag{3.139}$$

Finally we have all the pieces ready. We take as the *weighted residual statement for the heat conduction* problem the weighted balance residual (3.132) to which we add the natural boundary condition residuals (3.138) and (3.139).

$$\underbrace{\int_V \vartheta \left(c_V \frac{\partial T}{\partial t} - \operatorname{div} \left[\boldsymbol{\kappa}(\operatorname{grad}T)^T \right] - Q \right) \, dV}_{(3.132)}$$
$$+ \underbrace{\int_{S_2} \vartheta \left[\bar{q}_n - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS}_{(3.138)} + \underbrace{\int_{S_3} \vartheta \left[h(T - T_a) - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS}_{(3.139)} = 0 \;. \tag{3.140}$$

Expression (3.137) is introduced to replace the second volume term.

$$\int_V \vartheta c_V \frac{\partial T}{\partial t} \, dV + \int_V (\operatorname{grad}\vartheta) \, \boldsymbol{\kappa}(\operatorname{grad}T)^T \, dV - \int_V \vartheta Q \, dV$$
$$+ \int_{S_2} \vartheta \, (\boldsymbol{q} \cdot \boldsymbol{n}) \, dS + \int_{S_3} \vartheta \, (\boldsymbol{q} \cdot \boldsymbol{n}) \, dS$$
$$+ \int_{S_2} \vartheta \left[\bar{q}_n - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS + \int_{S_3} \vartheta \left[h(T - T_a) - \underline{(\boldsymbol{q} \cdot \boldsymbol{n})} \right] \, dS = 0, \tag{3.141}$$
$$\text{where } \vartheta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in S_1 \;.$$

The underlined surface terms cancel with the corresponding terms on the second line and we arrive at

$$
\begin{aligned}
&\int_V \vartheta c_V \frac{\partial T}{\partial t}\,\mathrm{d}V + \int_V (\mathrm{grad}\vartheta)\,\boldsymbol{\kappa}(\mathrm{grad}T)^T\,\mathrm{d}V - \int_V \vartheta Q\,\mathrm{d}V \\
&+ \int_{S_2} \vartheta\,\overline{q}_n\,\mathrm{d}S + \int_{S_3} \vartheta\,h(T - T_a)\,\mathrm{d}S = 0, \qquad \vartheta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in S_1 .
\end{aligned}
\tag{3.142}
$$

In this equation we have our result: a single weighted residual statement that the trial function should satisfy to be considered an approximate solution to the original BVP.

End Box 7

Box 8. Reduction of the number of coordinates from from three to two

For some physical situations we can make the observation that the temperature does not vary significantly along one coordinate direction, say along the z direction. Figure 3.1 shows a disk of thickness Δz. It is a slice of a structure of an unchanging cross-section which is very long in the z direction compared to the transverse dimensions. If we can neglect what is happening near the end sections, and if the component of the temperature gradient along the z direction is negligible, $\partial T/\partial z \approx 0$, a necessary condition for the formulation of a simplified model is met.

However, it is not a sufficient condition as it does not necessarily mean that the z component of the heat flux is also zero: the partial derivatives $\partial T/\partial x$, and $\partial T/\partial y$ multiply the first two columns in row three of (2.22) to yield

$$
q_z = \kappa_{zx}\partial T/\partial x + \kappa_{zy}\partial T/\partial y .
$$

However, for the two classes of materials (2.23) (isotropic) and (2.24) (orthotropic, with diagonal conductivity matrix) the two coefficients κ_{zx} and κ_{zy} are identically zero, which means that if the temperature gradient $\partial T/\partial z$ is zero, the heat flux in that direction also vanishes.

End Box 8

Box 9. FEM, a little bit of history

The concept of piecewise linear functions defined over tilings of arbitrary domains into triangles is quite ancient (at least in terms of the development of computational mechanics). The so-called "linear triangle" made its first appearance in a lecture by Courant. Courant was a German-born mathematician. He founded the Courant Institute of Mathematical Sciences (as it was renamed in 1964) which continues to be one of the most respected research centers in applied mathematics in the world. Courant applied something that resembled the finite element method in 1943 to Poisson's equation, which is a time-independent version of the heat conduction BVP. It was then picked up as a structural element in aerospace engineering to model Delta wing skin panels, as described in the 1956 paper by Turner, Clough, Martin and Topp (Figure 3.24(a)). Ray William Clough, was a Professor of Structural Engineering in the department of Civil Engineering at the University of California, Berkeley and one the founders of the Finite Element Method (FEM). His article in 1956 was one of the first applications of this computational method. He also coined the term "finite element" in an article from 1960.
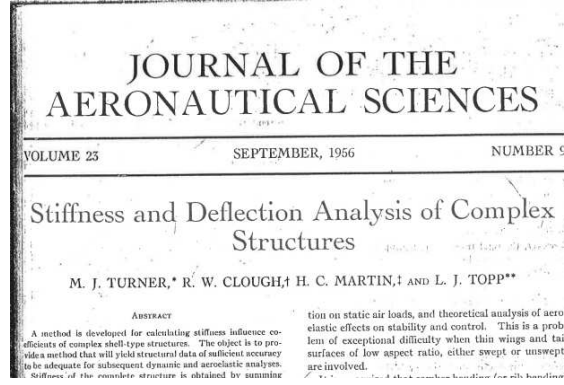
End Box 9

Fig. 3.24. The famous 1956 paper by Turner, Clough, Martin and Topp

---

Box 10. Derivatives of the basis functions

To evaluate the derivatives of the basis functions with respect to x and y we need the chain rule. Equations (3.28–3.30) define the functions over the standard triangle in terms of ξ and η. Therefore, to express $\partial N_i / \partial x$ we use the chain rule

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial x} \,,$$
$$\frac{\partial N_i}{\partial y} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial y} \,.$$

For the purpose of this discussion, the function that is being differentiated does not really matter. We will replace it with a $\heartsuit$, while we arrange the above equation into a matrix expression

$$\left[\frac{\partial \heartsuit}{\partial x},\ \frac{\partial \heartsuit}{\partial y}\right] = \left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]\begin{bmatrix}\dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y}\end{bmatrix} = \left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]\left[\tilde{J}\right] \,. \tag{3.143}$$

The derivatives are arranged in row matrices because these objects are **_gradients_** of the $\heartsuit$ function [compare with (2.25)]. The row matrix

$$\left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]$$

is the gradient of the function $\heartsuit$ with respect to the coordinates ξ, η and this is the gradient of the same function with respect to coordinates x, y

$$\left[\frac{\partial \heartsuit}{\partial x},\ \frac{\partial \heartsuit}{\partial y}\right] \,.$$

The matrix

$$\left[\tilde{J}\right] = \begin{bmatrix}\dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y}\end{bmatrix} \,, \tag{3.144}$$

is the ***Jacobian matrix*** of the mapping $\xi = \xi(x, y), \eta = \eta(x, y)$, which is the inverse of the map $x = x(\xi, \eta), y = y(\xi, \eta)$ of Eq. (3.35). The question is how to evaluate the partial derivatives of the type $\partial \xi / \partial x$, since the inverse of the map (3.35) is not known (at least not in general).

Here is an idea: If we start the chain rule from the other end (switching the role of the variables), we obtain

$$
\left[\frac{\partial \heartsuit}{\partial \xi}, \ \frac{\partial \heartsuit}{\partial \eta} \right] = \left[\frac{\partial \heartsuit}{\partial x}, \ \frac{\partial \heartsuit}{\partial y} \right] \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} ,
\tag{3.145}
$$

and inverting the Jacobian matrix $\left[\widetilde{J} \right]$ in equation (3.143) we get

$$
\left[\frac{\partial \heartsuit}{\partial \xi}, \ \frac{\partial \heartsuit}{\partial \eta} \right] = \left[\frac{\partial \heartsuit}{\partial x}, \ \frac{\partial \heartsuit}{\partial y} \right] \left[\widetilde{J} \right]^{-1} .
\tag{3.146}
$$

Comparing (3.145) and (3.146) yields

$$
[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \left[\widetilde{J} \right]^{-1} ,
\tag{3.147}
$$

where $[J]$ is the ***Jacobian matrix*** of the map (3.35).

Hence, the derivatives of the basis functions with respect to the coordinates x, y are evaluated as

$$
\left[\frac{\partial N_i(\xi, \eta)}{\partial x}, \ \frac{\partial N_i(\xi, \eta)}{\partial y} \right] = \left[\frac{\partial N_i(\xi, \eta)}{\partial \xi}, \ \frac{\partial N_i(\xi, \eta)}{\partial \eta} \right] [J]^{-1} .
\tag{3.148}
$$

End Box 10

Box 11. Form of the basis functions for the T3 triangle and the L2 line element

Triangle T3: When we map the point from the standard triangle ξ, η to the point x, y using (3.34) or, explicitly, (3.35), we obtain a linear relationship between those two pairs of coordinates, x, y expressed in terms of ξ, η. Therefore, inverting this relationship will result in a linear dependence of ξ, η on x, y. Substituting ξ, η in the expressions for the basis functions (3.28–3.30) will express the basis functions as linear functions of x, y. This will make perfect sense, since a linear function in x, y is uniquely determined by the values of three coefficients, and we have three nodes that each determine the basis function through the Kronecker delta property (3.31).

Line L2: Expanding (3.107) in terms of the basis functions, we obtain

$$
x(\xi) = \frac{x_1 + x_2}{2} + \xi \frac{x_2 - x_1}{2} , \quad y(\xi) = \frac{y_1 + y_2}{2} + \xi \frac{y_2 - y_1}{2}
\tag{3.149}
$$

This is clearly have linear relationship between each of x and y and ξ. The same reasoning as above leads us to conclude that the basis functions are in fact linear functions of x, y.

End Box 11

Box 12. Evaluating integrals along curves

Clearly we need some understanding of integrals along curves. The goal is to evaluate

$$\int_C f(\boldsymbol{p}) \, \mathrm{d}C \ , \tag{3.150}$$

where we will assume that the curve C may be "embedded" in a three-dimensional, two-dimensional, or one-dimensional Euclidean space (i.e. it may be a spatial curve, plane curve, or just an interval on the real line). Correspondingly, the point $\boldsymbol{p}$ on the curve C will have an appropriate number of components, three, two, or one.

To perform the integral, the elementary length $\mathrm{d}C$ is needed. The point $\boldsymbol{p}$ on the curve will be assumed to be the result of the mapping of the standard interval $-1 \le \xi \le +1$ (compare with the 1-D map (3.106), and refer to Fig. 3.25, where the map is two-dimensional)

$$\boldsymbol{p} = \boldsymbol{g}(\xi) \ . \tag{3.151}$$

For two closely spaced points on the curve, $\boldsymbol{p}(\xi)$ and $\boldsymbol{p}(\xi + \mathrm{d}\xi)$, where $\Delta\xi$ is the distance between
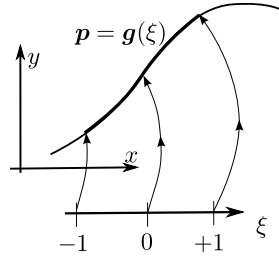

Fig. 3.25. Mapping of the standard interval to a Cartesian space

the two points in the standard interval, the second point may be obtained from the first using the first two terms of the Taylor series as

$$\boldsymbol{p}(\xi + \mathrm{d}\xi) = \boldsymbol{p}(\xi) + \frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi}\mathrm{d}\xi \ , \qquad 0 \le \varepsilon \le 1. \tag{3.152}$$

The two points may be connected with a vector approximately tracking the curve (see Fig. 3.26),

$$\boldsymbol{p}(\xi + \mathrm{d}\xi) - \boldsymbol{p}(\xi) = \frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi}\mathrm{d}\xi \ ,$$

whose length (squared) is

$$(\mathrm{d}C)^2 = \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi}\mathrm{d}\xi\right) \cdot \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi}\mathrm{d}\xi\right) = \left\|\frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi}\right\|^2 (\mathrm{d}\xi)^2 \ .$$

Skipping over the details, we may conclude that for infinitesimally short intervals

$$\frac{\partial \boldsymbol{p}(\xi + \varepsilon\mathrm{d}\xi)}{\partial\xi} \rightarrow \frac{\partial \boldsymbol{p}(\xi)}{\partial\xi}$$

and the following relationship is obtained

$$\mathrm{d}C = \left\|\frac{\partial \boldsymbol{p}(\xi)}{\partial\xi}\right\| \mathrm{d}\xi \ , \tag{3.153}$$

where $\dfrac{\partial \boldsymbol{p}(\xi)}{\partial\xi}$ is the vector ***tangent*** to the curve at ξ, and $\left\|\dfrac{\partial \boldsymbol{p}(\xi)}{\partial\xi}\right\|$ is the Jacobian to be used in the change-of-variables operation for the curve integral.
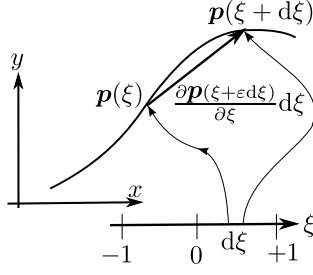
Fig. 3.26. Length of a curve

As an example, let us evaluate the integral $\int_C \, \mathrm{d}C$ along the curve generated by the map

$$\boldsymbol{p} = \boldsymbol{g}(\xi) = \sum_{i=1}^{2} N_i(\xi)\boldsymbol{x}_i \, , \quad -1 \le \xi \le +1 \, , \tag{3.154}$$

where the N_i's are given by (3.106). The integrand is $f(\boldsymbol{p}) = 1$ so that the meaning of the integral is the length of the curve.

The solution is as follows: The two endpoints are $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and because the functions (3.106) are linear, the point defined by (3.154) lies along a straight line connecting the two endpoints. In other words, the generated curve is a straight line segment.

The tangent vector is computed as

$$\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} = \sum_{i=1}^{2} \frac{N_i(\xi)}{\partial \xi}\boldsymbol{x}_i = \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{2} \, ,$$

and the Jacobian is

$$J = \left\| \frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} \right\| = \left\| \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{2} \right\| = \frac{h}{2}$$

where $h = \|\boldsymbol{x}_2 - \boldsymbol{x}_1\|$ is the length of the segment between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

The integral is evaluated as

$$\int_C \, \mathrm{d}C = \int_{-1}^{+1} J\mathrm{d}\xi = J \int_{-1}^{+1} \, \mathrm{d}\xi = \frac{h}{2} \int_{-1}^{+1} \, \mathrm{d}\xi = \frac{h}{2} \times 2 = h$$

So the length of the curve is h, as expected.

End Box 12

Box 13. Example of Abaqus input file and execution using the command line

The Abaqus/CAE graphical user interface is used to construct an input file to the solver. The input file can be found under the job name, for instance `Job-1.inp`. The input file describes completely what needs to be simulated and how. Most actions can be composed in the graphical user interface, but for a few select cases one needs to modify the input file by hand. In this example we show an input file from the example on page 51. The model is described by this `Job-1.inp` file:

```
*Heading
** Job name: Job-1 Model name: concrete-column-w-conv-pie-2d
```

```
** Tested with Abaqus/CAE Student Edition 6.14-2
*Preprint, echo=NO, model=NO, history=NO, contact=NO
**
** PARTS
*Part, name=THE-PART
*Node
      1,           0.,           0.
      2,    1.20739996, -0.323500007
      3,    1.20739996,  0.323500007
      4,    2.41479993, -0.647000015
      5,    2.41479993,  0.647000015
*Element, type=DC2D3
1, 1, 2, 3
2, 2, 4, 5
3, 2, 5, 3
*Elset, elset=Set-1, generate
 1,  3,  1
** Section: Section-1
*Solid Section, elset=Set-1, material=Material-1
1.,
*End Part
**
**
** ASSEMBLY
*Assembly, name=Assembly
**
*Instance, name=THE-PART-1, part=THE-PART
*End Instance
**
*Nset, nset=Set-1, instance=THE-PART-1
 4, 5
*Elset, elset=_Surf-1_S2, internal, instance=THE-PART-1
 2,
*Surface, type=ELEMENT, name=Surf-1
_Surf-1_S2, S2
*End Assembly
**
** MATERIALS
*Material, name=Material-1
*Conductivity
 1.8,
** ----------------------------------------------------------------
**
** STEP: the-step
*Step, name=the-step, nlgeom=NO
*Heat Transfer, steady state, deltmx=0.
1., 1., 1e-05, 1.,
**
** LOADS
** Name: the-body-load   Type: Body heat flux
*Dflux
THE-PART-1.Set-1, BF, 4.5
**
** INTERACTIONS
** Interaction: Int-1
*Sfilm
Surf-1, F, 0., 5.
**
```

```
** OUTPUT REQUESTS
*Restart, write, frequency=0
**
** FIELD OUTPUT: F-Output-1
*Output, field, variable=PRESELECT
*Output, history, frequency=0
** Note:
** ELSET=THE-PART-1.Set-1 refers to instance THE-PART-1, element set Set-1
** STIFF=YES means in heat transfer  "output the conductivity matrix"
** OUTPUT=USER,FILE=elmats means write the output into the file elmats.mtx
*ELEMENT MATRIX OUTPUT,ELSET=THE-PART-1.Set-1,STIFF=YES,OUTPUT=USER,FILE=elmats
*End Step
```

The line with the keyword `*ELEMENT MATRIX OUTPUT` can be used to obtain information about the elementwise quantities (matrices and vectors).

On Windows, the job can be submitted for execution by bringing up the command window and typing

```
abaqus job=job-1 ask_delete=off
```

In addition to the usual output files, the file `elmats.mtx` containing the matrices for the elements from the set `Set-1` will be located in the working folder. For the first element, the output will read (compare with the matrix on page 54):

```
**
** ELEMENT NUMBER         1 STEP NUMBER         1 INCREMENT NUMBER        1
** ELEMENT TYPE  DC2D3
*USER ELEMENT, NODES=         3, LINEAR
** ELEMENT NODES
**          1,          2,          3
        11
*MATRIX,TYPE=STIFFNESS
 0.48227599129621      ,
 -.24113799564810      ,  1.8001052273240
 -.24113799564810      , -1.5589672316759      ,  1.8001052273240
```

Similarly for the other elements of the set.

Alternatively, the input file may be modified from the Abaqus/CAE graphical user interface by bringing up the keyword editor (menu `Model->Edit keywords->`*model name*). The following line needs to be attached at the bottom of the file just above the line that states `*End Step`:

```
*ELEMENT MATRIX OUTPUT,ELSET=THE-PART-1.Set-1,STIFF=YES,OUTPUT=USER,FILE=elmats
```

The job for this model can then be executed from the Job Manager as usual.

End Box 13

3.17 Code listings

pnpConcreteColumn.py

Python
- script

```python
1 # Finite Element Modeling with Abaqus and Python for Thermal and
2 # Stress Analysis
3 # (C)  2017-2020, Petr Krysl
4 """
5 Concrete column with hydration heat and zero temperature on the boundary.
```

```python
 6 Numerical solution. This process is shown step-by-step, which means that
 7 the listing is much longer than it needs to be. Later we use loops, which
 8 makes the code much more succinct.
 9 """
10
11 import math
12 from numpy import array as array
13 from numpy import zeros as zeros
14 from numpy import ones as ones
15 from numpy import arange as arange
16 from numpy import dot as dot
17 from numpy import linalg as linalg
18
19 # These are the constants in the problem, k is kappa
20 a = 2.5 # radius on the columnthe
21 dy = a/2*math.sin(15./180*math.pi)
22 dx = a/2*math.cos(15./180*math.pi)
23 Q = 4.5 # internal heat generation rate
24 k = 1.8 # thermal conductivity
25 Dz = 1.0 # thickness of the slice
26
27 #  Gradients of the basis functions wrt the parametric coords
28 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
29 #Coordinates of the nodes. Node 1 in first row, and so on.
30 xall = array([[0, 0], [dx, -dy], [dx, dy], [2*dx, -2*dy], [2*dx, 2*dy]])
31 # Numbers of the degrees of freedom
32 dof = array([3, 1, 2, 5, 4])
33 # Number of free degrees of freedom
34 N_f = 3
35 # Number of all degrees of freedom
36 N = 5
37
38 # Global conductivity matrix and heat load vector.
39 K = zeros((N_f, N))
40 L = zeros((N_f,)).reshape(N_f, 1)
41
42 #First element
43 conn = array([1, 2, 3]) # The definition of the element, listing its nodes
44 zconn = conn - 1 # zero-based node indexes
45 x = xall[zconn, :]# The coordinates  of the three nodes
46 print('x = ', x)
47 J = dot(x.T, gradNpar) # Compute the Jacobian matrix
48 Se = linalg.det(J)/2 # The area of the triangle
49 print('Se = ', Se)
50 # Compute the gradient with respect to X, Y
51 gradN = dot(gradNpar, linalg.inv(J))
52 print('gradN = ', gradN)
53 # Some terms of the conductivity matrix
54 print(Se*dot(gradN[0, :], gradN[0, :].T)*k*Dz)
55 print(Se*dot(gradN[0, :], gradN[1, :].T)*k*Dz)
56 # The entire elementwise conductivity matrix
57 Ke1 = (Se*dot(gradN, gradN.T)*k*Dz)
58 print('Ke1 = ', Ke1)
59 # Element degree-of-freedom array,  converted to zero base
60 zedof = array(dof[zconn])-1
61 # Assemble contribution from element 1
62 for ro  in arange(len(zedof)):
63     for co  in arange(len(zedof)):
```

```python
64            K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke1[ro, co]
65
66  print('K = ', K)
67  # Compute heat load from element 1
68  LQe1 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
69  for ro  in arange(len(zedof)):
70      L[zedof[ro]] = L[zedof[ro]] + LQe1[ro]
71
72  ##Second element
73  conn = array([2, 4, 5])
74  zconn = conn - 1 # zero-based node indexes
75  x = xall[zconn, :]# The coordinates  of the three nodes
76  J = dot(x.T, gradNpar) # Compute the Jacobian matrix
77  Se = linalg.det(J)/2 # The area of the triangle
78  # Compute the gradient with respect to X, Y
79  gradN = dot(gradNpar, linalg.inv(J))
80  # The entire elementwise conductivity matrix
81  Ke2 = (Se*dot(gradN, gradN.T)*k*Dz)
82  print(Ke2)
83  # Element degree-of-freedom array,  converted to zero base
84  zedof = array(dof[zconn])-1
85  # Assemble contribution from element 2
86  for ro  in arange(len(zedof)):
87      for co  in arange(len(zedof)):
88          if (zedof[ro] < N_f):
89              K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke2[ro, co]
90
91  print(K)
92  # Compute heat load from element 2
93  LQe2 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
94  for ro  in arange(len(zedof)):
95      if (zedof[ro] < N_f):
96          L[zedof[ro]] = L[zedof[ro]] + LQe2[ro]
97
98  ##Third element
99  conn = array([2, 5, 3])
100 zconn = conn - 1 # zero-based node indexes
101 x = xall[zconn, :]# The coordinates  of the three nodes
102 J = dot(x.T, gradNpar) # Compute the Jacobian matrix
103 Se = linalg.det(J)/2 # The area of the triangle
104 # Compute the gradient with respect to X, Y
105 gradN = dot(gradNpar, linalg.inv(J))
106 # The entire elementwise conductivity matrix
107 Ke3 = (Se*dot(gradN, gradN.T)*k*Dz)
108 # Element degree-of-freedom array,  converted to zero base
109 zedof = array(dof[zconn])-1
110 # Assemble contribution from element 3
111 for ro  in arange(len(zedof)):
112     for co  in arange(len(zedof)):
113         if (zedof[ro] < N_f):
114             K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke3[ro, co]
115
116 print(K)
117 # Compute heat load from element 3
118 LQe3 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
119 for ro  in arange(len(zedof)):
120     if (zedof[ro] < N_f):
121         L[zedof[ro]] = L[zedof[ro]] + LQe3[ro]
```

```
122
123 print(L)
124
125 # Solution:
126 T = linalg.solve(K[0:N_f, 0:N_f], L)
127 print('Solution T = ', T)
```

Listing 3.1. pnpConcreteColumn.py

pnpConcreteColumnNZEBC.py

```python
1  # Finite Element Modeling with Abaqus and Python for Thermal and
2  # Stress Analysis
3  # (C)  2017-2020, Petr Krysl
4  """
5  Concrete column with hydration heat and non-zero temperature on the boundary.
6  Numerical solution. This process is shown step-by-step, which means that
7  the listing is much longer than it needs to be. Later we use loops, which
8  makes the code much more succinct.
9  """
10
11 import math
12 from numpy import array as array
13 from numpy import zeros as zeros
14 from numpy import ones as ones
15 from numpy import arange as arange
16 from numpy import dot as dot
17 from numpy import linalg as linalg
18
19 # These are the constants in the problem, k is kappa
20 a = 2.5 # radius on the columnthe
21 dy = a/2*math.sin(15./180*math.pi)
22 dx = a/2*math.cos(15./180*math.pi)
23 Q = 4.5 # internal heat generation rate
24 k = 1.8 # thermal conductivity
25 Dz = 1.0 # thickness of the slice
26
27 #  Gradients of the basis functions wrt the parametric coords
28 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
29 # Coordinates of the nodes. Node 1 in first row, and so on.
30 xall = array([[0, 0], [dx, -dy], [dx, dy], [2*dx, -2*dy], [2*dx, 2*dy]])
31 # Numbers of the degrees of freedom
32 dof = array([3, 1, 2, 5, 4])
33 # Number of free degrees of freedom
34 N_f = 3
35 # Number of all degrees of freedom
36 N = 5
37 # Prescribed temperatures at the nodes 4  and 5
38 Tfix = 10
39 # Vector of degrees of freedom (temperatures at the nodes)
40 T = zeros((N,)).reshape(N, 1)
41 # Set the temperatures at the data (given) degrees of freedom
42 T[N_f:N] = Tfix
43
44 # Global conductivity matrix and heat load vector.
45 K = zeros((N_f, N))
46 L = zeros((N_f,)).reshape(N_f, 1)
47
```

```
48  # First element
49  conn = array([1, 2, 3]) # The definition of the element, listing its nodes
50  zconn = conn - 1 # zero-based node indexes
51  x = xall[zconn, :]# The coordinates  of the three nodes
52  print('x = ', x)
53  J = dot(x.T, gradNpar) # Compute the Jacobian matrix
54  Se = linalg.det(J)/2 # The area of the triangle
55  print('Se = ', Se)
56  # Compute the gradient with respect to X, Y
57  gradN = dot(gradNpar, linalg.inv(J))
58  print('gradN = ', gradN)
59  #
60  ## Some terms of the conductivity matrix
61  print(Se*dot(gradN[0, :], gradN[0, :].T)*k*Dz)
62  print(Se*dot(gradN[0, :], gradN[1, :].T)*k*Dz)
63
64  # The entire elementwise conductivity matrix
65  Ke1 = (Se*dot(gradN, gradN.T)*k*Dz)
66  print('Ke1 = ', Ke1)
67  # Element degree-of-freedom array,  converted to zero base
68  zedof = array(dof[zconn])-1
69  # Assemble contribution from element 1
70  for ro  in arange(len(zedof)):
71      for co  in arange(len(zedof)):
72          if (zedof[ro] < N_f):
73              K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke1[ro, co]
74
75  print('K = ', K)
76
77  # Compute heat load from element 1
78  LQe1 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
79  for ro  in arange(len(zedof)):
80      if (zedof[ro] < N_f):
81          L[zedof[ro]] = L[zedof[ro]] + LQe1[ro]
82
83  # Second element
84  conn = array([2, 4, 5])
85  zconn = conn - 1 # zero-based node indexes
86  x = xall[zconn, :]# The coordinates  of the three nodes
87  J = dot(x.T, gradNpar) # Compute the Jacobian matrix
88  Se = linalg.det(J)/2 # The area of the triangle
89  # Compute the gradient with respect to X, Y
90  gradN = dot(gradNpar, linalg.inv(J))
91  # The entire elementwise conductivity matrix
92  Ke2 = (Se*dot(gradN, gradN.T)*k*Dz)
93  print(Ke2)
94  # Element degree-of-freedom array,  converted to zero base
95  zedof = array(dof[zconn])-1
96  # Assemble contribution from element 2
97  for ro  in arange(len(zedof)):
98      for co  in arange(len(zedof)):
99          if (zedof[ro] < N_f):
100             K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke2[ro, co]
101
102 print(K)
103 # Compute heat load from element 2
104 LQe2 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
105 for ro  in arange(len(zedof)):
```

```
106     if (zedof[ro] < N_f):
107         L[zedof[ro]] = L[zedof[ro]] + LQe2[ro]
108
109 # Third element
110 conn = array([2, 5, 3])
111 zconn = conn - 1 # zero-based node indexes
112 x = xall[zconn, :]# The coordinates  of the three nodes
113 J = dot(x.T, gradNpar) # Compute the Jacobian matrix
114 Se = linalg.det(J)/2 # The area of the triangle
115 # Compute the gradient with respect to X, Y
116 gradN = dot(gradNpar, linalg.inv(J))
117 # The entire elementwise conductivity matrix
118 Ke3 = (Se*dot(gradN, gradN.T)*k*Dz)
119 # Element degree-of-freedom array,  converted to zero base
120 zedof = array(dof[zconn])-1
121 # Assemble contribution from element 2
122 for ro  in arange(len(zedof)):
123     for co  in arange(len(zedof)):
124         if (zedof[ro] < N_f):
125             K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke3[ro, co]
126
127 print(K)
128 # Compute heat load from element 3
129 LQe3 = Se*Q*Dz/3*ones((3,)).reshape(3, 1)
130 for ro  in arange(len(zedof)):
131     if (zedof[ro] < N_f):
132         L[zedof[ro]] = L[zedof[ro]] + LQe3[ro]
133
134 # Compute loading from the prescribed temperatures
135 LT = -dot(K[0:N_f, N_f:N], T[N_f:N])
136 print(LT)
137
138 # Solution:
139 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], L[0:N_f] + LT[0:N_f])
140 print('Solution T=', T)
```

Listing 3.2. pnpConcreteColumnNZEBC.py

pnpLayeredWall1v3.py

```
1 # Finite Element Modeling with Abaqus and Python for Thermal and
2 # Stress Analysis
3 # (C)  2017-2020, Petr Krysl
4 """
5 Layered wall.  Temperature boundary conditions.
6 """
7
8 import math
9 from numpy import array as array
10 from numpy import zeros as zeros
11 from numpy import ones as ones
12 from numpy import arange as arange
13 from numpy import dot as dot
14 from numpy import linalg as linalg
15 from numpy import vstack  as vstack
16 import matplotlib.pyplot as plt
17
18 Dz = 1.0 # Thickness of the slice (it cancels out  in the end)
```

Python
- script

```python
19 kappaL = 0.05  # thermal conductivity, layer on the left
20 kappaR = 1.8  # thermal conductivity, layer on the right
21 # Coordinates of nodes
22 x = array([[0, 0], [0.07, 0], [0.3, 0], [0, 0.1], [0.07, 0.1], [0.3, 0.1]])
23 N = 6 # total number of nodes
24 N_f = 2 # total number of free degrees of freedom
25 # Mapping from nodes to  degrees of freedom
26 node2dof = array([3, 1, 4, 5, 2, 6])
27
28 # Connectivity of the left-region triangles (polyurethane)
29 connL = array([[4, 1, 5], [2, 5, 1]])
30 # Connectivity of the right-region triangles (concrete)
31 connR = array([[6, 5, 2], [2, 3, 6]])
32 pTe = -10  # Boundary conditions, exterior, interior, deg Celsius
33 pTi = +20
34 T = zeros(N).reshape(N, 1)
35 # Apply fixed temperatures at the left-hand side face
36 for index  in [1, 4]:
37     T[node2dof[index-1]-1] = pTe # left face, nodes 1 and 4
38
39 # Apply fixed temperatures at the right-hand side face
40 for index  in [3, 6]:
41     T[node2dof[index-1]-1] = pTi # right face, nodes 3 and 6
42
43 # gradients of the basis functions with respect to the param. coordinates
44 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
45
46 K = zeros((N_f, N)) # allocate the global conductivity matrix
47 LQ = zeros(N_f).reshape(N_f, 1)  # allocate the global heat loads vector
48
49 # Loop over the triangles in the mesh, left domain
50 kappa = kappaL
51 zconn = array(connL)-1
52 for j  in arange(zconn.shape[0]):
53     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
54     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
55     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
56     # Element degree-of-freedom array,  converted to zero base
57     zedof = array(node2dof[zconn[j, :]])-1
58     # Assemble elementwise conductivity matrix
59     for ro  in arange(len(zedof)):
60         for co  in arange(len(zedof)):
61             if (zedof[ro] < N_f):
62                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
63
64 # Loop over the triangles in the mesh, right domain
65 kappa = kappaR
66 zconn = array(connR)-1
67 for j  in arange(zconn.shape[0]):
68     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
69     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
70     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
71     # Element degree-of-freedom array,  converted to zero base
72     zedof = array(node2dof[zconn[j, :]])-1
73     # Assemble elementwise conductivity matrix
74     for ro  in arange(len(zedof)):
75         for co  in arange(len(zedof)):
76             if (zedof[ro] < N_f):
```

```
77                        K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
78
79 print(K)
80 # Compute loading from the prescribed temperatures
81 LT = -dot(K[0:N_f, N_f:N], T[N_f:N])
82 print(LT)
83 # Solve for the global temperatures at the free degrees of freedom
84 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], LQ + LT)
85 print('T=', T)
86
87 # Plotting: produce a contour plot of the temperature on the mesh
88 plt.figure()
89 plt.gca().set_aspect('equal')
90 # setup three 1-d arrays for the x-coord, the y-coord, and the z-coord
91 xs = x[:, 0].reshape(N,)# one value per node
92 ys = x[:, 1].reshape(N,)# one value per node
93 ix = node2dof[arange(N)]-1 # The number of the DOF for each node, zero-based
94 zs = T[ix].reshape(N,) # one value per node
95 triangles = vstack((connL-1, connR-1))# triangles are defined by conn arrays
96 plt.tricontourf(xs, ys, triangles, zs)
97 plt.colorbar()
98 plt.title('Contour plot of temperature')
99 plt.xlabel('x (m)')
100 plt.ylabel('y (m)')
101 plt.show()
```
Listing 3.3. pnpLayeredWall1v3.py

pnpLayeredWallq1.py

```
1 # Finite Element Modeling with Abaqus and Python for Thermal and
2 # Stress Analysis
3 # (C)  2017-2020, Petr Krysl
4 #
5 """
6  Layered wall.  Temperature boundary condition on the left, and heat flux
7  boundary condition on the right.
8  """
9
10 from numpy import array as array
11 from numpy import zeros as zeros
12 from numpy import arange as arange
13 from numpy import dot as dot
14 from numpy import linalg as linalg
15 from numpy import vstack  as vstack
16 from numpy import diff as diff
17 import matplotlib.pyplot as plt
18
19
20 Dz = 1.0 # Thickness of the slice (it cancels out  in the end)
21 kappaL = 0.05  # thermal conductivity, layer on the left
22 kappaR = 1.8  # thermal conductivity, layer on the right
23 # Coordinates of nodes
24 x = array([[0, 0], [0.07, 0], [0.3, 0], [0, 0.1], [0.07, 0.1], [0.3, 0.1]])
25 N = 6 # total number of nodes
26 N_f = 4 # total number of free degrees of freedom
27 # Mapping from nodes to  degrees of freedom
28 node2dof = array([5, 1, 3, 6, 2, 4])
```

Python
- script

```
29
30 # Connectivity of the left-region triangles (polyurethane)
31 connL = array([[4, 1, 5], [2, 5, 1]])
32 # Connectivity of the right-region triangles (concrete)
33 connR = array([[6, 5, 2], [2, 3, 6]])
34 # Connectivity  of the boundary L2 element  on the right
35 connRbdry = array([[3, 6]])
36 pTe = -10  # Boundary conditions, exterior, interior, deg Celsius
37 T = zeros(N).reshape(N, 1)
38 for index  in [1, 4]:
39     T[node2dof[index-1]-1] = pTe # left face with EBC, nodes 1 and 4
40 qnbarR = -30 # prescribed heat flux at the right boundary
41
42 # gradients of the basis functions with respect to the param. coordinates
43 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
44
45 K = zeros((N_f, N)) # allocate the global conductivity matrix
46 Lq = zeros(N_f).reshape(N_f, 1)  # allocate the global heat loads vector
47
48 # Loop over the triangles in the mesh, left domain
49 kappa = kappaL
50 zconn = array(connL)-1
51 for j  in arange(zconn.shape[0]):
52     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
53     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
54     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
55     # Element degree-of-freedom array,  converted to zero base
56     zedof = array(node2dof[zconn[j, :]])-1
57     # Assemble elementwise conductivity matrix
58     for ro  in arange(len(zedof)):
59         for co  in arange(len(zedof)):
60             if (zedof[ro] < N_f):
61                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
62
63 # Loop over the triangles in the mesh, right domain
64 kappa = kappaR
65 zconn = array(connR)-1
66 for j  in arange(zconn.shape[0]):
67     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
68     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
69     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
70     # Element degree-of-freedom array,  converted to zero base
71     zedof = array(node2dof[zconn[j, :]])-1
72     # Assemble elementwise conductivity matrix
73     for ro  in arange(len(zedof)):
74         for co  in arange(len(zedof)):
75             if (zedof[ro] < N_f):
76                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
77
78 # Boundary element heat flux term
79 qnbar = qnbarR
80 zconn = connRbdry - 1
81 for j  in arange(zconn.shape[0]):
82     he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
83     # Element degree-of-freedom array,  converted to zero base
84     zedof = array(node2dof[zconn[j, :]])-1
85     Leq = -qnbar*he*Dz/2*array([[1], [1]])
86     # Assemble elementwise heat load vector
```

```
87      for ro  in arange(len(zedof)):
88          if (zedof[ro] < N_f):
89              Lq[zedof[ro]] = Lq[zedof[ro]] + Leq[ro]
90
91 print(K)
92 # Compute loading from the prescribed temperatures
93 LT = -dot(K[0:N_f, N_f:N], T[N_f:N])
94 print(LT)
95 print(Lq + LT)
96 # Solve for the global temperatures at the free degrees of freedom
97 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], Lq + LT)
98 print('T=', T)
99
100 # Plotting
101 plt.figure()
102 plt.gca().set_aspect('equal')
103 # setup three 1-d arrays for the x-coord, the y-coord, and the z-coordinate
104 xs = x[:, 0].reshape(N,)# one value per node
105 ys = x[:, 1].reshape(N,)# one value per node
106 ix = node2dof[arange(N)]-1
107 zs = (T[ix]).reshape(N,)# one value per node
108 triangles = vstack((connL-1, connR-1))# triangles are defined by the conn arrays
109 plt.tricontourf(xs, ys, triangles, zs)
110 plt.colorbar()
111 plt.title('Contour plot of temperature')
112 plt.xlabel('x (m)')
113 plt.ylabel('y (m)')
114 plt.show()
```

Listing 3.4. pnpLayeredWallq1.py

pnpLayeredWallqboth.py

Python
- script

```
1 # Finite Element Modeling with Abaqus and Python for Thermal and
2 # Stress Analysis
3 # (C)  2017-2020, Petr Krysl
4 #
5 """
6  Layered wall.  Temperature boundary condition on the left, and heat flux
7  boundary condition on the right.
8  """
9
10 from numpy import array as array
11 from numpy import zeros as zeros
12 from numpy import arange as arange
13 from numpy import dot as dot
14 from numpy import linalg as linalg
15 from numpy import vstack  as vstack
16 from numpy import diff as diff
17 import matplotlib.pyplot as plt
18
19 Dz = 1.0 # Thickness of the slice (it cancels out  in the end)
20 kappaL = 0.05  # thermal conductivity, layer on the left
21 kappaR = 1.8  # thermal conductivity, layer on the right
22 # Coordinates of nodes
23 x = array([[0, 0], [0.07, 0], [0.3, 0], [0, 0.1], [0.07, 0.1], [0.3, 0.1]])
24 N = 6 # total number of nodes
25 N_f = 6 # total number of free degrees of freedom
```

```python
26 # Mapping from nodes to  degrees of freedom
27 node2dof=array([6, 1, 3, 5, 2, 4])
28 T = zeros(N).reshape(N, 1) # Vector of temperatures at all degrees of freedom
29
30 # Connectivity of the left-region triangles (polyurethane)
31 connL = array([[4, 1, 5], [2, 5, 1]])
32 # Connectivity of the right-region triangles (concrete)
33 connR = array([[6, 5, 2], [2, 3, 6]])
34 # Connectivity  of the boundary L2 element  on the left
35 connLbdry = array([[4, 1]])
36 qnbarL = +30
37 # Connectivity  of the boundary L2 element  on the right
38 connRbdry = array([[3, 6]])
39 qnbarR = -30
40
41 # gradients of the basis functions with respect to the param. coordinates
42 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
43
44 K = zeros((N_f, N)) # allocate the global conductivity matrix
45 Lq = zeros(N_f).reshape(N_f, 1)  # allocate the global heat loads vector
46
47 # Loop over the triangles in the mesh, left domain
48 kappa = kappaL
49 zconn = array(connL)-1
50 for j  in arange(zconn.shape[0]):
51     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
52     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
53     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
54     # Element degree-of-freedom array,  converted to zero base
55     zedof = array(node2dof[zconn[j, :]])-1
56     # Assemble elementwise conductivity matrix
57     for ro  in arange(len(zedof)):
58         for co  in arange(len(zedof)):
59             if (zedof[ro] < N_f):
60                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
61
62 # Loop over the triangles in the mesh, right domain
63 kappa = kappaR
64 zconn = array(connR)-1
65 for j  in arange(zconn.shape[0]):
66     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
67     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
68     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
69     # Element degree-of-freedom array,  converted to zero base
70     zedof = array(node2dof[zconn[j, :]])-1
71     # Assemble elementwise conductivity matrix
72     for ro  in arange(len(zedof)):
73         for co  in arange(len(zedof)):
74             if (zedof[ro] < N_f):
75                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
76
77 # Boundary element heat flux term, left-hand side
78 qnbar = qnbarL
79 zconn = array(connLbdry)-1
80 for j  in arange(zconn.shape[0]):
81     he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
82     # Element degree-of-freedom array,  converted to zero base
83     zedof = array(node2dof[zconn[j, :]])-1
```

```python
84      Leq = -qnbar*he*Dz/2*array([[1], [1]])
85      # Assemble elementwise heat load vector
86      for ro  in arange(len(zedof)):
87          if (zedof[ro] < N_f):
88              Lq[zedof[ro]] = Lq[zedof[ro]] + Leq[ro]
89
90  # Boundary element heat flux term, right-hand side
91  qnbar=qnbarR
92  zconn=array(connRbdry)-1
93  for j  in arange(zconn.shape[0]):
94      he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
95      # Element degree-of-freedom array,  converted to zero base
96      zedof = array(node2dof[zconn[j, :]])-1
97      Leq = -qnbar*he*Dz/2*array([[1], [1]])
98      # Assemble elementwise heat load vector
99      for ro  in arange(len(zedof)):
100         if (zedof[ro] < N_f):
101             Lq[zedof[ro]] = Lq[zedof[ro]] + Leq[ro]
102
103 print(K)
104 print(Lq)
105 # Solve for the global temperatures at the free degrees of freedom
106 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], Lq)
107 print('T=', T)
108
109 # Plotting
110 plt.figure()
111 plt.gca().set_aspect('equal')
112 # setup three 1-d arrays for the x-coord, the y-coord, and the z-coord
113 xs = x[:, 0].reshape(N,)# one value per node
114 ys = x[:, 1].reshape(N,)# one value per node
115 ix = node2dof[arange(N)]-1
116 zs = (T[ix]).reshape(N,)# one value per node
117 triangles = vstack((connL-1, connR-1))# triangles are defined by the conn arrays
118 plt.tricontourf(xs, ys, triangles, zs)
119 plt.colorbar()
120 plt.title('Contour plot of temperature')
121 plt.xlabel('x (m)')
122 plt.ylabel('y (m)')
123 plt.show()
```

Listing 3.5. pnpLayeredWallqboth.py

pnpConcreteColumnConv.py

```python
1  # Finite Element Modeling with Abaqus and Python for Thermal and
2  # Stress Analysis
3  # (C)  2017-2020, Petr Krysl
4  #
5  """
6  Concrete column with hydration heat and convection (film) condition
7  on the boundary. Numerical solution.
8  """
9  import math
10 from numpy import array as array
11 from numpy import zeros as zeros
12 from numpy import ones as ones
13 from numpy import arange as arange
```

Python
- script

```python
14 from numpy import dot as dot
15 from numpy import linalg as linalg
16 from numpy import vstack  as vstack
17 from numpy import diff as diff
18 import matplotlib.pyplot as plt
19
20 # These are the input constants in the problem:
21 a = 2.5  # radius of the column cross-section
22 dy = a / 2 * math.sin(15. / 180 * math.pi)
23 dx = a / 2 * math.cos(15. / 180 * math.pi)
24 Q = 4.5  # internal heat generation rate
25 k = 1.8  # thermal conductivity
26 Dz = 1.0  # thickness of the slice
27 h = 5.0  # surface heat transfer coefficient
28 Ta = 0.0  # ambient temperature (freezing water)
29
30 # Coordinates of nodes
31 x = array([[0, 0], [dx, -dy], [dx, dy], [2 * dx, -2 * dy], [2 * dx, 2 * dy]])
32 N = 5  # total number of nodes
33 N_f = 5  # total number of free degrees of freedom
34 # Mapping from nodes to  degrees of freedom
35 node2dof = array([3, 1, 2, 5, 4])
36 T = zeros(N).reshape(N, 1) # Vector of temperatures at all degrees of freedom
37
38 # gradients of the basis functions with respect to the param. coordinates
39 gradNpar = array([[-1, -1], [1, 0], [0, 1]])
40
41 # Connectivity of the mesh: interior mesh
42 conn = array([[1, 2, 3], [2, 4, 5], [2, 5, 3]])
43 # boundary mesh
44 connbdry = array([[4, 5]])
45
46 K = zeros((N_f, N))  # allocate the global conductivity matrix
47 L = zeros((N_f, 1))  # allocate the global heat loads vector
48
49 # Loop over the triangles in the mesh
50 kappa = k
51 zconn = conn - 1
52 for j  in arange(zconn.shape[0]):
53     J = dot(x[zconn[j, :], :].T, gradNpar) # compute the Jacobian matrix
54     gradN = dot(gradNpar, linalg.inv(J)) # compute the x,y grads of the b. funcs
55     Ke = (kappa*Dz*linalg.det(J)/2)*dot(gradN, gradN.T) # elementwise matrix
56     # Element degree-of-freedom array,  converted to zero base
57     zedof = array(node2dof[zconn[j, :]])-1
58     # Assemble elementwise conductivity matrix
59     for ro  in arange(len(zedof)):
60         for co  in arange(len(zedof)):
61             if (zedof[ro] < N_f):
62                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
63     LQe = linalg.det(J)/2*Q*Dz/3*ones((3, 1))
64     for ro  in arange(len(zedof)):
65         if (zedof[ro] < N_f):
66             L[zedof[ro]] = L[zedof[ro]] + LQe[ro]
67
68 # Calculations for the  convection boundary condition
69 H = zeros((N_f, N))  # allocate the global film-condition matrix
70 zconn = connbdry - 1
71 for j in arange(zconn.shape[0]):
```

```
72     he = linalg.norm(diff(x[zconn[j, :], :], axis=0))
73     # Element degree-of-freedom array,  converted to zero base
74     zedof = array(node2dof[zconn[j, :]]) - 1
75     He = h * Dz * he / 6 * array([[2, 1], [1, 2]])
76     # Assemble elementwise surface heat transfer matrix
77     for ro in arange(len(zedof)):
78         for co in arange(len(zedof)):
79             if (zedof[ro] < N_f):
80                 H[zedof[ro], zedof[co]] = H[zedof[ro], zedof[co]] + He[ro, co]
81     # Assemble elementwise heat load vector for surface heat transfer
82     Lea = Ta * h * he * Dz / 2 * array([[1], [1]])
83     for ro in arange(len(zedof)):
84         if (zedof[ro] < N_f):
85             L[zedof[ro]] = L[zedof[ro]] + Lea[ro]
86
87 # Solve for the global temperatures at the free degrees of freedom
88 print(K)
89 print(H)
90 print(L)
91 # Solve for the global temperatures at the free degrees of freedom
92 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f] + H[0:N_f, 0:N_f], L)
93 print('T=', T)
94
95 print('Temperatures at the nodes')
96 for index in arange(node2dof.shape[0]):
97     print('Node', index + 1, 'T=', T[node2dof[index] - 1])
98
99 # Plot filled contours
100 plt.figure()
101 plt.gca().set_aspect('equal')
102 # setup three 1-d arrays for the x-coordinate, the y-coordinate, and the
103 # z-coordinate
104 xs = x[:, 0].reshape(N,)  # one value per node
105 ys = x[:, 1].reshape(N,)  # one value per node
106 ix = node2dof[arange(N)] - 1
107 zs = (T[ix]).reshape(N,)  # one value per node
108 triangles = conn - 1  # the triangles are defined by the connectivity arrays
109 plt.tricontourf(xs, ys, triangles, zs)
110 plt.colorbar()
111 plt.title('Contour plot of temperature')
112 plt.xlabel('x (m)')
113 plt.ylabel('y (m)')
114 plt.show()
```

Listing 3.6. pnpConcreteColumnConv.py

4

FEA for 2-D Heat conduction with Quadrilaterals

Quadrilateral elements are an alternative to the triangles discussed in Chapter 3. The triangular elements are in a way too simple: the variation of temperature they can represent is not sufficiently rich. We say that the triangles are "stiff", which makes more sense when they are applied to stress analysis, but the intuitive insight this word attempts to generate is hopefully clear.

Quadrilateral elements address the excessive stiffness of the triangular elements by coupling together a larger number of nodes (four instead of three), which in the end leads to basis functions which are more than just linear polynomials. This endowes the finite element with that little something that makes it more accurate.

4.1 Quadrilateral element

The element Q4 has four nodes, and its standard shape is a bi-unit square. This square is to be understood as the Cartesian product of two standard intervals (refer to Section 3.14.1). Therefore, the basis functions of Q4 may also be formed as products of the basis functions on the standard interval L2 (3.106). Assuming the numbering of the nodes as shown in Figure 4.1, the basis function $N_{\underline{1}}$ may be written as the product of the basis function on the interval $-1 \leq \xi \leq +1$ and the basis function on the interval $-1 \leq \eta \leq +1$ where both functions correspond to the left-hand side endpoint ($\xi = -1$, $\eta = -1$)

$$N_{\underline{1}}(\xi, \eta) = \frac{\xi - 1}{-1 - 1} \times \frac{\eta - 1}{-1 - 1} = \frac{(\xi - 1)(\eta - 1)}{4} \ . \tag{4.1}$$

Similarly, for the remaining three functions we have

$$N_{\underline{2}}(\xi, \eta) = \frac{(\xi + 1)(\eta - 1)}{-4} \ , \ N_{\underline{3}}(\xi, \eta) = \frac{(\xi + 1)(\eta + 1)}{4} \ , \tag{4.2}$$

and

$$N_{\underline{4}}(\xi, \eta) = \frac{(\xi - 1)(\eta + 1)}{-4} \ . \tag{4.3}$$

As all basis functions are linear in ξ and η, the shape that they represent when raised as a surface above the standard square is a hyperbolic paraboloid.

The derivatives of the basis functions (i.e. the gradients of the basis functions) are computed in the standard way using equations (3.62). All we need is the gradient of the basis functions with respect to the parametric coordinates
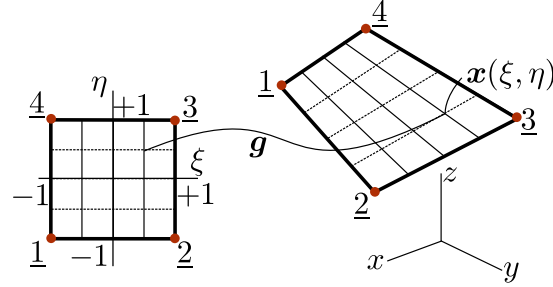
Fig. 4.1. Mapping the standard square to a general quadrilateral

$$
\left[\operatorname{grad}_{(\xi,\eta)} N \right] =
\begin{bmatrix}
\operatorname{grad}_{(\xi,\eta)} N_{\underline{1}} \\
\operatorname{grad}_{(\xi,\eta)} N_{\underline{2}} \\
\operatorname{grad}_{(\xi,\eta)} N_{\underline{3}} \\
\operatorname{grad}_{(\xi,\eta)} N_{\underline{4}}
\end{bmatrix}
=
\begin{bmatrix}
\dfrac{\partial N_{\underline{1}}}{\partial \xi} , & \dfrac{\partial N_{\underline{1}}}{\partial \eta} \\[2mm]
\dfrac{\partial N_{\underline{2}}}{\partial \xi} , & \dfrac{\partial N_{\underline{2}}}{\partial \eta} \\[2mm]
\dfrac{\partial N_{\underline{3}}}{\partial \xi} , & \dfrac{\partial N_{\underline{4}}}{\partial \eta} \\[2mm]
\dfrac{\partial N_{\underline{3}}}{\partial \xi} , & \dfrac{\partial N_{\underline{4}}}{\partial \eta}
\end{bmatrix}
= \frac{1}{4}
\begin{bmatrix}
\eta - 1 , & \xi - 1 \\
-\eta + 1 , & -\xi - 1 \\
\eta + 1 , & \xi + 1 \\
-\eta - 1 , & -\xi + 1
\end{bmatrix} ,
\tag{4.4}
$$

The new aspect compared to the three-node triangle is that the gradients of the basis functions with respect to the parametric coordinates now vary across the element, the gradients are not constant within the element anymore. Of course, that makes perfect sense, the slope of the surface of the "tent" function over the standard square varies from point to point. It does make the calculation of the elementwise quantities more difficult, which is where numerical integration comes in.

The integration of the elementwise expressions needs a more sophisticated approach than for the triangle: elementary, analytical, integrations are no longer possible for general-shape quadrilaterals. One must employ numerical quadrature (integration). The numerical approximation of the integral of the function f over the standard square is the sum of the function values

$$
\int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi,\eta)\, \mathrm{d}\xi \right) \mathrm{d}\eta \approx \sum_{q=1}^{M} f(\xi_q, \eta_q) W_q ,
\tag{4.5}
$$

at the $q = 1, \ldots, M$ quadrature points ξ_q, η_q, weighted with multipliers W_q. (Gaussian quadrature in one dimension: $\boxed{\text{See Box 14}}$. Gauss quadrature in two dimensions: $\boxed{\text{See Box 15}}$.) The Gauss quadrature is particularly suited to the task (small number of function evaluations, high accuracy). The most applicable rule to the four-node quadrilateral is the four point quadrature rule given in Table 4.1.

Four-point two-dimensional rule		
j	Coordinates ξ_j, η_j	Weights W_j
1	$-\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
2	$-\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$
3	$+\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
4	$+\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$

Table 4.1. Four-point Gauss rule on the standard square

The goal is to integrate some expression over the area of a general quadrilateral. Similarly to the triangle (both elements are isoparametric!), the map (3.34) will produce an image of the standard square in the Cartesian coordinates x, y – refer to Figure 4.2. Point (ξ, η) is mapped into
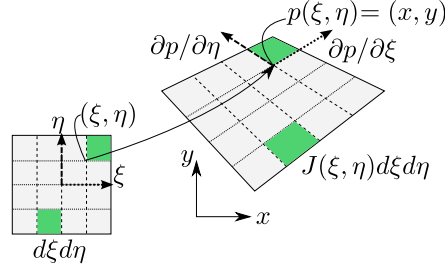
Fig. 4.2. Mapping of the standard square into a general Q4 quadrilateral. Point (ξ, η) is mapped into $p(\xi, \eta) = (x, y)$, and the area $\mathrm{d}\xi, \mathrm{d}\eta$ in the standard square is mapped to an area $J(\xi, \eta)\mathrm{d}\xi\mathrm{d}\eta$ by the Jacobian of the map

$$p(\xi, \eta) = (x, y) \ , \tag{4.6}$$

where

$$x = \sum_{\underline{k}=1}^{4} N_{\underline{k}}(\xi, \eta) x_{\underline{k}} \ , \quad y = \sum_{\underline{k}=1}^{4} N_{\underline{k}}(\xi, \eta) y_{\underline{k}} \ , \tag{4.7}$$

and the map (4.6) maps areas as

$$\mathrm{d}\xi\mathrm{d}\eta \longrightarrow \mathrm{d}\xi\mathrm{d}\eta \det\left[J\right] \ . \tag{4.8}$$

The differential area on the left-hand side is in the standard square, the differential area on the right is in the general quadrilateral in x, y, and $\det\left[J\right] = J$ is the Jacobian of the map (4.6) (which is of the same sort as (3.34) for the triangle).

As a consequence, we have the following change of coordinates in integrals:

$$\int_{S_{[x,y]}} f(x, y)\mathrm{d}x\mathrm{d}y = \int_{S_{[\xi,\eta]}} f(\xi, \eta) \det\left[J(\xi, \eta)\right] \mathrm{d}\xi\mathrm{d}\eta \ . \tag{4.9}$$

The numerical approximation for the quadrilateral is therefore

$$\int_{S_{[x,y]}} f(x, y)\mathrm{d}x\mathrm{d}y \approx \sum_{q=1}^{M} f(\xi_q, \eta_q) \det[J(\xi_q, \eta_q)] \, W_q \ , \tag{4.10}$$

where $\det[J(\xi_q, \eta_q)]$ is the value of the Jacobian at the quadrature point.

4.2 Example: using Gaussian quadrature for a rectangle

In this section we will investigate an illustrative example in detail. The goal is to compute the elementwise conductivity matrix of a rectangular Q4 finite element using a 2×2 point Gauss quadrature rule. We shall assume homogeneous isotropic thermal conductivity. We are interested in the rank of the resulting conductivity matrix: recall that for the three-node triangle, the elementwise conductivity matrix was singular.

We will consider the nodes located at

$$[p_1] = \begin{bmatrix} A, \, B \end{bmatrix} \ , \quad [p_2] = \begin{bmatrix} 0, \, B \end{bmatrix} \ , \quad [p_3] = \begin{bmatrix} 0, \, 0 \end{bmatrix} \ , \quad [p_4] = \begin{bmatrix} A, \, 0 \end{bmatrix} \ ,$$

where A, B will be taken as symbolic variables.
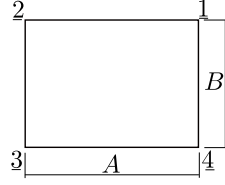
The weighted residual statement is still the same: refer to equation (3.45). When computing the elementwise conductivity matrix, the transition from the three-node triangle to the four-node quadrilateral is not very complicated. Expression (3.66) now needs to be evaluated for the domain of integration being a general quadrilateral

$$\int_{e=1} \text{grad} N_{\underline{k}} \, \kappa \text{grad} N_{\underline{m}}^T \, \Delta z \, \text{d}S \quad \underline{k} = \underline{1,2,3,4}, \; \underline{m} = \underline{1,2,3,4} \tag{4.11}$$

and there are four nodes (and four degrees of freedom) instead of three.

With the four-point Gauss rule (Table 4.1) this is approximated as

$$K_{\underline{km}} = \sum_{q=1}^{4} \text{grad} N_{\underline{k}}(\xi_q, \eta_q) \, \kappa \, \text{grad} N_{\underline{m}}^T(\xi_q, \eta_q) \, \Delta z \, \det[J(\xi_q, \eta_q)] W_q \;, \qquad \underline{k}, \underline{m} = \underline{1,2,3,4} \;. \tag{4.12}$$

where $\xi_q = \pm\sqrt{3}/3, \eta_q = \pm\sqrt{3}/3, W_q = 1$. The entire elementwise matrix can be in fact computed as

$$[K^{(e)}] = \sum_{q=1}^{4} [\text{grad} N(\xi_q, \eta_q)] \, \kappa \, [\text{grad} N(\xi_q, \eta_q)]^T \, \Delta z \, \det[J(\xi_q, \eta_q)] W_q \;, \tag{4.13}$$

which is the formula we will use.

The Python code to evaluate the matrix follows. Note that this script relies on the symbolic capabilities provided by the sympy module.

The basis functions (4.1)–(4.3) are the entries of this symbolic matrix

```
13 xi, eta = symbols('xi eta')
14 N = Matrix([(xi-1)*(eta-1)/4,
15              (xi+1)*(eta-1)/-4,
16              (xi+1)*(eta+1)/4,
17              (xi-1)*(eta+1)/-4]).reshape(4,1)
```

which yields the matrix of gradients of the basis functions with respect to the parametric coordinates by symbolic differentiation

```
25 gradNpar = diff(N, 'xi').row_join(diff(N, 'eta'))
26 print(gradNpar)
```

The row_join joins the derivative with respect to η as a second column to the first column which is the derivative with respect to ξ, producing as output this matrix of four rows and two columns

```
Matrix([[eta/4 - 1/4, xi/4 - 1/4], [-eta/4 + 1/4, -xi/4 - 1/4], ...
                [eta/4 + 1/4, xi/4 + 1/4], [-eta/4 - 1/4, -xi/4 + 1/4]])
```

With these definitions

```
21 A, B = symbols('A, B')
22 x = Matrix([[A, B], [0, B], [0, 0], [A, 0]])
```

we compute the Jacobian matrix as J=x.T*gradNpar which yields

```
Matrix([
[-A/2,    0],
[   0, -B/2]])
```

and the Jacobian `Jac=det(J)` as `A*B/4`. We may note that the Jacobian is constant, which is what we would expect since the standard square is mapped into a rectangle: All right angles stay right angles, and horizontal lines stretch by the same amount, and vertical lines also stretch by the same amount everywhere.

The basis function gradients with respect to x, y may be evaluated from the general formula (3.62)

```
38 gradN = simplify(gradNpar*(J**-1))
39 print(gradN)
```

resulting in

```
Matrix([
[-(eta - 1)/(2*A),  (-xi + 1)/(2*B)],
[ (eta - 1)/(2*A),   (xi + 1)/(2*B)],
[-(eta + 1)/(2*A),  -(xi + 1)/(2*B)],
[ (eta + 1)/(2*A),   (xi - 1)/(2*B)]])
```

We glean from the above that the x, y basis function gradients *vary from point to point* within the element.

The conductivity matrix can be now evaluated. First we simplify by noting that $\kappa, \Delta z, W_k = 1$, and $\det[J(\xi_k, \eta_k)] = (AB)/4$ are all constants and therefore may be taken out of the sum of the numerical quadrature . The four-point integration needs to be carried out only on the product of the gradients, which change from quadrature point to quadrature point, and yields the intermediate result (note the use of `subs()` to evaluate the basis function gradient expressions for particular values of ξ, η)

```
50 gN = gradN.subs(xi, -0.577350269189626).subs(eta, -0.577350269189626)
51 K1 = kap*Dz*Jac*gN*gN.T # term 1 of the sum
52 gN = gradN.subs(xi, -0.577350269189626).subs(eta, +0.577350269189626)
53 K2 = kap*Dz*Jac*gN*gN.T # term 2 of the sum
54 gN = gradN.subs(xi, +0.577350269189626).subs(eta, -0.577350269189626)
55 K3 = kap*Dz*Jac*gN*gN.T # term 3 of the sum
56 gN = gradN.subs(xi, +0.577350269189626).subs(eta, +0.577350269189626)
57 K4 = kap*Dz*Jac*gN*gN.T # term 4 of the sum
```

and further adding together the intermediate results from the for quadrature points

```
59 K = simplify(K1+K2+K3+K4)
60 print(K)
```

(note the use of `simplify()` to produce an uncluttered expression) gives, with a little bit of manual editing,

```
kap*Dz/(A*B)/6*Matrix([
[ 2*(A**2 + B**2), (*A**2 - 2*B**2),   -(A**2 + B**2), (-2*A**2 + B**2)],
[ (A**2 - 2*B**2),  2*(A**2 + B**2), (-2*A**2 + B**2),   -(A**2 + B**2)],
[  -(A**2 + B**2), (-2*A**2 + B**2),  2*(A**2 + B**2),  (A**2 - 2*B**2)],
[(-2*A**2 + B**2),   -(A**2 + B**2),  (A**2 - 2*B**2),  2*(A**2 + B**2)]])
```

where we have shortened the output by hand by replacing 0.333333333333333 with 1/3, and 0.166666666666667 with 1/6. This can be cleaned up as

$$\left[K^{(e)}\right] = \frac{\kappa\,\Delta z}{6AB} \begin{bmatrix} 2A^2+2B^2, & A^2-2B^2, & -A^2-B^2, & B^2-2A^2 \\ A^2-2B^2, & 2A^2+2B^2, & B^2-2A^2, & -A^2-B^2 \\ -A^2-B^2, & B^2-2A^2, & 2A^2+2B^2, & A^2-2B^2 \\ B^2-2A^2, & -A^2-B^2, & A^2-2B^2, & 2A^2+2B^2 \end{bmatrix} \tag{4.14}$$

The elementwise conductivity matrix $\left[K^{(e)}\right]$ has a rank equal to the number of nonzero eigenvalues. With Python we find the eigenvalues of this matrix as `K.eigenvals()` which yields

```
{B*Dz*kap/A: 1, A*Dz*kap/B: 1, (A**2*Dz*kap + B**2*Dz*kap)/(3*A*B): 1, 0: 1}
```

in the format "eigenvalue: multiplicity" . Since there is one zero eigenvalue, the rank of $\left[K^{(e)}\right]$ is equal to 3. The rank of the conductivity matrix was discussed for the T3 (triangle) element in Section 3.14.4. It was discussed there that the expected rank would be equal to the number of nodes minus one, which for the triangle gives $3 - 2 = 1$. The present result is consistent in that the defect (1) is the same as for the triangle: $4 - 3 = 1$. We can see that the elementwise matrix of the quadrilateral has the same property as the elementwise matrix for the triangle: in all the rows the entries add up to zero. The physical meaning is the same as before: the elementwise conductivity matrix produces zero output for the input of the temperatures being the same at all nodes.

4.3 Concrete column example again

We will consider again the example of the concrete column with zero temperature on the boundary and hydration heat load from Section 3.7. Figure 4.3 shows this time the mesh of a pie-shaped piece of the cross-section of the concrete column is a single quadrilateral, with connectivity [1,2,3,4]. The temperatures along the boundary with water, i.e. along the edge [2,3] and [3,4] are fixed degrees of freedom $T_2 = T_3 = T_4 = 0$, and the only unknown is T_1. The total number of degrees of freedom is $N = 4$, and the number of free (unknown) degrees of freedom is $N_f = 1$.
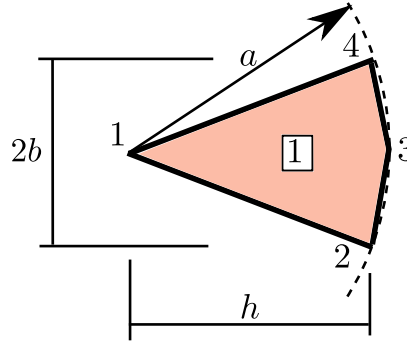
Fig. 4.3. Concrete column with prescribed zero temperature on the boundary and hydration heat source. Mesh with a single Q4 element.

The solution with a brief Python script is described below. The data of the problem is defined as (page 123)

```
19 a = 2.5  # radius of the column
20 b = a * math.sin(15 / 180 * math.pi)  # dimension
21 h = a * math.cos(15 / 180 * math.pi)  # dimension
22 Q = 4.5  # internal heat generation rate
23 k = 1.8  # thermal conductivity
24 Dz = 1.0  # thickness of the slice
25 x = array([[0, 0], [h, -b], [a, 0], [h, b]])  # Coordinates of nodes
```

4.3.1 Conductivity matrix

It will be convenient to define a function to evaluate the matrix of the basis function gradients with respect to the parametric coordinates at any given point ξ, η:

```
32 def gradNpar(xi, eta):
33     """
34     A one-liner to calculate the matrix of the basis function gradients
35     in the parametric coordinates.
```

```
36      """
37      return array([
38          [eta / 4 - 1. / 4, xi / 4 - 1. / 4],
39          [1. / 4 - eta / 4, - xi / 4 - 1. / 4],
40          [eta / 4 + 1. / 4, xi / 4 + 1. / 4],
41          [- eta / 4 - 1. / 4, 1. / 4 - xi / 4]])
```

The 2×2 Gauss rule (Table 4.1) will be used for the elementwise quantities. It is the default for this type of element in Abaqus. The locations and the weights are defined as

```
52 xe = array([[-0.577350269189626, -0.577350269189626],
53             [-0.577350269189626, +0.577350269189626],
54             [+0.577350269189626, -0.577350269189626],
55             [+0.577350269189626, +0.577350269189626]])
56 W = array([1, 1, 1, 1])
```

The task is to compute the elementwise conductivity matrix

$$[K^{(e)}] = \int_{e=1} [\mathrm{grad}N]\, \kappa [\mathrm{grad}N]^T\, \Delta z\, \mathrm{d}S \tag{4.15}$$

which is an expression analogous to (3.71) but of dimension 4×4, using the numerical approximation (4.13). This will be accomplished in the integration loop over the four quadrature points of the Gauss rule. Inside the loop we will be building up the elementwise conductivity matrix one quadrature point at a time, and therefore we need a preallocated array to which the intermediate results will be added (page 123).

```
60 Ke = zeros((4, 4))
```

In the loop we start by setting the parametric coordinates of the quadrature point, and calculating the Jacobian matrix at that point from (3.60).

```
63 for qp in arange(xe.shape[0]):
64     xi = xe[qp, 0]
65     eta = xe[qp, 1]
66     # Compute the Jacobian matrix
67     J = dot(x.T, gradNpar(xi, eta))
```

From the Jacobian matrix, we calculate the Jacobian. It varies from quadrature point to quadrature point, with the minimum value of 0.1868 (at the fourth quadrature point) and the maximum value of 0.6220 (at the first quadrature point).

```
69     detJ = linalg.det(J)
```

The gradient of the basis functions is calculated using the standard expression (3.62)

```
71     gradN = dot(gradNpar(xi, eta), linalg.inv(J))
```

Now we can add the contribution from the quadrature point to the preallocated elementwise matrix Ke.

```
73     Ke = Ke + k * Dz * dot(gradN, gradN.T) * detJ * W[qp]
```

The interior of the loop is executed four times, once for each quadrature point. The finished elementwise conductivity matrix is

```
array([[ 0.46772756, -0.02719481, -0.41333794, -0.02719481],
       [-0.02719481,  2.13771962, -0.77091145, -1.33961336],
       [-0.41333794, -0.77091145,  1.95516084, -0.77091145],
       [-0.02719481, -1.33961336, -0.77091145,  2.13771962]])
```

4.3.2 Heat load vector and solution

Now analogously to the heat load vector calculation in Section 3.11 we need to evaluate (3.74), which can be written as a matrix expression

$$\int_{e=1} [N]Q \, \Delta z \, \mathrm{d}S \tag{4.16}$$

where $[N]$ is the matrix that consists of the four basis functions on the quadrilateral in the rows. Of course, the integral will again be approximated with numerical quadrature

$$\int_{e=1} [N]Q \, \Delta z \, \mathrm{d}S \approx \sum_{q=1}^{4} [N(\xi_q, \eta_q)] \, Q \, \Delta z \, \det[J(\xi_q, \eta_q)] W_q \ , \tag{4.17}$$

This little helper function will evaluate the column array of the basis function values at any quadrature point:

```
82  def N(xi, eta):
83      """
84      A one-liner to calculate the matrix of the basis function values.
85      """
86      return array([(xi - 1) * (eta - 1) / 4,
87                    (xi + 1) * (eta - 1) / -4,
88                    (xi + 1) * (eta + 1) / 4,
89                    (xi - 1) * (eta + 1) / -4]).reshape(4, 1)
```

Now we are ready to start the quadrature point loop. Again, the load vector is preallocated in the correct size, `Le=zeros(4).reshape(4,1)`. We will need the Jacobian (the first few lines of this loop look precisely like the ones for the conductivity matrix above), and then according to the formula we need to add the contribution from (4.17)

```
104     Le = Le + Q * Dz * N(xi, eta) * detJ * W[qp]
```

for each quadrature point `qp`. The result of the integration is the elementwise load vector

```
array([[ 2.38508927],
       [ 1.81982141],
       [ 1.25455355],
       [ 1.81982141]])
```

Notice that the largest contribution of the heat load density Q is to the degree of freedom at the first node, and the smallest is to the degree of freedom at the third node. This corresponds to the magnitudes of the Jacobians we inspected above: the quadrature point closest to node 1 has the largest value of the Jacobian, and the quadrature point closest to node 3 has the smallest value of the Jacobian.

Because of the simplicity of the present FE mesh, the entry `K[0,0]` of the elementwise conductivity matrix is the entire free-free global conductivity matrix, and the entry `L[0]` of the elementwise load vector is the entire global load vector. The solution for the only degree of freedom then follows as

```
114 T = zeros((4, 1))
115 # Solve the global equations
116 T[0] = (1 / K[0, 0]) * L[0]
```

which results in the temperature 5.0993 for the only free degree of freedom. Clearly, compared to the exact value, the solution is not very good. But the *mesh* is not very good, so we cannot ask too much of it.

4.3.3 Postprocessing

In this section we will evaluate the heat flux at the quadrature points. This is often called the postprocessing step: the solution has already been obtained, we will merely process it to extract some more information.

The gradient of the temperature at any quadrature point can be calculated using the definition of the interpolated temperature within the element:

$$T(\xi, \eta) = \sum_{i=1}^{N} N_i(\xi, \eta) T_i \tag{4.18}$$

as

$$\text{grad} T(\xi, \eta) = \text{grad}\left(\sum_{i=1}^{N} N_i(\xi, \eta) T_i\right) = \sum_{i=1}^{N} T_i \, \text{grad} N_i(\xi, \eta) = [T]^T \left[\text{grad} N(\xi, \eta)\right] \tag{4.19}$$

where the last expression is a matrix multiplication, of the matrix of the degrees of freedom values and the matrix of the basis function gradients. Note that the matrix of the basis function gradients depends on where in the element the gradients are evaluated. The postprocessing for the heat flux vectors is again performed in a loop (page 123)

```
124 for qp in arange(xe.shape[0]):
125     xi = xe[qp, 0]
126     eta = xe[qp, 1]
127     # Compute the location of the quadrature point
128     qploc = dot(N(xi, eta).T, x)
```

where the location of the quadrature point in the global coordinates `qploc` also calculated to allow us to plot the results.

We need the gradients of the basis functions:

```
130     J = dot(x.T, gradNpar(xi, eta))
131     # The  gradient  of the basis functions with respect to x, y
132     gradN = dot(gradNpar(xi, eta), linalg.inv(J))
```

It remains to calculate the gradient of the temperature and the heat flux, and to print out the results:

```
134     gradT = dot(T.T, gradN)
135     # Heat flux vector
136     q = -k * gradT.T
137     print('Heat flux at', qploc, '  = ', q)
```

For the four quadrature points we get the following output:

```
Heat flux at [[ 0.91658369  0.        ]]   = [[ 3.76543041]
 [-0.        ]]
Heat flux at [[ 2.02654304  0.37357311]]   = [[ 3.67150541]
 [ 0.27906968]]
Heat flux at [[ 2.02654304 -0.37357311]]   = [[ 3.67150541]
 [-0.27906968]]
Heat flux at [[ 2.35995936  0.        ]]   = [[ 3.35882447]
 [-0.        ]]
```

The calculated heat flux vectors are visualized in Figure 4.4. Notice that the results are relatively crude: we would expect the heat flux vectors to become shorter towards the center whereas the opposite is true for this very coarse mesh.
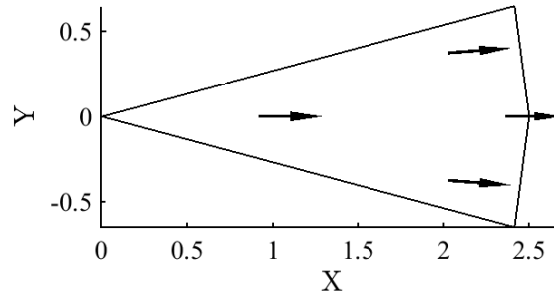
Fig. 4.4. Concrete column: single quadrilateral element mesh. Heat flux at quadrature points represented with arrows. The length of the arrow was scaled with 1/10.

4.4 Effects of element distortion

The accuracy of the quadrilateral element with nodes in arbitrary locations can be degraded by the distortion of the element. The Jacobian that maps the areas between the parametric bi-unit square and the physical space can cause some difficulties. We will illustrate here the possibility of finding a negative Jacobian in severely distorted quadrilaterals.

The Jacobian is the determinant of the Jacobian matrix (3.57). As discussed below equation (3.63), the Jacobian may be computed as the cross product of the two vectors

$$\det[J(\xi, \eta)] = \frac{\partial \boldsymbol{p}(\xi, \eta)}{\partial \xi} \times \frac{\partial \boldsymbol{p}(\xi, \eta)}{\partial \eta}$$

Note that $\partial \boldsymbol{p}(\xi, \eta)/\partial \xi$ is shown as dotted vector, and $\partial \boldsymbol{p}(\xi, \eta)/\partial \eta$ is displayed with a dashed vector. In the general quadrilateral these vectors change from point to point as shown in figure (a). Also shown in the bottom left part of the figure is the standard shape (square) in the parametric coordinates. Finally, the standard square and the general quadrilateral on the right show as filled polygons the areas between the grid lines. This is used as a visual measure of the distortion of the quadrilateral.

Now imagine the topmost corner of the quadrilateral is moved towards the bottommost corner. The original shape of the quadrilateral is shown for reference in Figure 4.5(b). Clearly the shift caused the areas of the filled polygons to change, with the top corner polygon being squished more than the one at the bottom.

Figure 4.5(c) continues the story: If the downward movement of the corner reaches the straight line that connects the leftmost and rightmost corners, break point is reached. At that instant the top filled polygon that was originally a convex quadrilateral is reduced to a triangle, and its area has at that point decreased considerably relative to Figure 4.5(a).

If the originally topmost corner is moved further along the same direction downwards, the area of the filled polygon at that corner will become *negative*. The quadrilateral shape is now concave (Figure 4.5(d)).

Elements of the shapes shown in Figures 4.5(c,d) must be avoided, as the numerical quadrature would lead to incorrect results, and the overall accuracy of the element would deteriorate significantly. The ideal shape is a square, followed by a rectangle of an aspect ratio that is not too high (e.g. aspect ratio smaller than $4 : 1$).
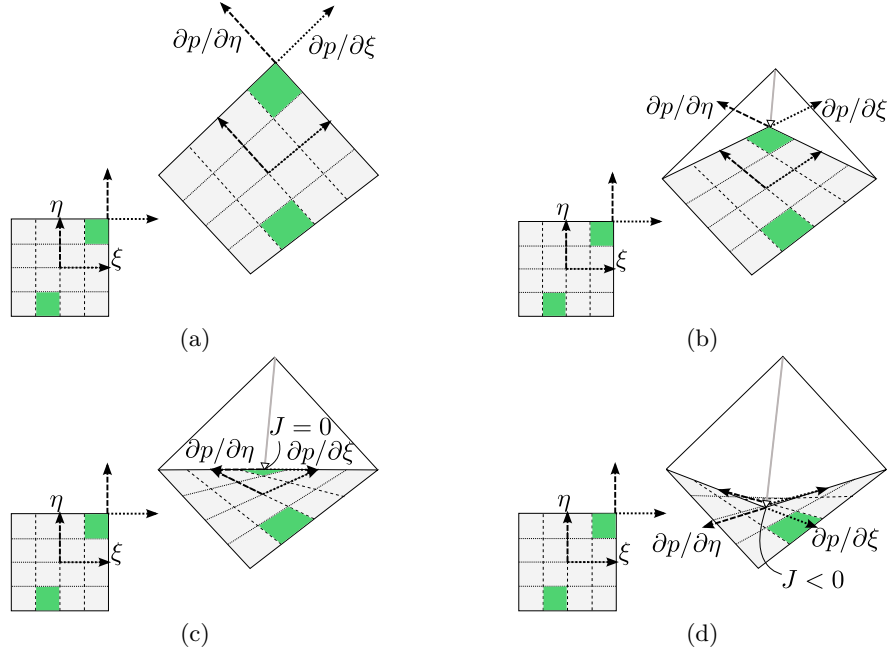
Fig. 4.5. Distortion of the elementary areas in the general quadrilateral mapped from the standard shape and then distorted by the movement of the top node

4.5 Effects of the choice of the numerical quadrature rule

In Section 4.2 the conductivity matrix of a rectangular element was constructed using the default quadrature rule for this type of element, Gauss 2×2 rule. Here we will explore the effect that switching to a one-point Gauss quadrature rule has on the elementwise conductivity matrix.

The elementwise conductivity matrix was computed in Section 4.2 using the formula (4.2). For the one-point Gauss rule, the sum degenerates into a single term:

$$[K^{(e)}] = [\mathrm{grad}N(\xi_q, \eta_q)] \; \kappa \; [\mathrm{grad}N(\xi_q, \eta_q)]^T \; \Delta z \; \det[J(\xi_q, \eta_q)]W_q \;, \tag{4.20}$$

where $\xi_1 = 0, \eta_1 = 0, W_1 = 4$. Note that the point $\xi_1 = 0, \eta_1 = 0$ maps to the geometrical center of the rectangular element.

Due to the simplicity of the rectangular shape and the minimal expense of the one-point quadrature, we can evaluate the conductivity matrix by trivial back-of-the-envelope calculations.

The basis function gradients with respect to x, y may be evaluated from the general formula (3.62), but for the centroid of the rectangular element they are also easily computed from the elementary rise-over-run formula. For instance,

$$\frac{\partial N_1}{\partial x} = \frac{1/2}{A}, \quad \frac{\partial N_1}{\partial y} = \frac{1/2}{B}$$

and

$$\frac{\partial N_2}{\partial x} = \frac{-1/2}{A}, \quad \frac{\partial N_2}{\partial y} = \frac{1/2}{B} \;.$$

Also, in Section 4.2 the Jacobian to be constant across the entire element $J = AB/4$.

The conductivity matrix can now be evaluated: The one-point integration yields

$$[K^{(e)}] = \frac{\kappa \Delta z}{4} \begin{bmatrix} A/B + B/A, & A/B - B/A, & -A/B - B/A, & B/A - A/B \\ A/B - B/A, & A/B + B/A, & B/A - A/B, & -A/B - B/A \\ -A/B - B/A, & B/A - A/B, & A/B + B/A, & A/B - B/A \\ B/A - A/B, & -A/B - B/A, & A/B - B/A, & A/B + B/A \end{bmatrix} . \tag{4.21}$$

As expected, the temperatures at the nodes of the same values produces zero output

$$[K^{(e)}][T^{(e)}] = [0] \quad \text{for } [T^{(e)}] = [1, 1, 1, 1]^T . \tag{4.22}$$

This is because the entries in each row sum up to zero. However, there is another possibility: sum only the entries in column 1 and 3, and we also get zero output.

$$[K^{(e)}][T^{(e)}] = [0] \quad \text{for } [T^{(e)}] = [1, 0, 1, 0]^T . \tag{4.23}$$

The same is possible for column 2 and 4.

$$[K^{(e)}][T^{(e)}] = [0] \quad \text{for } [T^{(e)}] = [0, 1, 0, 1]^T . \tag{4.24}$$

Altogether, we can see that there are now two possibilities for getting zero output from the elementwise conductivity matrix:

$$[T^{(e)}] = [1, 0, 1, 0]^T \quad \text{or} \quad [T^{(e)}] = [0, 1, 0, 1]^T . \tag{4.25}$$

The vector $[T^{(e)}] = [1, 1, 1, 1]^T$ is not a third, independent, possibility, as it may be obtained by adding together the two vectors in (4.25). In summary, this means that the rank of the elementwise conductivity matrix (4.21) is only two (2), not three (3) as would be expected.

The rank of the elementwise conductivity matrix can be understood in terms of an eigenvalue problem

$$[K^{(e)}][T^{(e)}] = \lambda[T^{(e)}]$$

The two vectors (4.25) are two eigenvectors, both associated with a zero eigenvalue. The defect of the matrix is the number of zero eigenvalues, so the rank being the dimension of the matrix minus the defect is two, 4-2=2.

The graphical representation of the first two modes that correspond to zero eigenvalues helps explain what is going on. The modes are representations of the temperature across the element, shown as a surface raised above the element to the third dimension. The slope of the temperature surface corresponds to the gradient of the temperature. We can see that the gradient of the temperature is nonzero everywhere, except at the midpoint of the element where the two dashed lines that lie in the temperature surface intersect. Those two lines have zero slope, and therefore at the midpoint the gradient of the temperature is zero. But that is exactly where the conductivity matrix was integrated by the one-point Gauss rule. The integration of the conductivity matrix may be thought of as imposition of constraints on the possible variations of temperature across the element that are allowed. In particular, the correct conductivity matrix should allow for a uniform temperature distribution, which would correspond to zero temperature gradient everywhere. Clearly imposing such a constraint in the quadrilateral at a single quadrature point is not sufficient, since there are two possible distributions of temperature which are not uniform but which do give a zero temperature gradient at the quadrature point (Figure 4.6).

The problem with a rank-deficient elementwise conductivity matrices is that when the conductivity matrix of the entire structure is assembled, the global conductivity matrix may be genuinely singular, or close to singular, even after sufficient boundary conditions have been applied to theoretically make the solution unique. The deficiency of the individual element matrices may show up in the form of the patterns of Figure 4.6.

Figure 4.7 shows a particular steady-state heat conduction case with nonzero internal heat generation rate and a combined essential and natural boundary condition. On the left the solution was obtained with the Q4 quadrilateral used with the 2×2 Gauss quadrature, on the right the same mesh is used with a one-point Gauss quadrature. For the latter scheme the individual elementwise conductivity matrices are singular, and the global conductivity matrix is close to singular. The singular matrix is prone to instabilities of the solution, and we can compare the patterns of zero-eigenvalue modes for the individual elements shown above with the oscillations that developed in the solution below.
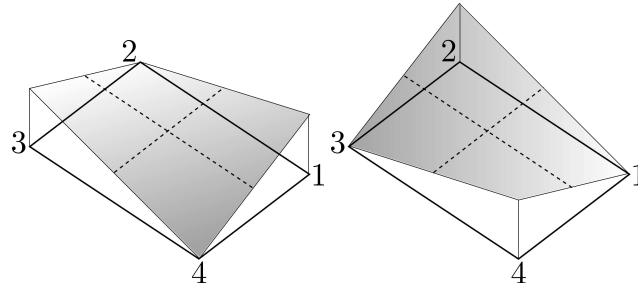
Fig. 4.6. Two possible variations of the temperature across the element that lead to zero gradient of the temperature at the centroid of the element
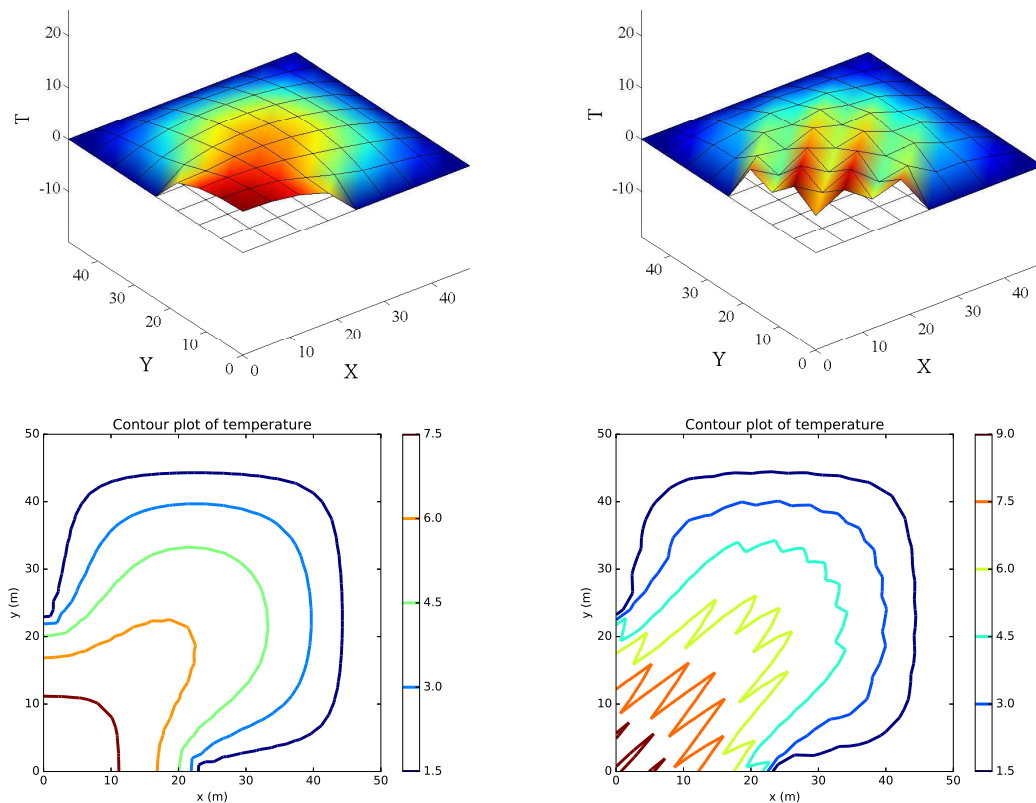

Fig. 4.7. Square with both insulated and zero-temperature boundary condition and internal heat generation rate. Solution to the problem obtained with full quadrature of the elementwise matrices (on the left), and solution obtained with one-point quadrature of the elementwise matrices (on the right).

> The quadrature rules in finite element programs are chosen to be both efficient, and sufficient. Efficient, to minimize the computing effort and to speed up the simulations. Sufficient, to avoid the appearance of spurious (in the sense of "not real") solutions.

4.6 Example finite element program

The solution to the above heat-conduction problem (refer to Figure 4.7) was obtained with a simple, but reasonably complete, implementation of finite element program for meshes composed of quadrilaterals. The region is square, of homogeneous and isotropic material.

Python
- script

The complete code is presented elsewhere (page), here we just look at the highlights: The first part of the script generates a topologically regular mesh of square four-node elements. The quadrature rule is chosen by commenting or uncommenting blocks

```
103 xe = array([[-0.577350269189626, -0.577350269189626],
104            [-0.577350269189626, +0.577350269189626],
105            [+0.577350269189626, -0.577350269189626],
106            [+0.577350269189626, +0.577350269189626]])
107 W = array([1, 1, 1, 1])
```

and

```
110 #xe = array([[+0., +0]])
111 #W = array([4.0])
```

In this case the first block defines a four point Gauss quadrature rule, and the commented-out block defines a one-point Gauss quadrature rule. If we wish to test the one-point quadrature rule (i.e. the effect of underintegration), we switch these two blocks: comment out the first one, and remove the comment letters (#) from the second one.

The global conductivity matrix in the global heat load vector are defined as

```
109 K = zeros((N_f, N_t))
110 L = zeros((N_f, 1))
```

Here, as before, N_f is the number of free degrees of freedom, and N_t is the total number of degrees of freedom. In defining a dense (full) matrix the script is fairly unsophisticated. A real workhorse finite element program would use a sparse matrix. The matrix K is full, meaning every single number in the matrix is stored and processed, including all the zeros. This can get expensive very quickly. The solution are sparse matrices: they require much less storage and are more efficient to process.

With the definition of one auxiliary variable (the number of nodes per element, that is 4) and the zero-based connectivity array

```
116 nnpe = conn.shape[1] # number of nodes per element
117 zconn = conn - 1
```

we are ready to launch into a loop over all the finite elements

```
118 for index  in arange(conn.shape[0]):
119     # Element degree-of-freedom array,  converted to zero base
120     zedof = dof[zconn[index, :]]-1
121     # Initialize the elementwise conductivity matrix.
122     Ke = zeros((nnpe, nnpe))
123     # Initialize the elementwise heat load vector
124     LQe = zeros((nnpe, 1))
```

The elementwise conductivity matrix and the elementwise heat-load vector due to the internal heat generation rate are allocated above. The next loop, the one over the quadrature points, now computes the expected Jacobian matrix, the gradient of the basis functions, and the contributions to the elementwise quantities are added.

```
126     for qp in range(xe.shape[0]):
127         xi, eta = xe[qp, 0], xe[qp, 1]
128         lx = x[zconn[index, :], :]
129         # Compute the Jacobian matrix
130         J = dot(lx.T, gradNpar(xi, eta))
131         # The Jacobian
132         detJ = linalg.det(J)
133         # The  gradient  of the basis functions with respect to x,y
134         gradN = dot(gradNpar(xi, eta), linalg.inv(J))
135         # Add the contribution to the conductivity matrix
136         Ke = Ke + kappa * Dz * dot(gradN, gradN.T) * detJ * W[qp]
137         # At the contribution to the elementwise heat load vector
138         LQe = LQe + Q * Dz * N(xi, eta) * ones((nnpe, 1)) * detJ * W[qp]
```

The elementwise quantities are then assembled in precisely the same way as discussed before.

```
139      # Assemble elementwise conductivity matrix
140      for ro  in range(len(zedof)):
141          for co  in range(len(zedof)):
142              if (zedof[ro] < N_f):
143                  K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
144      # Assemble the elementwise heat load vector
145      for ro  in range(len(zedof)):
146          if (zedof[ro] < N_f):
147              L[zedof[ro]] = L[zedof[ro]] + LQe[ro]
```

The resulting system of equations is solved `T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], L)`, using the free-free part of the conductivity matrix, and the results are processed with a contour plot.

Despite the simplicity of this code, it can process a decent-sized mesh in a matter of seconds: perhaps 20 seconds for 10,000 quadrilateral elements.

4.7 Background, explanations, details

Box 14. Gauss Quadrature over an Interval

The numerical quadratures that are in common use with polynomial finite elements are the **Gauss rules**. They are by now standard fare described in any number of textbooks (e.g. [3]). For completeness we present a simple version of the logic of Gaussian quadrature rules below.

We will start by explaining the basic idea on the example of a one-point integration rule. The idea is in fact common to a variety of quadrature rules: instead of the original function $f(\xi)$ we integrate an interpolation polynomial $L(\xi)$ that passes through selected points on the function curve.

For our selected example, a one-point rule, we will consider an interpolating polynomial that passes through a single point. Such a polynomial is a constant, $L(\xi) = \text{const} = f(\xi_1)$. The coordinate ξ_1 is the location of the integration point, and at this point it is unknown. We are now going to formulate a condition from which to compute the optimal location of this integration point. The function $f(\xi)$ is integrated on the standard interval $-1 \leq \xi \leq +1$ numerically using the rule, with $M = 1$, and ξ_1 and W_1 to be determined, as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \sum_{k=1}^{M} L(\xi_k)W_k = L(\xi_1)W_1$$

where the polynomial L interpolates at the integration point, $L(\xi_1) = f(\xi_1)$. Since for the case the given function is a constant, $f(\xi) = C$, we must have that

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi = 2C = L(\xi_1)W_1 = f(\xi_1)W_1 = CW_1 \tag{4.26}$$

the weight of the integration point must be $W_1 = 2$.

Next we will consider the location of the integration point. Our criterion will be to make the rule as accurate as possible. In particular, if the given function is a polynomial of a certain degree the integration rule should integrate it exactly. That means that the integral of the difference $F(\xi) = f(\xi) - L(\xi)$ must vanish

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = 0 \tag{4.27}$$

Because L interpolates at the integration point, the difference $F(\xi)$ assume value zero at the integration point, $F(\xi_1) = 0$. This can be accomplished by writing $F(\xi)$ as the product of two polynomials

$$F(\xi) = (\xi - \xi_1)q(\xi) \tag{4.28}$$

The first term, $(\xi - \xi_1)$, ensures that $F(\xi_1) = 0$. What could the second term look like? Consider a linear polynomial as an example

$$q(\xi) = A\xi + B$$

Substituting for $F(\xi)$ we obtain for the integral (4.27)

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = \int_{-1}^{+1} (\xi - \xi_1)(A\xi + B)\mathrm{d}\xi$$

If the above integral should become zero for arbitrary A and B

$$\int_{-1}^{+1} (\xi - \xi_1)(A\xi + B)\mathrm{d}\xi = 0 \quad \Rightarrow \quad A\int_{-1}^{+1} (\xi - \xi_1)\xi\mathrm{d}\xi = 0 \quad \text{and} \quad B\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = 0$$

However since we have only ξ_1 to play with, we cannot possibly make both $\int_{-1}^{+1} (\xi - \xi_1)\xi\mathrm{d}\xi = 0$ and $\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = 0$. Our guess of $q(\xi)$ as a linear polynomial was too ambitious. We should have picked a polynomial with a single unknown coefficient, $q(\xi) = D$ (a constant polynomial). Then if we substitute for $F(\xi)$ in the integral (4.27) we obtain

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = \int_{-1}^{+1} (\xi - \xi_1)D\mathrm{d}\xi$$

which means the condition for ξ_1 reads

$$\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = 0$$

Therefore

$$\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = \left[\xi^2/2 - \xi\xi_1\right]_{-1}^{+1} = -2\xi_1 = 0 \quad \Rightarrow \quad \xi_1 = 0$$

To summarize, the one-point Gauss quadrature rule is defined by the parameters $\xi_1 = 0$ and $W_1 = 2$. It can exactly integrate a linear polynomial, as evident from the fact that it can exactly integrate the polynomial (4.28).

Using the route introduced above leads to Gauss rules with an arbitrary number of points. Table 4.2 lists three most common ones. For completeness: these are not the only Gauss rules that are in use in the finite element methods. The rules we discussed here are the so-called open quadrature rules since the endpoints of the interval are not included amongst the integration points.

We will expand on the above one-point rule derivation by determining the parameters of a two-point Gauss quadrature rule. The goal is to integrate the function $p(\xi)$ by integrating its interpolating polynomial $L(\xi)$. Interpolation conditions are satisfied at the quadrature points. We have to find the location of the quadrature points ξ_1, ξ_2 so that the polynomial $p(\xi) - L(\xi)$ becomes zero at the quadrature points, and at the same time integrates to zero

$$\int_{-1}^{+1} p(\xi) - L(\xi)\, dx = 0$$

Consult the figure below– the red-filled areas must cancel:
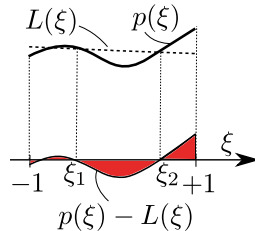
We will tailor the Gauss rule to work well for polynomials. We will assume therefore that both $p(\xi)$ and $L(\xi)$ are polynomials and therefore their difference is also going to be a polynomial

$$F(\xi) = p(\xi) - L(\xi)$$

It is important to realize that all such polynomials may be written as

$$F(\xi) = (\xi - \xi_1)(\xi - \xi_2)q(\xi)$$

The term $(\xi - \xi_1)$ makes sure that the interpolating condition holds at ξ_1 since it guarantees $F(\xi_1) = 0$. Similarly the term $(\xi - \xi_2)$ makes sure that the interpolating condition holds at ξ_2. The remaining term, $q(\xi)$, is at this point arbitrary, but must satisfy

$$\int_{-1}^{+1} F(\xi) \, dx = 0$$

In order to make this happen we can choose the locations of the quadrature points. Since we have two quadrature points, we need exactly two equations. Such equations will materialize if we choose $q(\xi) = A\xi + B$ as

$$\int_{-1}^{+1} F(\xi) \, dx = \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)q(\xi) \, dx = \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)(A\xi + B) \, dx = 0$$

means that because A and B are arbitrary we obtain two equations

$$\int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)\xi \, dx = 0 \,, \quad \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2) \, dx = 0$$

that have to be satisfied at the same time. These two equations can be solved for the locations of the quadrature points. The two integrals are evaluated using symbolic manipulation in Python with the `sympy` module: first we obtain the indefinite integrals

```
I1 = sympy.integrate((xi-xi1)*(xi-xi2), xi)
I2 = sympy.integrate(xi*(xi-xi1)*(xi-xi2), xi)
```

which become expressions in the unknowns `xi1`, `xi2` upon evaluation of these integrals between the limits $-1 \le \xi \le +1$.

```
I1v = I1.subs(xi, +1) - I1.subs(xi, -1)
I2v = I2.subs(xi, +1) - I2.subs(xi, -1)
```

The integrals are set equal to zero and the resulting system of equations is solved for the locations of the quadrature points:

```
solution = sympy.solve([I1v, I2v], xi1, xi2)
print(solution)
```

which gives

```
[(-sqrt(3)/3, sqrt(3)/3), (sqrt(3)/3, -sqrt(3)/3)]
```

Evidently the locations of the quadrature points are $\xi_1 = -\frac{1}{\sqrt{3}}$ and $\xi_2 = +\frac{1}{\sqrt{3}}$. (The other solution switches these two variables, but otherwise it has the same meaning.)

The weights of the two-point Gauss rule can now be determined so that when the original function $p(\xi)$ and the interpolating function $L(\xi)$ are one and the same function, the numerical integral comes out exact. Since the interpolating function is determined by two points, $p(\xi)$ must be a linear polynomial $(A\xi + B)$. Therefore we require that the exact integral of such a function

$$\int_{-1}^{+1} (A\xi + B) \, dx = 2B$$

is identically reproduced by the numerical quadrature rule

$$\int_{-1}^{+1} (A\xi + B)\, dx \approx (A\xi_1 + B)W_1 + (A\xi_2 + B)W_2 = 2B$$

Therefore we need

$$A\xi_1 W_1 + A\xi_2 W_2 = 0 \implies W_1 = W_2$$

and

$$BW_1 + BW_2 = 2B \implies W_1 + W_2 = 2$$

Hence we conclude $W_1 = W_2 = 1$. In summary, we obtain the following table for the two-point Gauss rule:

k	ξ_k	W_k
1	$-\frac{1}{\sqrt{3}}$	1
2	$+\frac{1}{\sqrt{3}}$	1

Since we have shown that the integration is exact for the difference of the interpolated and interpolating function being $F(\xi) = (\xi - \xi_1)(\xi - \xi_2)q(\xi) = (\xi - \xi_1)(\xi - \xi_2)(A\xi + B)$, which is a cubic polynomial, the two-point Gauss rule is *exact* for constant, linear, quadratic, and cubic polynomials.

Rule	Coordinates ξ_j	Weights W_j	Integrates exactly
1-point	0	2	linear polynomial
2-point	$-\sqrt{1/3}$	1	cubic polynomial
	$+\sqrt{1/3}$	1	
3-point	$-\sqrt{3/5}$	5/9	
	0	8/9	quintic polynomial
	$+\sqrt{3/5}$	5/9	

Table 4.2. Gauss quadrature rules

End Box 14

Box 15. Gauss integration over quadrilaterals

In this section we will extend the one-dimensional Gaussian integration rule for the standard interval to work for two-dimensional integration on the standard square. See Box 14 The solution can be best explained with a picture: call for Figure 4.1 for the standard bi-unit square ($-1 \leq \xi \leq +1$ and $-1 \leq \eta \leq +1$). The integration over the square may be split into two one-dimensional integrations as

$$\int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta)\, d\xi d\eta = \int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi, \eta)\, d\eta \right) d\xi$$

or,

$$\int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta) \, \mathrm{d}\xi \mathrm{d}\eta = \int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi, \eta) \, \mathrm{d}\xi \right) \mathrm{d}\eta$$

The one-dimensional Gaussian rule may be applied to the outer integral

$$\int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi, \eta) \, \mathrm{d}\xi \right) \mathrm{d}\eta \approx \sum_{k=1}^{M} \left(\int_{-1}^{+1} f(\xi, \eta_k) \, \mathrm{d}\xi \right) W_k$$

Now each of the integrals $\int_{-1}^{+1} f(\xi, \eta_k) \, \mathrm{d}\xi$ may be evaluated with a numerical rule, possibly a different one from the one used above. Normally there is no reason to choose a different numerical rule, and therefore we can write

$$\int_{-1}^{+1} f(\xi, \eta_k) \, \mathrm{d}\xi \approx \sum_{j=1}^{M} f(\xi_j, \eta_k) W_j$$

Substituting of the latter approximation into the former yields

$$\int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi, \eta) \, \mathrm{d}\xi \right) \mathrm{d}\eta \approx \sum_{k=1}^{M} \sum_{j=1}^{M} f(\xi_j, \eta_k) W_j W_k$$

Two-and three-dimensional Gaussian rules may be written as one-dimensional tables, exactly as for one-dimensional Gaussian rules, by listing the quadrature points and their weights as follows: the coordinates of the quadrature points are composed as tensor products of the coordinates of the one-dimensional rules, and the corresponding weights are the products of the weights of the one-dimensional rules. For instance, one-point rule in one dimension

One-point one-dimensional rule		
j	Coordinates ξ_j	Weights W_j
1	0	2

yields a one-point rule in two dimensions,

One-point two-dimensional rule		
j	Coordinates ξ_j, η_j	Weights W_j
1	0, 0	$2 \times 2 = 4$

and two-point rule in one dimension

Two-point one-dimensional rule		
j	Coordinates ξ_j	Weights W_j
1	$-\sqrt{1/3}$	1
2	$+\sqrt{1/3}$	1

gives the four-point rule in two dimensions

Four-point two-dimensional rule		
j	Coordinates ξ_j, η_j	Weights W_j
1	$-\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
2	$-\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$
3	$+\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
4	$+\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$

The numerical integration rule can therefore be expressed using a single summation as

$$\int_{-1}^{+1} \left(\int_{-1}^{+1} f(\xi, \eta) \, \mathrm{d}\xi \right) \mathrm{d}\eta \approx \sum_{q=1}^{M^2} f(\xi_q, \eta_q) W_q \ . \tag{4.29}$$

Figure 4.8 illustrates the above two-dimensional Gauss integration rule tables graphically. Higher-order Gaussian integration rules would be derived entirely analogously.
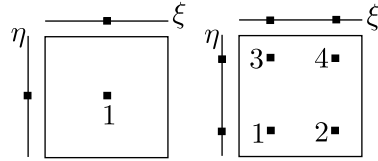
Fig. 4.8. Illustration of the relationship of one-dimensional Gaussian integration rules and two-dimensional rules on quadrilaterals.

End Box 15

4.8 Code listings

pnpQuadAB.py

Python
- script

```python
1  # Finite Element Modeling with Abaqus and Python for Thermal and
2  # Stress Analysis
3  # (C)  2017-2020, Petr Krysl
4  #
5  """
6   A single quadrilateral element of rectangular shape.  Elementwise
7   conductivity matrix. Evaluated symbolically with numerical quadrature.
8   """
9
10 from sympy import symbols, simplify, Matrix, det
11
12 # Define the basis function matrix
13 xi, eta = symbols('xi eta')
14 N = Matrix([(xi-1)*(eta-1)/4,
15             (xi+1)*(eta-1)/-4,
16             (xi+1)*(eta+1)/4,
17             (xi-1)*(eta+1)/-4]).reshape(4,1)
18 print(N)
19
20 #Define geometry
21 A, B = symbols('A, B')
22 x = Matrix([[A, B], [0, B], [0, 0], [A, 0]])
23
24 #  Differentiate to obtain the Basis function gradients
25 gradNpar = diff(N, 'xi').row_join(diff(N, 'eta'))
26 print(gradNpar)
27
28 # Compute the Jacobian matrix
29 J = x.T*gradNpar
30 print(simplify(J))
31 #  The Jacobian
32 Jac = det(J)
33 print(Jac)
34
35 #  The  gradient  of the basis functions with respect to x,y is a
36 #  symbolic expression which needs to be evaluated for particular values of
37 #  xi,eta
```

```python
38 gradN = simplify(gradNpar*(J**-1))
39 print(gradN)
40
41 # We introduce the the thermal
42 # conductivity  and the thickness of the slice, as symbolic variables.
43 kap, Dz = symbols('kap, Dz')
44
45 # Note that using subs() will substitute the values of the parametric
46 # coordinates.  The Jacobian and the material properties and the thickness
47 # of the slice are assumed constant and hence are not included in this sum.
48 # The weights of the for quadrature points are all 1.0.
49 # The formula (4.13) will be recognizable in the following lines.
50 gN = gradN.subs(xi, -0.577350269189626).subs(eta, -0.577350269189626)
51 K1 = kap*Dz*Jac*gN*gN.T # term 1 of the sum
52 gN = gradN.subs(xi, -0.577350269189626).subs(eta, +0.577350269189626)
53 K2 = kap*Dz*Jac*gN*gN.T # term 2 of the sum
54 gN = gradN.subs(xi, +0.577350269189626).subs(eta, -0.577350269189626)
55 K3 = kap*Dz*Jac*gN*gN.T # term 3 of the sum
56 gN = gradN.subs(xi, +0.577350269189626).subs(eta, +0.577350269189626)
57 K4 = kap*Dz*Jac*gN*gN.T # term 4 of the sum
58
59 K = simplify(K1+K2+K3+K4)
60 print(K)
61
62 print(K.eigenvals())
```

Listing 4.1. pnpQuadAB.py

pnpConcreteColumn1Q4.py

```python
1 # Finite Element Modeling with Abaqus and Python for Thermal and
2 # Stress Analysis
3 # (C)  2017-2020, Petr Krysl
4 #
5 """
6  Concrete column with temperature boundary condition.  Finite element mesh
7  that consists of a single four-node quadrilateral.
8  """
9
10 import math
11 from numpy import array as array
12 from numpy import zeros as zeros
13 from numpy import arange as arange
14 from numpy import dot as dot
15 from numpy import linalg as linalg
16
17 ##
18 # Evaluate for a given data
19 a = 2.5  # radius of the column
20 b = a * math.sin(15.0 / 180 * math.pi)  # dimension
21 h = a * math.cos(15.0 / 180 * math.pi)  # dimension
22 Q = 4.5  # internal heat generation rate
23 k = 1.8  # thermal conductivity
24 Dz = 1.0  # thickness of the slice
25 x = array([[0, 0], [h, -b], [a, 0], [h, b]])  # Coordinates of nodes
26
27 ##
28 # Define a little function to calculate the gradient of the basis functions
```

Python
- script

```python
29 # in the parametric coordinates.
30
31
32 def gradNpar(xi, eta):
33     """
34     A one-liner to calculate the matrix of the basis function gradients
35     in the parametric coordinates.
36     """
37     return array([
38         [eta / 4 - 1. / 4, xi / 4 - 1. / 4],
39         [1. / 4 - eta / 4, - xi / 4 - 1. / 4],
40         [eta / 4 + 1. / 4, xi / 4 + 1. / 4],
41         [- eta / 4 - 1. / 4, 1. / 4 - xi / 4]])
42
43 # The expressions actually come from  the following symbolic code:
44 #xi,eta = symbols('xi eta')
45 #N=Matrix([(xi-1)*(eta-1)/4, (xi+1)*(eta-1)/-4, (xi+1)*(eta+1)/4, (xi-1)*(eta+1)
       /-4])
46 # N=N.reshape(4,1)
47 # gradNpar=diff(N,'xi').row_join(diff(N,'eta'))
48 # print(gradNpar)
49
50 ##
51 # These are the integration point data
52 xe = array([[-0.577350269189626, -0.577350269189626],
53             [-0.577350269189626, +0.577350269189626],
54             [+0.577350269189626, -0.577350269189626],
55             [+0.577350269189626, +0.577350269189626]])
56 W = array([1, 1, 1, 1])
57
58 ##
59 # Initialize the elementwise conductivity matrix.
60 Ke = zeros((4, 4))
61 ##
62 # Loop over the quadrature points.
63 for qp in arange(xe.shape[0]):
64     xi = xe[qp, 0]
65     eta = xe[qp, 1]
66     # Compute the Jacobian matrix
67     J = dot(x.T, gradNpar(xi, eta))
68     # The Jacobian
69     detJ = linalg.det(J)
70     # The  gradient  of the basis functions with respect to x,y
71     gradN = dot(gradNpar(xi, eta), linalg.inv(J))
72     # Add the contribution to the conductivity matrix
73     Ke = Ke + k * Dz * dot(gradN, gradN.T) * detJ * W[qp]
74
75 print(Ke)
76
77 ##
78 # We will find it convenient to define a little function to evaluate the
79 # basis function values at a given quadrature point location.
80
81
82 def N(xi, eta):
83     """
84     A one-liner to calculate the matrix of the basis function values.
85     """
```

```
86      return array([(xi - 1) * (eta - 1) / 4,
87                    (xi + 1) * (eta - 1) / -4,
88                    (xi + 1) * (eta + 1) / 4,
89                    (xi - 1) * (eta + 1) / -4]).reshape(4, 1)
90
91  ##
92  # Initialize the elementwise heat-load vector .
93  Le = zeros((4, 1))
94  ##
95  # Loop over the quadrature points.
96  for qp in arange(xe.shape[0]):
97      xi = xe[qp, 0]
98      eta = xe[qp, 1]
99      # Compute the Jacobian matrix
100     J = dot(x.T, gradNpar(xi, eta))
101     # The Jacobian
102     detJ = linalg.det(J)
103     # Add the contribution to the heat load vector
104     Le = Le + Q * Dz * N(xi, eta) * detJ * W[qp]
105
106 print(Le)
107
108 # Global system matrix and global load vector
109 K = Ke
110 L = Le
111
112 ##
113 # The solution vector consists of all zeros, except in the first entry.
114 T = zeros((4, 1))
115 # Solve the global equations
116 T[0] = (1 / K[0, 0]) * L[0]
117 print('T_1 = ', T[0])
118
119 #
120 # Now we will postprocess to extract the heat flux vectors at the
121 # quadrature points.
122 ##
123 # Loop over the quadrature points.
124 for qp in arange(xe.shape[0]):
125     xi = xe[qp, 0]
126     eta = xe[qp, 1]
127     # Compute the location of the quadrature point
128     qploc = dot(N(xi, eta).T, x)
129     # Compute the Jacobian matrix
130     J = dot(x.T, gradNpar(xi, eta))
131     # The  gradient  of the basis functions with respect to x, y
132     gradN = dot(gradNpar(xi, eta), linalg.inv(J))
133     # Compute the gradient of temperature
134     gradT = dot(T.T, gradN)
135     # Heat flux vector
136     q = -k * gradT.T
137     print('Heat flux at', qploc, '  = ', q)
```

Listing 4.2. pnpConcreteColumn1Q4.py

IntegrEffectQ4.py

```
1  # Finite Element Modeling with Abaqus and Python for Thermal and
```

Python
- script

```python
# Stress Analysis
# (C)  2017-2020, Petr Krysl
#
"""
Square domain with partial temperature boundary condition
and partial insulated boundary condition.  Finite element mesh
that consists of four-node quadrilaterals.
 """

import math
from numpy import array, zeros, ones, arange, linspace
from numpy import unique, dot, linalg
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D

import time

tstart = time.time()

# Input parameters
kappa = 0.2 # thermal conductivity
Q = 0.005 # uniform heat source
sidel = 48 # length of the side of the square domain
nes = 18 # number of elements per side
Dz = 1.0  # thickness of the slice

# Generate the nodes and the connectivities of the quadrilaterals
xs = linspace(0, sidel, nes+1) # grid in the X direction
ys = linspace(0, sidel, nes+1) # grid in the Y direction
nx = len(xs) - 1 # number of element edges in the X direction
ny = len(ys) - 1 # number of element edges in the Y direction
nnodes = (nx+1) * (ny+1) # Number of nodes
nfes = nx * ny # Number of elements

# Generate the nodes
x = zeros((nnodes, 2)) # Coordinates of nodes: initialized to all zeros
k = 0
for j  in arange(0, (ny+1)):
    for i  in arange(0, (nx+1)):
        x[k, :] = [xs[i], ys[j]]
        k = k + 1

# Generate the elements: note that the nodes are numbered from 1
conn = zeros((nfes, 4), dtype=int)
k = 0
for i  in arange(0, (nx)):
    for j  in arange(0, (ny)):
        f = int(j * (nx+1) + i) + 1
        conn[k, :] = [f, f+1, f+(nx+1)+1, f+(nx+1)]
        k = k + 1

# Define the degrees of freedom
# Fixed temperature at the nodes on the boundary, except for a few
# in the lower-left corner.  The  node numbers are one-based
Fixed = [int(i)+1 for i in arange(nx/2, nx+1)]
Fixed = Fixed + [int(j*(nx+1))+1 for j in arange(ny/2, ny+1)]
Fixed = Fixed + [int(i+ny*(nx+1))+1 for i in arange(0, nx+1)]
```

```python
60 Fixed = Fixed + [int(j*(nx+1)+ny)+1 for j in arange(0, ny+1)]
61 Fixed = unique(Fixed)
62
63 dof = zeros((nnodes,1), dtype=int)
64 N_f = nnodes - len(Fixed) # number of free degrees of freedom
65 N_t = nnodes # total number of degrees of freedom
66
67 T = zeros((nnodes,1)) # vector of degrees of freedom
68
69 # Number the degrees of freedom: First we will number the prescribed
70 # degrees of freedom starting from the number of free degrees of freedom
71 k = N_f # first we number the prescribed degrees of freedom
72 for index  in Fixed:
73     dof[index-1] = k+1 # the DOF are 1-based
74     k = k + 1
75 k = 0 # next we number the free degrees of freedom
76 for index  in range(0, nnodes):
77     if (dof[index] == 0):
78         dof[index] = k+1 # the DOF are 1-based
79         k = k + 1
80
81
82 def gradNpar(xi, eta):
83     """
84     A one-liner to calculate the matrix of the basis function gradients
85     in the parametric coordinates.
86     """
87     return array([
88         [eta / 4 - 1. / 4, xi / 4 - 1. / 4],
89         [1. / 4 - eta / 4, - xi / 4 - 1. / 4],
90         [eta / 4 + 1. / 4, xi / 4 + 1. / 4],
91         [- eta / 4 - 1. / 4, 1. / 4 - xi / 4]])
92
93 def N(xi, eta):
94     """
95     A one-liner to calculate the matrix of the basis function values.
96     """
97     return array([(xi - 1) * (eta - 1) / 4,
98                   (xi + 1) * (eta - 1) / -4,
99                   (xi + 1) * (eta + 1) / 4,
100                  (xi - 1) * (eta + 1) / -4]).reshape(4, 1)
101
102 # These are the integration point data for four-point quadrature rule
103 xe = array([[-0.577350269189626, -0.577350269189626],
104            [-0.577350269189626, +0.577350269189626],
105            [+0.577350269189626, -0.577350269189626],
106            [+0.577350269189626, +0.577350269189626]])
107 W = array([1, 1, 1, 1])
108
109 # These are the integration point data for one-point quadrature rule
110 #xe = array([[+0., +0]])
111 #W = array([4.0])
112
113 K = zeros((N_f, N_t))
114 L = zeros((N_f, 1))
115
116 nnpe = conn.shape[1] # number of nodes per element
117 zconn = conn - 1
```

```python
118 for index  in arange(conn.shape[0]):
119     # Element degree-of-freedom array,  converted to zero base
120     zedof = dof[zconn[index, :]]-1
121     # Initialize the elementwise conductivity matrix.
122     Ke = zeros((nnpe, nnpe))
123     # Initialize the elementwise heat load vector
124     LQe = zeros((nnpe, 1))
125     # Loop over the quadrature points.
126     for qp in range(xe.shape[0]):
127         xi, eta = xe[qp, 0], xe[qp, 1]
128         lx = x[zconn[index, :], :]
129         # Compute the Jacobian matrix
130         J = dot(lx.T, gradNpar(xi, eta))
131         # The Jacobian
132         detJ = linalg.det(J)
133         # The  gradient  of the basis functions with respect to x,y
134         gradN = dot(gradNpar(xi, eta), linalg.inv(J))
135         # Add the contribution to the conductivity matrix
136         Ke = Ke + kappa * Dz * dot(gradN, gradN.T) * detJ * W[qp]
137         # At the contribution to the elementwise heat load vector
138         LQe = LQe + Q * Dz * N(xi, eta) * ones((nnpe, 1)) * detJ * W[qp]
139     # Assemble elementwise conductivity matrix
140     for ro  in range(len(zedof)):
141         for co  in range(len(zedof)):
142             if (zedof[ro] < N_f):
143                 K[zedof[ro], zedof[co]] = K[zedof[ro], zedof[co]] + Ke[ro, co]
144     # Assemble the elementwise heat load vector
145     for ro  in range(len(zedof)):
146         if (zedof[ro] < N_f):
147             L[zedof[ro]] = L[zedof[ro]] + LQe[ro]
148
149 # Solve for the global temperatures at the free degrees of freedom
150 T[0:N_f] = linalg.solve(K[0:N_f, 0:N_f], L)
151
152 tend = time.time()
153 print( 'Time =', tend-tstart)
154
155 if (len(T) < 10):
156     print('Tg=', Tg)
157
158 # Plot contours
159 plt.figure()
160 plt.gca().set_aspect('equal')
161 # setup three 1-d arrays for the x-coordinate, the y-coordinate, and the
162 # z-coordinate
163 xs = x[:, 0].reshape(nnodes,)  # one value per node
164 ys = x[:, 1].reshape(nnodes,)  # one value per node
165 ix = dof[arange(nnodes)] - 1
166 zs = (T[ix]).reshape(nnodes,)  # one value per node
167 triangles = conn[:, (0, 1, 2)] - 1  # the triangles are defined by the
        connectivity arrays
168 plt.tricontour(xs, ys, triangles, zs, linewidths=3)
169 triangles = conn[:, (0, 2, 3)] - 1  # the triangles are defined by the
        connectivity arrays
170 plt.tricontour(xs, ys, triangles, zs, linewidths=3)
171 plt.colorbar()
172 plt.title('Contour plot of temperature')
173 plt.xlabel('x (m)')
```

```
174 plt.ylabel('y (m)')
175 plt.show()
```

Listing 4.3. IntegrEffectQ4.py

5

FEA for 2-D Stress Analysis

In this chapter we will consider the finite element method applied to linear static stress analysis. Since the degrees of freedom now represent the components of displacement at the node, all the expressions become much more tedious to work with: the dimensions of the matrices increase significantly. This is especially true in three dimensions, which is why in this chapter we will discuss models that are formulated in two coordinates only. There are three such models: plane stress, plane strain, and axially symmetric. For simplicity we will start with plane stress; the plane strain model will follow naturally from our findings. The axially symmetric model differs from the first two in some respects, and needs a separate discussion.
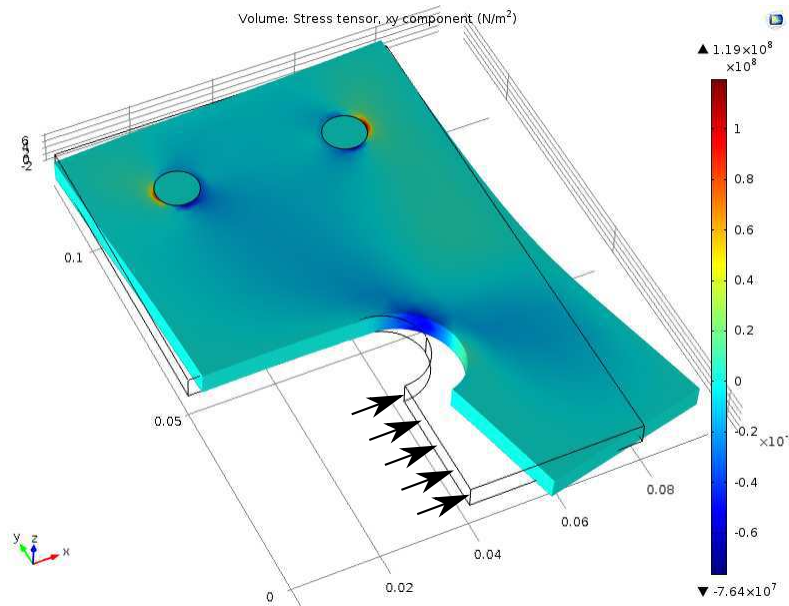
Fig. 5.1. Sample geometry: when the part is loaded so that the deformation results in the stresses $\sigma_x, \sigma_y, \tau_{xy}$ that are uniform throughout the thickness (in the Z direction), we have the so-called plane-stress conditions

5.1 Bracket plane-stress model

Figure 5.1 is intended to motivate the development of the plane-stress model. Consider a metal bracket manufactured out of a sheet of metal and spot-welded to another structure. The bracket is loaded as indicated by the arrows, and it deforms. While some deformation occurs out of the plane of

the sheet of metal, it is insignificant compared to the in-plane deformation. Importantly, the nonzero stresses are limited to the in-plane normal stresses σ_x and σ_y and the shear stress τ_{xy}; the other stresses are negligible. This suggests modeling only the in-plane deformations in the two coordinates x, y, and with suitable assumptions that are discussed in detail in Box 19 one can formulate the so-called plane stress model. See Box 19

The geometry is now described as a plane figure, for instance as in Figure 5.2. We limit ourselves initially to static deformation, without thermal effects. The balance equation is written in terms of the three in-plane stresses and the components of the body load $\bar{b}_x, \bar{b}_y$

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \bar{b}_x = 0$$
$$\frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \bar{b}_y = 0 \, . \tag{5.1}$$

This can be helpfully rewritten using an important matrix that is central to stress analysis: the strain-displacement ("symmetric gradient") matrix

$$[\mathcal{B}] = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \, . \tag{5.2}$$

It consists of partial-derivative operators, and when it is multiplied with a displacement vector on the right, it differentiates its components to produce strains. Its transpose is the so-called divergence,

$$[\mathcal{B}]^T = \begin{bmatrix} \partial/\partial x & 0 & \partial/\partial y \\ 0 & \partial/\partial y & \partial/\partial x \end{bmatrix} \, . \tag{5.3}$$

and with its help we can rewrite the balance equation as follows

$$[\mathcal{B}]^T [\boldsymbol{\sigma}] + [\bar{\boldsymbol{b}}] = [\mathbf{0}] \tag{5.4}$$

where we have arranged the stress components in vector form as

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \tag{5.5}$$

The stress is generated by elastic strains, and since we neglect thermal effects the elastic strains are equal to the total strains

$$[\boldsymbol{\sigma}] = [\boldsymbol{D}] \, [\boldsymbol{\epsilon}] \, . \tag{5.6}$$

To begin, we consider an isotropic material, and for the plane-stress model the material stiffness matrix reads:

$$[\boldsymbol{D}] = \frac{E}{(1 - \nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1 - \nu}{2} \end{bmatrix} \, , \tag{5.7}$$

where E is the Young's modulus, and ν is the Poisson ratio.

The total strains are also arranged as a vector

$$[\boldsymbol{\epsilon}] = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \tag{5.8}$$

and are computed from the displacement components

$$[\boldsymbol{u}] = \begin{bmatrix} u_x \\ u_y \end{bmatrix} \tag{5.9}$$

as

$$[\boldsymbol{\epsilon}] = [\mathcal{B}][\boldsymbol{u}] \ . \tag{5.10}$$

Now we can substitute for the stresses in the balance equation

$$[\mathcal{B}]^T \left([\boldsymbol{D}] [\boldsymbol{\epsilon}] \right) + [\overline{\boldsymbol{b}}] = [\boldsymbol{0}] \tag{5.11}$$

and subsequently also for the strains

$$[\mathcal{B}]^T \left([\boldsymbol{D}] [\mathcal{B}][\boldsymbol{u}] \right) + [\overline{\boldsymbol{b}}] = [\boldsymbol{0}] \tag{5.12}$$

and we obtain two coupled partial differential equations for the two displacement components u_x and u_y.

In order to solve these two partial differential equations uniquely we need boundary conditions. As for the heat conduction problem, a boundary condition needs to be specified along the entire surface of the modeled domain. In Figure 5.2 the surface consists of all the boundary segments along the circumference and the two circles for the holes inside. Complicating factor number one is that there are now *two* coupled variables: consequently *two* boundary conditions are now needed at *all* points of the boundary. Complicating factor number two is that because we are prescribing components of a vector, we must say in which coordinate system this is done, and sometimes for convenience, sometimes out of necessity, we have to use different coordinate systems for different parts of the boundary.
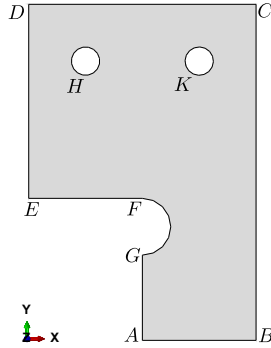
Fig. 5.2. Definition of the geometry of the bracket in two dimensions. Note that we are assuming here the thickness t in the third dimension to be constant across the entire domain.

The boundary conditions are expressed in terms of the displacement (the essential boundary condition) or in terms of the traction (the natural boundary conditions). The displacement component $u_s, s = x, y$ is prescribed on the part of the boundary $C_{u,s}$ (we write $C_{u,s}$ because the boundary surface is in the two-dimensional picture represented by a piece of a *Curve*)

$$u_s(\boldsymbol{x}) = \overline{u}_s(\boldsymbol{x}), \quad \boldsymbol{x} \text{ on } C_{u,s} \ , \tag{5.13}$$

and traction component $t_s, s = x, y$ is prescribed on the part of the boundary $C_{t,s}$

$$t_s(\boldsymbol{x}) = \overline{t}_s(\boldsymbol{x}), \quad \boldsymbol{x} \text{ on } C_{t,s} \ . \tag{5.14}$$

The traction on a piece of boundary surface with outer normal $\boldsymbol{n}$ is related to the stress existing in the solid at that point by the formula

$$\left[\mathcal{P}\boldsymbol{n}(\boldsymbol{x})\boldsymbol{\sigma}(\boldsymbol{x})\right]_s = t_s(\boldsymbol{x}), \tag{5.15}$$

where the matrix "*vector-stress vector dot product*" operator is defined as

$$\mathcal{P}\boldsymbol{n} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} . \tag{5.16}$$

With these pieces of information about the boundary conditions in the plane-stress analysis we can compare usefully with the heat conduction model from the previous chapters:

> The boundary condition expressed in terms of the displacement component is equivalent to prescribed temperature; the boundary condition expressed in terms of the heat flux component normal to the boundary is equivalent to prescribed traction component. This is well worth understanding: temperature and displacement are equivalent quantities, and heat flux and stress are equivalent quantities.

For the bracket of Figure 5.2 the boundary conditions are relatively simple to describe—refer to Figure 5.3. In Figure 5.3(a) the thick-line highlighted part of the boundary is where we prescribe traction components. The t_y component is zero along this entire part of the boundary, and the t_x component is zero everywhere along the boundary except for the segment GA, where it is prescribed as nonzero. Figure 5.3(b) shows that the displacement components are both prescribed along the circles that represents the holes, $u_x = u_y = 0$ (clamped or "encastré" condition).

(a)   (b)
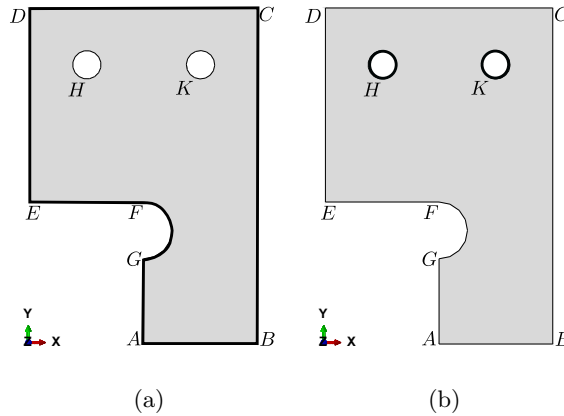
Fig. 5.3. Partitioning of the surface of the bracket for the application of the boundary conditions. (a) Part of the boundary where both traction components are prescribed. (b) Part of the boundary where both displacement components are prescribed.

There is a simple systematic way (algorithm) for applying boundary conditions in stress analysis. It relies on the simple observation that at any given point of the boundary for each of the components $s = x, y$ one of the quantities u_s and t_s, must be known and the other one must be unknown. The key is to choose an appropriate coordinate system in which to work. Often such a system will have orientation that places one of the coordinate directions normal to the surface.

For each of the coordinate directions $s = x, y$ then do:

1. Consider both u_s and t_s: If one of these is known (prescribed), the boundary condition is decided; otherwise if one of these is definitely unknown, the other one must be known, and if not explicitly given, then it will be equal to zero.
2. If the decision could not be reached above, restart by selecting a different coordinate system.

As an example, consider the circular arc FG in Figure 5.3. The appropriate coordinate system in this case will be in a cylindrical coordinate system with origin at the center of the arc. Since

the surface of the arc is not loaded by externally applied tractions, we know that the appropriate boundary condition is $t_n = t_m = 0$ (where n corresponds to the normal direction, and m is the tangential direction to the circle). The same boundary condition can be derived by considering the displacements of the points along the circular arc FG. Since we do not know how the points along this arc displace (i.e. the displacement components u_n and u_m are unknown), we must know the traction components, and since they are not explicitly given we must have $t_n = t_m = 0$.

5.2 Bookkeeping for plane stress FE models

It is necessary to start the discussion of the finite element model by gaining some understanding of what it means to have two degrees of freedom per node in the stress analysis model. Figure 5.4 shows a very small finite element model with four nodes and two elements, where three of the nodes are supported by rollers. As for the heat conduction model, the unknown (free) degrees of freedom will be numbered first, followed by the known (fixed, prescribed) degrees of freedom. Application of a roller at a node means that the degree of freedom restrained by the roller is a known quantity, either zero if the roller is immovable, or a nonzero value to which the roller makes the node displace orthogonally to the rolling direction. So starting with the y-component of the displacement at node 3, which is the degree of freedom 1 called U_1, and ending with U_8, the eight degrees of freedom are numbered. This information is conveniently recorded in a table:

Node numbers	1	2	3	4
DOF in x direction	3	5	6	4
DOF in y direction	2	7	1	8

> ✍ We are discussing here the so-called point supports applied at the nodes. This is a subtle subject that will be dealt with later: be aware that some point supports can cause the solution not to be meaningful!

There is no longer a one-to-one correspondence between nodes and degrees of freedom: there are twice as many degrees of freedom as there are nodes. There is one possibility of organizing the degrees of freedom that preserves a connection between the nodes and the degrees of freedom: we introduce a ***vector of degrees of freedom per node***, as opposed to a single degree of freedom per node. The numbers of the degree of freedom vectors can be taken as the node numbers.

Similarly to the expansion of the temperature (3.38), at any point on the mesh we can write the displacement vector as

$$[\boldsymbol{u}] = \sum_{j=1}^{N} N_j [\boldsymbol{u}_j] \tag{5.17}$$

where $[\boldsymbol{u}_j]$ is the displacement vector at node with degree of freedom vector j. The displacement $[\boldsymbol{u}_j]$ is the vector that consists of the two degrees of freedom

$$[\boldsymbol{u}_j] = \begin{bmatrix} u_{jx} \\ u_{jy} \end{bmatrix} . \tag{5.18}$$

We will use the notation $[\boldsymbol{u}_i]_s$ to refer to the sth entry of the vector $[\boldsymbol{u}_i]$ (where $s = x$ or $s = y$). So, for instance for node 1, the degree of freedom vector is $[3, 2]$. Hence in Figure 5.4, the degrees of freedom stored in the vector at node 1 are

$$[\boldsymbol{u}_1] = \begin{bmatrix} u_{1x} = U_3 \\ u_{1y} = U_2 \end{bmatrix} . \tag{5.19}$$

Let us now use (5.17) to compute strain. By definition (5.10)

$$[\epsilon] = [\mathcal{B}][u] = [\mathcal{B}] \sum_{j=1}^{N} N_j [u_j] = \sum_{j=1}^{N} [\mathcal{B}(N_j)][u_j] = \sum_{j=1}^{N} [B_j][u_j] \tag{5.20}$$

The nodal strain-displacement matrices $[B_j]$ are the result of applying the strain-displacement matrix of partial derivatives $\mathcal{B}$ of (5.2) to a basis function

$$[B_j] = [\mathcal{B}(N_j)] . \tag{5.21}$$

Explicitly spelled out

$$[B_j] = \begin{bmatrix} \dfrac{\partial N_j}{\partial x} & 0 \\ 0 & \dfrac{\partial N_j}{\partial y} \\ \dfrac{\partial N_j}{\partial y} & \dfrac{\partial N_j}{\partial x} \end{bmatrix} . \tag{5.22}$$

In words, the strain displacement matrices $[B_j]$ consist of the components of the gradients of the basis functions. The term $[B_j][u_j]$ is the contribution of the displacement at node j to the strain at a given point within the element.
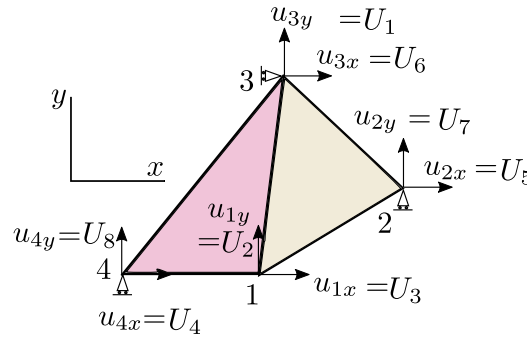
Fig. 5.4. Plane-stress FE model. Two-element mesh with four nodes. Three roller supports take away three fixed degrees of freedom (U_6, U_7, and U_8), leaving five free degrees of freedom in the model, $U_j, j = 1, ..., 5$.

5.3 Weighted residual equation for plane stress

The expression (5.17) is the trial function. The test function will be again one basis function at a time. With the trial and test functions at hand we are ready to formulate the weighted residual equation. The WR equation is presented here as a simplified version of the one derived for a fully three-dimensional analysis: See Box 21.

The balance equations of the plane-stress structure in the WR form can be written as follows:

$$\sum_{p=1}^{N} \int_S \left[[B_{j(q)}]^T [D] [B_{i(p)}] \right]_{r(q)s(p)} t \mathrm{d}S \, U_p$$
$$- \int_S N_{j(q)} [\bar{b}]_{r(q)} t \mathrm{d}S - \int_{C_{t,r}} N_{j(q)} \bar{t}_{r(q)} t \mathrm{d}C = 0 , \quad \text{for } q = 1, \dots, N_{\mathrm{f}}, \tag{5.23}$$

where the essential boundary conditions imply for the fixed degrees of freedom

$$U_p = \overline{U}_p, \text{ where node } i(p) \text{ is on } C_{u,s(p)} \quad \text{and } s = x \text{ or } y . \tag{5.24}$$

Here U_p are the degrees of freedom (horizontal and vertical displacements). The numbers

$$K_{qp} = \int_S \left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} t\mathrm{d}S \tag{5.25}$$

are the coefficients of the (global) stiffness matrix. Note that

$$\left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right] \tag{5.26}$$

is a 2×2 matrix. The subscripting $[\bullet]_{r(q)s(p)}$ extracts one of the entries of this matrix, in the row $r(p)$ and column $s(q)$. The meaning of the subscripts is as follows

- p, q are the global degree of freedom numbers;
- $j(q)$ is the number of the degree of freedom vector in which we can find degree of freedom q;
- $i(p)$ is the number of the degree of freedom vector in which we can find degree of freedom p;
- $r(p)$ is the direction of the degree of freedom p;
- $s(q)$ is the direction of the degree of freedom q.

As mentioned also above, the vector of the degrees of freedom is attached to the node. This degrees with the convention used in the heat conduction model that the degree of freedom j and the basis function N_j was attached to a node.

As an example, consider Figure 5.4. Let us take degrees of freedom 1 and 4. So, $p = 1$ and $q = 4$. We will find the degree of freedom $p = 1$ at node 3, as displacement in the vertical direction. The vector of degrees of freedom at that node is $[6, 1]$. Therefore we see that $j = 3$, and $r = y$. Analogously, we will find the degree of freedom $q = 4$ at node 4, as displacement in the horizontal direction. The vector of degrees of freedom at that node is $[4, 8]$. Therefore we see that $i = 4$, and $s = x$. Obviously, we consider the horizontal direction as the first (so that $s = x$ is equivalent to $s = 1$), and the vertical direction as the second (so that $r = y$ is equivalent to $r = 2$).

> The meaning of (5.23) is the following: it is a balance of forces. The first line represents the restoring (resistance) forces (think "spring constant times stretch"), the second line is the applied loading, both body forces (such as gravity), and loading on the surface (traction). The finite element method really is nothing else but a clever scheme of lumping the response of the continuous structure and the continuous applied loads into discrete springs and concentrated forces. We will discuss this in detail in Section 6.8

The integrals in the sum in the first term of equation (5.23) represent a rectangular stiffness matrix, which may be again split into a square matrix $\boldsymbol{K}_{ff}$ and a rectangular matrix $\boldsymbol{K}_{fd}$ as in Section 3.12.1. Then the term $-\boldsymbol{K}_{fd}\boldsymbol{U}_d$ would literally be the support-settlement loads.

5.4 Elementwise quantities for the three-node triangle

The elementwise expressions on the three-node triangle are expressed in terms of the displacements of the three nodes

$$[\boldsymbol{u}_1] = \begin{bmatrix} u_{1x} \\ u_{1y} \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}, \quad [\boldsymbol{u}_2] = \begin{bmatrix} u_{2x} \\ u_{2y} \end{bmatrix} = \begin{bmatrix} U_3 \\ U_4 \end{bmatrix}, \quad [\boldsymbol{u}_3] = \begin{bmatrix} u_{3x} \\ u_{3y} \end{bmatrix} = \begin{bmatrix} U_5 \\ U_6 \end{bmatrix}, \tag{5.27}$$

which should be correlated with Figure 5.5. The notation makes clear that there are 6 DOFs on the triangle.

5.4.1 Strains from displacements

First we will develop some understanding of the actions of the strain-displacement matrices. We will consider a simple example of a single finite element shown in Figure 5.6. The locations of the nodes and the definition of the connectivity of the triangle are

Python
- script
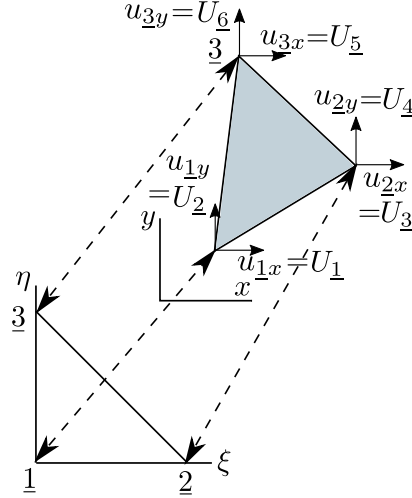
Fig. 5.5. Element degrees of freedom for the three node triangle (T3)

```
xall= array([[-1, -1/2], [3, 2], [1, 2]]) #Coordinates of the nodes
conn= [1,2,3] # The definition of the element, listing its nodes
```

The gradients of the basis functions are computed precisely as in Chapter 3 in the heat conduction model:

```
gradNpar= array([[-1,-1], [1,0], [0,1]])#Grads of basis fncs wrt param. coords
zconn=array(conn)-1
x=xall[zconn,:] # The coordinates  of the three nodes
J=dot(x.T, gradNpar) # Compute the Jacobian matrix
gradN=dot(gradNpar, linalg.inv(J))
```

which gives for the gradients of the basis functions

```
array([[ 0. , -0.4],
       [ 0.5, -0.4],
       [-0.5,  0.8]])
```
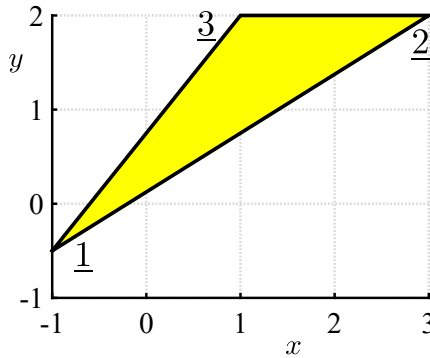

Fig. 5.6. The geometry of the single-element mesh

The general expression (5.20) for the computation of the strains is specialized to the three-node triangle as

$$[\epsilon] = [B_1][u_1] + [B_2][u_2] + [B_3][u_3] \tag{5.28}$$

where the three nodal strain-displacement matrices are calculated from the gradients of the basis functions (see (5.22)) by defining a little helper Python function

```
def Bn(gradNn):
    return array([[gradNn[0], 0],
                  [0, gradNn[1]],
                  [gradNn[1], gradNn[0]]])
```

Here `gradNn` is a 1×2 array of the components of the basis function gradients: `gradNn[0]` means derivative $\partial N_n/\partial x$ and `gradNn[1]` corresponds to derivative $\partial N_n/\partial y$. Let us inspect the strains that are produced by the motion of each of the three nodes individually (one moves, the other two stay fixed). For node $\underline{1}$ we obtain (`gradN[0,:]` is the gradient of the first basis function) using
`B1=Bn(gradN[0,:])`

```
array([[ 0. ,   0. ],
       [ 0. ,  -0.4],
       [-0.4,   0. ]])
```

Note the strains we get when this strain-displacement matrix is multiplied with $u_{\underline{1}x}$, in the first column and $u_{\underline{1}y}$ in the second column:

$$\epsilon_x = 0 , \quad \epsilon_y = -0.4u_{\underline{1}y} , \quad \gamma_{xy} = -0.4u_{\underline{1}x} \tag{5.29}$$

The x displacement of node $\underline{1}$ produces only shear strain, and in particular no strain ϵ_x is produced by the motion of node $\underline{1}$. Similarly, for node $\underline{2}$ we obtain the strain-displacement matrix as
`B2=Bn(gradN[1,:])`

```
array([[ 0.5,   0. ],
       [ 0. ,  -0.4],
       [-0.4,   0.5]])
```

Now we can see that the displacement $u_{\underline{2}x}$ contributes to the strain ϵ_x (just consider that the element side [2,3] changes length when node $\underline{2}$ moves). We get the contribution to the strain within the element from the motion of node $\underline{2}$

$$\epsilon_x = 0.5u_{\underline{2}x} , \quad \epsilon_y = -0.4u_{\underline{2}y} , \quad \gamma_{xy} = -0.4u_{\underline{2}x} + 0.5u_{\underline{2}y} \tag{5.30}$$

And, for node $\underline{3}$

```
array([[-0.5,   0. ],
       [ 0. ,   0.8],
       [ 0.8,  -0.5]])
```

$$\epsilon_x = -0.5u_{\underline{3}x} , \quad \epsilon_y = 0.8u_{\underline{3}y} , \quad \gamma_{xy} = 0.8u_{\underline{3}x} - 0.5u_{\underline{3}y} . \tag{5.31}$$

Therefore (5.28) turns out to be explicitly

$$\begin{bmatrix} \epsilon_x = 0.5u_{\underline{2}x} - 0.5u_{\underline{3}x} \\ \epsilon_y = -0.4u_{\underline{1}y} - 0.4u_{\underline{2}y} + 0.8u_{\underline{3}y} \\ \gamma_{xy} = -0.4u_{\underline{1}x} - 0.4u_{\underline{2}x} + 0.5u_{\underline{2}y} + 0.8u_{\underline{3}x} - 0.5u_{\underline{3}y} \end{bmatrix} \tag{5.32}$$

This is the complete description of how strains are produced for this particular triangular finite element by the motion of all and any of its three nodes.

An important observation should be made here:

Note that the strain components are uniform across the entire triangle. The three-node triangle is therefore usually called the CST (constant strain triangle).

Consider now a special type of displacement: let us take all the x components of the node displacements to be some value, the same at all three nodes, and similarly for the y components: $u_{1x} = u_{2x} = u_{3x} = a$ and $u_{1y} = u_{2y} = u_{2y} = b$. The strain will then become

$$\begin{bmatrix} \epsilon_x = 0.5a - 0.5a = 0 \\ \epsilon_y = -0.4b - 0.4b + 0.8b = 0 \\ \gamma_{xy} = -0.4a - 0.4a + 0.5b + 0.8a - 0.5b = 0 \end{bmatrix} \tag{5.33}$$

This displacement pattern is the so-called rigid-body translation, and the strains are all zero! That is the correct element response, there should be no strain when the element moves as a rigid body.

There is in fact one more displacement pattern where the element moves as a rigid body: rotation. The rotation about some point x_c, y_c can be described by the displacements

$$u_x = -(y - y_c)\psi , \quad u_y = (x - x_c)\psi \tag{5.34}$$

where the scalar ψ stands for the magnitude of the rotation angle. The displacements at the nodes are then obtained by substituting their coordinates to yield

$$\begin{aligned} u_{1x} &= -(y_1 - y_c)\psi , \ u_{2x} = -(y_2 - y_c)\psi , \ u_{3x} = -(y_3 - y_c)\psi , \\ u_{1y} &= (x_1 - x_c)\psi , \ u_{2y} = (x_2 - x_c)\psi , \ u_{3y} = (x_3 - x_c)\psi. \end{aligned} \tag{5.35}$$

The strains are then

$$\begin{bmatrix} \epsilon_x = 0.5[-(y_2 - y_c)\psi] - 0.5[-(y_3 - y_c)\psi] = 0 \\ \epsilon_y = -0.4[(x_1 - x_c)\psi] - 0.4[(x_2 - x_c)\psi] + 0.8[(x_3 - x_c)\psi] = 0 \\ \gamma_{xy} = -0.4[-(y_1 - y_c)\psi] - 0.4[-(y_2 - y_c)\psi] + 0.5[(x_2 - x_c)\psi] \\ \qquad +0.8[-(y_3 - y_c)\psi] - 0.5[(x_3 - x_c)\psi] = 0 \end{bmatrix} \tag{5.36}$$

where ψ can be totally arbitrary.

> The strains for the three node triangle are all zero for the displacement patterns that corresponds to the two rigid body translations and the rigid body rotation.

5.4.2 Stiffness matrix

Now we will develop the stiffness matrix of the three node triangle. The starting point is the first term of (5.23). For a row degree of freedom q (the displacement at node j in the direction r) and column p (the displacement at node i in the direction s), the entry qp of the stiffness matrix reads

$$K_{qp} = \int_S \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}] \right]_{r(q)s(p)} t \mathrm{d}S \tag{5.37}$$

Since the two indexes r, s take on values x, y (which are equivalent to $1, 2$), the matrix

$$\int_S \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}] \right] t \mathrm{d}S \tag{5.38}$$

is of dimension 2×2. As the degrees of freedom of the three-node triangle are organized into a vector

$$[\boldsymbol{u}^{(e)}] = \left[u_{1x}, u_{1y}, u_{2x}, u_{2y}, u_{3x}, u_{3y} \right]^T = \left[U_1, U_2, U_3, U_4, U_5, U_6 \right]^T , \tag{5.39}$$

we realize that the entire stiffness matrix of the triangle consists of the 2×2 submatrices (5.38)

$$\left[\,\boldsymbol{K}^{(e)}\,\right] = \begin{bmatrix} \int_S [\boldsymbol{B_1}]^T [\boldsymbol{D}][\boldsymbol{B_1}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_1}]^T [\boldsymbol{D}][\boldsymbol{B_2}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_1}]^T [\boldsymbol{D}][\boldsymbol{B_3}]\,t\mathrm{d}S \\[2mm] \int_S [\boldsymbol{B_2}]^T [\boldsymbol{D}][\boldsymbol{B_1}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_2}]^T [\boldsymbol{D}][\boldsymbol{B_2}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_2}]^T [\boldsymbol{D}][\boldsymbol{B_3}]\,t\mathrm{d}S \\[2mm] \int_S [\boldsymbol{B_3}]^T [\boldsymbol{D}][\boldsymbol{B_1}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_3}]^T [\boldsymbol{D}][\boldsymbol{B_2}]\,t\mathrm{d}S\,, & \int_S [\boldsymbol{B_3}]^T [\boldsymbol{D}][\boldsymbol{B_3}]\,t\mathrm{d}S \end{bmatrix} \tag{5.40}$$

Even neater expression results from constructing the entire elementwise 3×6 strain-displacement matrix as

$$[\boldsymbol{B}^{(e)}] = \left[\,[\boldsymbol{B_1}]\,,[\boldsymbol{B_2}]\,,[\boldsymbol{B_3}]\,\right] \tag{5.41}$$

as we can then write the elementwise stiffness matrix

$$\left[\,\boldsymbol{K}^{(e)}\,\right] = \int_S [\boldsymbol{B}^{(e)}]^T [\boldsymbol{D}][\boldsymbol{B}^{(e)}]\,t\mathrm{d}S \tag{5.42}$$

Now, if the thickness within the triangle is the same everywhere (as we assumed at the outset), then because all the matrices are constants, we have

$$\left[\,\boldsymbol{K}^{(e)}\,\right] = [\boldsymbol{B}^{(e)}]^T [\boldsymbol{D}][\boldsymbol{B}^{(e)}]\,t\int_S \mathrm{d}S = S^{(e)} t [\boldsymbol{B}^{(e)}]^T [\boldsymbol{D}][\boldsymbol{B}^{(e)}] \tag{5.43}$$

When the elementwise stiffness matrix is multiplied with nodal displacements, we get the elastic restoring forces as entries of the output of the matrix-vector multiplication

$$[\boldsymbol{F}^{(e)}] = \left[\,\boldsymbol{K}^{(e)}\,\right][\boldsymbol{u}^{(e)}] = S^{(e)} t [\boldsymbol{B}^{(e)}]^T [\boldsymbol{D}][\boldsymbol{B}^{(e)}][\boldsymbol{u}^{(e)}] \tag{5.44}$$

where the last expression follows from the substitution of the stiffness matrix (5.43).

For the special displacements of the rigid-body-motion kind discussed in the previous section, i.e.

$$[\boldsymbol{u}^{(e)}] = \begin{bmatrix} a \\ 0 \\ a \\ 0 \\ a \\ 0 \end{bmatrix}\,, \quad [\boldsymbol{u}^{(e)}] = \begin{bmatrix} 0 \\ b \\ 0 \\ b \\ 0 \\ b \end{bmatrix}\,, \quad [\boldsymbol{u}^{(e)}] = \begin{bmatrix} -(y_1 - y_c)\psi \\ (x_1 - x_c)\psi \\ -(y_2 - y_c)\psi \\ (x_2 - x_c)\psi \\ -(y_3 - y_c)\psi \\ (x_3 - x_c)\psi \end{bmatrix} \tag{5.45}$$

we get, since we have shown before that the strains produced by these displacement patterns were all zero, that all the forces vanish

$$[\boldsymbol{F}^{(e)}] = S^{(e)} t [\boldsymbol{B}^{(e)}]^T [\boldsymbol{D}]\underbrace{[\boldsymbol{B}^{(e)}][\boldsymbol{u}^{(e)}]}_{\boldsymbol{0}} = \boldsymbol{0} \tag{5.46}$$

In words: if the displacement vector consists of rigid body displacements, no elastic restoring forces are produced. Since

$$[\boldsymbol{D}][\boldsymbol{B}^{(e)}][\boldsymbol{u}^{(e)}] \tag{5.47}$$

are the stresses, we can see that no elastic restoring forces are produced because there are no stresses which follows from there being no strains.

> For rigid-body displacements there must be no strains, no stresses, no restoring forces. This is necessary for convergence to the correct solution to occur.

5.4.3 Singularity

For three linearly independent vectors $[\boldsymbol{u}^{(e)}]$ of (5.45), the output of (stiffness times displacement equals) forces yields zeros

$$[\boldsymbol{F}^{(e)}] = \left[\,\boldsymbol{K}^{(e)}\,\right][\boldsymbol{u}^{(e)}] = \boldsymbol{0} \tag{5.48}$$

which means that we can write

$$\left[\,\boldsymbol{K}^{(e)}\,\right][\boldsymbol{\phi}_\ell^{(e)}] = \lambda_\ell[\boldsymbol{\phi}_\ell^{(e)}] \quad \text{for } \lambda_\ell = 0 \,, \ell = 1, 2, 3 \,. \tag{5.49}$$

This is an eigenvalue problem for the stiffness matrix, where $[\boldsymbol{\phi}_\ell^{(e)}]$ are the three rigid-body displacement patterns (5.45), and the three corresponding eigenvalues are all equal to zero. Therefore, it follows from matrix theory that the rank of the elementwise stiffness matrix is only 3: its defect is 6-3=3. This is entirely correct, and we would expect the defect to be 3 for all plane-stress elements.

So the stiffness matrix of a single element is singular. Let us contemplate what happens when several elements are assembled: Figure 5.7 shows a sample mesh and the elementwise stiffness matrix of one triangle, both by itself, and also assembled into the global stiffness matrix of the entire structure. Further shown are the locations in the global displacement vector which multiply the entries of the elementwise stiffness matrix *after assembly*. Clearly, if the elementwise matrix was singular (it was!), the global stiffness matrix that has this (and only this) entire elementwise matrix assembled into it is also singular. If the displacement vector $\boldsymbol{u}$ corresponds to a rigid body displacement, the product $\boldsymbol{Ku}$ will be a zero vector. We can easily convince ourselves that if we take an arbitrary number of singular elementwise matrices and assemble them completely into the global stiffness matrix (recall: this is only possible if there are no constraints on the displacements of the nodes), the global stiffness matrix will have at least the defect of 3— it will inherit the properties of a single element stiffness matrix. All the elements connected together can still move as a rigid body. The only way in which we can remove this possibility is by introducing external constraints (supports).
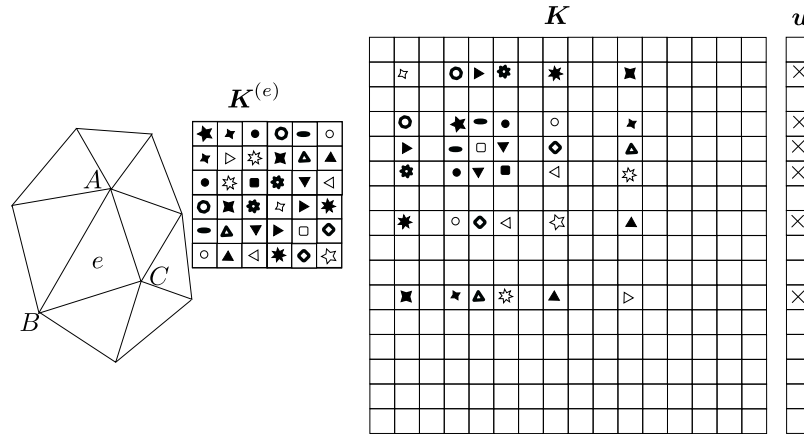
Fig. 5.7. The stiffness matrix of a single element before and after assembly. The displacement vector shows the locations of DOFs that multiply the entries of the elementwise stiffness matrix.

One such possibility is shown in Figure 5.4 where three roller supports were added to the structure consisting of two triangles. Before the supports were introduced, the overall stiffness matrix of such a structure would have been singular with a defect of 3. After the supports were applied the rigid-body displacements became impossible: the structure cannot translate in the x direction (roller at node 3), it cannot translate in the y direction (the two rollers at the nodes 4 and 2), and all the rollers together prevent rotation about any pivot in the plane.

5.5 Quadratic triangle T6

The quadratic triangle T6 makes it possible to design basis functions that can reproduce quadratic variations of the temperature or of the displacement. More precisely, it will do that in terms of the coordinates on the standard triangle. The quadratic triangle has the potential for very significantly improving the representation of stresses. Figure 5.8 shows the representation of the axial (bending) stress in a plane-stress model of a clamped beam with transverse load at the free end. The linear triangle T3 (Abaqus designation CPS3) evidently cannot represent the bending stress with any accuracy (not with this coarse mesh anyway). The quadratic triangle CPS6 with elements of the same size does a much better job.

Abaqus
- CAE file
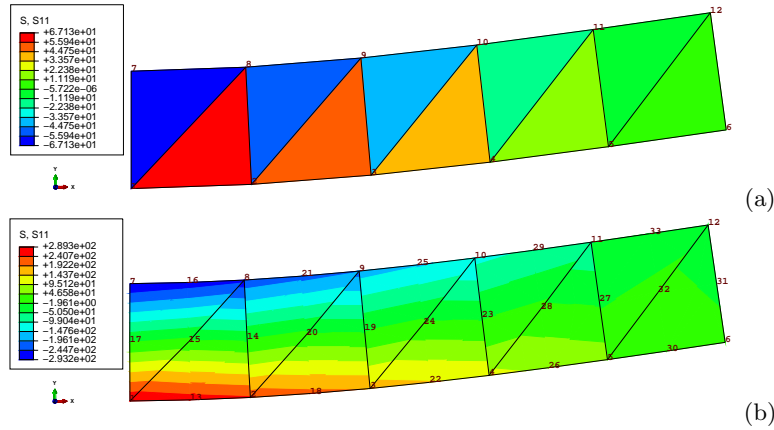
Fig. 5.8. Plane-stress beam model. Axial (bending) stress. (a) Linear triangles (T3). Maximum tensile stress 67.1. (b) Quadratic triangles (T6). Maximum tensile stress 289.3. Elementary beam theory: maximum tensile stress 300.
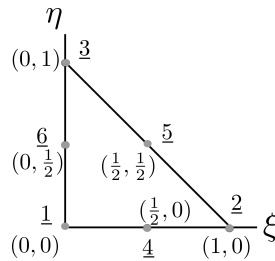
Fig. 5.9. Standard quadratic triangle.

In order to understand how the quadratic triangle works, the first task will be to formulate the basis functions on the standard triangle, Figure 5.9. To write down a polynomial for a particular basis function that is quadratic in ξ, η, six coefficients will be needed. To determine these coefficients, we will make use of the Kronecker delta property (3.31). Let us start with the basis function $N_{\underline{2}} = a_0 + a_1\xi + a_2\eta + a_3\xi\eta + a_4\xi^2 + a_5\eta^2$. Writing

$$N_{\underline{2}}(\xi_{\underline{k}}, \eta_{\underline{k}}) = \delta_{\underline{2k}}, \quad \text{for } \underline{k} = 1, ..., 6 ,$$

at all six nodes (see Table 5.1), provides us with six equations from which the six coefficients may be determined. That is however tedious and there's a quicker way: let us use common sense and guesswork instead. Looking along the η axis we see three ($\underline{1}, \underline{6}, \underline{3}$) and two ($\underline{4}, \underline{5}$) nodes respectively

align at the same ξ coordinate. Evidently, this makes it possible to design the function $N_{\underline{2}}$ as a Lagrange polynomial that is zero in these two locations, and equal to one at node $\underline{2}$

$$N_{\underline{2}} = \frac{(\xi - 0)(\xi - 1/2)}{(1 - 0)(1 - 1/2)} = \xi(2\xi - 1) \ .$$

Similarly, in the other direction we have for $N_{\underline{3}} = \eta(2\eta - 1)$.

The construction of the other basis functions may be systematized if we note that both $N_{\underline{2}}$ and $N_{\underline{3}}$ may be written as the normalized product of planes: for $N_{\underline{2}}$ the two planes are $\widehat{p}_2(\xi, \eta) = \xi$ and $\widetilde{p}_2(\xi, \eta) = \xi - 1/2$, and $N_{\underline{2}}$ is written as

$$N_{\underline{2}} = \frac{\widehat{p}_2(\xi, \eta)\widetilde{p}_2(\xi, \eta)}{\widehat{p}_2(1, 0)\widetilde{p}_2(1, 0)} = \xi(2\xi - 1) \ .$$

Similarly for $N_{\underline{3}}$ and $N_{\underline{1}}$: the recipe is to find two planes that go through three nodes and two nodes respectively (but not through the node at which the function is supposed to be equal to one), and normalize their product. For $N_{\underline{1}}$ the planes are $\widehat{p}_1(\xi, \eta) = 1 - \xi - \eta$ (this is the same N_1 as in (3.30)) and $\widetilde{p}_1(\xi, \eta) = 1 - 2\xi - 2\eta$ (compare with Figure 5.10)

$$N_{\underline{1}} = (1 - \xi - \eta)(1 - 2\xi - 2\eta) \ .$$

Table 5.1. Standard quadratic triangle: locations of the nodes

Coordinate	Node $\underline{1}$	Node $\underline{2}$	Node $\underline{3}$	Node $\underline{4}$	Node $\underline{5}$	Node $\underline{6}$
ξ	0	1	0	1/2	1/2	0
η	0	0	1	0	1/2	1/2

For the mid-edge nodes, $\underline{4}$, $\underline{5}$, $\underline{6}$, we find planes that pass through two triples of nodes. For instance, for node $\underline{6}$ (see Figure 5.10), the two planes are $\widehat{p}_6(\xi, \eta) = 1 - \xi - \eta$ and $\widetilde{p}_6(\xi, \eta) = \eta$ (same as $N_{\underline{3}}$ in (3.29))

$$N_{\underline{6}} = 4(1 - \xi - \eta)\eta \ .$$

The basis functions are summarized as

$$[N] = \begin{bmatrix} (\eta + \xi - 1)(2\eta + 2\xi - 1) \\ \xi(2\xi - 1) \\ \eta(2\eta - 1) \\ -4\xi(\eta + \xi - 1) \\ 4\eta\xi \\ -4\eta(\eta + \xi - 1) \end{bmatrix} \ . \tag{5.50}$$

5.5.1 Basis function gradients and strains

The basis function gradients in the x, y coordinates are obtained from the standard expression (3.62). The first ingredient is the matrix of gradients of the basis functions with respect to ξ, η

$$\left[\mathrm{grad}_{(\xi, \eta)} N \right] = \begin{bmatrix} -3 + 4\xi + 4\eta, & -3 + 4\xi + 4\eta \\ 4\xi - 1, & 0 \\ 0, & 4\eta - 1 \\ 4 - 8\xi - 4\eta, & -4\xi \\ 4\eta, & 4\xi \\ -4\eta, & 4 - 4\xi - 8\eta \end{bmatrix} \ . \tag{5.51}$$
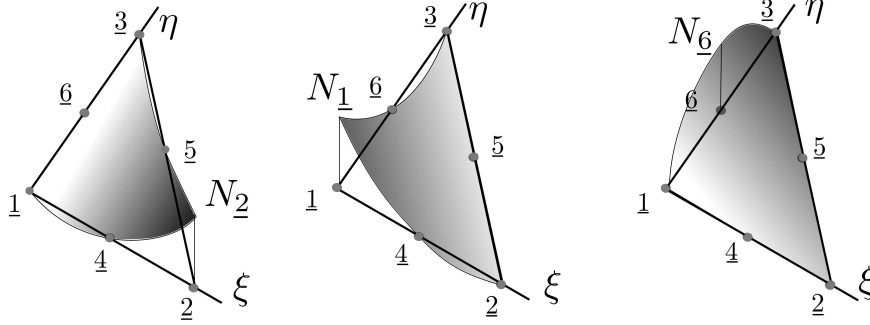
Fig. 5.10. Standard quadratic triangle: Basis functions $N_{\underline{2}}$, $N_{\underline{1}}$, and $N_{\underline{6}}$.

The obvious observation here is that the gradients of the basis functions vary from point to point, and the dependence on the parametric coordinates is linear. Consequently, also the Jacobian matrix will be linear in the parametric coordinates ξ, η as follows from (3.60):

$$[\boldsymbol{J}] = [\boldsymbol{x}]^T \left[\mathrm{grad}_{(\xi,\eta)} N \right] \ . \tag{5.52}$$

Here $[\boldsymbol{x}]$ is a 6×2 matrix of the coordinates of the nodes. However, in the special case of the physical triangle having straight edges and the nodes $\underline{4}, \underline{5}, \underline{6}$ located precisely at the midpoints of their respective edges (Figure 5.11) the Jacobian matrix is constant across the entire element. The formula for the gradients of the basis functions with respect to x, y (3.62) then yields linear variation of the derivatives (and hence the strains!). This is very useful in modeling of bending deformations, as evident from Figure 5.8.
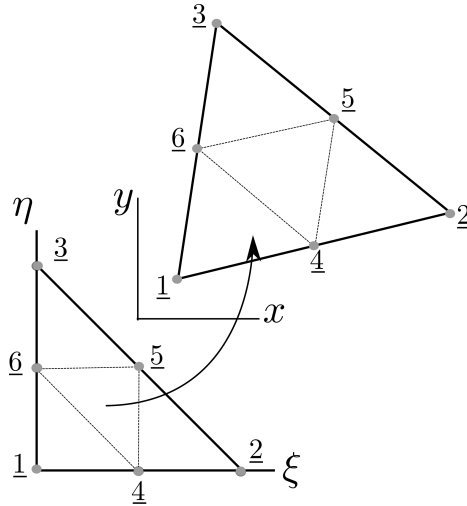
Fig. 5.11. Mapping of the quadratic standard triangle into a straight-sided triangle with the $\underline{4}, \underline{5}, \underline{6}$ nodes being precisely in the middle of the edges

However, when the Jacobian matrix is not constant within an element (as is the case when the edges are straight but the mid-side nodes are not in the middle of the edges, or when the edges are not straight, around holes or fillets), the inverse of the Jacobian matrix is a function of ξ, η, and it multiplies $\left[\mathrm{grad}_{(\xi,\eta)} N \right]$ (also a function of ξ, η) with the result that the derivatives of the basis functions do not vary linearly across the element anymore, and nor do the strains. A vivid example is provided in Figure 5.12: The locations of four nodes had been perturbed by dragging them along the edges. The edges are still straight, but nodes 15, 14, 20, and 19 are no longer at the midpoints

of their respective edges. As a consequence the stress field is now wrong: the maximum for instance occurs away from the expected location at the clamped edge, and the numerical values are also off.

(b)
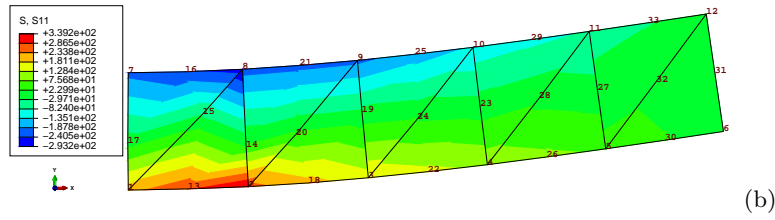
Fig. 5.12. Plane-stress beam model. Axial (bending) stress. Mesh of distorted quadratic triangles (T6)—call for the locations of nodes 15, 14, 20, 19. Maximum tensile stress 339.2. Note that the maximum does not occur in the expected location at the clamped edge. Elementary beam theory: maximum tensile stress 300.

It might seem wise then to avoid triangles with curved edges altogether, but there is a potential trade-off here between the decreased accuracy of curved triangles and an improvement of the ability to approximate curved shape: Figure 5.14 shows the mesh of a part where some of the triangles have curved edges because those edges are used to approximate the geometry of a circle. The ability to approximate curved geometry *sometimes* offsets the decreased accuracy due to the distortion of the triangle. Note that we will call an edge curved even when it appears to be straight. To explain
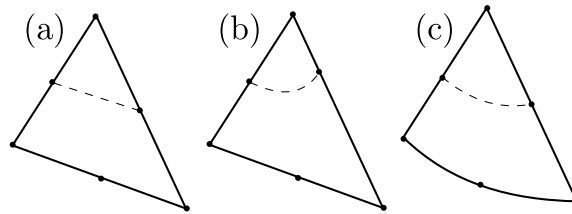
Fig. 5.13. Three six-node triangles. (a) triangle with constant Jacobian, (b) triangle with non-constant Jacobian because of the mid-edge node having been displaced from the mid-point position, (c) triangle with non-constant Jacobian due to an actual curved edge. Dashed line: zero level curve of the basis function at the topmost node.

this paradox, we will refer to the Jacobian along this edge. If the Jacobian is not constant along the edge, if it is a function of where it is computed, we would call the edge curved. Similarly for the six node triangle itself: even when its edges appear straight, if the Jacobian varies from point to point within the triangle, we call that triangle curved. Why? Because irrespectively whether the non-constant Jacobian results from actual curved edges or just from the position of the mid-edge nodes being dislocated from the midpoints of the edges, the non-constant Jacobian will result in degraded accuracy of the element.

5.6 Quadratic curve element L3

The L3 element is needed to provide a mesh on the boundary of the triangulation with quadratic (T6) elements: the boundary of a single quadratic triangle consists of three L3 curve elements.

The L3 element is developed from the L2 element by adding one more node inside the element. Therefore in order for the basis functions to satisfy the condition that they are either zero or one at a node, they will have to become quadratic polynomials. Nevertheless, the basis functions for this element are still the Lagrange interpolation functions on the standard interval, as shown in Figure 5.15. The quadratic Lagrange interpolation polynomial is developed as follows: for instance,
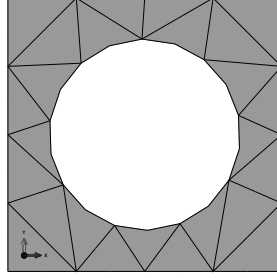
Fig. 5.14. Triangulation of a part that consists of quadratic triangles with curved edges around the circumference of the hole

$N_{\underline{1}}$ is sought as a product of two linear polynomials which each vanish at the location of the remaining nodes

$$(\xi - 0)(\xi - 1) .$$

This expression is then normalized to become $+1$ at $\xi = -1$. Thus the basis functions $N_{\underline{1}}$, $N_{\underline{2}}$, and $N_{\underline{3}}$ on the standard interval read

$$N_{\underline{1}}(\xi) = \frac{\xi(\xi - 1)}{2} , \quad N_{\underline{2}}(\xi) = \frac{\xi(\xi + 1)}{2} , \quad N_{\underline{3}}(\xi) = (1 - \xi^2) . \tag{5.53}$$
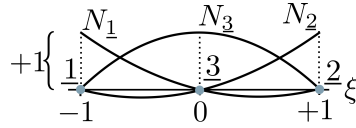
Fig. 5.15. Quadratic basis functions on the standard interval

5.6.1 Distributed traction on the boundary

Let us consider traction load uniformly distributed along the straight L3 element with the third node precisely at the midpoint of the edge. The elementwise mechanical load vector components for the quadratic L3 finite element are due to the last term in (5.23)

$$\int_{C_{t,r(q)}} N_{j(q)} \bar{t}_{r(q)} \, t \mathrm{d}C$$

where $\bar{t}_r$ is the r−component of the traction vector (either $r = x$, or $r = y$), t is the thickness of the plane-stress part, and $C_{t,r(q)}$ is the part of the boundary curve where the traction component is prescribed. For a single element, the degrees of freedom are q where $j(q) = \underline{1}, \underline{2}, \underline{3}$. The length of the L3 element in the x, y coordinates is $h^{(e)}$ and then the integral reduces to

$$\underset{3\times 1}{[L]} = \bar{t}_{r(q)} \, t \int_{L^{(e)}} N_{j(q)} \, \mathrm{d}C$$

where N_j is visualized in Figure 5.15. The areas under the quadratic curves are

$$\int_{L^{(e)}} N_1 \, \mathrm{d}C = \frac{1}{6} h^{(e)} , \quad \int_{L^{(e)}} N_2 \, \mathrm{d}C = \frac{1}{6} h^{(e)} , \quad \int_{L^{(e)}} N_3 \, \mathrm{d}C = \frac{4}{6} h^{(e)} , \tag{5.54}$$
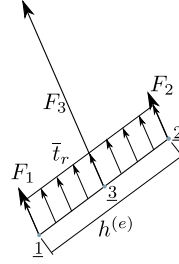
Fig. 5.16. Calculation of nodal forces from uniform distributed traction along an L3 element

This process is called force lumping, because the distributed load is lumped into concentrated forces acting at the nodes. The result is visualized in Figure 5.16: Clearly the lumping is rather unequal, as the middle node attracts two thirds of the total loading on the element.

The calculation will require numerical integration in more complicated cases, for instance when the edge is curved or when the direction or magnitude of the traction loading varies along the edge. See Box 12

5.7 Case study: applying boundary conditions

We will consider the structure shown in Figure 5.17. It is made of a relatively thin sheet ($t = 1.0$mm), and the material properties are $E = 10^7$MPa, $\nu = 0.3$. It is loaded by pressure on the surface of the circular hole in the plane of the sheet. The goal is to compute the maximum principal tensile stress.

The boundary conditions are entirely given in terms of tractions. For the interior surface (hole) we select a cylindrical coordinate system. In this cylindrical coordinate system, the radial component is prescribed as outward pressure $p = 1.0$MPa, and the shear component is zero. On the outer four straight edges we can use Cartesian coordinates, and the two components of the traction are both zero. Quite appropriately, we say that the structure is free-floating. The loading is entirely balanced by itself, so a statically admissible solution exists.
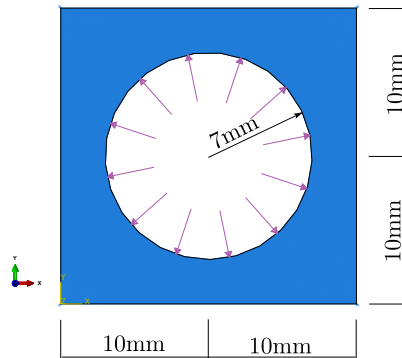
Fig. 5.17. Sketch of the plane-stress structure. The thickness is 1 mm.

5.7.1 Free-floating structure

The structure is first analyzed as specified: only the pressure loading is applied on the boundary. The displaced shape with the color-coded maximum principal stress is shown in Figure 5.18(a). The unexpected thing is the deformed shape which incorporates an apparently random rotation: the shape is symmetric and so is the loading, yet the displacement is not symmetric. Figure 5.18(b)

illustrates the pattern of the displacement by color coding the magnitude. The displacement is not particularly easy to interpret.

We can observe that Abaqus does not report an error or a warning in the Monitor tool. We have to look for the reasons for this behavior in the message file: three warnings are reported of "numerical singularity". Indeed as discussed in Section 5.4.3 the overall stiffness matrix of the structure is singular, with a defect of 3.
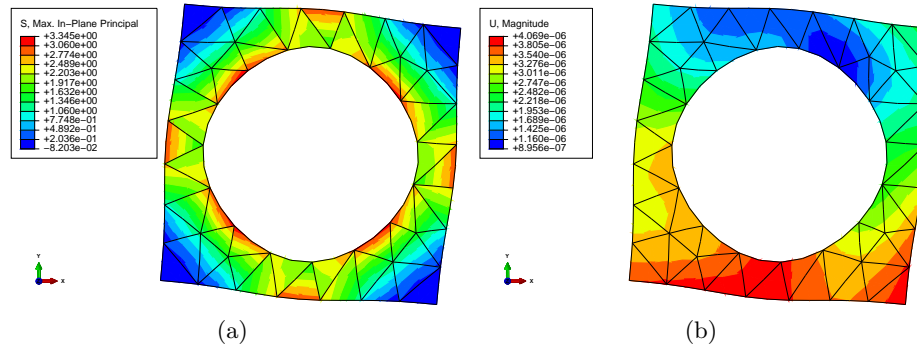
Fig. 5.18. Square with circular hole. Plane stress model. Full geometry without supports. (a) Largest principal stress. (b) Displacement magnitude.

5.7.2 Point supports

As the first pass at remedying the situation we will apply point supports to convert three degrees of freedom from "free" to "fixed". In Figure 5.19(a) this was done by setting the displacement in the x and y at the lower left corner to zero and by setting the y displacement at the lower right corner to zero. These supports are indicated by the orange triangles (projections of cones) in Figure 5.19(a). The plot of the largest principal stress shows the same results as in Figure 5.18(a), but after the supports were applied, the warning messages are gone, and the displacements are easier to interpret.

How good is the estimate of the maximum principal stress? Figure 5.19 (b) displays the largest principal stress without the so-called nodal averaging. The primary quantity—the displacement, in stress analysis, or the temperature, in heat conduction analysis—is guaranteed to be continuous from element the element. Nodal averaging makes quantities which are normally discontinuous from element to element, such as strains, stresses, heat flux, continuous by computing the value at the node (which is shared by several elements) by taking the element values and forming some sort of weighted average to represent the quantity at the node. The result is a visually pleasing picture, which usually conveys an overly optimistic view of the quality of the results. Figure 5.19(b) should be compared with Figure 5.20. The latter presents the stresses as continuous: that is flatly contradicted by the former figure, which is a lot more honest about the actual quality of the results.

> Useful rule of thumb: display the stresses without nodal averaging. If the discontinuity of the stresses across inter-element interfaces can be easily seen in the color-coded plots, the mesh should be refined.

Since the geometry and the loading have two planes of symmetry, the vertical and the horizontal plane passing through the center of the circle, an even better strategy than point supports at arbitrary locations would be to apply supports at the planes of symmetry. Consider an even function, such as $\cos x$: the slope of the curve is zero at $x = 0$: the even function is symmetric with respect to the plane $x = 0$. The displacements parallel to the plane of symmetry behave like this even function, and therefore right angles at the plane of symmetry before deformation remain right angles after
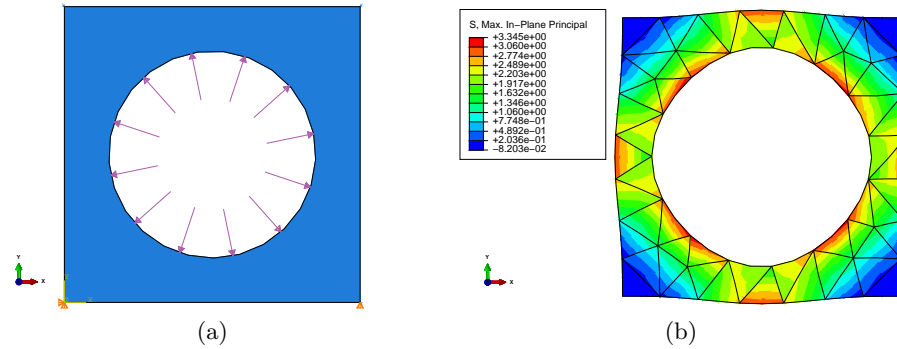
Fig. 5.19. Square with circular hole. Plane stress model. (a) Full geometry with supports at the corners of the bottom edge: x, y support at the lower left corner, y support at the lower right corner. (b) Largest principal stress (maximum 3.345MPa).
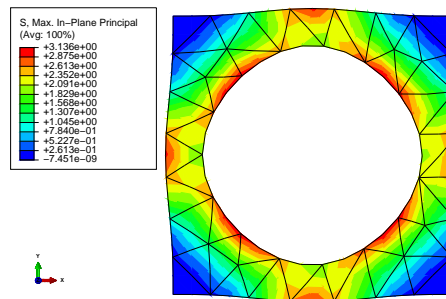
Fig. 5.20. Square with circular hole. Plane stress model. Full geometry with supports at the corners of the bottom edge. (b) Largest principal stress with *nodal averaging* (maximum 3.136MPa).

deformation: there is no shear along the plane of symmetry. At the same time, the displacement component normal to the plane of symmetry behaves like an odd function, it becomes zero on the plane of symmetry. Therefore a straightforward observation is that on the plane of symmetry we know that the shear tractions will be zero, and the normal component of displacement will be zero. Since traction boundary conditions with zero value of traction need not be prescribed, all we have to do is suppress the normal displacement component for any node that lies on the plane of symmetry. (For the full development of symmetry and anti-symmetry conditions refer to a detailed discussion: See Box 25)

In Figure 5.21 we are taking advantage of symmetry by partitioning the edges of the square, and by applying symmetry boundary conditions at these four points: x support at the midpoints of the horizontal edges, y support at the midpoints of the vertical edges.

Note well that the point supports lead to zero associated reactions: If the reactions were not zero, we would have changed the original assignment, where the only applied forces are in the interior of the circular hole and that load balances itself out entirely.

5.7.3 Free vibration (frequency) analysis

It is often very useful to perform the so-called frequency analysis in order to determine whether or not the applied boundary conditions are sufficient to remove the possibility of rigid body displacements.

The "frequency" analysis falls into the category of dynamics in the form of undamped harmonic vibration. The dynamic balance equation is modified from the static balance by the addition of the inertial force density

(a)   (b)
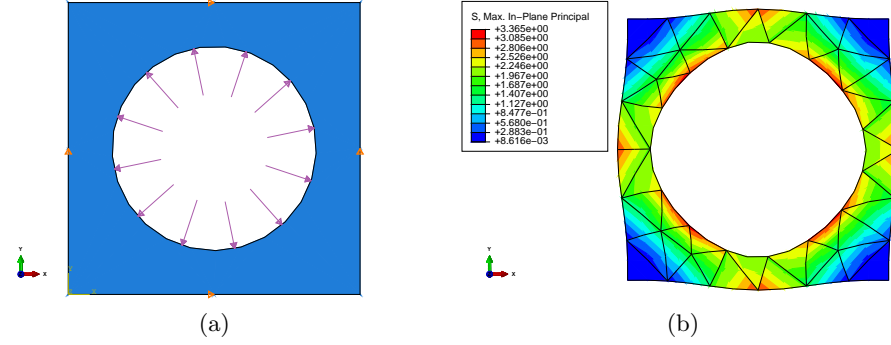
Fig. 5.21. Square with circular hole. Plane stress model. (a) Full geometry with supports at the midpoints of the edges: x support at the midpoints of the horizontal edges, y support at the midpoints of the vertical edges. (b) Largest principal stress (maximum 3.365MPa).

$$[\mathcal{B}]^T [\sigma] + [\overline{b}] = \rho [\ddot{u}] \ . \tag{5.55}$$

Here ρ is the mass density (mass per unit volume), and $[\ddot{u}]$ is the vector of the acceleration.

During free vibration *all external loads* are assumed to be *zero*, so the body load $[\overline{b}] = [\mathbf{0}]$. Furthermore, there are no non-zero applied tractions on the boundary. Finally, all prescribed displacement components are also assumed to be zero (this is referred to as homogeneous essential boundary conditions), and temperature changes are not considered either.

The displacements are assumed to be harmonic (sine, cosine) functions of time

$$[u] = [\widehat{u}] \cos(\omega t) \tag{5.56}$$

so that $[\ddot{u}] = -\omega^2 [\widehat{u}]$. Here ω is the frequency of the harmonic motion, and $\widehat{u}$ is the time-independent vector of displacement amplitudes. Then the dynamic balance equation (5.55) simplifies to

$$[\mathcal{B}]^T [\sigma] = -\omega^2 \rho [\widehat{u}] \ . \tag{5.57}$$

For finite elements in three dimensions, the weighted residual equation for the free-vibration dynamics follows from the dynamic equations of motion (for details: $\boxed{\text{See Box 28}}$): For each free degree of freedom q, i.e. for $1 \leq q \leq N_\mathrm{f}$, such that where $j(q)$ is a node number and $r(q) = x, y, z$ is a direction in the computational coordinate system, there will be one equilibrium (force balance) equation

$$-\omega^2 \sum_{p=1}^{N_\mathrm{f}} \int_V N_{j(q)} \rho N_{i(p)} \ \mathrm{d}V \delta_{r(q)s(p)} \ \widetilde{U}_p + \sum_{p=1}^{N_\mathrm{f}} \int_V \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} \ \mathrm{d}V \ \widetilde{U}_p = 0 \ , \tag{5.58}$$

where the homogeneous essential boundary conditions imply for the fixed degrees of freedom

$$\widetilde{U}_p = 0, \text{ where node } i(p) \text{ is on } S_{u,s(p)} \quad \text{and } s = x \text{ or } y \text{ or } z \ . \tag{5.59}$$

These equations represent harmonic balance equations for the mass of the structure lumped to the nodes and the elastic restoring forces lumped into springs that connect the nodes. The first term represents the harmonic inertial forces, the second term represents the elastic restoring forces.

The above is written in terms of components. The matrix notation allows us to write the tidy expression

$$-\omega^2 [\boldsymbol{M}][\widetilde{\boldsymbol{U}}] + [\boldsymbol{K}][\widetilde{\boldsymbol{U}}] = [\mathbf{0}] \ , \tag{5.60}$$

where $[\boldsymbol{M}]$ and $[\boldsymbol{K}]$ are square $N_\mathrm{f} \times N_\mathrm{f}$ matrices. Equation (5.60) is the **generalized eigenvalue problem**. The column matrix $[\widetilde{\boldsymbol{U}}]$ collects the free degrees of freedom for the entire structure. The

solution is sought as the pair ω_k and $[\widetilde{U}] = [\phi_k]$, where ω_k is the $k-$th circular frequency of free vibration (also called angular frequency), and $[\phi_k]$ is the $k-$th eigenmode (also called normal mode, or free vibration shape). By the way, the natural frequency in cycles per second (Hz) is calculated from the angular frequency as

$$f_k = \frac{\omega_k}{2\pi} \ . \tag{5.61}$$

Note that the frequencies are typically ordered by magnitude, from the smallest to the largest. The lowest-magnitude natural frequency is called the fundamental frequency.

Now, what makes this useful for us here is the fact that if any rigid body displacements are possible, at least one of the angular frequencies $\omega_k = 0$. A really simple example can convince us of that: Consider a single degree of freedom oscillator, for instance the carriage in Figure 5.22.
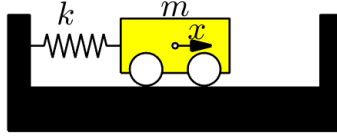
Fig. 5.22. Single degree of freedom oscillator. The carriage can move horizontally without friction. The stiffness of the spring k and the mass of the carriage m determine the angular frequency of vibration.

If the carriage moves to the right, the spring will pull it back. If the carriage moves to the left, the spring will push it back. This can set up oscillation, with the frequency

$$\omega_1 = \sqrt{\frac{k}{m}} \ . \tag{5.62}$$

This is what we find in elementary vibration textbooks. But we can transform it like this

$$\omega_1^2 = \frac{k}{m} \quad \Rightarrow \quad \omega_1^2 m = k \quad \Rightarrow \quad -\omega_1^2 m \widetilde{x} + k \widetilde{x} = 0 \ , \tag{5.63}$$

which now looks like Equation (5.60). Here $\widetilde{x}$ is the amplitude of the motion, which is totally arbitrary (except that $\widetilde{x} = 0$ is not allowed).

We can clearly deduce that when the spring is so weak that $k = 0$ (i.e. when the spring is absent), the natural frequency must be zero, $\omega_1 = 0$. This motion is called rigid body displacement: no forces are involved, because there is no deformation (recall that the stretch of the gap between the support and the carriage doesn't play a role, because the spring is absent).

The same observation may be made for a multiple-dof vibrating system: when we find from Equation (5.60) that one or more $\omega_k = 0$, the corresponding $[\widetilde{U}] = [\phi_k]$ describes motion of the system which does not produce any force because there is no deformation (viz. "rigid body motion").

How do we use this? When we are not sure whether or not the structure is sufficiently supported to prevent rigid body motion, we set up and run the "Frequency" analysis. If any of the frequencies are zero, the corresponding displacement will show us how the structure moves as a rigid body. We then simply have to prevent this from happening by adding supports. A note on numerical accuracy: solving the generalized eigenvalue problem Equation (5.60) means iteration (it is a nonlinear problem, where the only approach is iteration). The iteration needs to be stopped at some point, and the frequencies are not computed exactly, but with some small numerical error. Therefore, the frequencies that we expected to be zero, may be produced as small-magnitude nonzero numbers. We actually know that for symmetric stiffness and mass matrices, the angular natural frequencies need to come out as positive (or zero, when rigid body displacements are possible). Numerical inaccuracies may cause some of the zero frequencies to become very small *negative* numbers. This is nothing to worry about: Consider small-magnitude natural frequencies to be practically zero, irrespectively of sign.

5.7.4 Reduction using symmetry

The observed symmetry can be used not only to place supports on the symmetry plane, but also to physically reduce the structure. For instance, if we consider two planes of symmetry, horizontal, and vertical, we can model only a quarter of the entire square plate. On the edges where the cuts separate the modeled quarter from the rest, the symmetry boundary conditions are employed: compare with Figure 5.23(a).

(a)   (b)
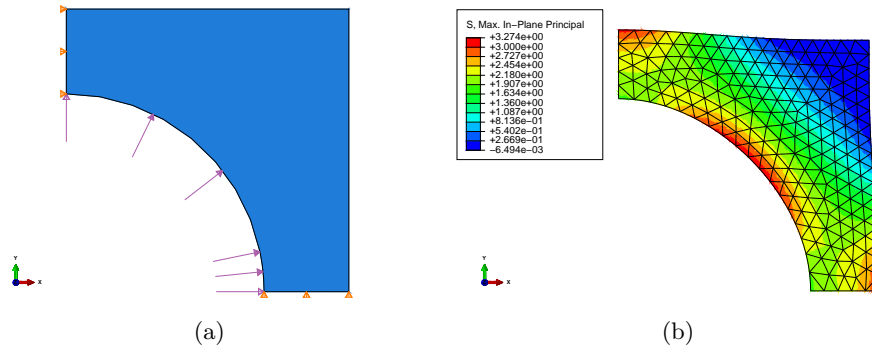
Fig. 5.23. Square with circular hole. Plane stress model. (a) Quarter geometry with supports at the edges on the planes of symmetry: x support at the vertical edge, y support at the horizontal edge. (b) Largest principal stress (maximum 3.274MPa).

Note well: even though the boundary conditions in terms of zero tractions are not applied explicitly, they are still enforced (albeit approximately). Figure 5.24 displays the shear stress σ_{xy}, which should be zero along the square sides because of the traction-free boundary conditions, and also along the symmetry planes because of the no-shear boundary conditions. In the present case, zero is color-coded as red. The values on the boundary are close to zero, but not precisely zero: the boundary condition is only approximated.
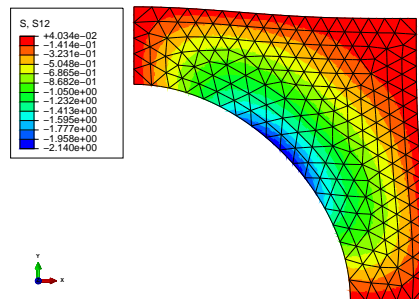
Fig. 5.24. Square with circular hole. Plane stress model. Quarter geometry with supports at the edges on the planes of symmetry: shear stress σ_{xy}

The vertical and horizontal symmetry planes do not exhaust the possibilities. There are also two symmetry planes at $\pm 45^o$. With all four symmetry planes taken into consideration we can reduce the original square to 1/8th as shown in Figure 5.25. The boundary condition on the inclined symmetry plane needs a custom coordinate system aligned with the symmetry plane (the Datum CSYS shown in green in Figure 5.25(a)). The displacement component orthogonal to the symmetry plane then needs to be set equal to zero, and the shear traction component along the symmetry plane is again zero. One advantage of modeling only a fraction of the entire geometry is that we can afford more resolution: the number of finite elements in Figures 5.23 and 5.25 is roughly the same, which means

that in the 1/8th model we get better representation of the solution for the same cost. Compared to the full model we are saving 7/8 of the solution effort.

(a)   (b)
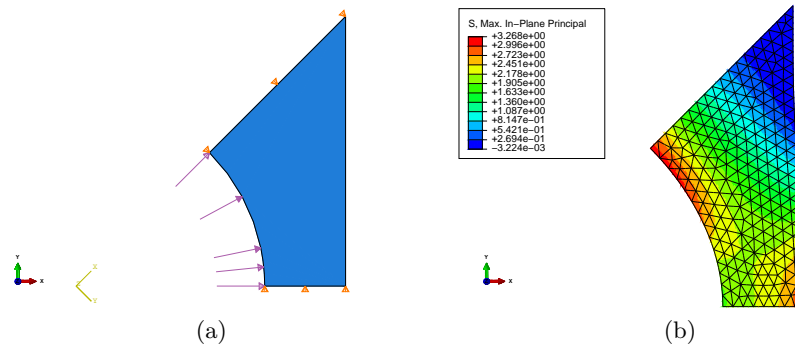
Fig. 5.25. Square with circular hole. Plane stress model. (a) 1/8 geometry with supports at the edges on the planes of symmetry: x support at the vertical edge, y support at the horizontal edge. (b) Largest principal stress (maximum 3.268MPa).

5.7.5 Graded mesh

So what *is* the maximum of the largest principal stress? We calculated four different numbers above, which one, if any, is the correct one? Generally speaking, the finer the mesh the better our chances of getting an answer close to the correct one. The finest mesh used was for the 1/8th model. All the meshes were uniform, meaning the elements were the same size everywhere (more or less). We can use the computing resources much more efficiently if we direct the FE program to put small elements to where the stress is changing the fastest. The detailed discussion of the phenomena and the proper procedures for controlling the error are the subject of Chapter 6.

The seeding of the geometry edges can be used to put much smaller elements around the red spots (where the largest principal stress has a strong gradient) as shown in Figure 5.26. Even though we are still displaying the stress without nodal averaging, we can no longer see strong discontinuities where they matter (around the red spots); because we put large elements in the top right corner, there are some stress discontinuities there, but the stress is practically zero in that area. This is our best solution so far. Without a serious convergence study we still don't know how far off the estimate in Figure 5.26 is, but at least we can see that the numbers are not changing very much anymore (3.234MPa is only around 1% below 3.268MPa).

5.8 Boundary conditions of the "Contact" type

Figure 5.27 shows a sketch of an aluminum slab of thickness 3 mm, perforated by four holes. The line marked S shows the plane of symmetry that will be used in the modeling. Perfectly-fitting pins are inserted in the holes labeled A and are forced to move away from each other in a straight line by 0.4 mm total distance by an actuator. In our analysis in this section we will assume the pins are not deformable (i.e. they are rigid), and we also assume that the pins are not fused with the slab, meaning that they can freely turn in their holes, without friction. The goal is to find the maximum of the von Mises stress.

In this section we intend to discuss the implications of choosing different representations of the boundary conditions applied to the interior of the holes labeled A. This will eventually lead to the introduction of the so-called contact boundary condition.
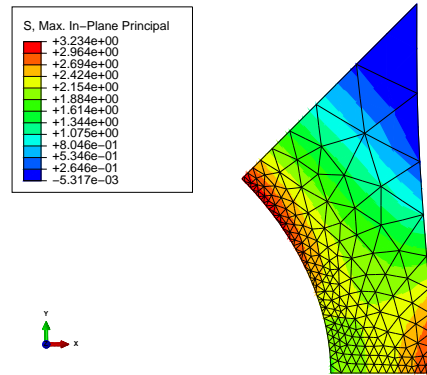
Fig. 5.26. Square with circular hole. Plane stress model. 1/8 geometry with supports at the edges on the planes of symmetry with graded mesh. Largest principal stress (maximum 3.234MPa).
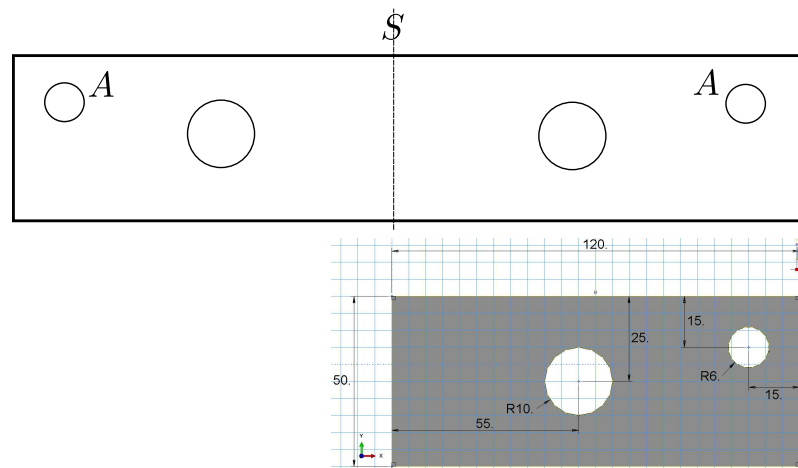
Fig. 5.27. Aluminum slab with circular holes. Pins in holes A are forced to move away from each other by a given distance.

5.8.1 Prescribed displacement of the circumference of the hole A

The goal is to simulate the effect of these pins moving away from each other, which in turn means that the slab needs to stretch to accommodate the motion. The final state is determined from the considerations of equilibrium: the pin exerts a force on the slab and vice versa. This force needs to be large enough to stretch the slab sufficiently far so that the pin can move by a given amount and still pass through the hole in the slab. This problem can be thought of in terms of reactions at supports due to prescribed displacement of the supports, which motivates the first modeling approach: we can assume that the points of the surface of the holes move in unison by the same amount as the center of the pin. In this way the circular pin will certainly fit into the hole after it moves and after the slab deforms.

This condition is implemented as prescribed displacement: along the circle that defines hole A we prescribe in the global Cartesian coordinate system the given nonzero displacement in the X direction and simultaneously zero displacement in the Y direction. The results are shown in Figure 5.28. We can see the color mapping of the von Mises stress (averaged at the nodes), and the deformed shape of the slab (with a $40\times$ magnification of the displacements) is overlaid with an outline of the shape of the slab before deformation. Upon closer inspection we can detect a problem: the surface of the hole A does not rotate, while the upward-bent shape of the slab indicates that the right-hand side end of the slab should rotate clockwise (which of course includes the neighborhood of the hole). In reality that is precisely what the boundary condition we prescribed was expected to do: fixing the

Abaqus
- CAE file
- tutorial

displacements of all the points along the circumference of the circle simply translated the circle as a rigid body, without rotation. That is okay to assume for the pin, but not for the slab. As stated above, the pin can freely rotate without friction in its hole, which also means that the slab should be able to freely rotate around the pin. The prescribed displacement boundary condition in effect assumes that neither the pin nor the slab are allowed to rotate, which is a contradiction to the situation we wish to model.
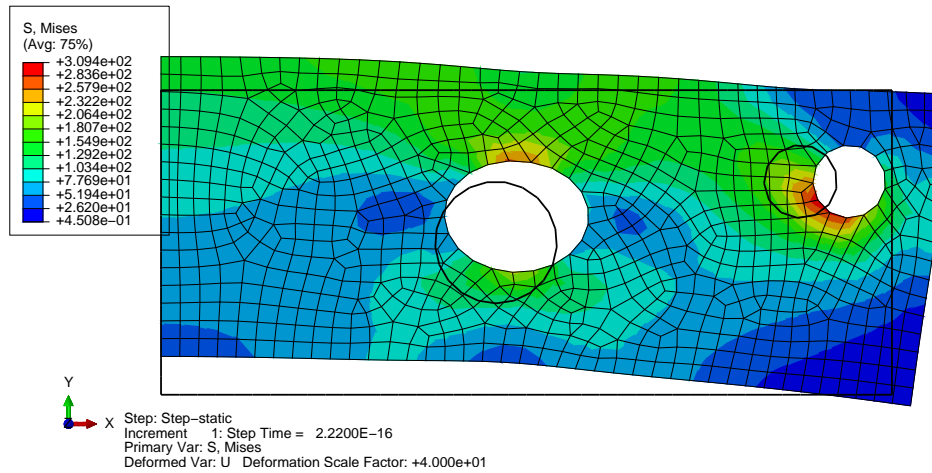
Fig. 5.28. Aluminum slab with circular holes. Model that prescribes the displacement of the surface of the hole.

5.8.2 Rigid-body constraint applied to the hole A

The observations of the previous section motivate the model here: we will assume that the surface of the hole A can move as a rigid body, and we will prescribe only the motion of the **reference point** of the rigid body (in our case that is naturally the center of the circle). In the global Cartesian coordinate system we will prescribe for the reference point the given nonzero displacement in the X direction and zero displacement in the Y direction, and we will *not* prescribe rotation of the rigid body so that the slab can freely rotate around the pin. Figure 5.29 shows the von Mises

Abaqus
- CAE file
- tutorial
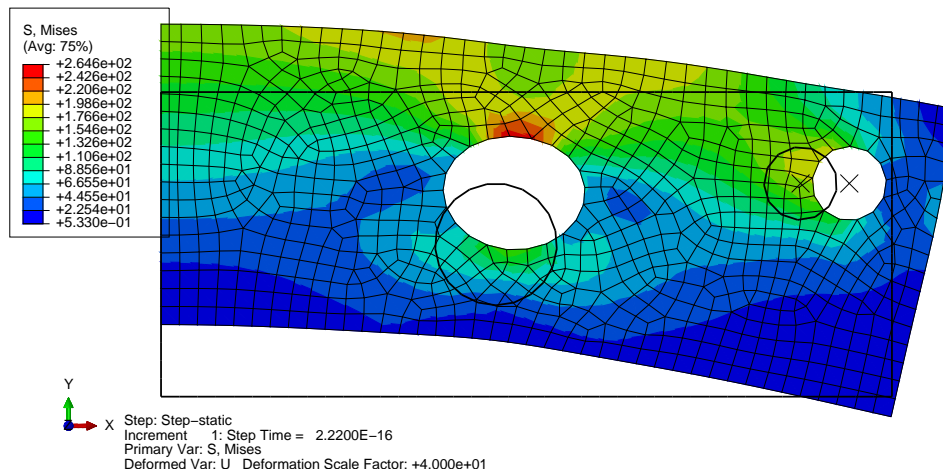
Fig. 5.29. Aluminum slab with circular holes. Model that ties the surface of the hole into a rigid body.

stress displayed on the deformed shape, and the shape before deformation is again shown as outline. Comparing with Figure 5.28 we can clearly see that the maximum of one Mises stress moved from the hole A to the circumference of the large hole, and the magnitude of the bending deformation visibly increased, which is consistent with the relaxed rotation constraint.

5.8.3 Modeling the contact of the hole A with the pin

The model discussed in Section 5.8.2 is not completely realistic in that the pin not only pushes against the material of the slab, but it also pulls on it. But, if the pin is to freely rotate inside the hole A, it is not likely to be able to pull on the material of the slab. Rather, if the pin moves away from the edge of the hole, it should cleanly separate, opening a gap between the slab and the pin.

The difficulty is that we don't know beforehand precisely where the pin will push against the hole and where it will pull away from the surface of the hole. Therefore, when we propose to apply this condition in terms of the tractions acting on the circle A, we have to admit that we don't know on which part of the circle we could prescribe zero tractions (that would be the surface of the hole exposed by the pin moving away from it), or nonzero tractions (on the part of the surface where the pin pushes against it). Moreover, we don't even know the magnitude of the tractions: all of this needs to be found by solving for the equilibrium of the system slab+pin. This is a chicken and egg problem: we can't find the equilibrium until we specify the boundary conditions, and we can't specify the boundary conditions until we know the equilibrium. This leads to a nonlinear model (as opposed to linear models discussed in this book) which needs to be solved by iteration. This type of situation is usually referred to as a ***contact problem***. Figure 5.30 shows the solution of the contact problem. The deformed shape shows clearly where the pin pushes against the slab and where it pulls away from it. The hole remains circular where the pin is in contact with the slab, but deforms into a non-circular shape where the pin and the slab are separated. The maximum of the von Mises stress moved back to the hole A, but is now situated next to the part of the surface where the pin-slab contact occurs.

Abaqus
- CAE file
- tutorial
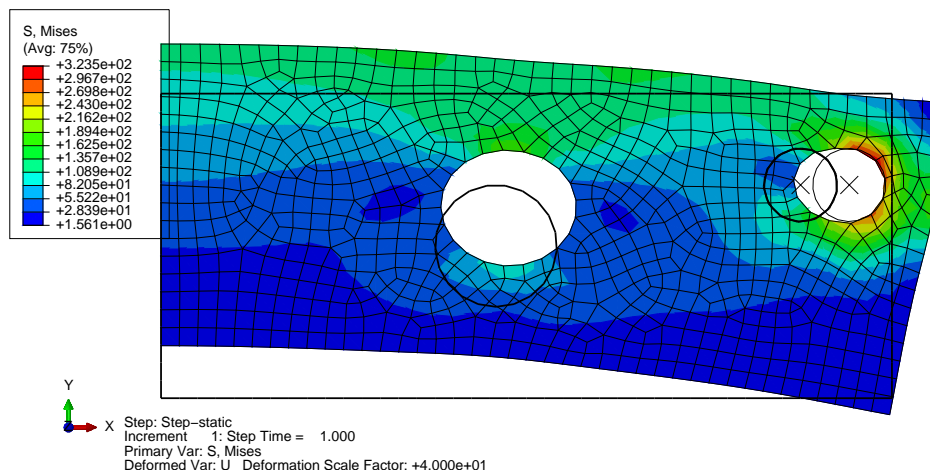
Fig. 5.30. Aluminum slab with circular holes. Model that incorporates contact between a rigid pin and the slab.

5.8.4 Summary

By progressively adding sophistication to our modeling we are able to capture more and more of the correct physical behavior. The cost is modeling complexity: we need to construct more and more complicated models. At some point we usually decide that we have reached a balance between

the amount of modeling difficulties and the payoff in terms of fidelity of the model, and we accept the solution as a good approximation. Consider the case of the slab discussed in this section: we could move on to more complicated models. The geometry could be considered changing during the solution (the so-called *nonlinear geometry* option), the contact interaction may involve not only pressure response along the contact surface, but also shear forces due to friction, and in fact we could also introduce a pin that deforms and is not rigid. We shall not do that here, but the reader is invited to construct these more complicated models and compare with the results shown above. If the differences are found to be negligible, the model presented in Section 5.8.3 will be good enough; otherwise that model may be deemed not acceptable, and the more complicated models would be needed.

5.9 Thermal loading

Often the material from which a structure is built up reacts to its environment by deformation. The material experiences the environment in possibly different ways or different measures in different locations, and stresses are produced.

Think about a very small piece of material that is exposed to the environment. We can assume that the resultant relative deformation is homogeneous and no stress is produced. For the sake of this argument, let us consider one particular environmental effect: **thermal expansion**. However, similar effects may be produced by shrinkage, swelling, and so on.

When the small sample of material is at a *reference temperature* it is unstressed, and we define its displacements and the associated strains to be zero in this state: the *reference state*. Then the temperature is increased by ΔT, and the material responds by displacement, and because by assumption the deformation is homogeneous, the entire sample experiences uniform strains. Based on experimental evidence, the following model is adopted for isotropic materials to describe the so-called thermal strains

$$[^{\theta}\boldsymbol{\epsilon}] = \begin{bmatrix} ^{\theta}\epsilon_x \\ ^{\theta}\epsilon_y \\ ^{\theta}\gamma_{xy} \end{bmatrix} = \Delta T \alpha \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} . \tag{5.64}$$

Evidently, the thermal expansion is assumed not to cause any shear strains. The factor α is the so-called **coefficient of thermal expansion** (CTE), and ΔT is the change of temperature from the reference value.

Therefore, we have a modification of the constitutive equation (5.6): the stress is produced by the net elastic strains (total strains from which thermal strains are subtracted)

$$[\boldsymbol{\sigma}] = [\boldsymbol{D}] \left([\boldsymbol{\epsilon}] - [^{\theta}\boldsymbol{\epsilon}] \right) , \tag{5.65}$$

where $[\boldsymbol{\epsilon}]$ is the total strain, and $[\boldsymbol{\epsilon}] - [^{\theta}\boldsymbol{\epsilon}]$ is the elastic strain. The total strain is related to the displacement via the strain-displacement relation (5.107), and thus we may write

$$[\boldsymbol{\sigma}] = [\boldsymbol{D}] \left([\mathcal{B}\boldsymbol{u}] - [^{\theta}\boldsymbol{\epsilon}] \right) .$$

The weighted residual equation (5.23) then picks up one more term, the thermal loading: For all DOFs $1 \le q \le N_{\mathrm{f}}$, require

$$\sum_{p=1}^{N} \int_S \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}] [\boldsymbol{B}_{i(p)}] \right]_{r(q)s(p)} t \mathrm{d}S \, U_p$$

$$- \int_S N_{j(q)} [\overline{\boldsymbol{b}}]_{r(q)} \, t \mathrm{d}S - \int_{C_{t,r}} N_{j(q)} \overline{t}_{r(q)} \, t \mathrm{d}C \tag{5.66}$$

$$- \int_S \left[[\boldsymbol{B}_{j(q)}]^T \boldsymbol{D} \, [^{\theta}\boldsymbol{\epsilon}] \right]_{r(q)} t \mathrm{d}S = 0 ,$$

where the essential boundary conditions imply for the fixed degrees of freedom

$$U_p = \overline{U}_p \quad , \text{ where node } i(p) \text{ is on } C_{u,s(p)} \quad \text{and } s = x \text{ or } y . \tag{5.67}$$

5.9.1 Thermal loads example

Consider again the three-node triangle from Section 5.4.1 (call for Figure 5.6). For this geometry, the thermal loads in terms of forces acting at nodes of the triangle are calculated by a few lines of Python: the strain-displacement matrices were calculated in Section 5.4.1, so now we add the definition of the variables and the elastic material stiffness matrix (page 192)

Python
- script

```
35 DeltaT, CTE, E, nu, t  = symbols('DeltaT CTE E nu t')
36 D = E/(1-nu**2)*Matrix([[1, nu, 0], [nu, 1, 0], [0, 0, (1-nu)/2]])
```

then we define the thermal strain

```
37 eth = DeltaT*CTE*Matrix([1, 1, 0]).reshape(3, 1)
```

and for instance for node 1 we calculate the nodal forces as

```
40 F1 = simplify(Se*t*B1.T*sig)
```

which yields

```
Matrix([
[                             0],
[1.0*CTE*DeltaT*E*t/(nu - 1)]])
```

Cleaning up the expressions for the nodal forces for all three nodes results in

$$\begin{bmatrix} {}^{\theta}F_{1x} \\ {}^{\theta}F_{1y} \end{bmatrix} = \frac{\alpha \Delta T E t}{1-\nu} \begin{bmatrix} 0 \\ -1 \end{bmatrix} \ , \quad \begin{bmatrix} {}^{\theta}F_{2x} \\ {}^{\theta}F_{2y} \end{bmatrix} = \frac{\alpha \Delta T E t}{1-\nu} \begin{bmatrix} 5/4 \\ -1 \end{bmatrix} \ , \quad \begin{bmatrix} {}^{\theta}F_{3x} \\ {}^{\theta}F_{3y} \end{bmatrix} = \frac{\alpha \Delta T E t}{1-\nu} \begin{bmatrix} -5/4 \\ 2 \end{bmatrix} \ . \tag{5.68}$$

The forces are also visualized in Figure 5.31 (divided by the factor $(\alpha \Delta T E t/(1-\nu))$). The following
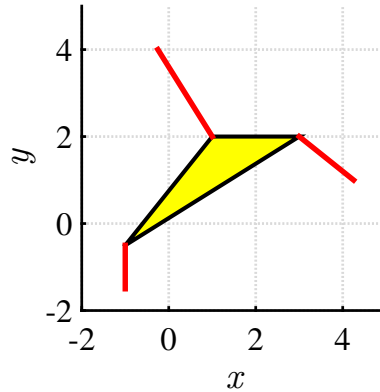
Fig. 5.31. Triangle from Section 5.4.1: thick red lines represent nodal forces corresponding to thermal expansion scaled by $(\alpha \Delta T E t/(1-\nu))$

insights may be derived from the formulas and the picture:

1. The directions of the forces are given by the directions of the opposite edges: the force at a node is orthogonal to the opposite edge and points outwards of the triangle (for positive change in temperature, which means expansion). This corresponds to our intuitive understanding of the change of area in a triangle: we change the area by moving a vertex at right angle to the opposite edge.
2. All the x-components of the forces sum to zero, and all the y-components also sum to zero.

Observation 2 holds for any shape of element. In other words: the thermal loads are self-equilibrated on each finite element.

5.9.2 Thermal strains in a bimetallic assembly

Consider an assembly of two bonded thin metal slabs. The inset is of aluminium, while the outer frame is of steel: see Figure 5.32. The assembly is exposed to an increase of 70°C from the stress-free reference state. Of interest are the deformations produced by the unequal coefficients of thermal expansion. Due to the thinness of the plate, we consider plane stress an adequate approximation. Because of two-way symmetry, only a quarter of the plate is modeled.
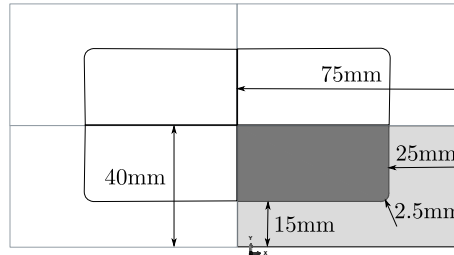
Fig. 5.32. Aluminium-steel plate assembly, plane stress. Thickness 5 mm. Interior: aluminium, exterior frame: steel. The shaded part is modeled.
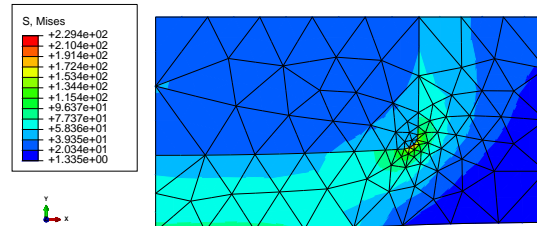
Fig. 5.33. Aluminium-steel plate assembly, plane stress. Von Mises stress. Graded mesh.

Figure 5.33 shows the von Mises stress peaking in the vicinity of the rounded corner at the interface where the two materials meet. We turn our attention to Figure 5.34. The fact that in the present model we have two materials adjacent to each other across a bonded interface gives us an opportunity to discuss the continuity of the stresses and strains. Stress can be converted to a traction vector acting on the surface with a given normal. For instance at the horizontal part of the interface the normal component of this traction vector will be the stress component σ_y. By the principle of action and reaction, this component of the stress vector should be continuous *across* the horizontal material interface. Figure 5.34(a) confirms this: even though the mesh could use some refinement, the part of the interface with normal along y shows reasonable approximation of continuity of σ_y. Strain, on the other hand, will not possess continuity of the same form. Rather, the amount of stretch *along an interface* should be continuous. Thus, along the vertical part of the material interface (with normal to the interface along the x direction) should have the strain component *along* the interface, i.e. ϵ_y, continuous, and Figure 5.34(b) offers a good approximation of such continuity.

5.10 Background, explanations, details

Box 16. Mechanical balance of a deformable body
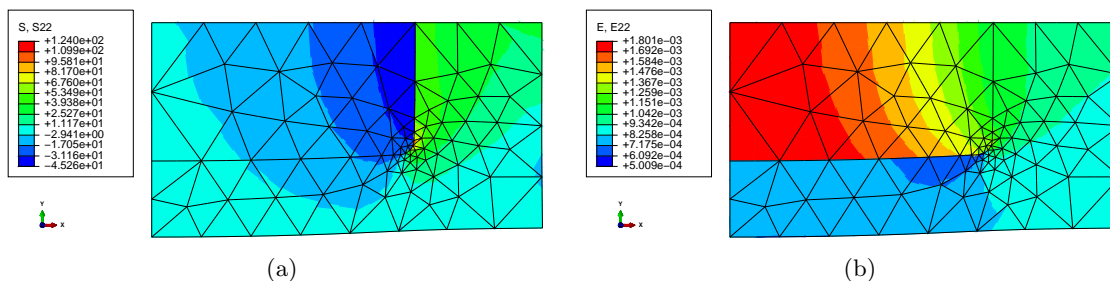
Fig. 5.34. Aluminium-steel plate assembly, plane stress. (a) Stress component σ_y (S22). (b) Strain component ϵ_y (E22)). The x coordinate is horizontal, the y coordinate is vertical.

We can consider a deformable body to be a collection of particles, and apply the Newton's equation of motion of elementary dynamics to each particle, $m\dot{\boldsymbol{v}} = \boldsymbol{F}$, where $\dot{\boldsymbol{v}}$ is the particle acceleration, m is the particle mass, and $\boldsymbol{F}$ is the applied force. The complicating circumstance is that a deformable body can be thought of as a collection (of infinitely many) particles, all interacting through contact. Furthermore, our goal is to formulate a continuum model rather than deal with the discrete collection of particles.

The model that we are going to formulate in this chapter is sometimes referred to as the model of elastodynamics. When inertial effects are not important, i.e. when the accelerations are negligible, the case of elastostatics results. Analyzing either of these is often referred to as ***stress analysis***. This denomination reflects the importance of quantitative assessment of stresses in structures in engineering practice, but in general other quantities are of interest as well: accelerations, velocities, displacements, natural frequencies, and so on.

Balance of linear momentum

Let us consider a body with some distributed force on parts of the boundary (the reactions must be included) and distributed force in the volume (for instance, gravity-induced load). For simplicity, we draw a sketch in two dimensions, but obviously we are thinking of a three-dimensional body; see Fig. 5.35. The distributed force on the boundary is therefore in units force/length^2, and units of the distributed force in the volume are force/length^3. The distributed force on the boundary is customarily called the ***traction***.
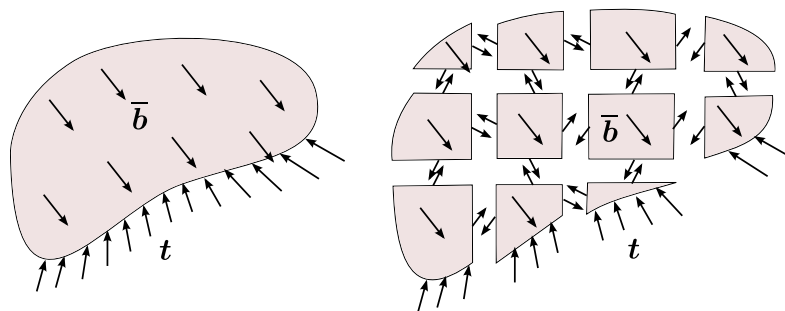
Fig. 5.35. A continuous body with applied distributed force on the boundary, and within the volume (on the left). The same body cut up into many small volumes (particles), with their interaction represented by distributed forces along the cuts (on the right).

The continuous body will be now divided into many very small (infinitesimally small) volumes, which we may consider "particles". The interaction between the particles is mediated by contact forces (tractions) along the cuts between the particles. Assuming we know these forces, the Newton's equation may be applied to each separately. However, we will apply this equation in the form of the

change of **linear momentum**

$$\frac{\mathrm{d}}{\mathrm{d}t}(m\boldsymbol{v}) = \boldsymbol{F} \; ,$$

from which the previous form of the equation of motion may be obtained provided m does not change. In our case, this will be true because each small volume holds a certain amount of material and does not exchange material with any other volume, so the mass of each volume is conserved.

As a consequence of the above, we may write for each small particle volume j the change of its linear momentum

$$\frac{\mathrm{d}}{\mathrm{d}t}(m_j\boldsymbol{v}_j) = \boldsymbol{F}_j \; , \tag{5.69}$$

where we use for the mass of the particle $m_j = \rho V_j$, with V_j the volume of the particle, and ρ the mass density, $\boldsymbol{v}_j$ the velocity, all at some point within the volume of the particle (we are using the mean-value theorem to express integrals over the volume of the particle!). The force $\boldsymbol{F}_j$ includes the body force $\bar{\boldsymbol{b}}$ and the tractions $\boldsymbol{t}$ on the surface of the particle volume

$$\boldsymbol{F}_j = \bar{\boldsymbol{b}} V_j + \int_{S_{\mathrm{int}}} \boldsymbol{t} \mathrm{d}S + \int_{S_{\mathrm{ext}}} \boldsymbol{t} \mathrm{d}S \; , \tag{5.70}$$

where the surface integral is split into two parts (see Fig. 5.36): the interior surfaces S_{int}, where two particle volumes are separated, and the exterior surfaces S_{ext}.
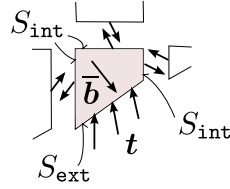
Fig. 5.36. Isolated particle volume.

Now we will collect the contributions of Eq. (5.69) by summing over all the particles

$$\sum_{j=1}^{N} \frac{\mathrm{d}}{\mathrm{d}t}(m_j\boldsymbol{v}_j) = \sum_{j=1}^{N} \boldsymbol{F}_j \; , \tag{5.71}$$

which may be rewritten in the limit of infinitely many particles as integrals

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\mathrm{d}m = \int_V \boldsymbol{b}\mathrm{d}V + \int_{S_{\mathrm{ext}}} \boldsymbol{t}\mathrm{d}S + \sum_{j=1}^{\infty} \int_{S_{\mathrm{int},j}} \boldsymbol{t}\mathrm{d}S \; , \tag{5.72}$$

where the last term (the sum) is over all the shared surfaces that separate the particle volumes. Using Newton's third law of action and reaction, we may conclude that whenever two particle volumes share a piece of their boundary, the traction at the material point A on the surface of particle 1 is equal in magnitude but opposite to the traction at the same material point (the one that has been split by the cut separating the two particles) at the corresponding point A on the surface of particle 2. Since the sum is over all the *pairs* of such surfaces, the last term in Eq. (5.72) cancels, and the final statement of the **balance of linear momentum** of the material in the volume V reads

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\,\mathrm{d}m = \int_V \boldsymbol{b}\,\mathrm{d}V + \int_S \boldsymbol{t}\,\mathrm{d}S \; , \tag{5.73}$$

where m is the total mass of the material inside the volume V, and S is the bounding surface of the volume V. While the surface S and the volume V change with deformation, and hence are time-dependent, the total mass of the material m does not change (the same particles that were inside the volume before deformation are there during the deformation).

Force Distributed on a Surface and Stress

The traction vector $\boldsymbol{t}$ may be written in terms of components in a surface-aligned Cartesian basis as $\boldsymbol{t} = t_n\boldsymbol{n} + t_1\boldsymbol{e}_1 + t_2\boldsymbol{e}_2$, where t_n is the normal component, and t_r are the shear components $(r = 1, 2)$. The Cartesian basis is defined at the given point on the surface by first taking the (outer, unit) surface normal as the third basis vector, and then picking arbitrary orthogonal directions in the tangent plane– see Fig. 5.37. The normal component is extracted as

$$t_n = \boldsymbol{n} \cdot \boldsymbol{t} \ . \tag{5.74}$$

The shear part of the traction $\boldsymbol{t}_s$ is obtained by subtracting the normal part of the traction from the traction vector $\boldsymbol{t}$

$$\boldsymbol{t}_s = \boldsymbol{t} - t_n\boldsymbol{n} \ . \tag{5.75}$$
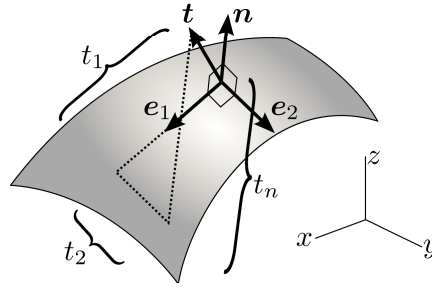
Fig. 5.37. Components of traction.

Next we need to relate the traction on the surface to the deformation of the material just below the surface. The deformation will be measured by strains, and the response of the material to the strains will be related to the tractions on the surface (and any body loads, if present) through the mathematical device of the **stress**.

First, inspect Fig. 5.38: it is possible to define such a Cartesian coordinate system in the vicinity of a given point that the coordinate planes will cut out a (curvilinear) tetrahedron from the solid. Our plan is to make this tetrahedron very small indeed, but to still contain the given point on the surface. An enlarged image of such a tetrahedron is shown on the right, and we see how the curved edges may be approximated by straight lines in the limit of a very small tetrahedron. The goal is to relate the traction at the given point to the tractions on the internal cut planes, because these tractions are representations of the stress in the volume.
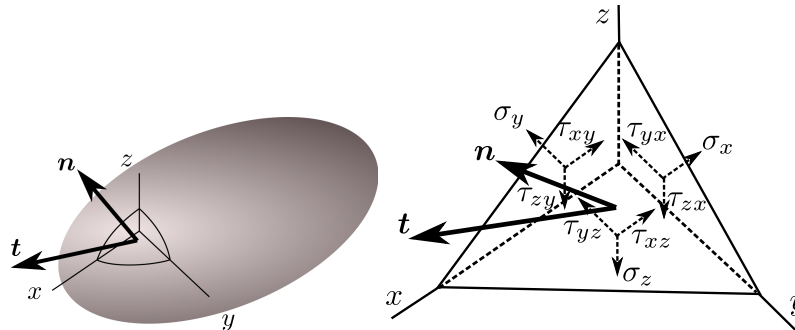
Fig. 5.38. Relating the components of traction to stress.

In anticipation of the definition of stress, the traction components on the three flat cut planes, with normals pointing against the three Cartesian basis vectors, are called σ_x, σ_y, σ_z (the normal

components), and τ_{xy}, τ_{yx}, τ_{xz}, τ_{zx}, τ_{yz}, τ_{zy}, for the shear components on all three planes. The areas of the triangular faces of the tetrahedron are related as $A_x = n_x A$, and so forth, where n_x, n_y, n_z are the components of the unit normal, and A_x is the area orthogonal to the x-axis and so on; this can be deduced from the volume of the tetrahedron in Fig. 5.39 written in terms of the heights d, i_x, i_y, i_z, and the corresponding areas.
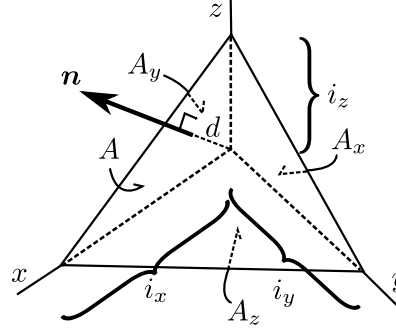
Fig. 5.39. Components of traction.

When we write the conditions of equilibrium in all three directions (the volume forces do not play a role; why?), the following three equations result

$$
\begin{aligned}
t_x &= \sigma_x n_x + \tau_{xy} n_y + \tau_{xz} n_z \ , \\
t_y &= \tau_{yx} n_x + \sigma_y n_y + \tau_{yz} n_z \ , \\
t_z &= \tau_{zx} n_x + \tau_{zy} n_y + \sigma_z n_z \ .
\end{aligned}
\tag{5.76}
$$

This equation relates the components of the traction on the surface with the components of the traction on the special surfaces – coordinate planes – inside the volume. The components of the traction on the internal surfaces are called **normal stresses** (σ_x, σ_y, σ_z), and **shear stresses** (τ_{xy}, τ_{yx}, τ_{xz}, τ_{zx}, τ_{yz}, τ_{zy}). The form of Eq. (5.76) suggests the matrix expression

$$
\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \ ,
\tag{5.77}
$$

where all matrices hold components in the Cartesian basis. A component-free version would read

$$
\boldsymbol{t} = \boldsymbol{\Sigma} \cdot \boldsymbol{n} \ ,
$$

where $\boldsymbol{\Sigma}$ would be defined as a Cartesian tensor, the **Cauchy stress tensor**. The traction vector and the normal would then also become tensors. However, in this book the tensor notation is avoided, and with a few exceptions tensors will not be needed. The two exceptions that may be mentioned here are coordinate transformations and the calculation of the **principal stresses** which are the eigenvalues of the matrix of the stress components.

First we will consider transformation of vectors from one Cartesian coordinate system into another. The Cartesian coordinate axes are defined by a triple of orthonormal basis vectors, one by the basis triple $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ and the other by the basis triple $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$. An arbitrary vector $\boldsymbol{u}$ may be written in terms of components and basis vectors in either coordinate system as

$$
\boldsymbol{u} = u_x \boldsymbol{e}_x + u_y \boldsymbol{e}_y + u_z \boldsymbol{e}_z = u_{\overline{x}} \boldsymbol{e}_{\overline{x}} + u_{\overline{y}} \boldsymbol{e}_{\overline{y}} + u_{\overline{z}} \boldsymbol{e}_{\overline{z}} \ .
\tag{5.78}
$$

This may be written as a matrix expression, with the basis vectors as columns of matrices

$$
\boldsymbol{u} = \begin{bmatrix} \boldsymbol{e}_x, & \boldsymbol{e}_y, & \boldsymbol{e}_z \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_{\overline{x}}, & \boldsymbol{e}_{\overline{y}}, & \boldsymbol{e}_{\overline{z}} \end{bmatrix} \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \ .
\tag{5.79}
$$

Now consider that the basis vectors themselves may be expressed in terms of their components on some basis: we will pick $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ as that basis. On this basis we can write the components of the vectors $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ themselves as

$$[\boldsymbol{e}_x] = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad [\boldsymbol{e}_y] = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad [\boldsymbol{e}_z] = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We can now write, entirely in components,

$$\begin{bmatrix} [\boldsymbol{e}_x], [\boldsymbol{e}_y], [\boldsymbol{e}_z] \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} .$$

The components of the vectors $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ are

$$[\boldsymbol{e}_{\overline{x}}] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z \end{bmatrix}, \quad [\boldsymbol{e}_{\overline{y}}] = \begin{bmatrix} \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z \end{bmatrix}, \quad [\boldsymbol{e}_{\overline{z}}] = \begin{bmatrix} \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix},$$

where $\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x$ is the cosine of the angle between the two vectors $\boldsymbol{e}_{\overline{x}}$ and $\boldsymbol{e}_x$, and so on, so that we can write entirely in terms of components

$$\begin{bmatrix} \boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}} \end{bmatrix} \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z \end{bmatrix}, \begin{bmatrix} \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z \end{bmatrix}, \begin{bmatrix} \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} \end{bmatrix} \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} .$$

where we have defined the transformation matrix

$$[\boldsymbol{R}_m] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} , \tag{5.80}$$

composed of the direction cosines. In words, the columns of the transformation matrix consist of the components of the vectors $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ on the basis vectors $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$. The transformation matrix is recognized as an orthogonal matrix (rotation matrix), whose inverse is its transpose

$$[\boldsymbol{R}_m]^{-1} = [\boldsymbol{R}_m]^T$$

Thus we have the ***transformation of vector components***

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} . \tag{5.81}$$

Now we take up again the relationship between the traction vector, the stress at a point, and the normal to a surface through the point (5.77)

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} ,$$

and we consider what happens when both vectors are expressed using components in a different coordinate system. We apply (5.81) as

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix}$$

so that we can write

$$
\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} \, ,
$$

or

$$
[\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} \, ,
$$

and finally

$$
\begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} \, .
$$

Since one also has

$$
\begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} \, ,
$$

we conclude that the ***stress tensor components transform*** when switching between coordinate systems as

$$
\begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \tag{5.82}
$$

Of course, as for vectors this transformation also works back and forth so that

$$
\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} [\boldsymbol{R}_m]^T \, . \tag{5.83}
$$

Let us look at an example: The stress components are given at the point A on the surface of the sphere in the Cartesian coordinate system with basis $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ as

$$
[\sigma] = \begin{bmatrix} -350, & -150, & 0 \\ -150, & -350, & 0 \\ 0, & 0, & -200 \end{bmatrix}
$$

Transform the stress components to the Cartesian coordinate system $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ which is defined so that $\boldsymbol{e}_{\overline{x}}$ is in the xy plane, at an angle of $45°$ from $\boldsymbol{e}_x$, and $\boldsymbol{e}_{\overline{z}} = \boldsymbol{e}_z$.
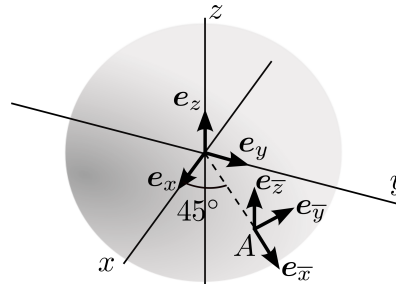
Solution: The transformation (5.82) needs to be applied. Therefore, the transformation matrix $[\boldsymbol{R}_m]$ is needed. The basis vector $\boldsymbol{e}_{\overline{x}}$ and $\boldsymbol{e}_{\overline{z}}$ follow the definition

```
exb = array([math.sqrt(2.)/2, math.sqrt(2.)/2, 0])
ezb = array([0, 0, 1.])
```

The third vector is obtained using the vector cross product

```
eyb = cross(ezb, exb)
```

Therefore we can write $[\boldsymbol{R}_m]$ with these three basis vectors as columns

```
# The vectors exb, eyb, ezb are going to become the rows
# of RmT; to get the actual rotation matrix we need
# to transpose
Rm = array([exb, eyb, ezb]).T
```

and the transformed stress components are obtained from the formula as

```
sigb = dot(Rm.T, sig).dot(Rm)
```

giving the resulting stress matrix

```
array([[-500.,    0.,     0.],
       [   0., -200.,     0.],
       [   0.,    0., -200.]])
```

This means that the shear stresses $\tau_{\overline{xy}}, \tau_{\overline{xz}}, \tau_{\overline{yz}}$ are all zero, and the normal stresses are $\sigma_{\overline{x}} = -500, \sigma_{\overline{y}} = -200, \sigma_{\overline{z}} = -200$. End of example.

The **principal directions** are normals to such special surfaces that the traction vector acting on the surface has only a normal component. We write this statement as

$$\begin{bmatrix} \widehat{t}_x \\ \widehat{t}_y \\ \widehat{t}_z \end{bmatrix} = \widehat{\sigma} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} \ . \tag{5.84}$$

In words, the traction vector $[\widehat{t}]$ has the same direction as the normal to the surface $[\widehat{n}]$ and (signed) magnitude $\widehat{\sigma}$. Substituting for the traction vector from (5.77) leads to

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} = \widehat{\sigma} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} \ , \tag{5.85}$$

This is a standard eigenvalue problem, and for the solution to exist one further equation must be true

$$\det \left(\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} - \widehat{\sigma} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = 0 \ . \tag{5.86}$$

This equation is equivalent to the search for the roots of the cubic characteristic equation

$$-\widehat{\sigma}^3 + C_2 \widehat{\sigma}^2 + C_1 \widehat{\sigma} + C_0 = 0$$

where $C_2 = (\sigma_x + \sigma_y + \sigma_z)$, $C_1 = (\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2 - \sigma_x \sigma_y - \sigma_x \sigma_z - \sigma_y \sigma_z)$, $C_0 = -\sigma_z \tau_{xy}^2 + 2\tau_{xy}\tau_{xz}\tau_{yz} - \sigma_y \tau_{xz}^2 - \sigma_x \tau_{yz}^2 + \sigma_x \sigma_y \sigma_z$. Consequently, this being a cubic equation there must be three such roots – **principal stresses**–which we denote

$$\widehat{\sigma}_j \quad j = 1, 2, 3 \ ,$$

and three mutually orthogonal **principal directions** (eigenvectors)

$$\begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_j \quad j = 1, 2, 3 \ .$$

The matrix of the stress components being real and symmetric guarantees that the roots of the characteristic equation are always real, and also the existence of the three eigenvectors that are orthonormal is guaranteed.

As an example we will find the principal stresses of the stress matrix

$$[\sigma] = \begin{bmatrix} -350, & -150, & 0 \\ -150, & -350, & 0 \\ 0, & 0, & -200 \end{bmatrix}$$

Solution: While there are formulas for the roots of cubic equations, in finite element programs principal stresses are normally computed numerically and here we shall solve the eigenvalue problem using numerical methods built-in into the Python `linalg.eig` solver: First we define the array of the stress components

```
sig = array([[-350.00, -150.00, 0],
        [-150.0, -350.00, 0],
           [0, 0, -200.0000]])
```

which is subsequently passed to the solver

```
D, V = linalg.eig(sig)
```

The principal stresses are in the array `D`

```
[-200. -500. -200.]
```

Note that they are not sorted. The principal directions are the columns of the array `V`

```
[[ 0.70710678  0.70710678  0.        ]
 [-0.70710678  0.70710678  0.        ]
 [ 0.          0.          1.        ]]
```

By convention in continuum mechanics the principal stresses should be ordered $\widehat{\sigma}_1 \geq \widehat{\sigma}_2 \geq \widehat{\sigma}_3$. Therefore, we can sort the array `D` and then reorder both arrays, switching the columns of the principal-direction matrix (note that the sort is in ascending order, but we need the descending order; hence we use the `order` array reshuffled from last to first, `order[::-1]`)

```
order = argsort(D)
print('Sorted principal stresses')
print(D[order[::-1]]) # reverse the order to make it descending
print('Sorted principal directions')
print(V[:, order[::-1]])
```

The code above prints

```
Sorted principal stresses
[-200. -200. -500.]
Sorted principal directions
[[ 0.          0.70710678  0.70710678]
 [ 0.         -0.70710678  0.70710678]
 [ 1.          0.          0.        ]]
```

so that the principal stresses and principal directions are explicitly

$$\widehat{\sigma}_1 = -200 , \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\widehat{\sigma}_2 = -200 , \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_2 = \begin{bmatrix} 0.70710678 \\ -0.70710678 \\ 0 \end{bmatrix}$$

and

$$\widehat{\sigma}_3 = -500 , \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_3 = \begin{bmatrix} 0.70710678 \\ 0.70710678 \\ 0 \end{bmatrix}$$

End of example.

Balance of angular momentum and stress symmetry.

It would appear that there are nine components of stress that need to be related to the deformation, but it is straightforward to show that in the matrix (5.77) the elements reflected with respect to the diagonal must be equal: Consider a rectangular volume of material (again, for convenience the drawing in Fig. 5.40 is of a two-dimensional nature, but the argument applies to three dimensions). When the **balance of angular momentum** is written for the rotation about the axis orthogonal to the plane of the paper, the normal stresses and any body forces will turn out to be negligible compared to the effect of the shear stresses, and from the resultant equation we obtain the symmetries

$$\tau_{xy} = \tau_{yx} \;, \quad \tau_{xz} = \tau_{zx} \;, \quad \tau_{yz} = \tau_{zy} \;. \tag{5.87}$$
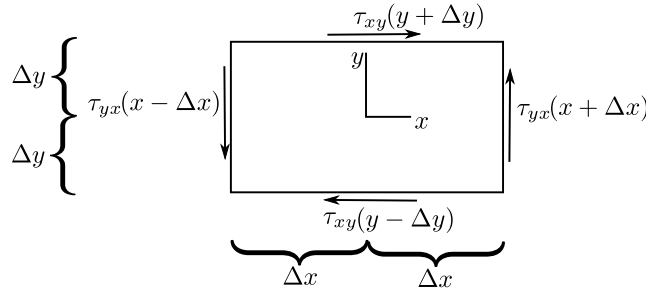
Fig. 5.40. Components of traction.

Consequently, there are only six components of the stress that are independent. It will be convenient to manipulate these six components as a vector (as opposed to a tensor)

$$[\boldsymbol{\sigma}] = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{xz}, \tau_{yz}]^T \;. \tag{5.88}$$

Equation (5.76) may be rewritten in terms of the stress vector $\boldsymbol{\sigma}$ as

$$\boldsymbol{t} = \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} \;, \tag{5.89}$$

where the matrix "**vector-stress vector dot product**" operator is defined as

$$\mathcal{P}_{\boldsymbol{n}} = \begin{bmatrix} n_x & 0 & 0 & n_y & n_z & 0 \\ 0 & n_y & 0 & n_x & 0 & n_z \\ 0 & 0 & n_z & 0 & n_x & n_y \end{bmatrix} \;. \tag{5.90}$$

Equation (5.89) may be used in a variety of ways: any of the three quantities may be given, which would then for another quantity being fixed produce the third as the result. Most useful are these two possibilities: $\boldsymbol{t}$ given, produce the stress vector in dependence on the normal; and $\boldsymbol{\sigma}$ given, produce the surface tractions for various normals.

Here is an illustration: Stress components at a point are given as

$$\sigma_x = 9 \;, \quad \sigma_y = -3 \;, \quad \sigma_z = 3 \;, \quad \tau_{xy} = -2 \;, \quad \tau_{xz} = 0 \;, \quad \tau_{yz} = 2$$

Find the magnitude of the normal and shear traction on the plane with normal $[n] = [2, 1, 2]^T/3$.

Solution: This can be done in two ways. Firstly we can organize the stress components in a square matrix and use equation (5.77). Secondly we can use (5.89) with the stress vector form.

The first approach: The traction vector is computed from the matrix representing the stress

```
sig = array([[9.00, -2.00, 0.00],
             [-2.00, -3.00, 2.00],
             [0.00, 2.00, 3.00]])
```

and the normal

```
n = array([2.0, 1.0, 2.0])
n = n/linalg.norm(n)
```

as t = dot(sig, n)

```
array([ 5.33333333, -1.        ,  2.66666667])
```

The normal component of the traction is tn = dot(t, n), that is 5.0. The shear part of the traction is the difference between the total traction vector and its normal part

```
ts = t - tn * n
```

and the shear traction vector is

```
array([ 2.        , -2.66666667, -0.66666667])
```

The magnitude of the shear traction vector is linalg.norm(ts), i.e. 3.399346342.
    The second approach computes the projection matrix from the normal

```
Pn = array([[n[0], 0, 0, n[1], n[2], 0],
            [0, n[1], 0, n[0], 0, n[2]],
            [0, 0, n[2], 0, n[0], n[1]]])
```

and then from the stress vector

```
sigv = array([9.00, -3.00, 3.00, -2.00, 0.00, 2.00])
sigv.shape = (6, 1)
```

evaluates, after the normal vector is made into a column n.shape = (3, 1) so that the matrices are compatible,

```
t = dot(Pn, sigv)
tn = dot(t.T, n)
ts = t - tn * n
print(linalg.norm(ts))
```

which results in the precisely the same results as before. End of example.
    Another example: Stress components at a point are given as

$$\sigma_x = 9 , \quad \sigma_y = -3 , \quad \sigma_z = 3 , \quad \tau_{xy} = -2 , \quad \tau_{xz} = 0 , \quad \tau_{yz} = 2$$

Find the largest possible shear stress at the given point. This means the maximum of the shear stress acting on all planes that can be passed through the point.
    Solution: The maximum possible shear stress is computed from the principal stress values as

$$\tau_{\max} = \frac{\sigma_1 - \sigma_3}{2}$$

i.e. as half the difference between the largest and the smallest principal stress. The principal stress values and the principal stress directions are computed from

```
sig = array([[9.00, -2.00, 0.00],
             [-2.00, -3.00, 2.00],
             [0.00, 2.00, 3.00]])

princstrs, princdirs = linalg.eig(sig)

order = argsort(princstrs)
print('Sorted principal stresses')
princstrs = princstrs[order[::-1]]
print(princstrs)
print('Sorted principal directions')
princdirs = princdirs[:, order[::-1]]
print(princdirs)
```

which gives

```
Sorted principal stresses
[ 9.34156488  3.54921973 -3.89078461]
Sorted principal directions
[[ 0.98434176  0.09670745  0.14737356]
 [-0.16810829  0.26356552  0.94988042]
 [-0.05301792  0.95978169 -0.27569587]]
```

and the maximum shear stress is therefore found as $\tau_{\max} = (9.342 - -3.891)/2 = 6.616$. Notably the maximum shear stress appears in the Tresca failure criterion

$$\tau_{\max} \leq \frac{\sigma_{\mathrm{f}}}{2}$$

where σ_{f} is the failure stress of the material in uniaxial tension. End of example.

Ductile materials (such as metals) are often considered in failure analyses in relation to the stress quantity that describes the distortion of the volume of the material (no change in volume), the von Mises equivalent stress.

One more example: Stress components at a point are given as

$$\sigma_x = 9 \,, \quad \sigma_y = -3 \,, \quad \sigma_z = 3 \,, \quad \tau_{xy} = -2 \,, \quad \tau_{xz} = 0 \,, \quad \tau_{yz} = 2$$

Find the Von Mises equivalent stress at the given point.

Solution: The von Mises equivalent stress may be computed from the principal stresses as

$$\sigma_{\mathrm{vm}} = \frac{1}{\sqrt{2}} \sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_1 - \sigma_3)^2 + (\sigma_3 - \sigma_2)^2}$$

or directly from the components of the stress as

$$\sigma_{\mathrm{vm}} = \frac{1}{\sqrt{2}} \sqrt{(\sigma_x - \sigma_y)^2 + (\sigma_x - \sigma_z)^2 + (\sigma_z - \sigma_y)^2 + 6(\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2)}$$

Substituting given stress components we obtain $\sigma_{\mathrm{vm}} = 11.489$. The von Mises equivalent stress is compared in the von Mises failure criterion

$$\sigma_{\mathrm{vm}} \leq \sigma_{\mathrm{f}}$$

with the failure stress of the material in uniaxial tension σ_{f}. End of example.

Equation of Local equilibrium

In complete analogy to the model of heat conduction, the global balance equation (5.72) (in this case, balance of linear momentum, for the heat conduction it was balance of heat energy (2.12)) needs to be converted to a local form. The local form expresses dynamic equilibrium of an infinitesimal particle as an equation that holds at a point.

There are three terms in the global balance (5.72), and to produce the local form we'll have to convert all three integrals to volume integrals. The first one involves the time derivative of the integral

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_m \boldsymbol{v} \, \mathrm{d}m \,.$$

However, that causes no difficulties since the mass m inside the volume V does not change with time. Therefore,

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_m \boldsymbol{v} \, \mathrm{d}m = \int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}m \,. \tag{5.91}$$

Introducing the **mass density** ρ (which as mass per unit volume depends on the deformation, and hence varies with time), we may write $\mathrm{d}m = \rho \mathrm{d}V$ and

$$\int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\,\mathrm{d}m = \int_V \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\rho\,\mathrm{d}V\ . \tag{5.92}$$

The divergence theorem may be now applied to the third term in (5.72), that is to the surface integral. However, if we introduce the abstract symbol for the divergence of stress by spelling out its definition, we generate more questions than answers. Therefore, it will be instructive to get to the needed form of the divergence theorem in a roundabout way.

Consider a small volume (parallelepiped) with faces parallel to coordinate planes of the global Cartesian basis (Fig. 5.41, and refer also to Fig. 5.35); for simplicity, the box is drawn as two-dimensional, and it is drawn twice so that we can display the normal and the shear stresses separately. The center of the box is at x, y, z, and the stress components may be expanded into a truncated Taylor series. For instance,

$$\sigma_x(x + \xi\Delta x, y + \eta\Delta y, z + \zeta\Delta z) \approx \sigma_x(x, y, z) + \frac{\partial\sigma_x(x, y, z)}{\partial x}\xi\Delta x$$
$$+ \frac{\partial\sigma_x(x, y, z)}{\partial y}\eta\Delta y + \frac{\partial\sigma_x(x, y, z)}{\partial z}\zeta\Delta z\ ,$$

where $-1 \leq \xi \leq +1$, $-1 \leq \eta \leq +1$, and $-1 \leq \zeta \leq +1$.
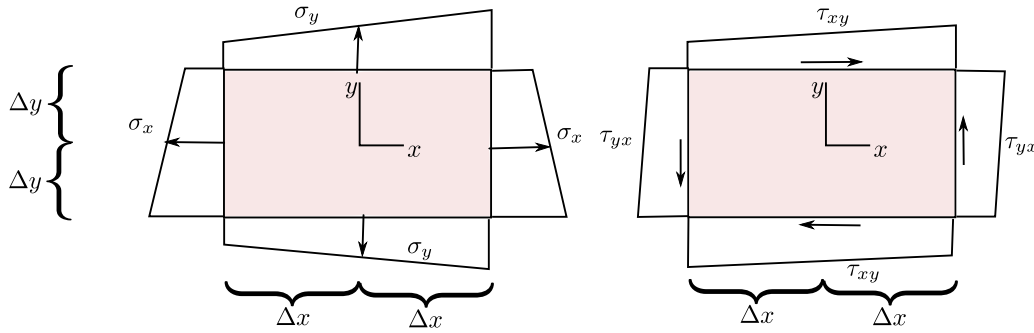
Fig. 5.41. Components of traction as functions of x, y along the sides of the box.

The box is loaded only by the tractions on its boundary, there are no body loads. Equilibrium in the x-direction requires integration of the stress σ_x over the vertical sides of the box, τ_{xy} over the horizontal sides, and τ_{xz} over the faces parallel to the plane of the paper. For instance, integrating σ_x over the side at $\xi = 1$ leads to

$$\Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \sigma_x(x + \Delta x, y + \eta\Delta y, z + \zeta\Delta z)\,\mathrm{d}\eta\mathrm{d}\zeta \approx$$
$$\Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1}\left[\sigma_x(x, y, z) + \frac{\partial\sigma_x(x, y, z)}{\partial x}\Delta x\right.$$
$$\left. + \frac{\partial\sigma_x(x, y, z)}{\partial y}\eta\Delta y + \frac{\partial\sigma_x(x, y, z)}{\partial z}\zeta\Delta z\right]\,\mathrm{d}\eta\mathrm{d}\zeta\ .$$

The terms with η and ζ integrate to zero, and the result is

$$4\Delta y\Delta z\left[\sigma_x(x, y, z) + \frac{\partial\sigma_x(x, y, z)}{\partial x}\Delta x\right]\ .$$

Next, integrating σ_x over the side at $\xi = -1$ leads to

$$\Delta y \Delta z \int_{-1}^{+1} \int_{-1}^{+1} -\sigma_x(x + \Delta x, y + \eta \Delta y, z + \zeta \Delta z) \, \mathrm{d}\eta \mathrm{d}\zeta \approx$$

$$\Delta y \Delta z \int_{-1}^{+1} \int_{-1}^{+1} \left[-\sigma_x(x, y, z) + \frac{\partial \sigma_x(x, y, z)}{\partial x} \Delta x \right.$$

$$\left. - \frac{\partial \sigma_x(x, y, z)}{\partial y} \eta \Delta y - \frac{\partial \sigma_x(x, y, z)}{\partial z} \zeta \Delta z \right] \mathrm{d}\eta \mathrm{d}\zeta \, .$$

The terms with η and ζ integrate to zero, and the result is

$$4 \Delta y \Delta z \left[-\sigma_x(x, y, z) + \frac{\partial \sigma_x(x, y, z)}{\partial x} \Delta x \right] \, .$$

Adding these two together gives the total contribution of the stress σ_x as

$$8 \Delta x \Delta y \Delta z \frac{\partial \sigma_x(x, y, z)}{\partial x} = \Delta V \frac{\partial \sigma_x(x, y, z)}{\partial x} \, ,$$

with the elementary volume $\Delta V = 8 \Delta x \Delta y \Delta z$. The same exercise is now repeated for the stress components τ_{xy} and τ_{xz}, giving the total force on the elementary volume in the x-direction

$$\Delta b_x^* = \Delta V \left[\frac{\partial \sigma_x(x, y, z)}{\partial x} + \frac{\partial \tau_{xy}(x, y, z)}{\partial y} + \frac{\partial \tau_{xz}(x, y, z)}{\partial z} \right] \, , \tag{5.93}$$

and analogously in the other two directions

$$\Delta b_y^* = \Delta V \left[\frac{\partial \tau_{yx}(x, y, z)}{\partial x} + \frac{\partial \sigma_y(x, y, z)}{\partial y} + \frac{\partial \tau_{yz}(x, y, z)}{\partial z} \right] \, , \tag{5.94}$$

and

$$\Delta b_z^* = \Delta V \left[\frac{\partial \tau_{zx}(x, y, z)}{\partial x} + \frac{\partial \tau_{zy}(x, y, z)}{\partial y} + \frac{\partial \sigma_z(x, y, z)}{\partial z} \right] \, . \tag{5.95}$$

Now the same argument that was established around Eq. (5.70) will be pursued: put together the total force on the body by collecting the contributions from all the elementary volumes. This can be done in two ways:

1. Add up all the tractions on the bounding faces of the elementary volumes. The tractions on the shared faces (internal surfaces) will cancel; only the tractions on the exterior surface will be left:

$$\int_S \boldsymbol{t} \, \mathrm{d}S$$

2. Add up all the resultant equivalent volume forces (5.93-5.95), which in the limit will become a volume integral

$$\int_V \boldsymbol{b}^* \, \mathrm{d}V$$

where the imaginary force $\boldsymbol{b}^*$ has components on the Cartesian basis

$$[\boldsymbol{b}^*] = \begin{bmatrix} \dfrac{\partial \sigma_x(x, y, z)}{\partial x} + \dfrac{\partial \tau_{xy}(x, y, z)}{\partial y} + \dfrac{\partial \tau_{xz}(x, y, z)}{\partial z} \\[2ex] \dfrac{\partial \tau_{yx}(x, y, z)}{\partial x} + \dfrac{\partial \sigma_y(x, y, z)}{\partial y} + \dfrac{\partial \tau_{yz}(x, y, z)}{\partial z} \\[2ex] \dfrac{\partial \tau_{zx}(x, y, z)}{\partial x} + \dfrac{\partial \tau_{zy}(x, y, z)}{\partial y} + \dfrac{\partial \sigma_z(x, y, z)}{\partial z} \end{bmatrix} \tag{5.96}$$

and may be recognized as the ***stress divergence***.

These two forces are equal, and we have the following form of the divergence theorem

$$\int_V \boldsymbol{b}^* \, \mathrm{d}V = \int_S \boldsymbol{t} \, \mathrm{d}S \ .$$

Using the template of the "vector-stress vector dot product" operator (5.90), we may write the stress divergence as

$$\boldsymbol{b}^* = \mathcal{B}^T \boldsymbol{\sigma} \ , \tag{5.97}$$

where the **stress-divergence operator** $\mathcal{B}^T$ is defined as

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & 0 & \partial/\partial y & \partial/\partial z & 0 \\ 0 & \partial/\partial y & 0 & \partial/\partial x & 0 & \partial/\partial z \\ 0 & 0 & \partial/\partial z & 0 & \partial/\partial x & \partial/\partial y \end{bmatrix} \ . \tag{5.98}$$

This operator (un-transposed) will make its appearance shortly yet again as the *symmetric gradient operator* to produce strains out of displacements. Using the definitions of both of these useful operators, the **divergence theorem** may be written in terms of stress as

$$\int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} \, \mathrm{d}S \ . \tag{5.99}$$

Putting the three integrals from (5.72) into the volume-integral form leads to a pointwise expression of local equilibrium:

$$\int_V \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}V = \int_V \overline{\boldsymbol{b}} \, \mathrm{d}V + \int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V \quad \Rightarrow \quad \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \overline{\boldsymbol{b}} + \mathcal{B}^T \boldsymbol{\sigma} \ . \tag{5.100}$$

This is a statement of **dynamic equilibrium** of a point particle: On the left-hand side we have the inertial force (mass times acceleration), on the right-hand side is the body load and the force generated by a stress gradient across the particle. Analogously to the heat conduction problem, this local balance equation contains too many variables. The stress plays the role of the heat flux, and it also will be replaced by reference to measurable variables – the strains.

End Box 16

Box 17. Strains and displacements

The strains measure the *relative* deformation, and based on the effect they represent when expressed in Cartesian coordinates, they may be divided into two groups: the normal strains (stretches), and the shear strains.

The strains are an expression of the local variations in the positions of material particles after deformation. The deformation (motion) is expressed as displacements. The **displacement** $\boldsymbol{u}$ is expressed in the Cartesian coordinates by components, and connects the locations of a given **material point** (particle) A before deformation and after deformation

$$[\boldsymbol{u}(A,t)] = \begin{bmatrix} x(A,t) \\ y(A,t) \\ z(A,t) \end{bmatrix} - \begin{bmatrix} x(A,0) \\ y(A,0) \\ z(A,0) \end{bmatrix} \ . \tag{5.101}$$

It will be useful to approach the meaning of strains from the point of view of what happens to tangents to material curves during the deformation. A **material curve** consists of the same material
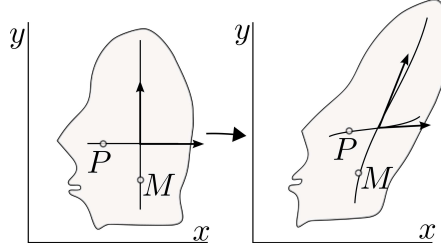
Fig. 5.42. Material curves, and tangents to material curves. Left: before deformation, right: after deformation.

points (particles) at any point in time. A visual may be useful: recall that some specimens have a square grid etched upon them before they are being mechanically tested (deformed). The etching curves that go in one direction may be thought of as sets of points whose one coordinate changes and the other is being held fixed. Figure 5.42 shows a blob of material with two material curves before and after deformation. Before deformation, the curve that is horizontal consists of points P such that the coordinates are

$$[P] = \begin{bmatrix} x \\ y = \text{constant} \end{bmatrix} ,$$

and the curve that is vertical consists of points M such that

$$[M] = \begin{bmatrix} x = \text{constant} \\ y \end{bmatrix} .$$

The parameter that varies along the curve through the point P is x. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 \\ 0 \end{bmatrix} . \tag{5.102}$$

The parameter that varies along the curve through the point M is y. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} . \tag{5.103}$$

The tangent vectors (5.102) and (5.103) are of course just the basis vectors of the Cartesian coordinates.

After deformation, the curve that used to be horizontal consists of points P such that

$$[P] = \begin{bmatrix} x + u_x \\ (y = \text{constant}) + u_y \end{bmatrix} ,$$

and the curve that is vertical consists of points M such that

$$[M] = \begin{bmatrix} (x = \text{constant}) + u_x \\ y + u_y \end{bmatrix} .$$

Since these are material curves, they are still parameterized by the same parameters as before deformation. Consequently, for the originally horizontal curve we have the tangent vector after deformation

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 + \dfrac{\partial u_x}{\partial x} \\ \dfrac{\partial u_y}{\partial x} \end{bmatrix} . \tag{5.104}$$

The parameter that varies along the curve through the point M is y. Therefore, after deformation the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} \dfrac{\partial u_x}{\partial y} \\ 1 + \dfrac{\partial u_y}{\partial y} \end{bmatrix} \ . \tag{5.105}$$

The **stretches** measure the relative change in length of the tangent vectors at the same material point before and after deformation. For instance, the tangent vector (5.102) is of unit length before deformation, and the vector (5.104) is of length

$$\sqrt{(1 + \frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} = \sqrt{1 + 2\frac{\partial u_x}{\partial x} + (\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} \ .$$

If we now make the *assumption that the derivatives of the displacement components are very small in magnitude,*

$$|\frac{\partial u_k}{\partial j}| \ll 1, \quad k, j = x, y, z \ , \tag{5.106}$$

the length of the tangent vector may be expressed as

$$\sqrt{1 + 2\frac{\partial u_x}{\partial x} + (\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} \approx 1 + \frac{\partial u_x}{\partial x} \ ,$$

and the relative change in length (the stretch in the x direction) is

$$\frac{1 + \dfrac{\partial u_x}{\partial x} - 1}{1} = \epsilon_x \ .$$

The **shears** measure the change in the angle between originally orthogonal directions of pairs of the Cartesian axes. Therefore, we could measure the change in the angle between the tangents of two intersecting material curves before and after deformation. For the two curves in Fig. 5.42, the initial angle is $\pi/2$; the cosine of the angle after the deformation is

$$\frac{\partial}{\partial x}[P]^T \frac{\partial}{\partial y}[M] = (1 + \frac{\partial u_x}{\partial x})\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}(1 + \frac{\partial u_y}{\partial y})$$

which, again using the assumption (5.106), gives for the change of the angle

$$\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} = \gamma_{xy} \ .$$

In this way we define all six strain components: three stretches, and three shears. In fact, we could have defined nine strains (components of the **strain tensor**), which would correspond to the nine components of the Cauchy stress tensor. However, we will stick to the vector representation in this book.

The six strain components are a mixture of the derivatives of the displacement components, and may be expressed in an operator equation, using the definition (5.97)

$$\boldsymbol{\epsilon} = \mathcal{B}\boldsymbol{u} \ , \tag{5.107}$$

where $\mathcal{B}$ is called the **symmetric gradient** (or strain-displacement) operator.

The components of the strain are

$$[\boldsymbol{\epsilon}]^T = [\epsilon_x, \epsilon_y, \epsilon_z, \gamma_{xy}, \gamma_{xz}, \gamma_{yz}]^T \tag{5.108}$$

Note the transpose, the strain vector has six rows and one column. The first three components are the stretches, the last three components are the shears.

End Box 17

Box 18. Elastic constitutive equation

The stress now needs to be linked to the strains. Then we will be in a position to replace it in the balance equation (5.100) by reference to the primary variable, the displacement. As for the heat conduction model, this link is the constitutive equation.

The constitutive equation that will be of interest in this book is the model of **_linear elasticity_**. It is expressed as a linear relationship between the strain and the stress, and since these are vectors, the linear relationship, the **_constitutive equation_**, is expressed as a matrix product

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon} \; , \tag{5.109}$$

where $\boldsymbol{D}$ is a 6×6 matrix of the elastic coefficients (also known as the elasticities); $\boldsymbol{D}$ may be also referred to as the **_material stiffness_** matrix. If the material stiffness matrix changes from point to point, we call the material inhomogeneous; otherwise when the matrix is uniform across the material, we call the material homogeneous.

The total energy density stored in the material in the volume $\mathrm{d}V$ is

$$\phi(\boldsymbol{\epsilon}) = \frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}\boldsymbol{\epsilon} \, \mathrm{d}V \; . \tag{5.110}$$

Mathematically, the expression $\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}\boldsymbol{\epsilon}$ is known as a **_quadratic form_**. One interesting property of the quadratic form is that the unsymmetrical part of the matrix $\boldsymbol{D}$ does not contribute to the energy:

$$\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}\boldsymbol{\epsilon} = \left(\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}\boldsymbol{\epsilon} \right)^T = \frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}^T \boldsymbol{\epsilon} \quad \Rightarrow \quad \frac{1}{2}\boldsymbol{\epsilon}^T (\boldsymbol{D} - \boldsymbol{D}^T)\boldsymbol{\epsilon} = 0 \; .$$

Because the energy of deformation is a fundamental quantity, from physical principles, and from the point of view of mathematical modeling, this is a very good reason for postulating a priori the **_symmetry of the material stiffness, $\boldsymbol{D} = \boldsymbol{D}^T$_**.

Material stiffness of linearly elastic materials

The strain displacement operator (symmetric gradient operator) (5.107) links displacements in terms of their components in the global Cartesian basis to strains. The strains could be expressed in the same Cartesian basis, but need not be. In fact, it will be most useful not to express the strains in the same global Cartesian coordinate system. The motivating factor is the constitutive equation: For some materials it will be important to keep track of the local orientation of the material volume. For instance, fiber reinforced materials will have very different stiffness properties along the fibers as opposed to orthogonally to the fibers.

The components of the material stiffness matrix (or the material compliance matrix) are to be understood as being expressed in the local coordinate system $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$, $\boldsymbol{e}_{\overline{z}}$ attached to the material point in the form of the rotation matrix (5.80).

General anisotropic material. Because of the symmetry of the material stiffness, for the most general elastic material the number of elastic coefficients is only 21 out of the total of 36 elements of the material stiffness matrix. Still, to identify all of these constants represents a major experimental effort, and few engineering materials are characterized as fully anisotropic.

Orthotropic material. If a material has three mutually orthogonal planes of symmetry, it is known as an orthotropic material. For instance wood is often characterized as such type of material, and fiber-reinforced composites are in more sophisticated analyses also treated as orthotropic. The compliance matrix

$$\boldsymbol{C} = \boldsymbol{D}^{-1} \,,$$

has a pleasingly simple appearance

$$\boldsymbol{C} = \begin{bmatrix} E_1^{-1}, & -\dfrac{\nu_{12}}{E_1}, & -\dfrac{\nu_{13}}{E_1}, & 0, & 0, & 0 \\[2ex] -\dfrac{\nu_{12}}{E_1}, & E_2^{-1}, & -\dfrac{\nu_{23}}{E_2}, & 0, & 0, & 0 \\[2ex] -\dfrac{\nu_{13}}{E_1}, & -\dfrac{\nu_{23}}{E_2}, & E_3^{-1}, & 0, & 0, & 0 \\[2ex] 0, & 0, & 0, & G_{12}^{-1}, & 0, & 0 \\[2ex] 0, & 0, & 0, & 0, & G_{13}^{-1}, & 0 \\[2ex] 0, & 0, & 0, & 0, & 0, & G_{23}^{-1} \end{bmatrix} \,.$$

All nine coefficients are independent, and need to be provided as input [4]. The material stiffness matrix is a bit of a mess. The nonzero elements are

$$D_{11} = \frac{C_{22}C_{33} - C_{23}C_{23}}{C}, \quad D_{12} = \frac{C_{13}C_{23} - C_{12}C_{33}}{C},$$
$$D_{13} = \frac{C_{12}C_{23} - C_{13}C_{22}}{C}, \quad D_{22} = \frac{C_{33}C_{11} - C_{13}C_{13}}{C},$$
$$D_{23} = \frac{C_{12}C_{13} - C_{23}C_{11}}{C}, \quad D_{33} = \frac{C_{11}C_{22} - C_{12}C_{12}}{C},$$
$$D_{44} = G_{12}, \qquad\qquad D_{55} = G_{13}, \qquad\qquad D_{66} = G_{23}.$$

Here $C = C_{11}C_{22}C_{33} - C_{11}C_{23}C_{23} - C_{22}C_{13}C_{13} - C_{33}C_{12}C_{12} + 2C_{12}C_{23}C_{13}$.

Transversely isotropic material. If a material has an infinite number of planes of symmetry passing through an axis (in this case, the local x-axis), and one plane of symmetry orthogonal to this axis, it is known as a transversely isotropic material. Unidirectionally reinforced composites are of this type, as are for instance muscles. The direction of the fibers is special (oriented along the local x-axis), but the material behaves isotropically in the planes orthogonal to the fibers. Layered materials are also modeled as transversely isotropic: here the direction orthogonal to the layers is special. Figure 5.43 offers an illustration of these two types.
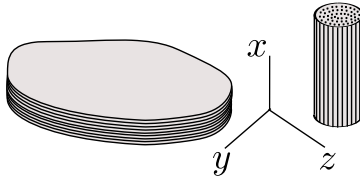
Fig. 5.43. Transversely isotropic model is appropriate for layered or fiber-reinforced materials

The compliance matrix is obtained from the orthotropic compliance by setting $E_2 = E_3$, $\nu_{12} = \nu_{13}$, $G_{12} = G_{13}$, and importantly

$$G_{23} = \frac{E_2}{2(1 + \nu_{23})} \,,$$

requiring five independent constants, E_1, E_2, ν_{12}, G_{12}, and ν_{23}.

Isotropic material. If a material has an infinite number of planes of symmetry of all possible orientations, it is known as an isotropic material. The compliance matrix of isotropic material is based on two material properties, for instance the Young's modulus E and Poisson's ratio ν

$$
\boldsymbol{C} = \begin{bmatrix}
E^{-1}, & -\dfrac{\nu}{E}, & -\dfrac{\nu}{E}, & 0, & 0, & 0 \\[2mm]
-\dfrac{\nu}{E}, & E^{-1}, & -\dfrac{\nu}{E}, & 0, & 0, & 0 \\[2mm]
-\dfrac{\nu}{E}, & -\dfrac{\nu}{E}, & E^{-1}, & 0, & 0, & 0 \\[2mm]
0, & 0, & 0, & G^{-1}, & 0, & 0 \\[2mm]
0, & 0, & 0, & 0, & G^{-1}, & 0 \\[2mm]
0, & 0, & 0, & 0, & 0, & G^{-1}
\end{bmatrix} .
$$

The shear modulus G is not independent, but is expressed as

$$
G = \frac{E}{2(1+\nu)} .
$$

The material stiffness is then

$$
\boldsymbol{D} = \begin{bmatrix}
\lambda + 2G & \lambda & \lambda & 0 & 0 & 0 \\
\lambda & \lambda + 2G & \lambda & 0 & 0 & 0 \\
\lambda & \lambda & \lambda + 2G & 0 & 0 & 0 \\
0 & 0 & 0 & G & 0 & 0 \\
0 & 0 & 0 & 0 & G & 0 \\
0 & 0 & 0 & 0 & 0 & G
\end{bmatrix} ,
$$

where we introduce the Lamé constant

$$
\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}
$$

for convenience.

End Box 18

Box 19. Plane Strain and Plane Stress, and Axisymmetric Model Reduction

We can reduce the three-dimensional elastodynamic model to require solutions in terms of only two space variables (and the time). This is possible by introducing various assumptions for strains or stresses (and some restrictions on the form of the constitutive equation and loads).

Plane strain model reduction

The starting point is the observation that for some solids of uniform cross-section (right angle cylinders), such as the one shown in Fig. 5.44, the set of *assumptions* that **(i)** $u_z(x, y, z) = 0$, and $u_x = u_x(x, y)$, $u_y = u_y(x, y)$ seems to approximate the deformation well. As examples consider a long pipe under internal pressure (well removed from any valves or caps), or a straight concrete dam, or the midsection of a wall panel. The reduction to two space variables will be possible if the third balance equation is satisfied by design
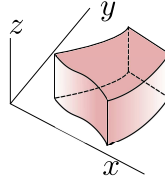
Fig. 5.44. Right-angle cylinder

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z = \rho \ddot{u}_z \ .$$

The identically vanishing u_z also implies $\epsilon_z = 0$. Furthermore, we have for the shear strains

$$\gamma_{zx} = 0, \quad \gamma_{zy} = 0 \ .$$

Provided the constitutive equation allows, the conjugate shear stresses may also vanish. The condition for τ_{zx} is **(ii)**

$$\tau_{zx} = [\boldsymbol{D}]_{5,(1:6)}[\boldsymbol{\epsilon}] = 0 \ ,$$

where $[\boldsymbol{D}]_{5,(1:6)}$ is taken to mean columns 1 through 6 of row 5. This condition is satisfied provided the coefficients of the material stiffness that multiply the nonzero strains are zero,

$$D_{51} = D_{52} = D_{54} = 0 \ .$$

Symmetry also implies $D_{15} = D_{25} = D_{45} = 0$. Analogously, for the stress τ_{zy} the conditions on the material stiffness coefficients applies to row 6 (column 6). The situation is illustrated in Fig. 5.45: coefficients which could be nonzero are filled, coefficients which must be zero are marked as such, and coefficients which are probably best set to zero are blanks (those would couple τ_{xy} and σ_z).

The general *orthotropic* material (See Box 18) complies with these conditions provided **(iii)** one of the orthotropy axes is aligned with the $z-$axis. A more general material model could be devised, but for practical purposes it is sufficient to consider the orthotropic material.

Fig. 5.45. Entries of the material stiffness matrix for plane strain

As a consequence of the assumptions on the displacements, and of the constraints on the form of the material stiffness, the third equilibrium equation becomes

$$\frac{\partial \sigma_z}{\partial z} + \bar{b}_z = 0 \ .$$

But since we already have $\epsilon_x = \epsilon_x(x, y)$ and $\epsilon_y = \epsilon_y(x, y)$, we can write

$$\sigma_z = [\boldsymbol{D}]_{3,(1:2)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \ ,$$

and we see that $\sigma_z = \sigma_z(x, y)$ provided **(iv)** all the coefficients $[\boldsymbol{D}]_{3,(1:2)}$ and $[\boldsymbol{D}]_{(1:2),3}$ are functions of x, y but not of z. Then, the equilibrium equation simply becomes **(v)** a definition of the admissible loads

$$\bar{b}_z = 0 \ .$$

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(v)** are

- top and bottom plane: $\bar{u}_z = 0$, $\bar{t}_x = 0$, $\bar{t}_y = 0$;
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the fixed components depend on z.

The initial conditions satisfy the modeling assumptions and constraints **(i)-(v)** provided the initial displacements and velocities do not depend on z.

Since $\epsilon_z = 0$, the constitutive equation is written for the nonzero normal stresses and the shear stress as

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:2),(1:2)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \ . \tag{5.111}$$

The stress-divergence operator for the plane strain model with just two balance equations and three stresses assumes the form

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & \partial/\partial y \\ 0 & \partial/\partial y & \partial/\partial x \end{bmatrix} \ . \tag{5.112}$$

At this point we are still dealing with the three-dimensional elasticity problem. However, when we substitute all the assumptions and constraints into the Eqs. (7.47), (7.49), (7.53), (7.54), (7.50), and (7.52), the following observations are made. Firstly, the values of all the degrees of freedom in the direction of the z-axis are known to be zero. Secondly, all the loads in this direction are also zero. Consequently, the third equilibrium equation may be ignored, and the test and trial function will have only two components, x and y. Thirdly, all the volume integrals may be precomputed in the z direction as all the integrands are independent of z. Thus, for instance the stiffness matrix integral (7.51) may be written as

$$K_{qp} = \left[\int_V \mathcal{B}^T \left(N_{j(q)}(\boldsymbol{x}) \right) \boldsymbol{D} \mathcal{B} \left(N_{i(p)}(\boldsymbol{x}) \right) \ \mathrm{d}V \right]_{r(q)s(p)} = \tag{5.113}$$

$$\left[\int_S \mathcal{B}^T \left(N_{j(q)}(\boldsymbol{x}) \right) \boldsymbol{D} \mathcal{B} \left(N_{i(p)}(\boldsymbol{x}) \right) \ \Delta z \ \mathrm{d}S \right]_{r(q)s(p)} \ . \tag{5.114}$$

Here Δz is the thickness of the slab in the z direction, S is the cross-sectional area of the cylinder. Only σ_x, σ_y, τ_{xy} (and correspondingly ϵ_x, ϵ_y, γ_{xy}) matter, and the strain-displacement matrix is the transpose of (5.112).

Plane stress model reduction

The plane stress model idealizes the following situation: imagine a thin slab of uniform thickness (the technical term would be ***membrane***) with both plane surfaces at the top and bottom (see Fig. 5.46) traction free, and the boundary conditions on the cylindrical surface independent of the z coordinate. Based on energy considerations we may presume that only the in-plane stresses would play a major role.

Therefore, we formulate a model starting from **(i)** the assumptions $u_x = u_x(x, y)$, $u_y = u_y(x, y)$, and u_z symmetric with respect to the midplane of the membrane. To reduce the problem to two dimensions, we have to try to get rid of the third equilibrium equation.

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z = \rho \ddot{u}_z \ .$$
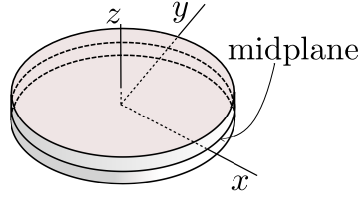
Fig. 5.46. Right-angle cylinder/thin slab/membrane

Neither of the stresses in it will be exactly zero, since the corresponding strains are in general nonzero. Nevertheless, making an inference from the uniform, and small, thickness of the membrane, we adopt yet another assumption, **(ii)** $|\tau_{zx}| \ll 1$, $|\tau_{zy}| \ll 1$, and $|\sigma_z| \ll 1$. Next, **(iii)** the through-the-thickness body load component is assumed to be zero, $\bar{b}_z = 0$. Finally, in order for the third equilibrium equation to be truly negligible, **(iv)** we invoke the symmetry with respect to the midplane, and establish that the integral of this equation through the thickness of the membrane will vanish, i.e. the resultant orthogonal to the plane of the membrane will vanish

$$\int_{-h/2}^{h/2} \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z \; \mathrm{d}z = \int_{-h/2}^{h/2} \rho \ddot{u}_z \; \mathrm{d}z = 0 \; .$$

Furthermore, in order to comply with the symmetry requirement, we shall assume that **(v)** the material is orthotropic, with one orthotropy axis orthogonal to the midplane. The material stiffness matrix will therefore have the appearance shown in Fig. 5.47.

Fig. 5.47. Entries of the material stiffness matrix for plane stress

As the last step, the strain ϵ_z is eliminated from the constitutive equation. Assuming $\sigma_z \approx 0$, we have

$$\sigma_z = [\boldsymbol{D}]_{3,(1:3)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} \approx 0 \; ,$$

which gives

$$\epsilon_z = -D_{33}^{-1}[\boldsymbol{D}]_{3,(1:2)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \; .$$

Therefore, we obtain

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \big[[\boldsymbol{D}]_{(1:2),(1:2)}, [\boldsymbol{D}]_{(1:2),3} \big] \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} =$$

$$\big([\boldsymbol{D}]_{(1:2),(1:2)}, -D_{33}^{-1}[\boldsymbol{D}]_{(1:2),3}[\boldsymbol{D}]_{3,(1:2)} \big) \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \; , \tag{5.115}$$

which together with $\tau_{xy} = D_{44}\gamma_{xy}$ may be substituted into the first two equilibrium equations, rendering them functions of x, y only, provided **(vi)** the body load is $\bar{b}_x = \bar{b}_x(x, y)$, $\bar{b}_y = \bar{b}_y(x, y)$.

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(vi)** are

- top and bottom plane traction-free ($\bar{t}_z = 0$, $\bar{t}_x = 0$, $\bar{t}_y = 0$);
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the fixed components depend on z.

The initial conditions satisfy the modeling assumptions and constraints **(i)-(vi)** provided the initial displacements and velocities do not depend on the z coordinate.

Model reduction for axial symmetry

The last model reduction approach to be discussed in this chapter, is the case of *torsionless* axial symmetry (Fig. 5.48). The main assumption is that **(i)** all planes passing through the y axis are symmetry planes. (The y axis will be referred to as the axis of symmetry.) Therefore, points in any particular cross-section move only in the radial and axial direction, and do not leave the plane of the cross-section (the circumferential displacement is identically zero). Furthermore, points on circles in planes orthogonal to the axis of symmetry, with centers on the axis of symmetry, experience the same radial and axial displacements.
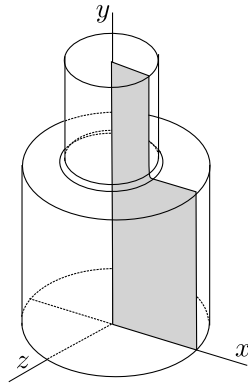
Fig. 5.48. Axially symmetric geometry

Let us consider one particular cross-section, for instance as indicated in Fig. 5.48. Since the displacements in the plane of the cross-section are symmetric with respect to the axis of symmetry, we will consider only the part to one side of the axis of symmetry (hatched in Fig. 5.48). The reduction from three equations of equilibrium to two equations in the plane of the cross-section, where x is the radial direction, and y is the axial direction, will succeed provided the equilibrium equation in the direction orthogonal to the plane of the cross-section (z) is satisfied. Thus, we consider

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z = \rho \ddot{u}_z ,$$

where as the first simplification we note $u_z = 0 \rightarrow \ddot{u}_z = 0$. Furthermore, by symmetry we must have $\bar{b}_z = 0$.

Concerning the stresses: As indicated in Fig. 5.49, by assumption the circles in planes orthogonal to the axis of symmetry are transformed by the deformation again into circles in planes orthogonal to the axis of symmetry. Therefore, right angles between the plane of the cross-section and the tangent to the circle where it intersects the cross-section will remain right angles, and the shear strains are $\gamma_{zx} = 0$, and $\gamma_{zy} = 0$. If the material is **(ii)** orthotropic, with one axis of orthotropy orthogonal to the cross-section (for instance a wound composite, with the reinforcing fibers running approximately in circles around the axis of symmetry), we may conclude that $\tau_{zx} = 0$, and $\tau_{zy} = 0$. Finally, since

the material on a given circle experiences the same stress in all cross sections, $\partial \sigma_z / \partial z = 0$ (while in general $\sigma_z \neq 0$). Conclusion: the equation of motion in the z direction for each cross-section plane (and in particular the one in which the two-dimensional model is formulated) is satisfied exactly.
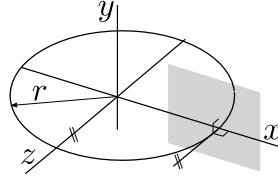
Fig. 5.49. Axially symmetric geometry: geometry of circles in planes orthogonal to the axis of symmetry

It remains to express the circumferential strain in terms of the displacements in the plane of the cross-section. By inspection of Fig. 5.49, displacement of the circle in the y direction does not change its circumference, while displacement radially means that the circle of radius x will experience strain

$$\epsilon_z = \frac{2\pi(x + u_x) - 2\pi x}{2\pi x} = \frac{u_x}{x} .$$

The strain-displacement operator for this model may therefore be written as

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ 1/x & 0 \\ \partial/\partial y & \partial/\partial x \end{bmatrix} . \tag{5.116}$$

The stress divergence operator (the transpose of (5.116)) gives the reduced form for the equations of equilibrium (5.100)

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\sigma_z}{x} + \bar{b}_x = \rho \ddot{u}_x$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \bar{b}_y = \rho \ddot{u}_y$$

where the presence of σ_z should be noted: recall that the equation of motion in the radial direction holds for a wedge-shaped element, hence the contribution of the circumferential stress to the radial direction must be included.

The constitutive equation is obtained from the full three-dimensional relationship by extracting appropriate rows and columns

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:3),(1:3)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \end{bmatrix} . \tag{5.117}$$

End Box 19

Box 20. Boundary conditions for stress analysis

The boundary conditions for general three-dimensional stress analysis may be expressed as (see (5.90))

$$t_i = (\mathcal{P}_n \boldsymbol{\sigma})_i = \bar{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \tag{5.118}$$

as the **_traction_** (natural) **_boundary condition_** for the ith component, and

$$u_i = \bar{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \tag{5.119}$$

as the **_displacement_** (essential) **_boundary condition_** for the ith component. When setting up a finite element model, note that the natural boundary condition need only be specified explicitly when $\bar{t}_i \neq 0$; the case of $\bar{t}_i = 0$ is implicitly active when we say nothing. In particular, no boundary condition needs to be explicitly defined for any component for a traction-free surface.

End Box 20

Box 21. Derivation of weighted residual equations for stress analysis

Incorporation of the natural boundary conditions

The natural boundary condition leads to the residual

$$r_{t,i} = (\mathcal{P}_n \boldsymbol{\sigma})_i - \bar{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \;, \tag{5.120}$$

which will be incorporated into the balance residual equation, and the displacement boundary condition (5.119) gives the residual

$$r_{u,i} = u_i - \bar{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \;, \tag{5.121}$$

that will be made zero by the choice of the trial functions (which takes care of the step 2).

The weighted residual equations are integrals of the residuals over the corresponding surface. Since the displacement boundary condition residual is identically zero, it may be ignored. The traction boundary condition residual equation reads

$$\int_{S_{t,i}} r_{t,i} \vartheta_i \, \mathrm{d}S \;,$$

or, expanded,

$$\int_{S_{t,i}} r_{t,i} \vartheta_i \, \mathrm{d}S = \int_{S_{t,i}} \left[(\mathcal{P}_n \boldsymbol{\sigma})_i - \bar{t}_i \right] \vartheta_i \, \mathrm{d}S = 0 \;. \tag{5.122}$$

Rearranging the balance equation (5.100) into the residual form leads to

$$\boldsymbol{r}_B = \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \bar{\boldsymbol{b}} - \mathcal{B}^T \boldsymbol{\sigma} \;, \tag{5.123}$$

which is a statement of imbalance when the force residual is nonzero.

The balance weighted residual reads

$$\int_V \boldsymbol{\vartheta} \cdot \boldsymbol{r}_B \, \mathrm{d}V = \int_V \boldsymbol{\vartheta} \cdot \left(\rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \bar{\boldsymbol{b}} - \mathcal{B}^T \boldsymbol{\sigma} \right) \, \mathrm{d}V \;, \tag{5.124}$$

where $\boldsymbol{\vartheta}$ is a vector test function (with three components). At this point, we only require that the test function be sufficiently smooth for the integral to exist. The dot product of the residual and the vector test function is written in the "dot" form; when the weighted residual equation is written in terms of the components, transposes must be used as

$$\int_V \boldsymbol{\vartheta} \cdot \boldsymbol{r}_B \, \mathrm{d}V = \int_V [\boldsymbol{r}_B]^T [\boldsymbol{\vartheta}] \, \mathrm{d}V = \int_V [\boldsymbol{\vartheta}]^T [\boldsymbol{r}_B] \, \mathrm{d}V \;.$$

Shifting of derivatives

While the first two terms on the right-hand side of (5.123) present no difficulties, the stress term needs to be treated similarly to the previous two models to move one derivative from the stress to the test function. Therefore, in the next few paragraphs we focus on the integral

$$\int_V \boldsymbol{\vartheta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V \ .$$

It will be sought as one constituent of the chain-rule result (analogously to Eq. (3.133) for the heat conduction). The inner product of $\boldsymbol{\vartheta}$ and $\boldsymbol{\sigma}$ may be expressed using the vector-stress vector dot product operator (5.90) in the form $\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}$ (which is a vector). Therefore, we need an identity for the chain rule applied to the divergence $\mathrm{div}\left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right)$: in one

$$\mathrm{div}\left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right) = (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} + \boldsymbol{\vartheta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \ . \tag{5.125}$$

It may not be immediately clear *why* the right-hand side has this form, but to verify this formula is straightforward, albeit tedious. Expressing the rightmost term of (5.125) by the other two, we obtain

$$\int_V \boldsymbol{\vartheta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_V \mathrm{div}\left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right) \, \mathrm{d}V - \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V \ .$$

The divergence theorem (2.18) may be applied to the first term on the right to yield

$$\int_V \boldsymbol{\vartheta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right) \cdot \boldsymbol{n} \, \mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V \ .$$

The traction boundary condition (5.118) references $\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}$. To extricate this form from $\left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right) \cdot \boldsymbol{n}$ we note that the result of this dot product is a scalar (a number). This indicates that the stress is involved in a double dot product: the stress tensor is dotted with one vector, which is subsequently dotted with the second vector. Indeed, it is easily verified by multiplying through that

$$\left(\mathcal{P}_{\boldsymbol{\vartheta}}\boldsymbol{\sigma}\right) \cdot \boldsymbol{n} = \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right) \cdot \boldsymbol{\vartheta} \ .$$

As a result we obtain

$$\int_V \boldsymbol{\vartheta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \boldsymbol{\vartheta} \cdot \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right) \, \mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V \ .$$

It will be useful to summarize the balance weighted residual (5.124) now as

$$\int_V \boldsymbol{\vartheta} \cdot \boldsymbol{r}_B \, \mathrm{d}V = \int_V \boldsymbol{\vartheta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}V - \int_V \boldsymbol{\vartheta} \cdot \overline{\boldsymbol{b}} \, \mathrm{d}V$$
$$- \int_S \boldsymbol{\vartheta} \cdot \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right) \, \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V \ . \tag{5.126}$$

Boundary conditions

The surface will now be split, for each component, into the part where traction is known, and the part where displacement is being fixed

$$\int_S \boldsymbol{\vartheta} \cdot \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right) \, \mathrm{d}S = \sum_{i=x,y,z} \int_{S_{t,i}} \vartheta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \, \mathrm{d}S + \int_{S_{u,i}} \vartheta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \, \mathrm{d}S \ ,$$

where $\vartheta_i = \left(\boldsymbol{\vartheta}\right)_i$ is the ith component of the test function. On the $S_{t,i}$ subset the traction component i is known from (5.118), but on the $S_{u,i}$ subset of the bounding surface, the component i of the

traction is not known, it represents the reaction. To be able to ignore the reactions, we will resort to the same trick as for the other PDE models, namely we will put in place the requirement

$$\vartheta_i = 0 \quad \text{on} \quad S_{u,i} \ .$$

Therefore, with the constraint on the trial function to satisfy the essential boundary conditions on $S_{u,i}$, and a constraint on the test function to vanish on $S_{u,i}$, we have the weighted balance residual

$$\int_V \boldsymbol{\vartheta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\vartheta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \vartheta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V \tag{5.127}$$

To the weighted balance residual (5.127) we now add the weighted residuals of the natural boundary condition (5.122) and set the result equal to zero.

$$\int_V \boldsymbol{\vartheta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\vartheta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \vartheta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V$$

$$+ \sum_{i=x,y,z} \int_{S_{t,i}} \vartheta_i \left[\left(\mathcal{P}_n\boldsymbol{\sigma}\right)_i - \overline{t}_i\right] \ \mathrm{d}S = 0 \tag{5.128}$$

We can see that the terms with $\vartheta_i \left(\mathcal{P}_n\boldsymbol{\sigma}\right)_i$ cancel and we obtain the final form of the **weighted residual equation**

$$\int_V \boldsymbol{\vartheta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\vartheta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \left(\boldsymbol{\vartheta}\right)_i \overline{t}_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V = 0$$

$$u_i = \overline{u}_i \quad \text{and} \quad \left(\boldsymbol{\vartheta}\right)_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \ .$$

$$\tag{5.129}$$

So far we have been using the velocity and the stress vector for convenience and brevity, but to produce a displacement-based computational model these will have to be replaced by references to the displacement field. Using

$$\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \quad \Rightarrow \quad \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}^2\boldsymbol{u}}{\mathrm{d}t^2} = \ddot{\boldsymbol{u}} \ ,$$

and the constitutive equation (5.109) and the displacement-strain relation (5.107), the form that goes into the discretization process reads

$$\int_V \boldsymbol{\vartheta} \cdot \rho\ddot{\boldsymbol{u}} \ \mathrm{d}V + \int_V (\mathcal{B}\boldsymbol{\vartheta}) \cdot \boldsymbol{D}\mathcal{B}\boldsymbol{u} \ \mathrm{d}V - \int_V \boldsymbol{\vartheta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \left(\boldsymbol{\vartheta}\right)_i \overline{t}_i \ \mathrm{d}S = 0 \tag{5.130}$$

$$u_i = \overline{u}_i \quad \text{and} \quad \left(\boldsymbol{\vartheta}\right)_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \ .$$

Box 22. Method of weighted residuals as the principle of virtual work

An alternative route to Eq. (5.129) is via the **principle of virtual work**. The test function is interpreted as a **virtual displacement**. The constraint on the test function is postulated a priori: virtual displacement is **kinematically admissible**. The various terms in (5.129) are interpreted as the virtual work of the inertial forces, applied body load, applied tractions, and internal forces.

This approach tends to seem somewhat arbitrary, since a number of concepts are postulated to be taken on faith. In this book we therefore avoid this viewpoint. Nevertheless, it may be useful to be aware of these possible interpretations of the various terms as virtual quantities (especially work).

End Box 22

Box 23. Integration for two-dimensional models

As discussed earlier, the volume integrals for plane strain, such as the one required for the stiffness matrix, incorporate the third dimension as a "thickness". Integration through the third dimension reduces to multiplication with the thickness.
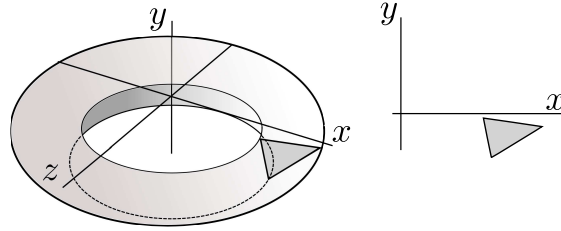
Fig. 5.50. Volume integration for axisymmetric analysis

The volume integrals for the axial-symmetry model reduction approach differ from plane stress or plane strain in that the "gratis" integration in the third direction is performed along the circumference of the rotationally symmetric body, not "through the thickness". For instance, the stiffness matrix integral (7.51) may be written as

$$K_{qp} = \left[\int_V \mathcal{B}^T \left(N_{j(q)}(\boldsymbol{x}) \right) \boldsymbol{DB} \left(N_{i(p)}(\boldsymbol{x}) \right) \ \mathrm{d}V \right]_{r(q)s(p)} = \tag{5.131}$$

$$\left[\int_S \mathcal{B}^T \left(N_{j(q)}(\boldsymbol{x}) \right) \boldsymbol{DB} \left(N_{i(p)}(\boldsymbol{x}) \right) \ 2\pi x \ \mathrm{d}S \right]_{r(q)s(p)} . \tag{5.132}$$

where $2\pi x \ \mathrm{d}S$ is the volume element of the ring generated by revolving the area $\mathrm{d}S$ around the axis of symmetry y—compare with Fig. 5.50. The volume integration may be performed by numerical quadrature over the area S but using a *volume* Jacobian

$$\int_V f \ \mathrm{d}V = \int_S f 2\pi x \ \mathrm{d}S \approx \sum_{k=1}^{M} f(\xi_k, \vartheta_k) J_{\mathrm{vol}}(\xi_k, \vartheta_k) W_k \ ,$$

where the volume Jacobian is defined as ($\det \left[J(\xi_k, \vartheta_k) \right]$ is the surface Jacobian)

$$J_{\mathrm{vol}}(\xi_k, \vartheta_k) = 2\pi x(\xi_k, \vartheta_k) \det \left[J(\xi_k, \vartheta_k) \right] \ . \tag{5.133}$$

In this way, all volume integrals are performed in exactly the same way, be they defined over a three dimensional manifold (solid), a two-dimensional manifold (surface equipped with a thickness, or swept around an axis of symmetry), a one dimensional manifold (curve equipped with a cross-sectional area), or a zero-dimensional manifold (point endowed with a chunk of volume).

Traction loads, such as prescribed pressure on part of the bounding surface, need to be evaluated with surface integrals. Quite analogously to the volume integration, in the reduced analysis the surface integrals are numerically evaluated along curves, but the Jacobians are surface Jacobians. Compare with Figs. 5.46 and 5.51: for instance, for axial symmetry the surface Jacobian is

$$J_{\mathrm{surf}}(\xi_k) = 2\pi x(\xi_k) \det \left[J(\xi_k) \right] \ . \tag{5.134}$$

Here $\det \left[J(\xi_k) \right]$ is the curve Jacobian (3.153).
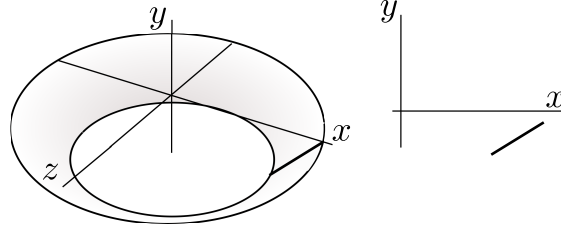
Fig. 5.51. Surface integration for axisymmetric analysis

End Box 23

Box 24. Inadmissible "concentrated" boundary conditions
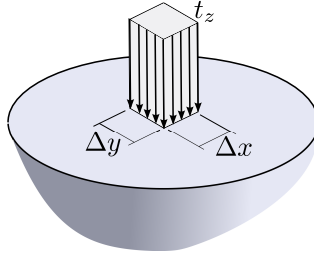
Fig. 5.52. Concentrated force as the limit of traction on infinitesimally small area.

Consider that a resultant force of magnitude F is to be applied along the z-direction, that is orthogonal to the surface shown in Fig. 5.52, as traction t_z applied to the area $\Delta x \Delta y$. As we need to have

$$F = t_z \Delta x \Delta y ,$$

if $\Delta x \to 0, \Delta y \to 0$, the traction component must approach infinity $t_z \to \infty$. In addition, since from the boundary conditions we have $\sigma_z = t_z$, we must conclude that in the immediate vicinity of the infinitesimal patch on the surface, at least some of the stresses must approach infinity as the traction component approaches infinity. The problem of a force applied to an infinite half space has been solved analytically by Boussinesq and others [7], and perhaps the most significant conclusion is that the displacement under the force is infinite. Consequently, for any finite force, the *energy* in the system is *infinite*. As a consequence, we should remember the following caveats when using a *concentrated force as a boundary condition*:

1. Trying to obtain a converged solution for the displacement under the force or for the energy is pointless;
2. Displacements and stresses near the point of application of the force are most likely wrong for any purpose;
3. Displacements or stresses removed from the point of application of the force may be useful, but we have to always ask ourselves whether the concentrated force is truly needed or whether we use it only because we haven't thought the problem through.

Furthermore, as a corollary, we must conclude that if we apply a *displacement boundary condition at a point*, the associated reaction will be zero in the limit, which is wrong, unless we can guarantee for instance from global force equilibrium conditions that the reaction *should* be zero.

Very similar analysis may be performed for a *distributed load along a curve*: Figure 5.53. To maintain a finite value of the distributed load (force per unit length) as $\Delta y \to 0$, the traction t_z must approach infinity. The same list of caveats applies.
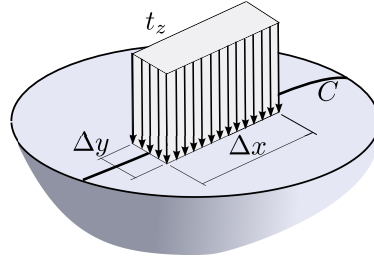
Fig. 5.53. Distributed load along a curve as the limit of traction on infinitesimally small area.

To summarize, the following should be remembered for concentrated force boundary conditions:

Do not use the concentrated force or force along a curve boundary condition *unless* it is essential. Remember the caveats.

Furthermore, this should be remembered for concentrated displacement boundary conditions (support at a point, or support along a curve):

Do not use the concentrated support boundary condition *unless* the associated reaction is guaranteed to be zero.

End Box 24

Box 25. Symmetry and anti-symmetry

Considerable benefits may be often derived when the solution is expected to possess either symmetry, or anti-symmetry.

For the solution to display **symmetry with respect to reflection** in a symmetry plane, all the ingredients that go into the definition of the problem must display the same kind of symmetry: the geometry, the material, the boundary conditions, the initial conditions.

Let us first look at the conditions that must hold for the *displacements* on the plane of symmetry: Figure 5.54. By inspection, we see that the arrow representing displacement at point P is reflected into an arrow at point P' which is best described in components which are (i) in the plane of symmetry: these are the same for both arrows; and (ii) orthogonal to the plane of symmetry: these have opposite signs. Therefore, if P is made to approach the plane of symmetry, its mirror image merges with it when they both reach the plane of symmetry (point M), and since the two displacements then must be the same, we may conclude that the orthogonal component of displacement $u_\perp$ at a point on the plane of symmetry must be zero

$$u_\perp = 0 \ . \tag{5.135}$$

Now for the *tractions* on the plane of symmetry. By the symmetry conditions, the shear part of the traction at point M on the surface with normal $\boldsymbol{n}$ must be equal to the shear part of the traction on the surface with the opposite normal $-\boldsymbol{n}$, i.e.
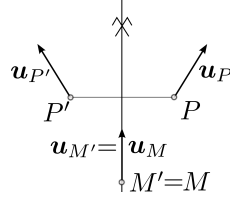
Fig. 5.54. Symmetric displacement pattern.

$$\boldsymbol{t}_{s(-n)} = \boldsymbol{t}_{s(n)} \ .$$

Furthermore, using equation (5.75) we have

$$\boldsymbol{t}_{s(n)} = \boldsymbol{t}_{(n)} - \left(\boldsymbol{n} \cdot \boldsymbol{t}_{(n)}\right) \boldsymbol{n} \ ,$$

and substituting from (5.90) with $\boldsymbol{\sigma}$ being the stress at the point M (which is the same irrespectively of the normal)

$$\boldsymbol{t}_{s(n)} = \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma} - (\boldsymbol{n} \cdot \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}) \, \boldsymbol{n} \ ,$$

$$\boldsymbol{t}_{s(-n)} = \mathcal{P}_{-\boldsymbol{n}}\boldsymbol{\sigma} - (-\boldsymbol{n} \cdot \mathcal{P}_{-\boldsymbol{n}}\boldsymbol{\sigma}) \, (-\boldsymbol{n}) = -\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma} + (\boldsymbol{n} \cdot \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}) \, \boldsymbol{n} = -\boldsymbol{t}_{s(n)} \ .$$

In order for both requirements to be satisfied,

$$\boldsymbol{t}_{s(n)} = \boldsymbol{0} \ , \tag{5.136}$$

must hold.
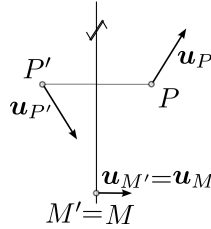
Fig. 5.55. Anti-symmetric displacement pattern.

The state of **anti-symmetry with respect to reflection in a plane** is defined by conditions that specify it as the opposite of symmetry. The situation is illustrated in Fig. 5.55. The arrow representing *displacement* at point P is transformed into an arrow at point P' which in terms of components gives (i) opposite sign parallel with the plane of anti-symmetry; and (ii) same sign in the direction orthogonal to the plane of anti-symmetry. Therefore, when P is made to approach the plane of symmetry and its mirror image merges with it at point M, we conclude that the two components of displacement $u_{\parallel,i}$ at a point on the plane of symmetry must be zero

$$u_{\parallel,i} = 0 \quad \text{for the two in-plane directions } i \ . \tag{5.137}$$

For the tractions, an analysis quite similar to that leading to Eq. (5.136), but applied to the normal component of the traction, leads to the condition

$$t_\perp = 0 \ . \tag{5.138}$$

Therefore, we can summarize the boundary conditions on the plane of symmetry or on the plane of anti-symmetry with the delightfully simple Table 5.2.

The boundary conditions that need to be explicitly prescribed in finite element analyses are boxed in the Table 5.2: zero tractions are incorporated automatically (natural boundary conditions!) and need not be explicitly specified. Note that the unknown tractions are the reactions.

Table 5.2. Boundary conditions on the plane of symmetry or anti-symmetry

Quantity	Symmetry	Anti-symmetry
Tractions ∥	0	unknown
Displacements ∥	unknown	0
Traction ⊥	unknown	0
Displacement ⊥	0	unknown

End Box 25

5.11 Code listings

pnpT3thermal.py

Python
- script

```python
1  # Finite Element Modeling with Abaqus and Python for Thermal and
2  # Stress Analysis
3  # (C)  2017-2020, Petr Krysl
4  #
5  """
6  Strain patterns calculated for a single triangle.
7  """
8
9  from numpy import array, dot, linalg
10
11 ##
12 xall = array([[-1, -1/2], [3, 2], [1, 2]]) #Coordinates of the nodes
13 conn = [1, 2, 3] # The definition of the element, listing its nodes
14
15 gradNpar = array([[-1, -1], [1, 0], [0, 1]])  #Gradients of the basis fncs wrt
       the param. coords
16 zconn = array(conn)-1
17 x = xall[zconn,:] # The coordinates  of the three nodes
18 J = dot(x.T, gradNpar) # Compute the Jacobian matrix
19 gradN = dot(gradNpar, linalg.inv(J))
20
21 def Bn(gradNn):
22     """
23     One-liner function to compute the nodal strain-displacement matrix.
24     """
25     return array([[gradNn[0], 0],
26                   [0, gradNn[1]],
27                   [gradNn[1], gradNn[0]]])
28
29 B1=Bn(gradN[0,:])
30 B2=Bn(gradN[1,:])
31 B3=Bn(gradN[2,:])
32
33 from sympy import symbols, Matrix, simplify
34
35 DeltaT, CTE, E, nu, t  = symbols('DeltaT CTE E nu t')
36 D = E/(1-nu**2)*Matrix([[1, nu, 0], [nu, 1, 0], [0, 0, (1-nu)/2]])
37 eth = DeltaT*CTE*Matrix([1, 1, 0]).reshape(3, 1)
```

```
38 sig = D*eth
39 Se = linalg.det(J)/2
40 F1 = simplify(Se*t*B1.T*sig)
41 F2 = simplify(Se*t*B2.T*sig)
42 F3 = simplify(Se*t*B3.T*sig)
43 print('Thermal forces')
44 print('Node 1:', F1)
45 print('Node 2:', F2)
46 print('Node 3:', F3)
```

Listing 5.1. pnpT3thermal.py

6

How to deal with errors

6.1 Where is this applicable?

It is important to realize that in this chapter we can talk about both the heat conduction finite element models and the stress analysis finite element models. Table 6.1 summarizes the correspondence between the heat conduction and stress analysis models. In each row we have quantities that correspond to each other in the models.

Designation of type	Heat conduction	Stress analysis
Primary quantity	Temperature	Displacement
Derived quantity	Temperature gradient	Strains
Flux quantity	Heat flux	Stress

Table 6.1. Comparison of the heat conduction and stress analysis models

The implication is that when we state something for the heat conduction model in terms of temperature, it will also apply to the stress analysis model expressed in terms of displacement. For instance, we know that the heat conduction finite element model can always represent linear variation of temperature exactly. Analogously, we know that the stress analysis finite element model can exactly represent linear displacements.

The value of this analogy will become obvious below, where when we say something about the error for the much simpler model of heat conduction, it will be directly applicable to the more complicated model of stress analysis.

6.2 Simple examples

First we look at an example. We pick a problem that has an analytical solution so that one can define true errors for the quantity v as

$$E_{v,h} = v_{\mathrm{ex}} - v_h \tag{6.1}$$

as the difference between the true solution v_{ex} and the computed solution v_h, where by the notation $\cdot_h$ we mean that the computed solution was obtained with a finite element mesh and h indicates some measure of the properties of the mesh (here it means the element size).

The variable can be anything of interest in the solution of the problem. For instance, for a heat conduction problem the variable v could stand for the temperature, or the heat flux, or perhaps the temperature gradient at some point (local, or pointwise, quantity). It could also be a global quantity, such as the total heat flux through a face of the domain, or the global range of temperatures (maximum minus minimum).

In the examples below we consider heat conduction through a wall, and we observe the error in the temperature and the error in the heat flux. The appealing characteristic of this problem is its one-dimensional character. The analytical solution is easily derived, and the finite element models are also very straightforward.

6.2.1 Uniformly heated wall

A wall is exposed to given temperature $T = 0^oC$ at both faces. Heat is generated internally at the uniform rate Q. We use a mesh of seven equal-length L2 finite elements. Our goal is to compute the exact error of the temperature and of the heat flux. Set $L = 6$m, $\kappa = 4\text{W/m/}^oK$, $Q = 0.1\text{W/m}^3$.
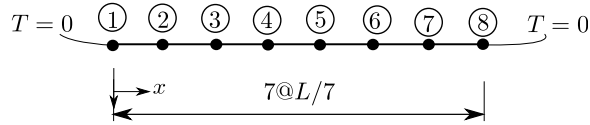
Fig. 6.1. Wall with prescribed temperatures at the boundary and uniform internal heat generation. Finite element mesh of seven L2 elements.

The solution is abbreviated as follows: The numbering of the nodes is shown in Figure 6.1. The free degrees of freedom 1,...,6 are assigned to the nodes 2,...,7. The elements are numbered left to right. Therefore, the elementwise conductivity matrices and source heat load vectors are assembled as

$$[K] = \frac{\kappa}{L/7} \begin{bmatrix} 2, & -1, & 0, & 0, & 0, & 0 \\ -1, & 2, & -1, & 0, & 0, & 0 \\ 0, & -1, & 2, & -1, & 0, & 0 \\ 0, & 0, & -1, & 2, & -1, & 0 \\ 0, & 0, & 0, & -1, & 2, & -1 \\ 0, & 0, & 0, & 0, & -1, & 2 \end{bmatrix}$$

and

$$[F] = \frac{QL}{7} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The solution is

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \frac{QL^2}{49\kappa} \begin{bmatrix} 3 \\ 5 \\ 6 \\ 6 \\ 5 \\ 3 \end{bmatrix}$$

The two degrees of freedom $T_7 = T_8 = 0^oC$ are known, and so we can plot the finite element solution using the trial function expansion

$$T(x) = \sum_{i=1}^{8} N_i(x)T_i$$
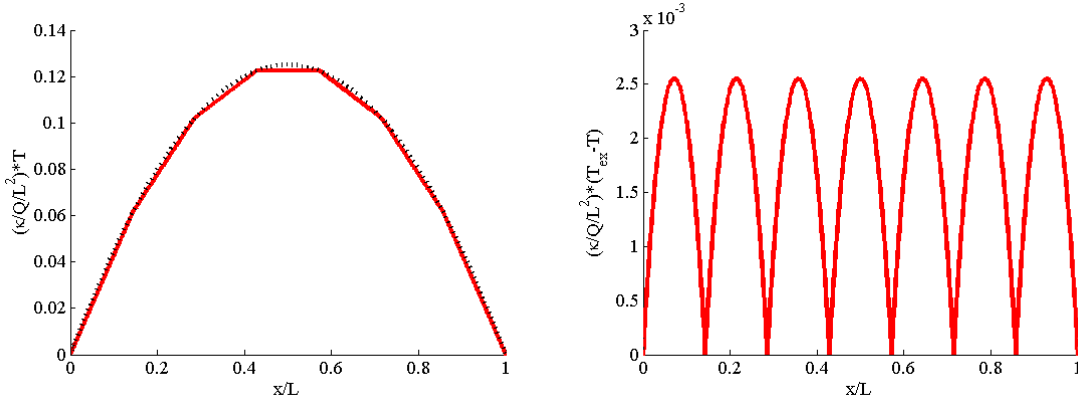
Fig. 6.2. Wall with prescribed temperatures at the boundary and uniform internal heat generation. The temperature distribution on the left, dotted curve exact, solid curve approximate. The error of the temperature on the right.

and compare with the analytical solution. In Figure 6.2 (on the left) the dotted curve is the analytical solution, the finite element solution is the solid curve. The error is there, but perhaps difficult to see at this resolution. It *is* visible that the finite element solution agrees with the exact solution at the nodes (it doesn't happen in general, only in 1-D!). The character of the error of the temperature is clearly brought out in Figure 6.2 (on the right) which displays the error

$$E_{T,h}(x) = T_{\text{ex}}(x) - T_h(x)$$

On each of the seven L2 elements the temperature error is represented by a little parabolic arc. The error is zero at the nodes.

> The error of the primary variable (temperature) apparently depends on the curvature of the exact solution: the finite element solution has curvature zero within each element. So the more curved the exact solution, the bigger the error. Conversely, if the true solution was linear, the finite element solution would match it exactly.

The heat flux is computed using the definition

$$q = -\kappa T'$$

The gradient of the temperature is computed using the definition of the trial function

$$T'(x) = \sum_{i=1}^{8} N_i'(x) T_i$$

It can be computed conveniently element-by-element taking advantage of the fact that over each element only two basis functions are nonzero. We write for the L2 element that connects nodes K, M

$$T'(x) = N_K'(x) T_K + N_M'(x) T_M$$

where we easily work out $N_K'(x) = -1/h$ and $N_M'(x) = +1/h$, with h being the length of the element, so that

$$T'(x) = \frac{T_M - T_K}{h}$$

and the heat flux is within the element K, M

$$q = -\kappa \frac{T_M - T_K}{h}$$

Note that within each element the heat flux is uniform, which goes hand-in-hand with the linear variation of the temperature within the element. For instance, for element 1 we get

$$q = -\kappa \frac{\frac{QL^2}{49\kappa} \times 3 - 0}{L/7} = -\frac{3QL}{7} = -0.2571$$

Compare the calculated value with Figure 6.3. The horizontal lines represent the uniform heat flux within each element. The true heat flux is indicated with the continuous dotted line. The difference
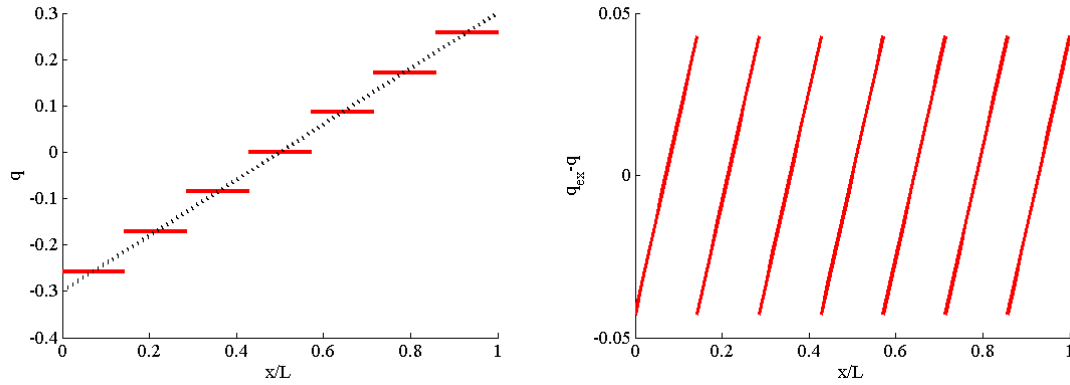
Fig. 6.3. Wall with prescribed temperatures at the boundary and uniform internal heat generation. The heat flux on the left, dotted curve exact, solid curve approximate. Error of the heat flux on the right.

between the discontinuous curve of the finite element heat fluxes and the continuous line of the true heat flux is the error of the heat flux. In Figure 6.3 on the right the error in the heat flux is shown and we see that it is represented by a linear function within each element.

> The finite element solution for the heat flux can only produce constant values within each element. How big can the heat flux error get? The larger the element, the larger it gets: the error of the heat flux depends on the size of the element linearly. Conversely, if we make the elements smaller, the absolute value of the heat flux error will also decrease.

6.3 Interpolation errors

In this section we will consider interpolation errors. The finite element solution in general does not interpolate the exact solution– meaning that the finite element nodal value is different from the value of the exact solution at the node, but it turns out that the interpolation errors are related to the actual errors in the numerical solution. We will not investigate this relationship, suffice it to say that the errors of the actual finite element solution and the errors of the interpolation of the true solution on the finite element mesh are related, and we can understand where errors occur and how to control them in terms of the simpler errors, the errors of interpolation.

We will estimate the difference between the "true" distribution of temperature, $T(\boldsymbol{x})$, and an interpolation of this function on a finite element mesh, $\Pi_h T(\boldsymbol{x})$. Here h means the element "size", or characteristic dimension. Typically, element size is taken to mean edge length, or the diameter of the smallest ball that completely encloses an element –see Figure 6.4.
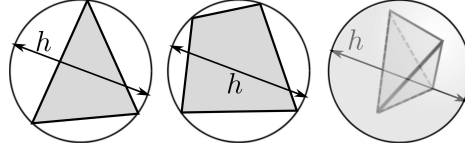
Fig. 6.4. Element size h as a "diameter" of an element: diameter of the tight-fitting circle that encloses a triangle or a quadrilateral, and of the smallest sphere that is circumscribed to a tetrahedron

6.3.1 Interpolation error of temperature

Let us pretend we know the true distribution of temperature. We can approximate this true distribution on the finite element mesh by interpolation: The interpolating function is defined as

$$\Pi_h T(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}_k) \ , \tag{6.2}$$

where $\boldsymbol{x}_k$ is the location of the node k, and $T(\boldsymbol{x}_k)$ is the value of the temperature at the location of the node k. Figure 6.5 offers a visual of what it means to approximate the true curve by a broken line (which is the representation of the interpolating function on a mesh consisting of L2 elements).
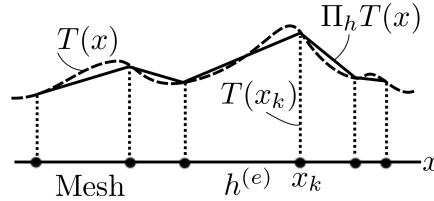
Fig. 6.5. Interpolating the temperature function on a mesh

Then we can write the error of the interpolation as proportional to some power of the element size   See Box 26

$$|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})| \approx h^\beta \ , \tag{6.3}$$

For instance, specializing this to the simplest finite elements, L2 in one dimension, T3 and Q4 in two dimensions, and four-node tetrahedra T4 and six node hexahedra H8 in three dimensions, we can write

$$|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})| \leq C\left(\partial^2 T\right) h^2 \ , \tag{6.4}$$

where we agree to mean by $C\left(\partial^2 T\right)$ some constant whose magnitude depends on the curvatures of the function T. Curvature here means second derivatives of the temperature. (Parabolic distribution would have constant curvature.)

From a practical standpoint, it is really important to realize that the curvature of the temperature distribution is proportional to the first derivatives of the gradient of temperature, and since in our linear model the gradient of temperature is proportional to the heat flux, the curvature of the temperature can be measured by looking at how fast the heat flux changes from point to point. So the constant $C\left(\partial^2 T\right)$ may be understood as measuring the *rate of change of the heat flux* in the immediate neighborhood of a given point.

The value of Equation (6.4) is twofold:

- Firstly, it states that the errors of interpolation will get bigger the higher the curvature of the function of the exact temperature T (that is the faster the heat flux is changing) and the bigger the elements (i.e. the error will increase with h^2). So to find locations in the mesh where the errors are large, we look for places where the heat flux changes strongly, and where the elements are large;

- Secondly, if we are interested in the error at a given location, the curvature of the true solution at that location needs to be considered given, and the Equation (6.4) then says that the error at that location can be controlled by decreasing the element size, as the error will diminish as $O(h^2)$ as $h \to 0$ (order-of estimate: reduce h with a factor of two, and the error will decrease with a factor of four, i.e. $= 2^2$).

6.3.2 Interpolation error of temperature gradient

To estimate errors for the gradient of temperature, we start with the interpolation (6.22), of which we take the gradient. See Box 26 We obtain

$$|\mathrm{grad}T(\boldsymbol{x}) - \mathrm{grad}\Pi_h T(\boldsymbol{x})| \leq C\left(\partial^2 T\right)\gamma h \ . \tag{6.5}$$

Here $C\left(\partial^2 T\right)$ is the same curvature-dependent constant as above: large curvature of the temperature means a large error constant, and vice versa. In addition, there is another factor related to the "shape" of the element. This factor comes in through the needs to differentiate the temperature: poorly shaped elements would tend to amplify errors due to differentiation. More below.

The value of Equation (6.5) is again twofold:

- Firstly, it states that the errors of interpolation for the gradient of temperature will get bigger the higher the curvature of the function of the exact temperature T, the larger the elements (i.e. the error will increase with h), and the larger the quality measure γ (i.e. the worse the shape of the triangle);
- Secondly, considering the curvatures of the temperature function at a fixed location as given, equation (6.5) states that the error can be reduced by making the element size h smaller, so the error will decrease as $O(h)$ as $h \to 0$. Note that this is one order lower than for the temperature itself: when we reduce reduce h with a factor of two, and the error will decrease with the same factor.

The visualize the effect of the shape of the element, let us consider a triangular element with three nodes. To estimate the magnitude of the gradient of the basis function, we invoke the picture of the basis function as a plane that assumes value one at one node and drops off to zero along the opposite edge. Therefore, the largest magnitude of the basis function gradient will be produced by the smallest height in the triangle. The shortest height $d_{\min}$ may be estimated from the radius of the largest inscribed circle, ρ (see Figure 6.6), as $d_{\min} \approx O(\rho)$. This can be linked to the so-called "shape quality" of a triangle using the **quality measure**

$$\gamma = \frac{h}{\rho} \ . \tag{6.6}$$
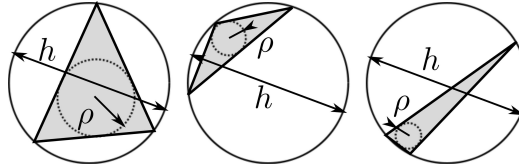
Fig. 6.6. Triangle quality measures using the radius of the inscribed circle and the diameter of the circumscribed circle. Good (almost equilateral) triangle on the left; bad triangles (obtuse, needle-like) on the right.

Considered from yet another angle, take the right angle triangle in Figure 6.7. The temperatures at the three corners of the triangle are T_1, T_2, and T_3. Because of its shape we can calculate the gradient across this triangle to be

$$\mathrm{grad}\Pi_h T(\boldsymbol{x}) = \left[\frac{T_2 - T_1}{d}, \frac{T_3 - T_1}{b}\right] \tag{6.7}$$

Now let's take $T_1 = T_2$. The correct slope in the x direction is then zero.

$$\mathrm{grad}\,\Pi_h T(\boldsymbol{x}) = \left[0, \frac{T_3 - T_1}{b}\right] \tag{6.8}$$

If the triangle is very skinny, $d \ll b$, any small change of T_2 could lead to large error in the gradient. For instance, change T_2 to read $T_2 = T_1 + \varepsilon(T_3 - T_1)$, where we take ε very small (for instance $\varepsilon = 1/1000$). Then $T_2 - T_1 = \varepsilon(T_3 - T_1)$, and the gradient then becomes

$$\mathrm{grad}\,\Pi_h T(\boldsymbol{x}) = \left[\frac{\varepsilon(T_3 - T_1)}{d}, \frac{T_3 - T_1}{b}\right] = \left[\frac{\varepsilon b}{d}\frac{(T_3 - T_1)}{b}, \frac{T_3 - T_1}{b}\right] \tag{6.9}$$

It is now easy to see that if the fraction $\varepsilon b/d$ is comparable to 1.0, a very small difference in temperatures at the second node ($T_2 = T_1 \longrightarrow T_2 = T_1 + \varepsilon(T_3 - T_1)$) results in an error of the gradient of the temperature which is comparatively speaking huge.
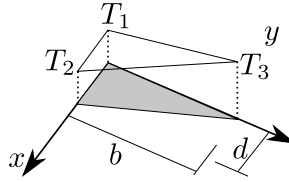
Fig. 6.7. Illustration of how a poorly shaped triangle can magnify errors in temperature to become large errors in temperature gradient

Importantly, Equation (6.5) allows us to make a general observation: the quantity calculated in the finite element solution is the temperature, the gradient (or, alternatively, the heat flux) is obtained by *differentiation* of the computed temperature, which immediately results in a reduction of the order of dependence on the mesh size. Phrased differently: the temperature results will converge faster than the temperature-gradient results (or, equivalently, heat flux results), because h^2 approaches zero much quicker than h as $h \to 0$.

> For linear elements the error of the heat flux depends linearly on the element size. It also depends on the rate of change of the heat flux: the faster the heat flux changes around the point, the greater the error will be at that location. Finally, the error depends on the shape of the elements: poorly shaped elements will leads to larger errors.

6.4 Estimation of True Solution

The observation that the error depends on the element size we made earlier in this chapter is very important. The element size is one of the knobs we can use to control the error. We can create a so-called **graded mesh** by changing the element size to reflect the distribution of error: large error – small elements and vice versa. If we set up the finite element procedure to solve the problem repeatedly with graded meshes designed to respect the distribution of error, we obtain the so-called **adaptive refinement** technique, or $h-$**adaptive refinement method**. The h stands for the element size as the control of the error.

On the other hand, we could try to reduce the error by increasing the number of terms matched in the Taylor series (6.20). This could be achieved by using higher order polynomials as basis functions. The resulting procedure would be called the $p-$**adaptive refinement method**, where the p stands for the polynomial order as the control of the error.

In this book, we will select the polynomial order of the elements only by choosing the element type, linear or quadratic. We will not do this adaptively which would involve increasing the polynomial order in a targeted fashion, locally, and to much higher order than just quadratic. Hence, we will focus on the control of the error by adjusting the element size.

6.4.1 The uses of the element-size control

As discussed above, we will adopt the point of view of the $h-$adaptive refinement method. In order to be able to control the error, it must depend on the element size h, and furthermore it must be expressible as a *positive* (but not necessarily integral) *power* of h. Only then decreasing h will lead to a reduction of the error. For instance, for quantity q we require for the error

$$E_q(h) = q_{\text{ex}} - q_h \approx Ch^\beta \,, \tag{6.10}$$

as the error then can be reduced by decreasing the element size

$$\lim_{h \to 0} E_q(h) = \lim_{h \to 0} Ch^\beta = 0 \quad \text{for} \;\; \beta > 0 \,. \tag{6.11}$$

The exponent of the element size β is called the ***convergence rate*** (or rate of convergence).

> We *require* the error to depend on the mesh size, but how do we get it? The answer is: by element design. If we use a properly designed finite element, one that is appropriate for the task at hand, the properties of the FE will guarantee convergence.

Equation (6.5) tells us how to reduce the error. Consider that at a given location $\boldsymbol{x}$, $C\left(\partial^2 T\right)$ cannot be controlled as it is determined by the behavior of the exact solution at $\boldsymbol{x}$. What we *can* influence is the shape of the elements (γ), and the element size (h). Now let the point $\boldsymbol{x}$ range across the computational domain. At some locations $C\left(\partial^2 T\right)$ is small, and at others it is large.
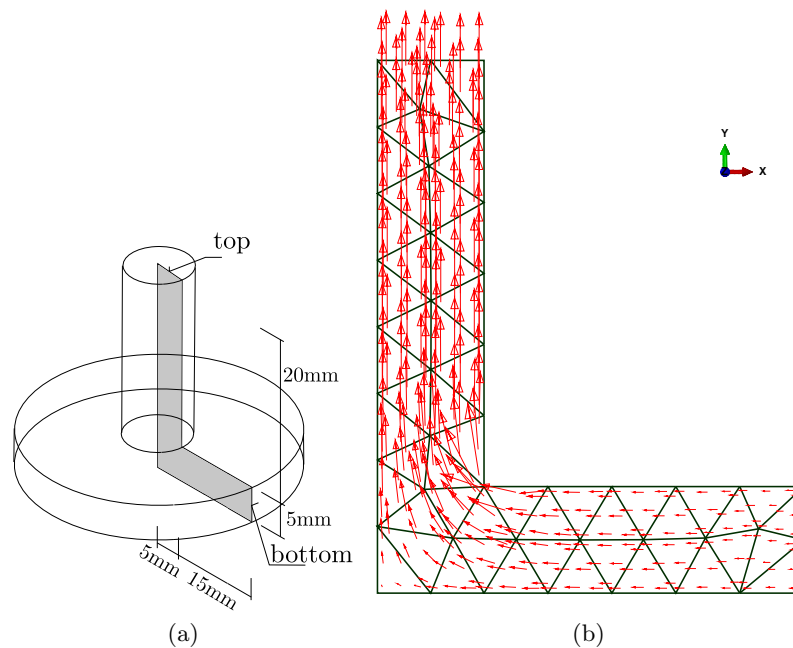
Fig. 6.8. Copper axially-symmetric part. Temperatures prescribed at the top and bottom surfaces, all others insulated. (a) Geometry, the generating section is shown shaded. (b) Heat flux, shown as arrows whose length indicates the magnitude.

As an illustration, let us contemplate Figure 6.8. The copper part is axially symmetric, and temperatures are prescribed at the top (circular) and bottom (cylindrical) surfaces, while all the other surfaces are insulated. The generating cross-section was used for axially symmetric modeling.

The heat flux arrows are shown at the quadrature points. The constant $C\left(\partial^2 T\right)$ will be large where the heat flux arrows either strongly change direction or strongly stretch or shrink (or both). Intuitively, those are the locations where reducing the error will make a difference. In Figure 6.8 this is the vicinity of the reentrant corner. The error is not large in the upper vertical portion of the part even though the heat flux arrows are large: the heat flux does not change there, all the arrows are roughly the same size and the same direction.

In the locations where the error constant is large, the elements should be made smaller. How much? The answer to that question is somewhat elusive: in general a trial and error procedure or iteration will be required to construct a mesh that delivers the answer within acceptable error bounds. Automatic procedures to estimate the *relative* desired element size are becoming available in commercial softwares, and the trend is nowadays to provide some means for the user of finite element softwares to estimate and control error. Abaqus provides an automatic adaptive procedure. Figure 6.9(a) shows the temperature distribution on the adaptively refined mesh (the fifth iteration of the process). Clearly the adaptive process thinks elements should be made much smaller around the corner, and can be actually very large in the top portion of the domain.

(a)   (b)
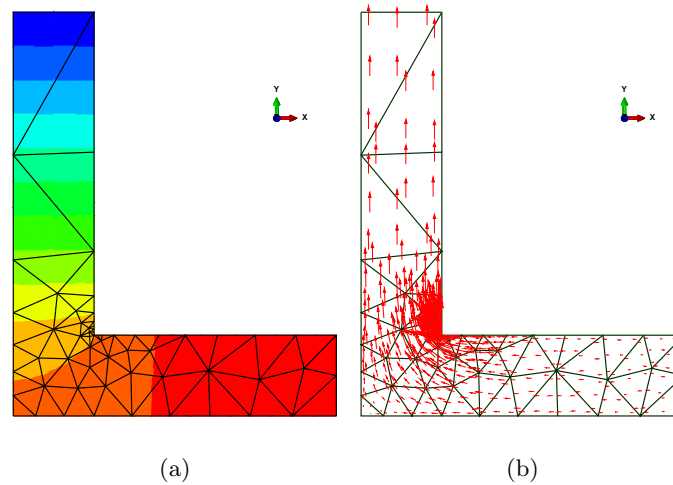
Fig. 6.9. Copper axially-symmetric part. Temperatures prescribed at the top and bottom surfaces, all others insulated. Adaptively refined mesh and the solution. (a) Temperature distribution. (b) Heat flux, shown as arrows.

> To summarize, the $h-$adaptive method will attempt to control the error by designing **graded meshes**, with small elements located in regions of expected high error, and proportionally large elements elsewhere.
>
> A very good indication of the presence of large errors are the so-called **reentrant corners** (concave corners), where the solution typically displays singularities in the form of infinite curvature(s) of the temperature directly in the corner: the heat flux direction (and perhaps also the magnitude) changes discontinuously in the corner.

6.5 Richardson extrapolation

The second important use of the fact that the error decreases asymptotically as some power of the element size is a procedure to improve the estimate of the exact (true) answer based on a series of calculated solutions: in other words, element-size-based extrapolation.

Richardson extrapolation is a way of extracting an asymptotic estimate of some quantity of interest from a series of its computed values. If we assume that the true error in the quantity q may be expanded in a Taylor series at element size $h = 0$, we may write

$$E_q(h) = q_{\mathrm{ex}} - q_h \approx Ch^\beta \,, \tag{6.12}$$

where q_{ex} is the unknown true value of the quantity, q_h is the approximate value for nonzero h, $E_q(h)$ is the true error, C is an unknown constant of the leading term h^β, with β, again, unknown. Provided C, and β do not depend on h for small element sizes (this is presumed to hold in the so-called **asymptotic range**), we might be able to compute all of the three quantities q_{ex}, C, and β, if three numerical solutions, q_{h_i}, are obtained for three different element sizes h_i, $h_1 < h_2 < h_3$. (It does not matter whether the solutions are obtained with uniform or graded meshes, but there's a caveat: see below.)

Remarkably, the estimate of the exact solution is available from a very easily solvable equation if the condition

$$\frac{h_2}{h_1} = \frac{h_3}{h_2} = \alpha \,,$$

holds, where $\alpha < 1$ is the so-called refinement factor, as we may then combine

$$\frac{q_{\mathrm{ex}} - q_{h_1}}{q_{\mathrm{ex}} - q_{h_2}} = \frac{h_1^\beta}{h_2^\beta} \quad \text{and} \quad \frac{q_{\mathrm{ex}} - q_{h_2}}{q_{\mathrm{ex}} - q_{h_3}} = \frac{h_2^\beta}{h_3^\beta} \,,$$

to yield an estimate of the true solution as

$$q_{\mathrm{ex}} = \frac{q_{h_2}^2 - q_{h_1}q_{h_3}}{2q_{h_2} - q_{h_1} - q_{h_3}} \,. \tag{6.13}$$

Alternatively put, we have the **estimated true error**

$$E_q(h_j) = q_{\mathrm{ex}} - q_{h_j} \,. \tag{6.14}$$

It is then straightforward to extricate the other two quantities. The constant C is of limited value, but the exponent β is the **rate of convergence**. The computation is implemented in the Python function `richextrapol`. The detailed derivation of the Richardson extrapolation procedure (applicable even for non-uniform refinement factor α) is provided in $\boxed{\text{See Box 27}}$.

Extrapolation is a tricky procedure in general. Therefore, before we attempt the Richardson extrapolation (6.13) to estimate the true error, the so-called **approximate error** should be considered.

$$E_{q,j} = q_{h_{j+1}} - q_{h_j} \,. \tag{6.15}$$

It relies only on computed quantities, $q_{h_{j+1}}$, and q_{h_j} for two different meshes with element sizes h_{j+1} and h_j. We can expect it to be more "honest" about the approach to the true solution than the extrapolation formula. The property of the approximate error that makes it so useful is that it depends on the element size in the same way as the true error. We can see that by adding and subtracting the true value to/from the approximate error as

$$E_{q,j} = q_{h_{j+1}} - q_{h_j} = q_{h_{j+1}} - q_{h_j} + q_{\mathrm{ex}} - q_{\mathrm{ex}} \,.$$

Regrouping the terms we obtain

$$E_{q,j} = \left(q_{h_{j+1}} - q_{\mathrm{ex}}\right) + \left(q_{\mathrm{ex}} - q_{h_j}\right) = -E_q(h_{j+1}) + E_q(h_j) \,. \tag{6.16}$$

i.e. as the difference of the true errors for the element sizes h_j and h_{j+1}. Consequently, we see that in the limit of the element size decreasing towards zero the approximate error will behave as a

power of the element size, and the convergence rate will be the same as that of the true error of equation (6.12). We show this by introducing (6.12) into (6.16) to obtain

$$E_{q,j} = -E_q(h_{j+1}) + E_q(h_j) = -Ch_{j+1}^{\beta} + Ch_j^{\beta}$$

If all the meshes are related by a constant mesh refinement factor $\alpha < 1$ such that the element sizes for two successive meshes are $h_{j+1} = \alpha h_j$, we arrive at

$$E_{q,j} = -Ch_{j+1}^{\beta} + Ch_j^{\beta} = -C(\alpha h_j)^{\beta} + Ch_j^{\beta} = \left(1 - \alpha^{\beta}\right)\left(Ch_j^{\beta}\right) = \left(1 - \alpha^{\beta}\right) E_q(h_j) \qquad (6.17)$$

In words, the approximate error $E_{q,j}$ is within a constant factor of the true error $E_q(h_j)$, provided a constant refinement factor α is used.

6.6 Graded meshes

A word on the meaning of the *element size in graded meshes* is in order. The element size in *graded* meshes varies from point to point, as opposed to *uniform* meshes, where the element size does not vary from location to location. If we take one particular graded mesh M_0, and produce a series of progressively finer meshes from M_0 by scaling the element size as a function of $\boldsymbol{x}$ by the same number $\alpha < 1$ everywhere in the domain, we may extrapolate with respect to the **refinement factor** α, instead of the element size which would be the natural choice for uniform meshes. For instance, mesh M_1 would be produced with element size $h_1(\boldsymbol{x}) = \alpha h_0(\boldsymbol{x})$, mesh M_2 with element size $h_2(\boldsymbol{x}) = \alpha h_1(\boldsymbol{x}) = \alpha^2 h_0(\boldsymbol{x})$, and so on. Note that we are keeping the refinement factor constant when going from mesh M_0 to M_1 to M_2 and so on. The extrapolation formula (6.13) relies on this.

Graded meshes not produced by the process just described are *not* suitable for extrapolation. An example is shown in Figure 6.10 which presents three meshes for an adaptive solution process (with quadratic triangles). The goal of the adaptive remeshing is to design a mesh that distributes the error equally to all the elements and consists of the target number of elements. As we can see, the mesh is strongly focused around the perimeter of the hole. Importantly, while from mesh 1 to mesh 2 to mesh 3 the elements around the hole are getting smaller, at the same time they are actually getting much bigger near the right-hand-side edge (with the distributed tensile load). So, there is no consistency in the change of the element size within the mesh: while at one point the elements are getting smaller, elsewhere they may be getting larger. It is not possible to extrapolate from the results for this mesh sequence, as there is *no* variable with respect to which the extrapolation could be performed.

Abaqus
- CAE file
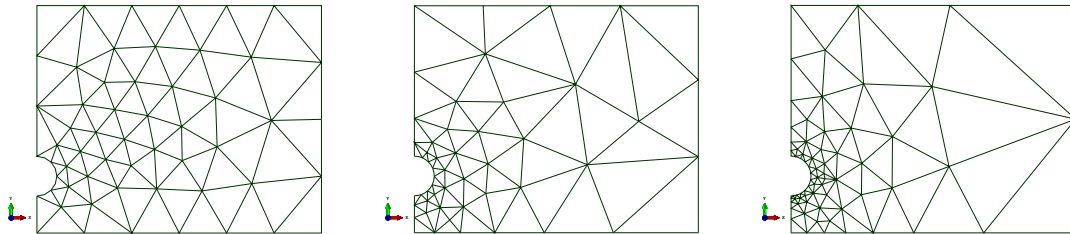
Fig. 6.10. Thin aluminium slab with a circular opening. Left to right: Initial mesh (mesh 1), mesh 2, mesh 3. Each of the meshes corresponds to one adaptive solution in succession.

Meshes produced by adaptive processes (such as those of Figure 6.10) will not allow extrapolation to be performed. There is no constant refinement factor α to produce mesh $j+1$ from mesh j, $h_{j+1}(\boldsymbol{x}) = \alpha h_j(\boldsymbol{x})$. Rather, the meshes are generated from the condition that the local error be equilibrated across the mesh (each element should have about the same "amount of error"). This naturally means that the refinement factor varies from point to point. The precondition of the application of the Richardson extrapolation is therefore absent.

6.6.1 Thin aluminium slab: estimation of maximum tensile stress

The maximum tensile stress in a thin aluminium slab with a circular hole placed off-center was studied by Szabó and Babuška [8]. Using the p-version of the finite element method they estimated this stress as 259.2 MPa. The dimensions of the three-dimensional slab are shown in Figure 6.11, and the material properties were taken as representative of the aluminium alloy 6061-T6, $E = 70$ GPa, $\nu = 0.35$. The part is here studied as a plane-stress case (as in [8]). The plane of symmetry was applied, plus a point support in the y direction to suppress rigid body displacement.
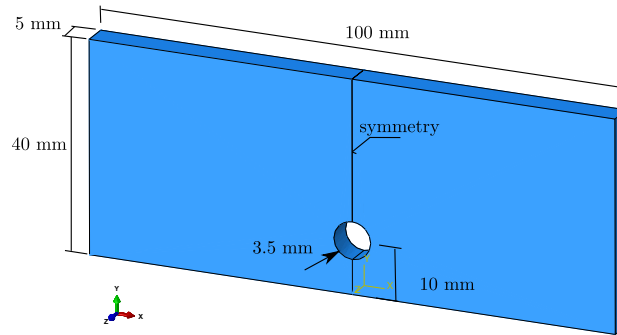
Fig. 6.11. Thin aluminium slab: dimensions and orientation
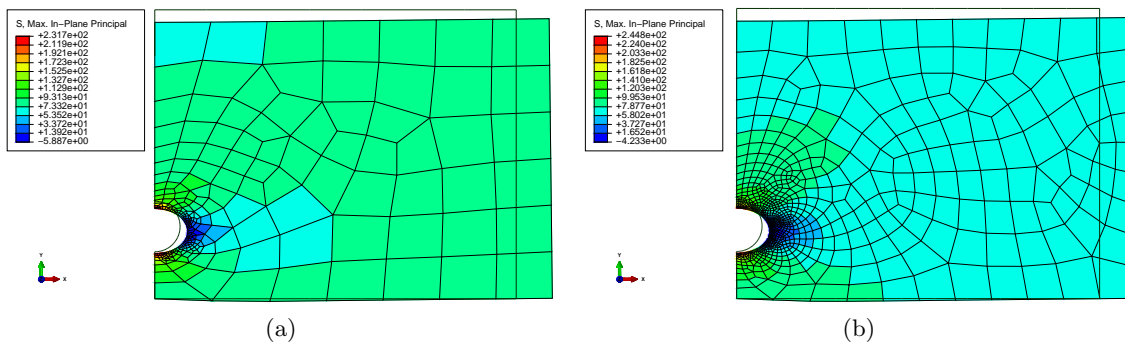
Fig. 6.12. Thin aluminium slab with a circular opening. (a) Largest principal stress for model 1. (b) Largest principal stress for model 3.

Three models were used to calculate the maximum of the largest principal stress. Figure 6.12(a) shows the coarsest mesh (model 1), and Figure 6.12(b) shows the finest mesh (model 3). The element size was varied with the mesh refinement factor $\alpha = 4/5$. The seeds for the coarsest mesh are shown in Figure 6.13 — the part seed dictates what the element size should be away from places where edge seeds are used. Three of the edges have associated edge seeds. The two straight edges are associated

Abaqus
- CAE file

with local biased seeds, where the minimum and maximum element size along the edge is given. The circular edge is associated with a local unbiased seed where only the average element size is given (0.13). The mesh generated from these prescriptions for model 1 was shown in Figure 6.12(a). For model 2, all the numbers in Figure 6.13 are multiplied by the refinement factor $\alpha = 4/5$. For model 3 the process is repeated, so that the seed numbers in Figure 6.13 are multiplied with $\alpha^2 = (4/5)^2 = 16/25$. This procedure is crucial for the applicability of extrapolation: provided the meshes are generated using this algorithm, the required limit values can be extrapolated with the respect to the sequence of numbers $\alpha^0 = 1$, $\alpha^1 = \alpha = 4/5$, and $\alpha^2 = 16/25$ as the independent variable.
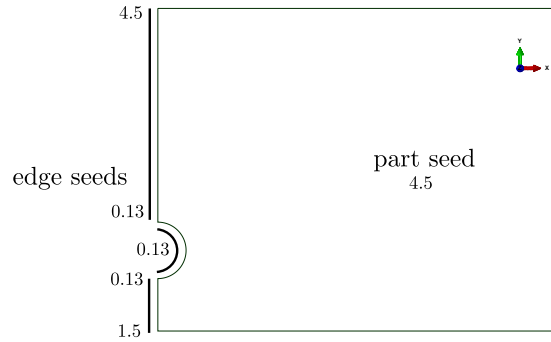
Fig. 6.13. Thin aluminium slab with a circular opening. Association of mesh seeds with the part and the edges for model 1

When we inspect the distribution of the principal stress, especially in the close-up of Figure 6.14, we can appreciate that even with the finest mesh used in this study the results are not anywhere near "converged": the representation of the stress is blocky, and the transitions from element to element are clearly visible. Nevertheless, such results can be used for extrapolation. The value of extrapolation then is that it (a) provides an improved estimate of the solution without the need for excessively fine mesh, and (b) can be used to judge how reliable such estimate will be.
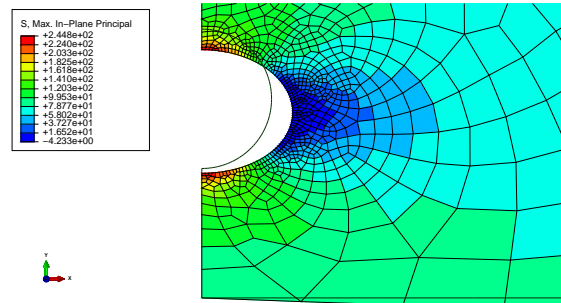
Fig. 6.14. Thin aluminium slab with a circular opening. Close-up of the largest principal stress for model 3 near the circular opening.

Now we can discuss the actual extrapolation: The results obtained with the three meshes are summarized in Table 6.2. We can use the Python function `richextrapol`. We define the solution list (page 221)

Python
- script

```
86    sol = [231.7, 239.1, 244.8]
```

and the list of the element sizes only needs to define values in the correct ratios (these are actually the powers of the refinement factor): the absolute values of the element sizes will not be needed in the extrapolation:

Model	Result in MPa
1	231.7
2	239.1
3	244.8

Table 6.2. Un-symmetric plate with a hole. Results of the refinement study. Target quantity: Maximum of the largest principal stress.

```
87    h = [(4./5.)**i for i in range(0, 3)]
```

i.e. `[1.0, 0.8, 0.64]`. The function `richextrapol` can then be applied as

Python - script

```
88    xestim, beta, c, residual = richextrapol(sol, h)
89    print('xestim, beta, c =', xestim, beta, c)
```

giving

```
xestim, beta, c = 263.91176470588067 1.1697126080157385 32.21176470588068
```

to yield the estimate of the limit value of the largest principal stress (263.9 MPa), but also the convergence rate (1.17), and the constant in (6.12). Finally, also the accuracy with which these quantities have been calculated are returned. The estimated true errors can be presented on a log-log scale graph: Figure 6.15. We can appreciate that the error of the best solution (model 3) with respect to the predicted true solution is around 8%. We would probably strive for engineering accuracy, say below 5%. This would be an argument then for not using the actual computed numbers, but rather to attempt to go to the limit to improve our guess of the true solution.
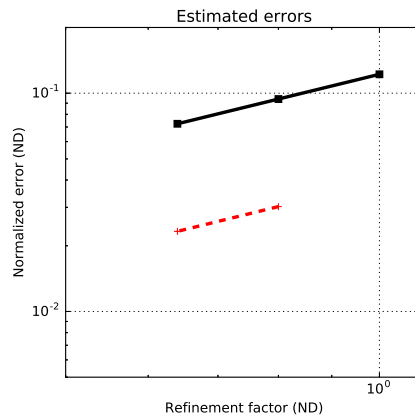
Fig. 6.15. Estimated normalized true errors (solid line), and normalized approximate errors (dotted line).

We should always check the quality of the extrapolation using the approximate errors. The approximate errors of the maximum of the largest principal stress are computed as

```
91    Ea = diff(sol)
```

that is

```
[ 7.4  5.7]
```

If the approximate errors indeed depend on the element size through a power relationship then there is a way of presenting the data graphically to confirm this. Take the absolute value of (6.17) and write

$$|E_{q,j}| = \left|\left(1 - \alpha^\beta\right) C \left(h_j^\beta\right)\right| = \left|\left(1 - \alpha^\beta\right) C\right| \left|h_j^\beta\right|$$

Now take the log of both sides to obtain

$$\log |E_{q,j}| = \log \left|\left(1 - \alpha^\beta\right) C\right| + \beta \log h_j \ ,$$

which is a linear relationship between $\log |E_{q,j}|$ and $\log h_j$. On a log-log plot the slope of the straight line is the convergence rate β.

The estimated true errors and the approximate errors are visualized in Figure 6.15. The estimated true errors are shown with a solid line, plotted as (page 221)

```
99      nEt = [(xestim-s)/xestim for s in sol]
100     plt.loglog(h, nEt, 'k-s', linewidth=3)
```

The approximate errors can also be normalized, for instance by the largest computed solution. Here we normalize by the largest computed value and plot as:

```
101     plt.loglog(h[1:3], Ea/max(sol), 'r--+', linewidth=3)
```

We can appreciate that the estimated true errors and the normalized approximate errors seem to lie on straight lines whose slope is approximately 1.0 (compare with the predicted convergence rate of 1.17). The behavior of the approximate error matching the estimated true error is a confirmation of the validity of the extrapolation process.

An alternative way of estimating the true error may be deduced from equation (6.17). We express the true error $E_q(h_j)$ in terms of the approximate error and a factor which depends on the convergence rate. So, provided we can reliably estimate the convergence rate (when we plot the approximate errors they lie on a straight line on a log-log plot of slope β), we can express the true error as

$$E_q(h_j) = \frac{E_{q,j}}{1 - \alpha^\beta} \ .$$

Substituting the definitions of the true error and of the approximate error we get

$$q_{\mathrm{ex}} - q_{h_j} = \frac{q_{h_{j+1}} - q_{h_j}}{1 - \alpha^\beta}$$

or

$$q_{\mathrm{ex}} = q_{h_j} + \frac{q_{h_{j+1}} - q_{h_j}}{1 - \alpha^\beta} \ , \tag{6.18}$$

which can be used as an alternative to the estimator (6.13).

Consider again the data computed above. Computing the approximate error and eyeballing the slope (i.e. the convergence rate) on the graph as 1.0, the alternative estimation of equation (6.18) yields

```
110     print(sol[1]+(sol[2]-sol[1])/(1-(4/5)**1.0))
```

i.e. 267.60. This is not too far off of the extrapolated value of 263.9 obtained from the general Richardson extrapolation formula (6.13). Of course, we can directly compute the slope of the approximate-error line (rise over run on a log-log scale)

```
111     print((log(Ea[1])-log(Ea[0]))/(log(h[2])-log(h[1])))
```

as 1.1697. Using this convergence rate estimate, the alternative estimation of the limit from approximate errors would yield

```
112     print(sol[1]+(sol[2]-sol[1])/(1-(4/5)**1.1697))
```

i.e. 263.9120. This value is within 2% of the stress reported by Szabó and Babuška [8]. Using finer meshes for the extrapolation may lead to an even better estimate.

6.7 Meshes and Mesh generation

In this book we focus on the family of *continuum finite elements*. The other families in Abaqus are for instance structural elements (beams and shells) and specialized elements such as connectors. Of the continuum finite elements we discuss only the basic (general-purpose) types for two-coordinate models (plane strain, plane stress, axially symmetric) and three-coordinate models (fully three-dimensional).

The general-purpose continuum finite elements are of two elementary shapes. This aspect is best explained with the help of a picture—please consider Figure 6.16. In three dimensions the two elementary shapes are the tetrahedron and the hexahedron. The former has four bounding faces that are triangular, the latter has six bounding faces which are quadrilateral. There are varieties of both elementary shapes with different numbers of nodes. For instance tetrahedron with four nodes (T4) and tetrahedron with 10 nodes (T10).

Furthermore, the bounding faces of these three-dimensional shapes can be understood as shapes with two intrinsic coordinates–surfaces. A tetrahedron has a boundary that consists of four triangles. Depending on how many nodes the tetrahedron has, the triangles may have three nodes (T3) or six nodes (T6). The triangles, in turn, are bounded by three curves. Again, these are finite elements with one intrinsic coordinate (distance along the curve), and may have two or three nodes. These curve finite elements also have a boundary, which consists of two distinct points. The points may also be understood as finite elements, which may have uses such as representing a concentrated mass etc.

We say that the tetrahedron, its bounding triangle, the bounding curve of the triangle, and the points at the ends of the curve, form a *compatible group*: compatible here means "fitting together" without gaps or cracks.

Similar group of compatible finite elements may be constructed for the hexahedral elementary shape. As a consequence of having two elementary shapes, the finite element programs commonly offer a choice between hexahedra and tetrahedra, quadrilaterals and triangles, to reflect this basic distinction.
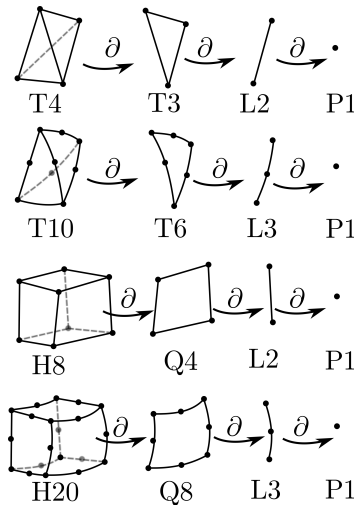
Fig. 6.16. The compatible groups of finite elements and their boundaries. The symbol ∂ means "extract the boundary". Note that only one finite element of the boundary set is shown. For instance, the T4 boundary consists of four T3 finite elements, the boundary of one T3 consists of three L2 elements, and the boundary of one L2 consists of two P1's.

6.7.1 Mesh generation

While it is certainly possible to design meshes by hand, this activity is limited by its laborious nature. Beyond say a dozen elements meshes generated by hand tend to take too long and the process is too error-prone. What one generally needs is a software component that can take a description of the geometric part and produce a finite element mesh whose shape approximates the part. These components are called ***mesh generators***.

Mesh generators tend to fall into two groups: structured and unstructured (free). The structured mesh generator would work from a few basic recipes, such as how to fill a region that looks like a box (possibly with curved sides, but with nice and clean six-face topology). Figure 6.17(a) presents a simple example.

(a)   **(b)**
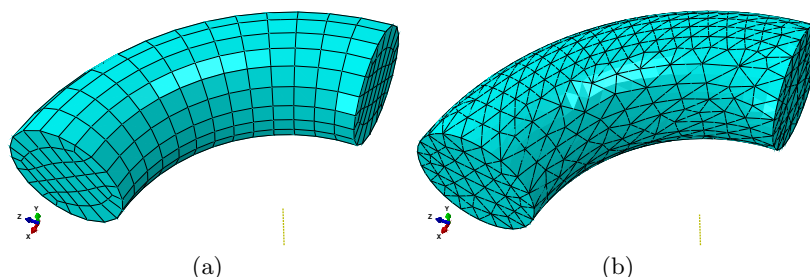
Fig. 6.17. Geometrical part: circular cylinder with a circular axis. (a) Structured mesh. Hexahedra. (b) Unstructured (free) mesh. Tetrahedra.

The unstructured (free) mesh generator would try to fill the space of the part element-by-element, using an algorithm, and generally in a much freer (less constrained) procedure than the structured mesh generator. Figure 6.17(b) shows an unstructured mesh of the same part, but filled with tetrahedra instead of hexahedra. This illustrates one powerful reason for having both hexahedral and tetrahedral elementary shape at your disposal: the hexahedral shapes tend to be better at representing the solution, but the tetrahedral shapes are so much better at meshing in an unstructured way. In fact, there are mathematical theorems that say that tetrahedra will be able to fill practically arbitrary shapes, whereas no such theorems exist for hexahedra. Hence it is much less likely to have an automatic generator of tetrahedra fail, whereas generation of hexahedral meshes fails often.

Figure 6.18 presents a persuasive example: the same cylindrical shape as before, except that a straight cylindrical hole has been drilled from one circular face. The unstructured generator of hexahedral elements will refuse the job: it cannot do it. The unstructured generator of tetrahedral elements built the mesh without a problem.
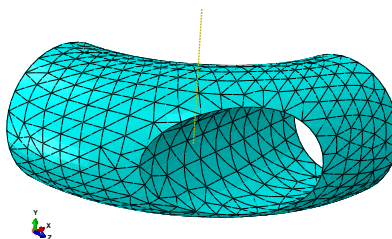
Fig. 6.18. Circular cylinder with a circular axis. Circular hole drilled orthogonally to one of the terminal faces.

6.7.2 Element distortions

We have seen that the errors in the finite element solution depends not only on the element size h but also on the distortion of the element (i.e. the so-called shape quality).

Automatic mesh generation takes away the drudgery but also the control. In contrast with hand-designed meshes, the user has limited control over the shapes of the automatically generated elements.

> The user still retains the veto power: the quality of the elements can be checked and displayed, and if the mesh has too many poorly shaped elements it can be (and probably should be) rejected.

Figure 6.19 shows three typical distortions: aspect ratio, skew angle, and taper. The elongated rectangle and the parallelogram shapes correspond to a constant Jacobian matrix, and are not as deleterious as the taper. The quality measures for the quadrilateral can be expressed for instance as acceptable range for the aspect ratio and for the minimum and maximum corner angle (as in Abaqus).

(a)   (b)   (c)
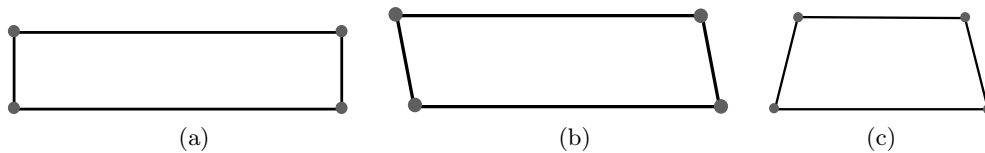
Fig. 6.19. Distortions of four-node quadrilaterals. (a) Large aspect ratio (in this case around 5:1). (b) Skew angle. (c) Taper.

Abaqus provides a tool to check the shape quality of the generated elements. Figure 6.20 shows a graded quadrilateral mesh, where two elements have been found lacking in shape quality (too distorted based on the user-specified criteria of aspect ratio and corner angles). The printout

```
Part: shaft
 Quad elements:  124
   Min angle on Quad Faces < 10:   0 (0%)
   Average min angle on quad faces:  73.66,  Worst min angle on quad faces:  27.10
   Max angle on Quad faces > 160:  0 (0%)
   Average max angle on quad faces:  108.07,  Worst max angle on quad faces:  145.77
   Aspect ratio > 5:  2 (1.6129%)
   Average aspect ratio:  1.68,  Worst aspect ratio:  12.00
  Number of elements :  124,   Analysis errors:  0 (0%),  Analysis warnings:  1 (0.806452%)
```

indicates that the criterion of interior angles passed, but two elements were found to exceed the maximum aspect ratio of 5.0.

In three-node triangles, the minimum and the maximum angle are usually checked. Figure 6.21 shows two distorted shapes: a needlelike triangle and a sliver-like triangle with both large and small interior angle. Both shapes should be avoided.

In quadratic quadrilaterals and triangles the situation becomes considerably more complicated as the location of the mid-edge nodes will also induce distortions. Figure 6.22 shows three examples of distortions. All result in the Jacobian matrix being not constant, which makes the variation of the derivatives of the basis functions quite complicated. In the first case, the mid-edge node was moved away from the center of the edge, so even though the edges are technically straight, the Jacobian now varies (wildly) across the element. The other two distortions are usually justifiable because they are introduced to approximate a curved boundary. They tend to lower the accuracy though. Similar figures could be drawn for quadratic triangles, with similar conclusions.

Elements typically get even worse distortions in three dimensions. Three-dimensional elements, tetrahedra and hexahedra, have many patterns of distortion. As a general rule, distortion with a constant Jacobian is mostly acceptable, distortion with non-constant Jacobian needs to be watched. Too many distorted elements, and the accuracy goes out the window.
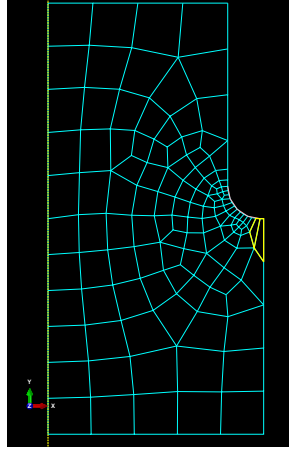
Fig. 6.20. Two elements highlighted as their shape quality raises concern.
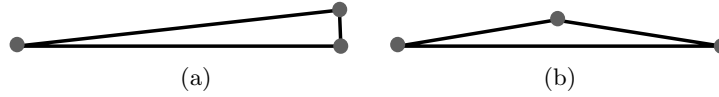
Fig. 6.21. Distortions of three-node triangles. (a) Small interior angle. (b) Both a small and a large interior angle.
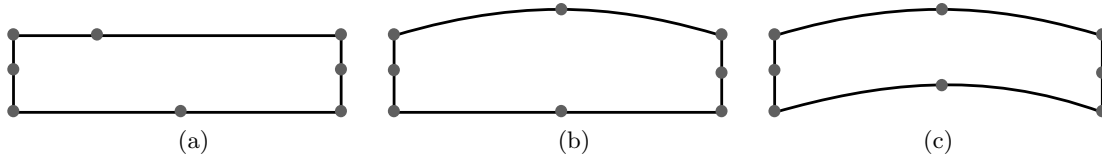
Fig. 6.22. Distortions of four-node quadrilaterals. (a) Place mid-edge node off-center. (b) Curved edge to approximate the shape of the part. (c) Two curved edges to mesh a curved thin part.

6.7.3 Interior mesh and Boundary mesh

The shapes of the finite elements in the toolkit are linked together through the "extract the boundary" operation (symbol ∂ in Figure 6.16). This capability is crucial because some finite element expressions need to be evaluated over the interior of the domain (the volume), while others should be evaluated over the boundary of the domain (the surface). Also, the volume and surface integrals may need to be computed for models of different number of space dimensions. Accordingly, we need different meshes for the calculations of integrals over the interior and for the calculations over the boundary.

Figure 6.16 summarizes how the various types of finite elements discussed in this book fit together. Some of these types have been presented already, some make their appearance later in the book. It is important that these families exist; if we had an element whose boundary could not be represented by another element in our library, we could not evaluate the terms on the boundary of the domain. That would be a fatal flaw.

Table 6.3 shows the terms in the Galerkin equations for the general heat conduction IBVP in three dimensions. The types of elements used for the volume integrals will be selected by the user depending, among other things, on the dimension of the model (one, two, three), the desired accuracy, efficiency considerations, and so on. The types of elements used for the surface integrals will then be determined by this choice.

For instance, if the model is fully three-dimensional we could select four-node tetrahedra T4 for the volume integrals, which would imply the use of three-node T3 triangles for the surface integrals. Another example: the model is two-dimensional (the domain is a two-dimensional figure), and we

could choose to cover the interior of the domain with four-node Q4 quadrilaterals. This would then imply that the boundary is to be covered with two-node L2 finite elements. Or, if we decided to use 6-node triangles of type T6 for the interior, the boundary integrals would be evaluated with L3 finite elements.

Term	Interior (volume) integral	Boundary (surface) integral
Conductivity matrix	$\int_V (\mathrm{grad}\eta)\,\boldsymbol{\kappa}(\mathrm{grad}T)^T\,\mathrm{d}V$	
Capacity matrix	$\int_V \eta c_V \dfrac{\partial T}{\partial t}\,\mathrm{d}V$	
Prescribed heat flux		$-\int_{S_2} \eta\,\bar{q}_n\,\mathrm{d}S$
Internal heat generation	$\int_V \eta Q\,\mathrm{d}V$	
Surface heat transfer matrix		$\int_{S_3} \eta\,hT\,\mathrm{d}S$
Surface heat transfer load		$\int_{S_3} \eta\,hT_a\,\mathrm{d}S$

Table 6.3. Heat conduction IBVP in three dimensions. Classification of terms into interior (volume) and boundary (surface) integrals.

6.8 Lumping: How the FEM works

The finite element method is a systematic approach to lumping. What is lumping? It is the conversion of continuous properties (attributes) to concentrated, or lumped, properties and attributes. Figure 6.23 illustrates what that means. Figure 6.23(a) shows a continuous structure with distributed loading. The mathematical model works with the continuous geometry and the distributed traction loading. Figure 6.23(b) shows the geometry replaced with a finite element mesh approximation. The geometry is now a collection of triangles. The loading in the form of distributed traction is applied on the edges of the triangles.

Fig. 6.23. From a continuous mathematical model to a discrete finite element model: lumping.

Figure 6.23(c) shows the discrete model produced by the finite element machinery. The motion of the solid in response to the loading is expressed as displacements of discrete points (the nodes, showed as filled circles). The continuous property of elastic resistance has now been lumped into discrete

springs that connect the nodes. The distributed traction loading has been converted (lumped) into concentrated forces applied at the nodes.

The method of weighted residuals is in effect an optimal way of determining the lumping of the properties so that the response of the original continuous structure under the original loads is approximated as closely as possible by a discrete system of masses and springs and forces derived from a given finite element mesh.

So, if the finite element method produces always lumped forces, is there anything wrong with applying a lumped force? Figure 6.24(a) shows a rectangle domain with an applied concentrated force. Figure 6.24(b) shows a lumped finite element model that results from meshing the rectangle with six quadrilaterals. The given applied concentrated force carries over from the continuous model and is applied directly to the middle node of the vertical edge on the right. Figure 6.24(c) shows the finite element discrete model obtained by meshing the rectangle with four times as many quadrilateral finite elements. The concentrated force is again applied at the original value at the middle node on the right vertical edge. The problem is that in the finer mesh the lumped springs are also finer (softer). The finer the mesh, the smaller and weaker the springs. In order to balance the force, which maintains its magnitude, the softer springs have to stretch more. The weaker the springs, the more they have to stretch. In the limit, the springs will have infinitesimal stiffness, and they will have to stretch by an infinite amount to balance the force: not good. Not good because this is not physically meaningful.
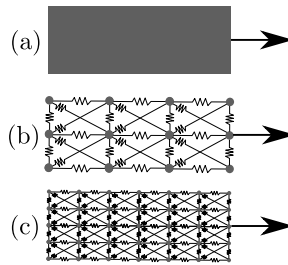
Fig. 6.24. Inadmissible modeling with a concentrated force. (a) Continuous model: rectangle with a concentrated applied force. (b) Coarse discrete model. (c) Finer discrete model.

As an example consider Figure 6.25: a block of material is clamped at the bottom, and a concentrated force is applied at the top-right corner. The mesh is then adaptively refined at the corner, and while the overall deflection of the block does not seem to change with the refinement, the point where the fixed force is applied can be seen to be progressively pulled into a thread by the applied force. The smaller the elements, the less resistance they offer, and the more they have to stretch and deform in order to balance the force. In the limit, the corner will get more and more stretched out into a thread, and the deflection of the point of application will tend to infinity.
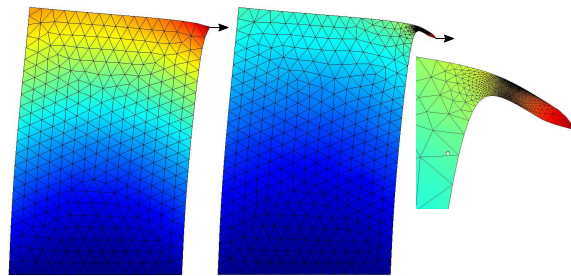
Fig. 6.25. Block of material with a fixed concentrated force applied horizontally at the top-right corner. Left: initial mesh. Right: refined mesh in the vicinity of the corner. Rightmost: zoomed detail of the corner.

Now consider distributed loading. Figure 6.26 illustrates how the concentrated forces that result from lumping of a distributed loading (Figure 6.26(a)) are still appropriate after refinement of the mesh. The finite element model of a rectangle loaded by distributed load at the top edge produces concentrated forces acting at the nodes in Figure 6.26(b). When the mesh is refined, such as in the Figure 6.26(c), the forces are again obtained by lumping the original distributed loading, and they end up smaller than for the coarse mesh. The finer the mesh, the smaller the lumped concentrated forces at the nodes. And, since the springs and the forces are now both getting "smaller" at the same time, they are matched. In the limit we get a perfectly fine physically meaningful answer.
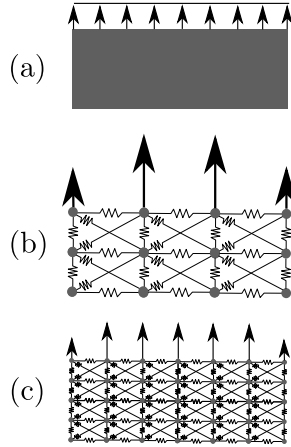
Fig. 6.26. Admissible concentrated forces resulting from the finite element lumping process. (a) Continuous model: rectangle with a distributed loading. (b) Coarse discrete model. (c) Finer discrete model.

> Avoid concentrated forces and nonzero reactions at point supports. Mesh refinement will not lead to convergence, and the results will be meaningless.

6.9 Background, explanations, details

Box 26. Interpolation errors

We will estimate the difference between the "exact" distribution of temperature, $T(\boldsymbol{x})$, and an interpolation of this function on a finite element mesh, $\Pi_h T(\boldsymbol{x})$. Here h means the mesh "size", or characteristic dimension. Typically, mesh size is taken to mean edge length, or the diameter of the smallest ball that completely encloses an element.

Interpolation error for temperature

The interpolating function is defined as

$$\Pi_h T(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}_k) \ , \tag{6.19}$$

where $\boldsymbol{x}_k$ is the location of the node k, and $T(\boldsymbol{x}_k)$ is the value of the temperature at the location of the node k. For interpolation on a mesh consisting of three-node triangles, when $\boldsymbol{x}$ is in the interior of the element Δ_e, only three basis functions N_k are nonzero at $\boldsymbol{x}$. The basic tool is the Taylor series
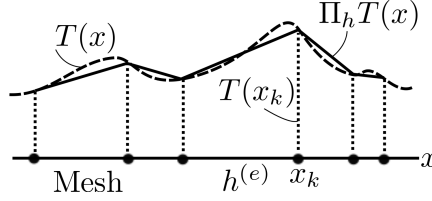
Fig. 6.27. Interpolating the temperature function on a mesh

which we use to expand the temperature at $\boldsymbol{x}$

$$T(\boldsymbol{y}) = T(\boldsymbol{x}) + \mathrm{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{y} - \boldsymbol{x}) + R_1(\boldsymbol{y}, \boldsymbol{x}) \;, \tag{6.20}$$

where the remainder is written as

$$R_1(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x}) \cdot \boldsymbol{H}(T)(\boldsymbol{y} - \boldsymbol{x}) \;. \tag{6.21}$$

The matrix of second derivatives (Hessian) is evaluated at $\boldsymbol{\xi}$ somewhere between the points $\boldsymbol{y}$ and $\boldsymbol{x}$

$$[\boldsymbol{H}(T)] = \begin{bmatrix} \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_1 \partial x_1} & \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_1 \partial x_2} \\[2mm] \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_2 \partial x_1} & \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_2 \partial x_2} \end{bmatrix} \;.$$

The Taylor series (6.20) may be used to express the value of the temperature at the nodes– plug in $\boldsymbol{x}_k$ for $\boldsymbol{y}$– which then may be substituted into the interpolation (6.19) to yield

$$\Pi_h T(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}_k) =$$

$$\sum_k N_k(\boldsymbol{x}) \left[T(\boldsymbol{x}) + \mathrm{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) + R_1(\boldsymbol{x}_k, \boldsymbol{x}) \right] \;.$$

Due to the construction of the basis functions, we have these important equalities

$$\sum_k N_k(\boldsymbol{x}) = 1 \;, \quad \sum_k N_k(\boldsymbol{x}) \boldsymbol{x}_k = \boldsymbol{x} \;. \tag{6.22}$$

Therefore, the first term in (6.22) simplifies as

$$\sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}) = T(\boldsymbol{x}) \sum_k N_k(\boldsymbol{x}) = T(\boldsymbol{x}) \;,$$

and the second will vanish

$$\sum_k N_k(\boldsymbol{x}) \mathrm{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) = \mathrm{grad}T(\boldsymbol{x}) \cdot \sum_k N_k(\boldsymbol{x})(\boldsymbol{x}_k - \boldsymbol{x}) = 0 \;.$$

Substituting into (6.22) gives

$$\Pi_h T(\boldsymbol{x}) = T(\boldsymbol{x}) + \sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x}) \;,$$

or, reshuffling to get the error on one side,

$$T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x}) = -\sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x}) \;. \tag{6.23}$$

To estimate the magnitude of the difference, $|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})|$, we compute

$$\left| \sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x}) \right| \leq \max |R_1(\boldsymbol{x}_k, \boldsymbol{x})| \, \left| \sum_k N_k(\boldsymbol{x}) \right| = \max |R_1(\boldsymbol{x}_k, \boldsymbol{x})| \;,$$

and make use of standard norm inequalities

$$|\boldsymbol{v} \cdot \boldsymbol{A}\boldsymbol{v}| \leq \|\boldsymbol{v}\|\|\boldsymbol{A}\boldsymbol{v}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{v}\|^2 \ .$$

This may be applied to the definition of the remainder (6.21) together with (see Fig. 6.28)

$$\|\boldsymbol{x}_k - \boldsymbol{x}\| \leq h \ ,$$

and an estimate of the norm of the matrix of second derivatives of the temperature $\boldsymbol{H}(T)$ to give

$$|T(\boldsymbol{x}) - \varPi_h T(\boldsymbol{x})| \leq Ch^2\|\boldsymbol{H}(T)\| \ . \tag{6.24}$$

Here C is a generic constant with respect to h. If we wrap the norm of the matrix of the second derivatives into the constant, we may write

$$|T(\boldsymbol{x}) - \varPi_h T(\boldsymbol{x})| \leq C\left(\partial^2 T\right) h^2 \ , \tag{6.25}$$

where we agree to mean by $C\left(\partial^2 T\right)$ some constant whose magnitude depends on the curvatures of the function T. Importantly, $C\left(\partial^2 T\right)$ may also be understood as measuring the *rate of change of the heat flux* in the immediate neighborhood of $\boldsymbol{x}$.
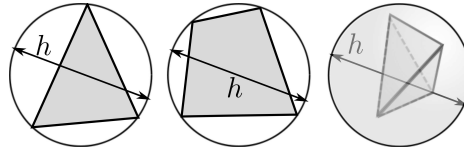
Fig. 6.28. Mesh size h as a "diameter" of an element: diameter of the tight-fitting circle that encloses a triangle or a quadrilateral, and of the smallest sphere that is circumscribed to a tetrahedron

The value of Eq. (6.25) is twofold:

- Firstly, it states that the errors of interpolation will get bigger the higher the curvature of the function of the exact temperature T (that is the faster the heat flux is changing) and the bigger the elements (i.e. the error will increase with h^2);
- Secondly, if we are interested in the interpolation error at a particular location, we may consider the curvatures at that location as given, and the Eq. (6.25) then says that the error will decrease as $O(h^2)$ as $h \to 0$ (order-of estimate: reduce h with a factor of two, and the error will decrease with a factor of four).

Interpolation error for temperature gradient

To estimate errors for the gradient of temperature, we start with the interpolation (6.22), of which we take the gradient

$$\begin{aligned}
\operatorname{grad}\varPi_h T(\boldsymbol{x}) = \sum_k \operatorname{grad}N_k(\boldsymbol{x})T(\boldsymbol{x}_k) = \\
\sum_k \operatorname{grad}N_k(\boldsymbol{x})\left[T(\boldsymbol{x}) + \operatorname{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) + R_1(\boldsymbol{x}_k, \boldsymbol{x})\right] = \\
\sum_k \operatorname{grad}N_k(\boldsymbol{x})T(\boldsymbol{x}) + \sum_k \operatorname{grad}N_k(\boldsymbol{x})\operatorname{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) \\
+ \sum_k \operatorname{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x}) = \\
T(\boldsymbol{x})\sum_k \operatorname{grad}N_k(\boldsymbol{x}) + \operatorname{grad}T(\boldsymbol{x}) \cdot \sum_k \operatorname{grad}N_k(\boldsymbol{x})(\boldsymbol{x}_k - \boldsymbol{x}) \\
+ \sum_k \operatorname{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x}) \ .
\end{aligned} \tag{6.26}$$

Differentiating (6.22) we obtain

$$\sum_k \mathrm{grad}N_k(\boldsymbol{x}) = 0 , \quad \sum_k \boldsymbol{x}_k \mathrm{grad}N_k(\boldsymbol{x}) = \boldsymbol{1} , \tag{6.27}$$

which upon substitution into (6.26) yields

$$\mathrm{grad}\Pi_h T(\boldsymbol{x}) = \mathrm{grad}T(\boldsymbol{x}) + \sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x}) .$$

$$\tag{6.28}$$

Again, an estimate of the magnitude is desired,

$$|\mathrm{grad}T(\boldsymbol{x}) - \mathrm{grad}\Pi_h T(\boldsymbol{x})| = |\sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x})| \leq$$

$$\max|R_1(\boldsymbol{x}_k, \boldsymbol{x})| \sum_k |\mathrm{grad}N_k(\boldsymbol{x})| .$$

To estimate the magnitude of the gradient of the basis function, we invoke the picture of the basis
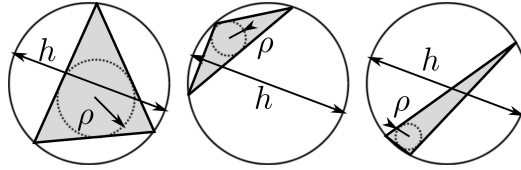
Fig. 6.29. Triangle quality measures using the radius of the inscribed circle and the diameter of the circumscribed circle. Good (almost equilateral) triangle on the left; bad triangles (obtuse, needle-like) on the right.

function as a plane that assumes value one at one node and drops off to zero along the opposite edge. Therefore, the largest magnitude of the basis function gradient will be produced by the smallest height in the triangle. The shortest height $d_{\min}$ may be estimated from the radius of the largest inscribed circle, ρ (see Fig. 6.29), as $d_{\min} \approx O(\rho)$. This can be linked to the so-called "shape quality" of a triangle using the **quality measure**

$$\gamma = \frac{h}{\rho} ,$$

as $d_{\min} \approx O(\gamma^{-1})h$. The magnitude of the basis function gradient may be then estimated as

$$\max \mathrm{grad}N_k(\boldsymbol{x}) = \frac{1}{d_{\min}} \approx \frac{\gamma}{h} .$$

Putting everything together, we obtain

$$|\mathrm{grad}T(\boldsymbol{x}) - \mathrm{grad}\Pi_h T(\boldsymbol{x})| \leq Ch^2 \frac{\gamma}{h} \|\boldsymbol{H}(T)\| = C\left(\partial^2 T\right)\gamma h . \tag{6.29}$$

The value of Eq. (6.29) is again twofold:

- Firstly, it states that the errors of interpolation for the gradient of temperature will get bigger the higher the curvature of the function of the exact temperature T, the larger the elements (i.e. the error will increase with h), and the larger the quality measure γ (i.e. the worse the shape of the triangle);
- Secondly, considering the curvatures at a fixed location as given, the equation (6.29) states that the error will decrease as $O(h)$ as $h \to 0$ (note that this is one order lower than for the temperatures themselves: reduce h with a factor of two, and the error will decrease with the same factor).

Importantly, Eq. (6.29) allows us to make a general observation: the quantity calculated in the finite element solution is the temperature, the gradient (or, alternatively, the heat flux) is obtained by *differentiation* of the computed temperature, which immediately results in a reduction of the order of dependence on the mesh size. Phrased differently: the temperature results will converge faster than the temperature-gradient results (or, equivalently, heat flux results), because h^2 approaches zero much quicker than h as $h \to 0$.

End Box 26

Box 27. Richardson extrapolation

If we assume that the true error in the quantity q may be expanded in a Taylor series at element size $h = 0$, we may write $E_q(h)$ as the true error of the quantity q for some nonzero element size h in the form

$$E_q(h) = q_{\mathrm{ex}} - q_h = Ch^\beta + R , \tag{6.30}$$

where q_{ex} is the true solution, q_h is the approximate solution obtained with element size h, C is some constant which does not depend on the element size, $\beta > 0$ is the so-called convergence rate, and R is the remainder term. If the remainder term R is negligible compared to Ch^β, we may write

$$E_q(h) = Ch^\beta . \tag{6.31}$$

Then, provided we obtain the solution of the finite element problem for three different element size values $h_1 < h_2 < h_3$, so that we have a sequence of solutions q_{h_j}, we can write the so-called approximate error as the difference of two successive solutions

$$E_{q,j} = q_{h_{j+1}} - q_{h_j} . \tag{6.32}$$

Replacing the approximate solutions with the true errors and the true solution $q_{h_j} = q_{\mathrm{ex}} - E_q(h_j)$, we obtain

$$E_{q,j} = -E_q(h_{j+1}) + E_q(h_j) . \tag{6.33}$$

Substituting from (6.31), we get

$$E_{q,j} = -Ch_{j+1}{}^\beta + Ch_j{}^\beta = C\left(-h_{j+1}{}^\beta + h_j{}^\beta\right) . \tag{6.34}$$

This equation may be written twice, for two successive approximate errors,

$$E_{q,1} = C\left(-h_2{}^\beta + h_1{}^\beta\right) \quad \text{and} \quad E_{q,2} = C\left(-h_3{}^\beta + h_2{}^\beta\right) , \tag{6.35}$$

which may be readily solved for C to yield

$$\frac{E_{q,1}}{-h_2{}^\beta + h_1{}^\beta} = \frac{E_{q,2}}{-h_3{}^\beta + h_2{}^\beta} . \tag{6.36}$$

This nonlinear equation may be solved for the convergence rate β. Then the constant C follows as

$$C = \frac{E_{q,1}}{-h_2{}^\beta + h_1{}^\beta} = \frac{E_{q,2}}{-h_3{}^\beta + h_2{}^\beta} , \tag{6.37}$$

and the estimate of the true error for any element size then follows from (6.31). Obviously, with the estimate of the true error we also have the estimate of the true solution q_{ex}.

End Box 27

6.10 Code listings

richextrapol.py

```python
1  # Finite Element Modeling with Abaqus and Python for Thermal and
2  # Stress Analysis
3  # (C)  2017-2020, Petr Krysl
4  """
5  Richardson extrapolation.
6  """
7
8  import math
9  from numpy import diff
10
11 def bisection(fun, xl, xu, tolx, tolf):
12     if (xl > xu):
13         xl, xu = xu, xl
14     fl = fun(xl);
15     fu = fun(xu);
16     while True:
17         xr = (xu + xl) * 0.5  # bisect interval
18         fr = fun(xr)  # value at the midpoint
19         if fr * fl < 0.0: # (fr < 0.0 && fl > 0.0) || (fr > 0.0 && fl < 0.0)
20             xu, fu = xr, fr # upper --> midpoint
21         else:
22             xl, fl = xr, fr # lower --> midpoint
23         if (abs(xu-xl) < tolx) or (abs(fr) <= tolf):
24             return xl, xu # We are done
25     return xl, xu
26
27 def richextrapol(xs, hs):
28     """
29     Richardson extrapolation.
30
31      xestim, beta, c, residual = richextrapol(xs,hs)
32
33     Richardson extrapolation. This function is applicable to fixed ratio
34     between the element sizes, hs[0]/hs[1] == hs[1]/hs[2], in which case
35     the solution follows explicitly, But it is also applicable to non-uniform
36     refinement factor (for arbitrary element sizes).
37
38     Arguments
39         xs = list of the calculated quantities,
40         hs = list of the element sizes
41
42     Returns
43         xestim= estimate of the asymptotic solution from the data points
44             in the xs array
45         beta= convergence rate
46         c = constant in the estimate "error = c*h**beta"
47         residual = residual after equations from which the above quantities were
48             solved (this is a measure of how accurately was the system solved)
49     """
50     # Normalize the solutions so that we work with nice numbers
51     nxs = [x/xs[0] for x in xs]
52     c, beta = 0.0, 1.0 # some defaults to be overwritten below
53     if abs(hs[0]/hs[1] - hs[1]/hs[2]) > 1.0e-6:
54         # In the hard case, with a non-uniform refinement factor, we must
```

```
55              #           solve a nonlinear equation 1st.
56          nea1 = nxs[1]-nxs[0]
57          nea2 = nxs[2]-nxs[1]
58          tol = 0.000000001
59          h0, h1, h2 = hs[0], hs[1], hs[2]
60          eqn = lambda b: nea1 - nea2 * ((-h1**b+h0**b) / (-h2**b+h1**b))
61          lo = tol # lower bound on the convergence rate
62          hi = 10.0
63          b1, b2 = bisection(eqn, lo, hi, tol, tol)
64          beta = (b1 + b2) / 2.0
65          c = xs[0]*nea1/(-hs[1]**beta+hs[0]**beta)
66          xestim = xs[2] + c * hs[2]**beta
67      else:
68          # In the easy case, with a uniform refinement factor, the solution
69          #           is obtained explicitly.
70          xestim = ((-(nxs[0]*nxs[2]-nxs[1]**2)/(2*nxs[1]-nxs[0]-nxs[2])))*xs[0]
71          if (xestim-xs[0]) <= 0:
72              beta = math.log((xestim-xs[1])/(xestim-xs[2]))/math.log(hs[1]/hs[2])
73          else:
74              beta = math.log((xestim-xs[0])/(xestim-xs[2]))/math.log(hs[0]/hs[2])
75          c = (xestim-xs[0])/hs[0]**beta
76      #
77      # just to check things, calculate the residual
78      residual = [0, 0, 0]
79      for I in range(0, 3):
80          residual[I] = (xestim-xs[I])-c*hs[I]**beta# this should be close to zero
81      #
82      return xestim, beta, c, residual

84 def _test():
85      """
86      Test of Richardson extrapolation.
87      """
88      xs = [93.0734, 92.8633, 92.7252]
89      hs = [0.1000, 0.0500, 0.0250]
90      xestim, beta, c, residual = richextrapol(xs, hs)
91      print('xestim, beta, c, residual =', xestim, beta, c, residual)
92      print('to be compared with ', 92.46031652777476, 0.6053628424093497,
       -2.471055221256022, \
93          [0.0, 5.534461777756405e-13, 3.6376457401843254e-13])
94      #
95      xs = [13.124299546191557, 12.192464513175167, 12.026065694522512]
96      hs = [1.0, 0.5, 0.25]
97      xestim, beta, c, residual = richextrapol(xs, hs)
98      print('xestim, beta, c, residual =', xestim, beta, c, residual)
99      print('to be compared with ', 11.989892116202842, 2.485429379525128,
       -1.1344074299887144, \
100         [0.0, -8.93729534823251e-15, -1.5959455978986625e-15])
101     #
102     xs = [18.279591331058882, 18.260877354949294, 18.255333231883284]
103     hs = [0.25, 0.125, 0.0625]
104     xestim, beta, c, residual = richextrapol(xs, hs)
105     print('xestim, beta, c, residual =', xestim, beta, c, residual)
106     print('to be compared with ', 18.25299931817398, 1.7550849296401745,
       -0.3029825595040225, \
107         [0.0, -3.975153539670373e-13, -1.1776647365624449e-13])
108     #
109     sol = [231.7, 239.1, 244.8]
```

```
110    h = [(4./5.)**i for i in range(0, 3)]
111    xestim, beta, c, residual = richextrapol(sol, h)
112    print('xestim, beta, c =', xestim, beta, c)
113    print('to be compared with ', 263.91176470588067, 1.1697126080157385,
       32.21176470588068)
114    #
115    # This Tests the case of non-uniform refinement factor
116    sol = [7.446, 7.291, 6.87] # f = lambda h: -3.333 * h**1.2 + 6.66
117    h = [0.3, 0.25, 0.1]
118    xestim, beta, c, residual = richextrapol(sol, h)
119    print('xestim, beta, c =', xestim, beta, c)
120    print('to be compared with ', 6.66, 1.2, -3.333)
121
122 _test()
123
124 def examples():
125    """
126    Examples from the textbook.
127    """
128    sol = [231.7, 239.1, 244.8]
129    h = [(4./5.)**i for i in range(0, 3)]
130    xestim, beta, c, residual = richextrapol(sol, h)
131    print('xestim, beta, c =', xestim, beta, c)
132    #
133    Ea = diff(sol)
134    print(Ea)
135    # Plotting
136    import matplotlib.pyplot as plt
137    plt.figure()
138    plt.xlim((0.5, 1.1))
139    plt.ylim((0.005, 0.2))
140    # Normalized true error estimate
141    nEt = [(xestim-s)/xestim for s in sol]
142    plt.loglog(h, nEt, 'k-s', linewidth=3)
143    plt.loglog(h[1:3], Ea/max(sol), 'r--+', linewidth=3)
144    #
145    plt.grid()
146    plt.title('Estimated errors')
147    plt.xlabel('Refinement factor (ND)')
148    plt.ylabel('Normalized error (ND)')
149    plt.show()
150    #
151    from math import log
152    print(sol[1]+(sol[2]-sol[1])/(1-(4/5)**1.0))
153    print((log(Ea[1])-log(Ea[0]))/(log(h[2])-log(h[1])))
154    print(sol[1]+(sol[2]-sol[1])/(1-(4/5)**1.1697))
155
156 #examples()
```

Listing 6.1. richextrapol.py

7

Further Developments of FEA for General Stress Analysis

The concepts introduced in the previous chapters are refined and expanded here. The present chapter intends to firstly review finite element techniques for three-dimensional stress analysis. The performance of the three-dimensional solid elements, the tetrahedra and the hexahedra, is illustrated using a variety of examples. Secondly, we deal with the plane-strain and axially symmetric reduced models of stress analysis. Thirdly, we discuss a couple of practically important modeling techniques: the principle of equivalent loads (St. Vénant), and the use of point supports to stabilize free-floating structures. Fourthly, dynamics is introduced by way of examples: free-vibration analysis and harmonic forced vibration. Finally, we introduce considerations appropriate for thin-walled solids (shells) and we discuss laminated structures.

7.1 Stress analysis in three dimensions

There are now three displacements at a given point, u_x, u_y, and u_z, all expressed in the global Cartesian basis. The six total strains are arranged in a vector as

$$[\boldsymbol{\epsilon}] = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \epsilon_{xy} \\ \epsilon_{xz} \\ \epsilon_{yz} \end{bmatrix} \tag{7.1}$$

and are derived from the displacements as $\boldsymbol{\epsilon} = \mathcal{B}\boldsymbol{u}$ with the the symmetric gradient operator $\mathcal{B}$

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 & 0 \\ 0 & \partial/\partial y & 0 \\ 0 & 0 & \partial/\partial z \\ \partial/\partial y & \partial/\partial x & 0 \\ \partial/\partial z & 0 & \partial/\partial x \\ 0 & \partial/\partial z & \partial/\partial y \end{bmatrix} \tag{7.2}$$

The stress components are also arranged in a vector

$$[\boldsymbol{\sigma}] = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{bmatrix} \tag{7.3}$$

and are related to the strains by the constitutive equation

$$[\boldsymbol{\sigma}] = [\boldsymbol{D}] \left([\boldsymbol{\epsilon}] - [^\theta\boldsymbol{\epsilon}]\right) , \tag{7.4}$$

where the elastic strains are the difference of the total strains and the thermal strains $[^\theta\boldsymbol{\epsilon}]$.

For an isotropic material we can write the material stiffness matrix as

$$\boldsymbol{D} = \begin{bmatrix} \lambda + 2G & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda + 2G & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda + 2G & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix} ,$$

where we introduce the Lamé constant

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{7.5}$$

and the shear modulus G

$$G = \frac{E}{2(1+\nu)} . \tag{7.6}$$

Both of these constants are expressed in terms of the Young's modulus E and the Poisson's ratio ν. For an isotropic material the thermal strains are

$$[^\theta\boldsymbol{\epsilon}] = \alpha \Delta T \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} . \tag{7.7}$$

The solution of the stress analysis problem is to satisfy a balance equation which, in the form valid at a point, is written as

$$[\mathcal{B}]^T [\boldsymbol{\sigma}] + [\overline{\boldsymbol{b}}] = [\boldsymbol{0}] . \tag{7.8}$$

Equation (7.8) is valid for statics (negligible accelerations). The ***stress-divergence operator*** $\mathcal{B}^T$ is the transpose of (7.2)

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & 0 & \partial/\partial y & \partial/\partial z & 0 \\ 0 & \partial/\partial y & 0 & \partial/\partial x & 0 & \partial/\partial z \\ 0 & 0 & \partial/\partial z & 0 & \partial/\partial x & \partial/\partial y \end{bmatrix} . \tag{7.9}$$

7.2 Weighted residual equation for three dimensions

For finite elements applied in three dimensions the weighted residual equation is a fairly simple variation on the weighted residual equation we wrote down for the plane-stress analysis (5.23): For each free degree of freedom q, i.e. for $1 \leq q \leq N_\mathrm{f}$, there will be one equilibrium (force balance) equation

$$\sum_{p=1}^{N_\mathrm{f}} \int_V \left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} \mathrm{d}V \ U_p + \sum_{p=N_\mathrm{f}+1}^{N} \int_V \left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} \mathrm{d}V \ U_p$$

$$\tag{7.10}$$

$$- \int_V N_{j(q)}[\overline{\boldsymbol{b}}]_{r(q)} \ \mathrm{d}V - \int_{S_{t,r(q)}} N_{j(q)} \overline{t}_{r(q)} \ \mathrm{d}S - \int_V \left[[\boldsymbol{B}_{j(q)}]^T \boldsymbol{D} \ [^\theta\boldsymbol{\epsilon}]\right]_{r(q)} \ \mathrm{d}V = 0.$$

The necessary changes are limited to the number of degrees of freedom per node (three displacements, instead of two), and the integral over the interior of the domain is now a true volume integral. Finally, the integral over the surface where a traction component is prescribed, $S_{t,r(q)}$, is a true surface integral. Note that again loads will be produced by nonzero displacements prescribed in some direction s on the part of the boundary $S_{u,s}$ ("support-settlement" loads).

The first term will result in a square stiffness matrix multiplied with the vector of the unknown displacement degrees of freedom; the second term will result in the contribution to the load vector due to the prescribed (known) displacement degrees of freedom (i.e. the support-settlement load); the third and fourth term are contributions to the load vector due to the known body load and traction loads on the boundary. The last term is the thermal-load contribution.

The nodal strain-displacement matrix is written explicitly by applying the partial derivatives to the argument N_k, i.e. the basis function associated with node k, (compare with the expression for the plane stress model (5.21))

$$[\boldsymbol{B}_j] = \mathcal{B}(N_j) = \begin{bmatrix} \dfrac{\partial N_j}{\partial x} & 0 & 0 \\[2mm] 0 & \dfrac{\partial N_j}{\partial y} & 0 \\[2mm] 0 & 0 & \dfrac{\partial N_j}{\partial z} \\[2mm] \dfrac{\partial N_j}{\partial y} & \dfrac{\partial N_j}{\partial x} & 0 \\[2mm] \dfrac{\partial N_j}{\partial z} & 0 & \dfrac{\partial N_j}{\partial x} \\[2mm] 0 & \dfrac{\partial N_j}{\partial z} & \dfrac{\partial N_j}{\partial y} \end{bmatrix} . \tag{7.11}$$

7.3 Tetrahedra

Tetrahedra are one elementary shape with which true 3-D solid shapes can be meshed. The second elementary shape, the hexahedron, will be covered in the next section. Tetrahedra are either linear or quadratic, where "linear" and "quadratic" refers to the variation of the interpolated quantities within the element. Linear elements described linear variation of displacements (temperatures), and analogously for the quadratic elements.

7.3.1 Tetrahedron T4

The tetrahedron with four nodes at the corners (element T4) is a straightforward extension of the triangle T3 to three dimensions. The standard tetrahedron is shown in Figure 7.1. The basis functions in the parametric coordinates are designed to be linear functions of ξ, η, ζ, and there are four corners at which to use the Kronecker delta property. It is straightforward to deduce that

$$N_1(\xi, \eta, \zeta) = 1 - \xi - \eta - \zeta \ , \ N_2(\xi, \eta, \zeta) = \xi \ , \ N_3(\xi, \eta, \zeta) = \eta \ , \ N_4(\xi, \eta, \zeta) = \zeta \ . \tag{7.12}$$

Using the nodal strain-displacement matrices we can compute the strain vector inside a finite element from the nodal displacements as

$$[\boldsymbol{\epsilon}] = \sum_k [\boldsymbol{B}_k][\boldsymbol{u}_k] \tag{7.13}$$

Entirely analogously to the three-node triangle, the strain-displacement matrices $[\boldsymbol{B}_k]$ consists only of constants. Consequently the strains are constant across the entire element (recall that the three-node triangle is also known as the constant-strain triangle, CST). Table 7.1 defines two integration rules for tetrahedra [3]. The one-point rule is adequate for stiffness (or conductivity) matrix evaluation, as we deduced from the strains being constant across the element.
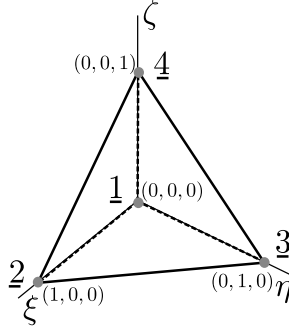
Fig. 7.1. Standard tetrahedron

Table 7.1. Numerical integration rules on the standard tetrahedron; $a = 0.1381966$, $b = 0.5854102$.

Rule	Coordinates ξ_j, η_j, ζ_j	Weights W_j	Integrates exactly
1-point	1/4, 1/4, 1/4	1/6	linear polynomial
4-point	a, a, a b, a, a a, b, a a, a, b	1/24 1/24 1/24 1/24	quadratic polynomial

> The four-node tetrahedron is delightfully simple. It is also dismally ineffective: when we're in doubt whether it will work for our particular problem, we should assume that it won't.

The stiffness matrix of the four-node tetrahedron is really the smallest stiffness matrix of any three-dimensional element, it is 12×12. It has the structure shown in Figure 7.2, where we illustrate how the matrix is composed of 3×3 blocks corresponding to the nodal strain-displacement matrices. Consider a single element, tetrahedron with four nodes, $\underline{1}, \underline{2}, \underline{3}$ and $\underline{4}$ (Figure 7.2). Let us also assume
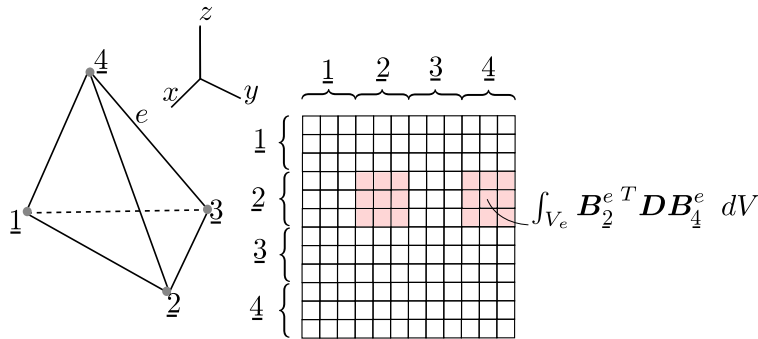
Fig. 7.2. Composition of the element stiffness matrix of submatrices (blocks)

that all the degrees of freedom are free (the stiffness matrix is then singular). The stiffness matrix of this single element has 12 rows and columns.

$$K_{qp} = \left[\int_{V_e} \boldsymbol{B}_{j(q)}^{e}{}^T \boldsymbol{D} \boldsymbol{B}_{i(s)}^e \, \mathrm{d}V \right]_{r(q)s(p)}, \quad j, i = \underline{1}, \underline{2}, \underline{3}, \underline{4}, \quad \text{and } r, s = 1, 2, 3 \quad (\text{i.e. } x, y, z) .(7.14)$$

For j, i fixed, say $j = \underline{2}$ and $i = \underline{4}$, the matrix

$$\int_{V_e} \boldsymbol{B}_j^{eT} \boldsymbol{D} \boldsymbol{B}_i^e \, \mathrm{d}V , \tag{7.15}$$

is a 3×3 submatrix of the element stiffness matrix: compare with the illustration in Figure 7.2 (the off-diagonal block). Similarly, for $j = \underline{2}$ and $i = \underline{2}$ we obtain the diagonal block. Therefore, the entire element stiffness matrix may be computed in one shot as

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \; \mathrm{d}V \; , \tag{7.16}$$

using the blocked matrix (the elementwise strain-displacement matrix)

$$\boldsymbol{B}_e = [\boldsymbol{B}_A^e, \boldsymbol{B}_B^e, \boldsymbol{B}_C^e, \boldsymbol{B}_D^e] \; . \tag{7.17}$$

Correspondingly, the displacements at the nodes will be ordered into a column vector of displacement components in the global Cartesian basis

$$\boldsymbol{U}_e = \begin{bmatrix} \boldsymbol{u}_A \\ \boldsymbol{u}_B \\ \boldsymbol{u}_C \\ \boldsymbol{u}_D \end{bmatrix} \; . \tag{7.18}$$

The important point we must make here is that these submatrices express the ***stiffness coupling*** between the nodes of the element. The diagonal sub matrices express how the three-dimensional displacement of a node affects the elastic restoring force acting on that same node; the off-diagonal submatrices express how the three-dimensional displacement of one node affects the elastic restoring force acting on another node.

7.3.2 Simplex elements

The point P1, the segment L2, the triangle T3, and the tetrahedron T4, are all examples of the so-called simplex elements. By definition, an n-dimensional simplex is the convex hull of $n+1$ points (vertices) in the n-dimensional space. (We can think of the convex hull in terms of a rubber band or rubber sheet stretched over $n + 1$ points in space.) Tiling domains with simplex elements is attractive, because a number of mathematical properties guarantee the success of automatic tools for mesh generation. This is to be contrasted with the generation of quadrilaterals in two dimensions, and of bricks (shapes bounded by six quadrilateral faces) in three dimensions: not an easy task– mesh generators often fail to produce good-quality meshes, or often they just fail to produce any mesh at all.

While the linear simplex elements perform adequately in the heat conduction models, in other types of analyses their inherent simplicity tends to work against them. For instance, in stress analysis the response of meshes composed of linear simplex elements is quite poorly represented – they are "too stiff".

7.3.3 Tetrahedron T10

The elements of the simplex shape are saved by the quadratic types: the six-node triangle and the 10-node tetrahedron. These elements can usually deliver acceptable results, and the meshes are relatively easily generated by automatic software components (mesh generators).

The quadratic tetrahedron is a simple extension of the quadratic triangle to three dimensions (refer to Figure 7.3 for the definition of the order of the nodes: start with the vertices, and then number the midpoints of the edges). The same idea of constructing the basis functions as (normalized) products of planes will work, yielding for the basis functions the quadratic expressions in the parametric coordinates ξ, η, ζ

$$\begin{aligned} N_1 &= (1 - \xi - \eta - \zeta)[2(1 - \xi - \eta - \zeta) - 1] \; , \quad N_2 = \xi(2\xi - 1) \; , \\ N_3 &= \eta(2\eta - 1) \; , \quad N_4 = \zeta(2\zeta - 1) \; , \quad N_5 = 4(1 - \xi - \eta - \zeta)\xi \; , \\ N_6 &= 4\xi\eta \; , \quad N_7 = 4\eta(1 - \xi - \eta - \zeta) \; , \quad N_8 = 4(1 - \xi - \eta - \zeta)\zeta \; , \\ N_9 &= 4\xi\zeta \; , \quad N_{10} = 4\eta\zeta \; . \end{aligned} \tag{7.19}$$

Since the basis functions are quadratic in the parametric coordinates, their gradients with respect to the parametric coordinates will be linear functions of ξ, η, ζ. Provided the Jacobian matrix in equation (3.57) is constant (independent of ξ, η, ζ), the gradients of the basis functions with respect to x, y, z as computed from (3.56) are going to be linear in the x, y, z coordinates. Hence, computing the elements of the stiffness matrix can be done exactly with the four-point rule from Table 7.1. In contrast, for curved elements (when a mid-edge node is moved from the mean of the locations of the corners) the four-point rule will not be able to integrate the stiffness matrix exactly. However, when the distortion of the element is not excessive, the integration error can be accepted.
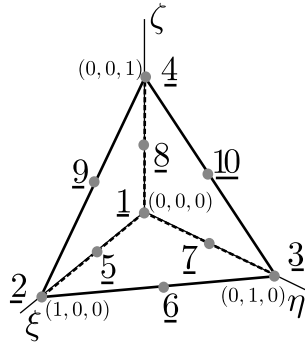
Fig. 7.3. The standard quadratic tetrahedron for the T10 element.

Four-point integration is considered a "full integration" of the stiffness (conductivity) matrix. The linear stress-analysis element in Abaqus is called C3D10: C= continuum, 3D= three-dimensional, 10= number of nodes.

7.4 Hexahedra

Hexahedra are sometimes referred to as bricks. As already indicated in Figure 6.16, the hexahedra are compatible with quadrilateral finite elements, since quadrilaterals constitute their boundaries.

7.4.1 Eight-node hexahedron (H8)

To extend the quadrilateral Q4 (whose basic properties were discussed in Chapter 4) to three dimensions is quite straightforward: instead of a Cartesian product of two intervals on the standard square, we consider the Cartesian product of three intervals on the standard cube (Figure 7.4). The Q4 quadrilateral is compatible with the discretization on the faces of the hexahedron, which is essential for the implementation of the integrals of the surface terms (heat flux or traction loads).

The numbering of the nodes is given in Figure 7.4. The basis function $N_{\underline{1}}$ may be written as the product of one one-dimensional Lagrange interpolation function on the interval $-1 \le \xi \le +1$, one on the interval $-1 \le \eta \le +1$, and one on the interval $-1 \le \zeta \le +1$; all functions correspond to the left-hand side end of the interval

$$N_{\underline{1}}(\xi, \eta, \zeta) = \frac{\xi - 1}{-1 - 1} \times \frac{\eta - 1}{-1 - 1} \times \frac{\zeta - 1}{-1 - 1} = \frac{(\xi - 1)(\eta - 1)(\zeta - 1)}{8} \; . \tag{7.20}$$

This process is analogous for the remaining functions.

Gauss integration at $2 \times 2 \times 2$ points is considered a "full integration" of the stiffness (conductivity) matrix. The linear stress-analysis element in Abaqus is called C3D8.
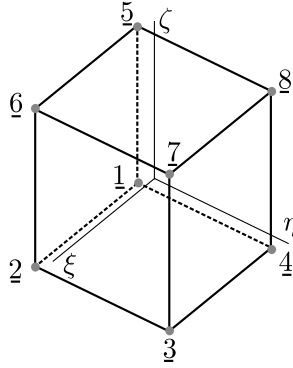
Fig. 7.4. Numbering of the nodes of the hexahedron H8. Node 1 is at $\xi = -1, \eta = -1, \zeta = -1$, node 2 is at $\xi = +1, \eta = -1, \zeta = -1$,..., and node 8 is at $\xi = -1, \eta = +1, \zeta = +1$

7.4.2 Thin simply-supported square plate with uniform distributed load

The plate is illustrated in Figure 7.5. This is one of the classic benchmarks for plate structures, see for example Reference [10]. To match the thin-plate analytical solution is typically quite challenging for *plate* finite elements (that is structural elements specialized for bending deformations). Here we will attempt to compute this solution with *solid* elements. The essential boundary condition is applied by suppressing the displacement degrees of freedom orthogonal to the plate surface along the circumference.
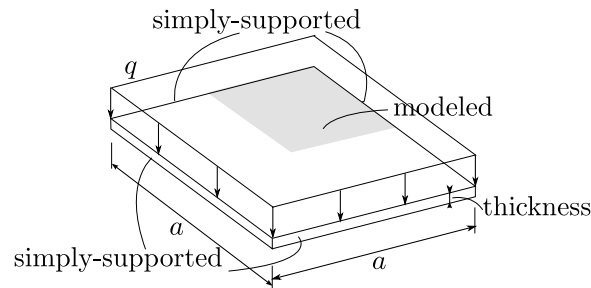
Fig. 7.5. Simply-supported square plate with uniform distributed load

Due to the symmetry of the problem, we will discretize just one quarter of the geometry, and apply symmetry boundary conditions. For a moderately thick plate, thickness/span = 1/30, the deflected shape is shown in Figure 7.6(a); the thin plate, thickness/span = 1/200, is shown in Figure 7.7(a).

7.4.3 Shear locking

If we use the linear hexahedron C3D8, we find that the computed deflections are way too small, especially for the thin plate. Even with many thousands of elements, the normalized computed deflection is just a fraction of the analytical solution. We say that the C3D8 element *locks* in bending (which is sometimes referred to as shear locking)— call for Figure 7.7(b). As the plate is so thin, most of the energy should be in bending, but the element deformation mode puts most of it into shear and as a result the model is way too stiff. Even for thicker plates (up to the span/thickness ratio of 30, when the plate would be considered rather thick), the element C3D8 delivers only disappointing accuracy: see Figure 7.6(b).

The equivalent element in two dimensions (where a picture is much easier to produce) is the four-node quadrilateral Q4, in plane-stress analysis called CPS4 (and CPE4 for plane strain). Figure 7.8 provides a visual explanation of the phenomenon of shear locking. In order for the element to

(a)   (b)
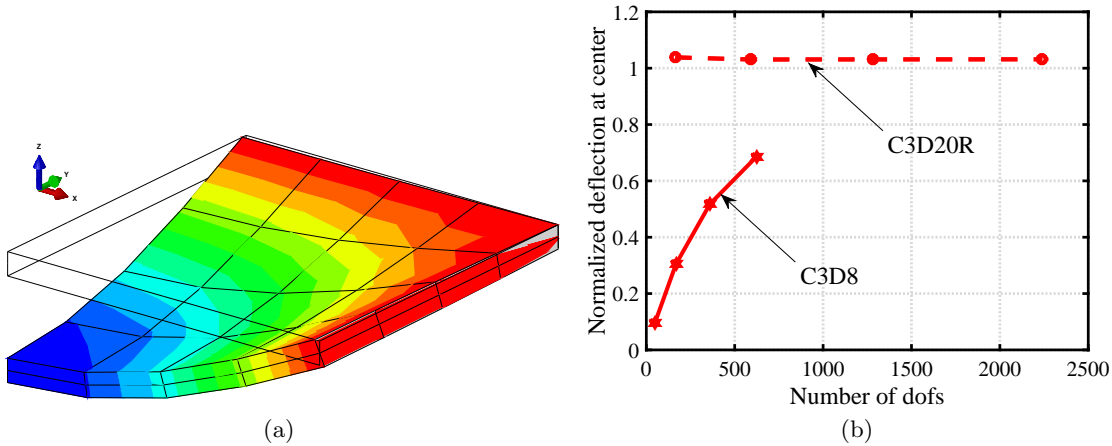
Fig. 7.6. Simply-supported square plate with uniform distributed load. Moderately thick plate (thickness/span = 1/30). (a) Deflected shape (highly magnified): note the ratio of the thickness to the span. Mesh with four edges on the side of the modeled quarter-domain of the plate. (b) Comparison of the normalized deflection for the standard eight-node hexahedron (C3D8) and the reduced-integration 20-node hexahedron (C3D20R). Four results are shown for four meshes: 2, 4, 6, and 8 element edges on the side of one quarter of the plate. The deflection is normalized by the thin-plate solution, which explains why the C3D20R solution appears to converge to a number greater than 1.0: the thick plate is shear-flexible and the shear flexibility makes the deflection larger than for bending-only deformation characteristic of thin plates.

(a)   (b)
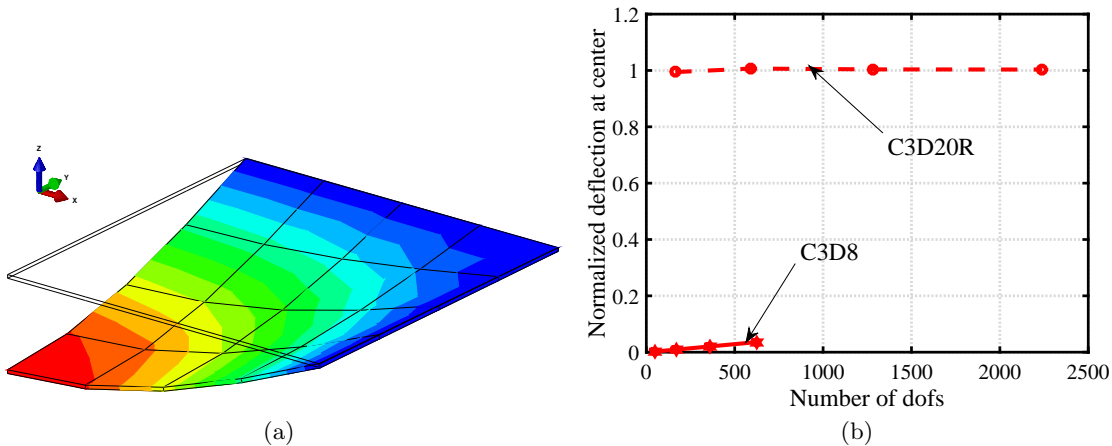
Fig. 7.7. Simply-supported square plate with uniform distributed load. Thin plate (thickness/span = 1/200). (a) Deflected shape (highly magnified): note the ratio of the thickness to the span. (b) Comparison of the normalized deflection for the standard eight-node hexahedron (C3D8) and the reduced-integration 20-node hexahedron (C3D20R). For results are shown for four meshes: 2, 4, 6, and 8 element edges on the side of one quarter of the plate.

assume a "bent" configuration, it really needs to experience shear strains γ because its sides must stay straight.

Consider a beam under pure bending loads through the applied moments M. As a response, the sides of the beam structure will tilt by the angle α. The energy in pure bending can be given as $BEC\alpha^2$, where BEC is the bending stiffness coefficient (which can be determined as a function of the dimensions of the beam, and the material properties, but we will skip over those details). The amount of work that the bending moment M acting on the angle of rotation α produces is $M\alpha$. Under equilibrium conditions we have

$$BEC\alpha^2 = \frac{1}{2}M\alpha \tag{7.21}$$

so that $\alpha = \frac{1}{2}M/BEC$ Note in particular that there is no shear-strain energy, because there are no shear strains in pure bending.

Consider now this beam modeled with one quadrilateral element. The finite element cannot deform without experiencing shear strain. Therefore, the energy stored in the finite element includes both the energy of bending $BEC\alpha^2$, and the energy of the shear strain, $SEC\alpha^2$. Here SEC is the shear stiffness coefficient, which again could be determined from the dimensions of the element and the material properties. The energy stored in the finite element is therefore

$$(BEC + SEC)\widehat{\alpha}^2 = \frac{1}{2}M\widehat{\alpha} \tag{7.22}$$

And, so we can determine the deformation of the finite element $\widehat{\alpha} = \frac{1}{2}M/(BEC + SEC)$. Consequently, the ratio of the deformation of the finite element $\widehat{\alpha}$ to the correct deformation α is

$$\frac{\widehat{\alpha}}{\alpha} = \frac{BEC}{(BEC + SEC)} \tag{7.23}$$

When the beam (and the quadrilateral element) are very stocky, the energy stored in bending is comparable with the energy stored in shear. On the other hand, when the aspect ratio increases and the beam and the finite element become thin, most of the energy should be stored in bending and only very little in shear. Unfortunately, for the finite element the ratio of the shear energy to the bending energy SEC/BEC *increases* as the element becomes thin, for instance the ratio could well be in excess of ≈ 100 or more. Then the finite element deformation is only about 1% of the correct value: the finite element locks.
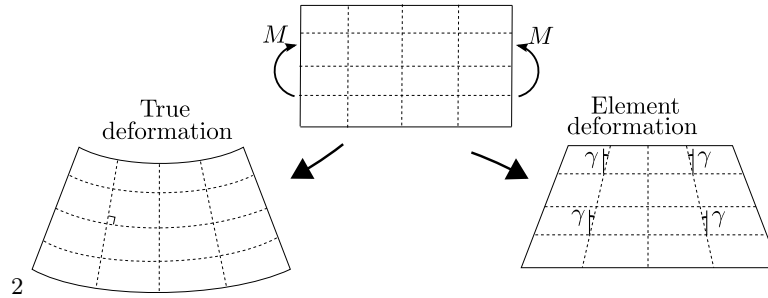
Fig. 7.8. Illustration of a deformation pattern that leads to shear locking

For span-to-thickness ratio greater than around 1:5 we shouldn't use the plane-vanilla quadrilaterals or hexahedra: they would be much too stiff. The cure is described in the following section (use quadratic elements), and in Section 7.10 where we introduce advanced stress analysis elements.

7.4.4 Quadratic element H20

The quadratic tetrahedron T10 managed to improve upon T4. The quadratic hexahedron H20 is a significant step up from the hexahedron H8.

Contrary to (likely) expectations, the quadratic element will not be derived by taking the Cartesian product of the basis functions of the quadratic one-dimensional element L3 (Section 5.6). The result would have been an element with 27 nodes: $3 \times 3 \times 3 = 27$, the so-called quadratic Lagrangean hexahedron.
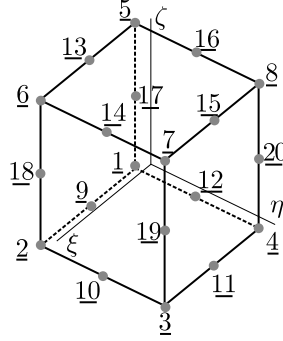
Fig. 7.9. Numbering of the nodes of the hexahedron H20

The quadratic hexahedron H20 is a member of the so-called ***serendipity*** family. The nodes associated with basis functions are located at the corners and the midpoints of the edges: see Figure 7.9; eight corners plus the midpoints of 12 edges gives 20 basis functions. The derivation of the basis functions may proceed as follows: start with the basis functions of the hexahedron H8, and add in the quadratic functions associated with the midpoints. Then modify the original linear functions so that all the basis functions add up to one at any point within the element.

The functions that are being added for the midpoints are produced as products of one quadratic function (in one variable) and two linear functions (in the remaining two variables). For instance, consider the basis function N_{11}: the variation of this function along the edge $3, 4$ should be the quadratic $(1 - \xi^2)$ (Figure 7.10). Furthermore, N_{11} should vanish at all the nodes except $\underline{11}$. The function $(1 - \xi^2)$ is zero along the faces $\underline{2}, \underline{3}, \underline{7}, \underline{6}$ and $\underline{4}, \underline{1}, \underline{5}, \underline{8}$. To make it vanish also along the two faces $\underline{7}, \underline{8}, \underline{5}, \underline{6}$ and $\underline{1}, \underline{2}, \underline{6}, \underline{5}$, we multiply it with the two functions $(\eta + 1)$ and $(\zeta - 1)$. Finally, this product is normalized to assume value $+1$ at the node $\underline{11}$. The result is

$$N_{11} = \frac{(1 - \xi^2)(\eta + 1)(\zeta - 1)}{8} \ .$$

The basis functions for all the other mid-side nodes are obtained in the same fashion. The final step is to subtract half of each mid-side basis function from the two corner basis functions that are borrowed from H8 along that edge so that the basis functions again add up to one everywhere within the element

$$\sum_{\underline{k}=1}^{20} N_{\underline{k}}(\xi, \eta, \zeta) = 1 \ .$$

Thus, for instance for $N_{\underline{1}}$ we have

$$N_{\underline{1}} = \frac{(\xi - 1)(\eta - 1)(\zeta - 1)}{8} - \frac{N_{\underline{11}}}{2} - \frac{N_{\underline{10}}}{2} - \frac{N_{\underline{19}}}{2} \ .$$

Detailing all the other basis functions would take up too much space: please consult [3] or the Abaqus manuals.

Gauss integration at $3 \times 3 \times 3$ points is considered a "full integration" of the stiffness (conductivity) matrix. This element would be known as C3D20 in Abaqus. However, the theoretically insufficient Gauss scheme $2 \times 2 \times 2$ points often leads to much improved results, giving a less constrained model with considerably improved flexibility. Such an element is labeled "reduced-integration", and it is
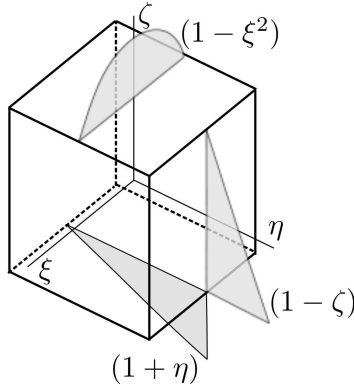
Fig. 7.10. Visualization of the three functions whose product gives the basis function $N_{\underline{11}}$ for the hexahedron H20

known as C3D20R. The simply-supported plate problem was modeled with the 20 node hexahedron C3D20R.

Consider the results presented earlier in Figures 7.6 and in 7.7: The hexahedron C3D20R with the reduced $2 \times 2 \times 2$ numerical integration scheme performs very well for both the thick and the thin plate: refer to Figures 7.6 and 7.7. For this particular problem it delivers engineering-accuracy results with very coarse meshes.

One note is in order: The deflections are normalized by the analytical solution obtained for a *thin* plate, i. e. without consideration of shear deformations [10]. However the numerical results are produced by solid elements, which do incorporate shear. That is why the numerical results apparently converge to a deflection higher than that predicted analytically.

> The 20-node hexahedron will usually work very well. The contraindications are few, for instance in contact problems: consult the manuals.

7.4.5 Quadratic element Q8

The quadratic quadrilateral Q8 is compatible with the faces of the brick element H20 (compare Figure 7.9 with Figure 7.11). Therefore, to write down the basis functions for the quadrilateral, we may for instance substitute $\zeta = -1$ into the basis functions $N_{\underline{j}}, \underline{j} = \underline{1}, \underline{2}, \underline{3}, \underline{4}$ (corner functions), and $N_{\underline{j}}, \underline{j} = \underline{9}, \underline{10}, \underline{11}, \underline{12}$ (mid-side functions) of the hexahedron.
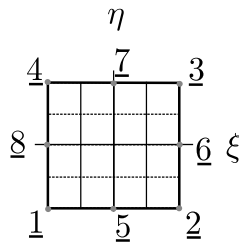
Fig. 7.11. Numbering of the nodes of the quadrilateral Q8

The Abaqus plane-stress element of the Q8 form is called CPS8 when used with full Gauss integration of 3×3 points, and CPS8R when used with the reduced Gauss integration of 2×2 points. Again, the reduced-integration version of the element is more flexible and therefore has an edge on the fully-integrated element.

7.5 Model reduction for plane strain

The plane-stress model developed in Chapter 5 is easily extended to handle the case of the so-called plane strain. The starting point is the observation that for some solids of uniform cross-section (right angle prisms), the set of *assumptions* that $u_x = u_x(x, y)$, $u_y = u_y(x, y)$ and $u_z(x, y, z) = 0$ seems to approximate the deformation well. In words, only the in-plane displacements are non-zero, and the third displacement component is identically zero. A classical example that is modeled as plane strain is a pipe with fixed ends. The following figures show 3-D simulations of a piping connector between two massive, stiff components. The middle portion (well removed from the two ends) can be modeled profitably with the plane strain model. See Box 19

Figure 7.12 shows (half of) the cross-section of a counter-flow pipe which is loaded in the larger-diameter opening with interior pressure. Symmetry boundary conditions are applied, and the end-points of the pipe are clamped. Figure 7.13(a) illustrates the distribution of the maximum principal
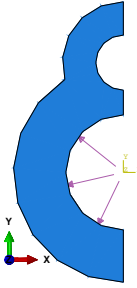
Fig. 7.12. Half of the cross-section of the piping connector. Pressure in the larger-diameter pipe is indicated by arrows.

strain. Clearly, away from the ends the distribution of the maximum principal strain is essentially independent of the position in the z (axial) direction. This would correspond to the strain being generated by in-plane (x, y) components of displacement. This can be correlated with Figure 7.13(b) where the $\epsilon_z = 0$ can be observed to hold away from the ends of the pipe. Figure 7.14(a) shows the
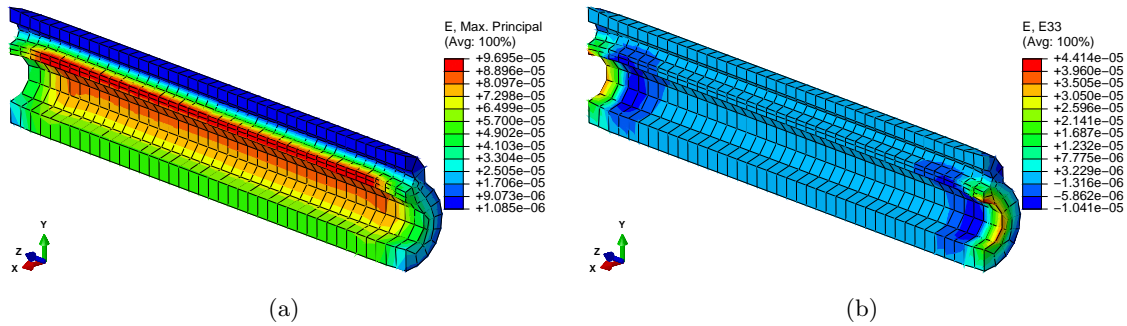
Fig. 7.13. Piping connector. (a) Largest principal strain. (b) Axial strain (ϵ_z). Note that the axial strain is essentially zero in the middle portion of the structure.

von Mises stress. Again, the essentially two-dimensional distribution along the pipe within a certain distance from the end sections can be appreciated. The axial stress, σ_z is *not* zero, as indicated in Figure 7.14(b).

The equations derived for plane stress in Chapter 5 apply, the only modification that is required consists of replacing the material stiffness matrix. For an isotropic material, the plane-strain material stiffness matrix reads (compare with the plane-stress matrix (5.7))
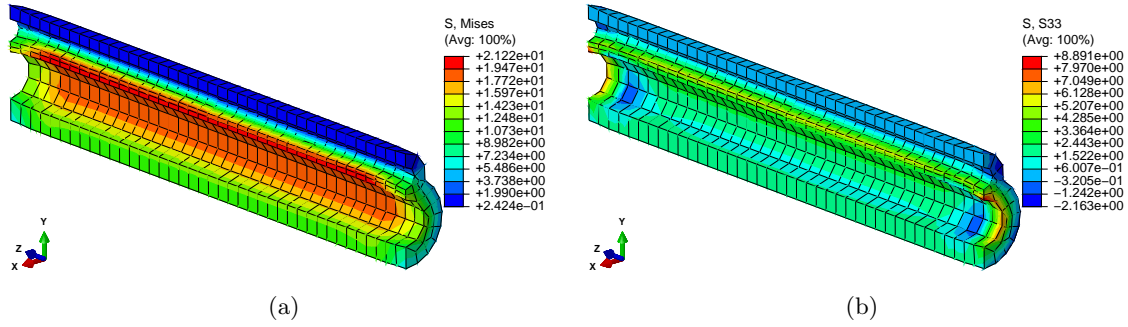
Fig. 7.14. Piping connector. (a) Von Mises stress. (b) Axial stress (σ_z).

$$[\boldsymbol{D}] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2} \end{bmatrix} \tag{7.24}$$

7.6 Model reduction for axial symmetry

The last model reduction approach to be discussed in this chapter, is the case of *torsionless* axial symmetry (Figure 7.15). The main assumption is that all planes passing through the y axis are symmetry planes. (The y axis will be referred to as the axis of symmetry.) Everything is assumed to have this symmetry: the geometry, the material, the loading and support conditions. Therefore, points in any particular cross-section will move only in the radial and axial direction, and will not leave the plane of the cross-section (the circumferential displacement is identically zero). Furthermore, points on circles in planes orthogonal to the axis of symmetry, with centers on the axis of symmetry, experience the same radial and axial displacements. See Box 19
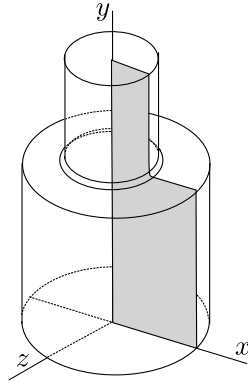
Fig. 7.15. Axially symmetric geometry. The dark shape is the generating section, which is revolved by 360° around the axis of symmetry to generate the solid. Note that the geometry may be as shown or it may have a hole along the axis of symmetry (like a pipe), a through-hole or partial hole depending on where the section extends all the way to the axis of symmetry.

Let us consider one particular cross-section, for instance as indicated in Figure 7.15. Since the displacements in the plane of the cross-section are symmetric with respect to the axis of symmetry, we will consider only the part to one side of the axis of symmetry (filled in Figure 7.15).

As indicated in Figure 7.16, the circles in planes orthogonal to the axis of symmetry are by assumption transformed by the deformation again into circles in planes orthogonal to the axis of

symmetry. Therefore, right angles between the plane of the cross-section and the tangent to the circle where it intersects the cross-section will remain right angles, and the shear strains are $\gamma_{zx} = 0$, and $\gamma_{zy} = 0$. For an isotropic material, we may conclude that $\tau_{zx} = 0$, and $\tau_{zy} = 0$.
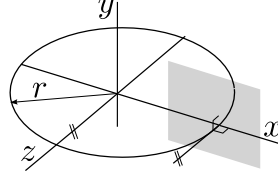
Fig. 7.16. Axially symmetric geometry: geometry of circles in planes orthogonal to the axis of symmetry. The generating section is indicated as a filled shape.

The circumferential stress is an important part of the balance of an axially symmetric part. It can be expressed in terms of the displacements in the plane of the generating cross-section as follows: By inspection of Figure 7.16, displacement of the circle in the y direction does not change its circumference, while displacement radially means that the circle of radius x will experience strain

$$\epsilon_z = \frac{2\pi(x + u_x) - 2\pi x}{2\pi x} = \frac{u_x}{x} \ .$$

The strain-displacement operator for this model may therefore be written as

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ 1/x & 0 \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \ . \tag{7.25}$$

The stress divergence operator (the transpose of (7.25)) gives the reduced form for the equations of static equilibrium (7.8)

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\sigma_z}{x} + \bar{b}_x = 0$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \bar{b}_y = 0$$

where the presence of σ_z should be noted: recall that the equation of motion in the radial direction holds for a wedge-shaped element, hence the contribution of the circumferential (hoop) stress to the radial direction must be included.

The constitutive equation is obtained from the full three-dimensional relationship by extracting appropriate rows and columns

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \end{bmatrix} = [\boldsymbol{D}] \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \end{bmatrix} \ , \tag{7.26}$$

where the material stiffness matrix reads

$$[\boldsymbol{D}] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 \\ \nu & 1-\nu & \nu & 0 \\ \nu & \nu & 1-\nu & 0 \\ 0 & 0 & 0 & \dfrac{(1+\nu)(1-2\nu)}{2(1+\nu)} \end{bmatrix} \tag{7.27}$$

When a single part is not supported in any way, in the axially symmetric model it has one rigid body mode, translation in the axial direction. Consider the piece of a steel cylindrical shaft with a filleted shoulder in Figure 7.17. The part is under self-equilibrated tensile loads, 20 ksi applied to the top circular cross-section of radius 5/6 in, and an appropriately reduced distributed loading of $(5/6)^2 \times 20$ ksi is applied on the bottom circular cross-section of radius 1 in.

Abaqus - CAE file

(a)   (b)
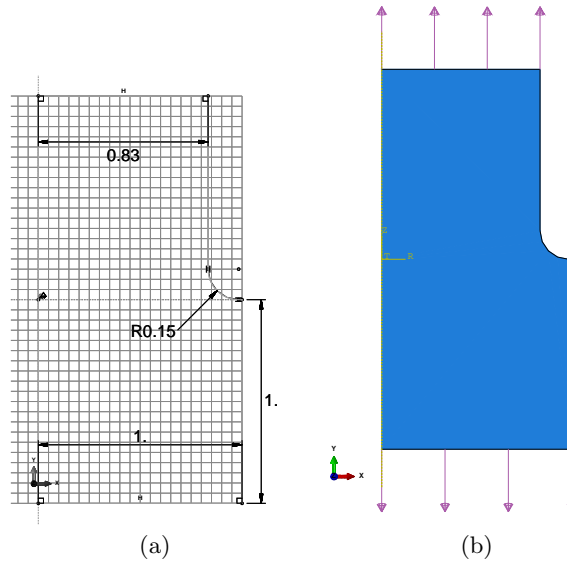
Fig. 7.17. Shaft with cross-section transition. (a) Sketch. Units: inches. Note that the top radius is 5/6 in: the dimension shows rounding. (b) Loading by tension tractions, the loading is assumed to be in self-equilibrium.

When the solution is obtained without applying any displacement boundary conditions, the displacement in the direction along the axis of symmetry is huge, and depends on the mesh used. For instance, with the mesh in the linked model database, the displacement is on the order of 17,000 inches. The reason is the (nearly) singular stiffness matrix and the following effect of numerical round off: the simulation is actually run off of a text file that describes the model, the ".inp" file that the Job Manager writes out. Thus we find that distributed loading defined by these lines:

```
** Name: tension  bottom   Type: Surface traction
*Dsload
bottom, TRVEC, 13888.9, 0., -1., 0.
** Name: tension top   Type: Surface traction
*Dsload
top, TRVEC, 20000., 0., 1., 0.
```

where `TRVEC, 13888.9` refers to the magnitude of the traction applied on the bottom face and `TRVEC, 20000.` refers to the magnitude of the traction applied on the top face. Using these numbers, the traction at the bottom is actually only approximately able to balance the traction applied at the top. The total force applied at the top is (in lbf)

```
math.pi*(5./6.)**2*20000
```

i.e. 43633.231, which is to be compared to the total force applied on the bottom face

```
math.pi*(1.0)**2*13888.9
```

i.e. 43633.266. We can see that the two forces only agree to six decimal places, so a force on the order of 10^{-2} lbf is the imbalance. However tiny this is compared to the applied forces, given the near singularity of the stiffness matrix it is still able to produce a huge displacement.

> 👣 Never rely on the applied loads to be perfectly in balance. Always stabilize the structure
>   to catch any imbalance that can be introduced by the errors of computer arithmetic.

The problem is easily fixed by applying a point constraint in the axial direction. The associated reaction will turn out to be on the order of 10^{-2} lbf, which is not zero (as it ideally should be for that point constraint to be admissible), but since the applied forces are several orders of magnitude bigger than the reaction, this inaccuracy is acceptable.

7.7 Free vibration (frequency) analysis

Free vibration (frequency) analysis was covered in Section 5.7.3. Solving free vibration problems accurately is tougher than solving statics problems: The procedure to solve for the natural frequencies and the mode shapes is nonlinear. Furthermore, we are typically interested in frequencies which are fairly low. Such frequencies correspond to deformations of the structure in which the structure is most flexible. If the finite elements that we use cannot handle these deformation modes accurately, the frequencies will come out with large errors. An example is the deformation of thin plates: The lowest frequencies correspond to bending of the plates. If the finite element cannot represent bending accurately, the errors will potentially be huge. The next Section illustrates the problem with tetrahedral finite element models.

7.7.1 Modal analysis of a circular clamped plate

As our first example, we consider the vibration of a moderately thick circular plate as shown in Figure 7.18. The cylindrical surface is fully clamped (all displacements zero). The material is isotropic. The analytical solution has been analytically worked out in dependence on the number of "nodes" (locations of approximately zero displacement) radially and circumferentially [2]. Therefore, this is a very good example with which to test a finite element solver (a so-called benchmark).
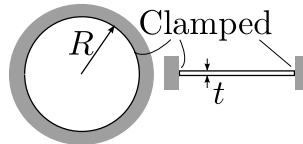
Fig. 7.18. Clamped circular plate (drum).

The finite element solution adopts the quadratic tetrahedron C3D10 (the quadratic T10). The natural frequencies obtained for a relatively coarse mesh are illustrated in Figure 7.19. Evidently, the element T10 produces estimates of the frequencies which are of good engineering accuracy.

The graph of Figure 7.20 may serve as a crude guide to the relative accuracy of the two tetrahedral elements, the linear T4 and the quadratic T10. The four natural frequencies are computed for finer and finer meshes (the analytical results are indicated with horizontal red lines). The quadratic element solutions approach the reference values quickly. The linear tetrahedral element solutions approach the reference values very slowly. (Slow convergence is bad: a large number of elements, and hence large expense, are needed for acceptable accuracy.)

> 👣 Sometimes the linear tetrahedron will be as bad as in this example, but it could also
>   be much worse. Insights of this nature are critical to an effective FE analyst. Picking
>   the wrong element for the job can thoroughly spoil our day.

This example is also available in a tutorial form. Please follow the link in the margin.
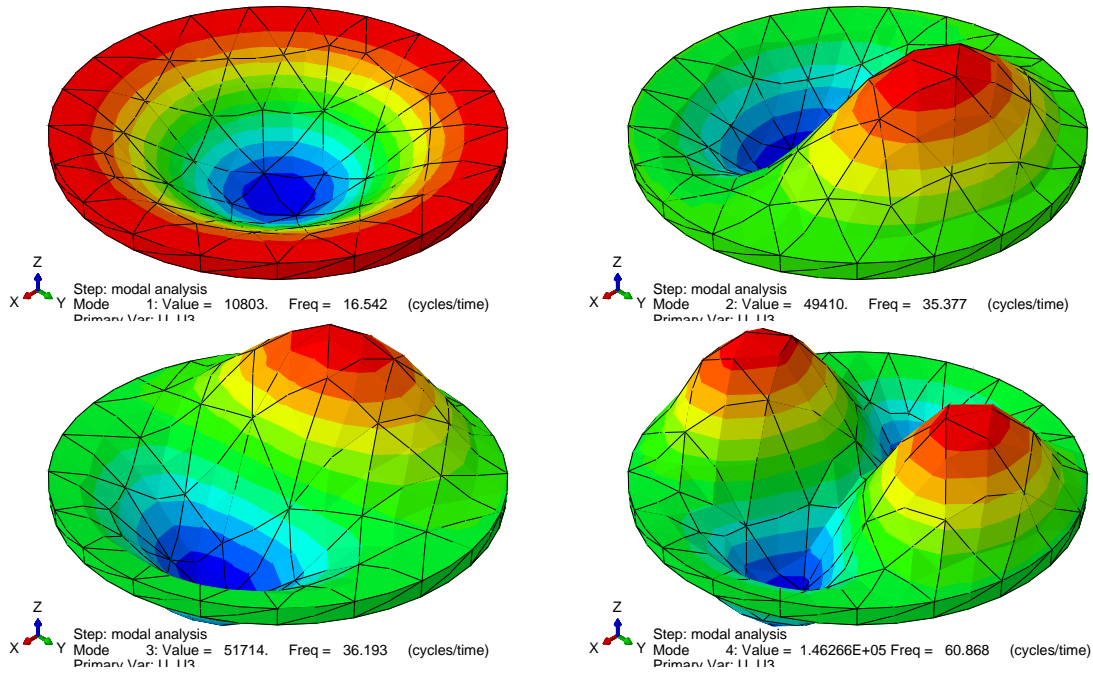
Fig. 7.19. Mode shapes of thick clamped plate. The analytical solution for the natural frequencies yields for these modes $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz
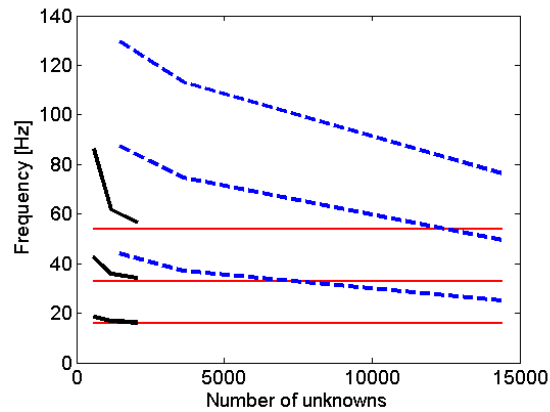
Fig. 7.20. Comparison of the first four natural frequencies of the drum computed with the two tetrahedral elements, the T4 (dashed line) and the T10 (solid line). The analytical solution for the natural frequencies, $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz, is indicated with horizontal lines.

7.8 Principle of equivalent loads (Saint-Venant's principle)

Shear traction components are often generated by frictional contact between interacting bodies. However, contact problems are well outside the scope of this textbook, they are nonlinear and involve inequality constraints. The other situation in which we might wish to apply shear tractions is when we formulate simplified models in which the effect of the omitted part of a structure is introduced as a resultant (force or torque) into the model.

As an example, we will consider a shaft of circular cross-section, with one through-hole. The focus of our interest is the local stress concentration around the hole when the shaft is subjected to a known torque. As we do not wish to model the actual transmission of the torque into the shaft,

the geometry of the shaft is reduced to just the small neighborhood of the hole, and the torque generated at the end points is applied as prescribed shear tractions in the end cross-sections of the short stump.
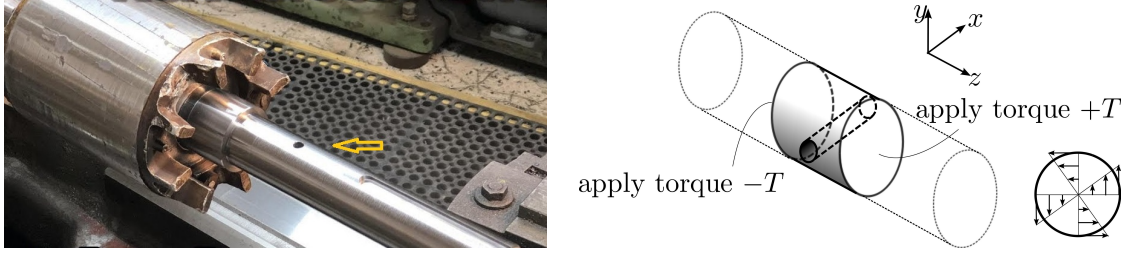
Fig. 7.21. Example of boundary conditions: shaft with a through-hole loaded by torque.

The way in which we distribute the shear tractions is only an approximation of the stress distribution that would exist in the complete part. Based on experimental observations accompanied by analyses of a few particular cases, the so-called **_Saint-Venant's principle_** [9, 1], may be invoked: If a set of self-equilibrated tractions is applied on a limited subset of the boundary, its effect will be negligible beyond a certain range. By necessity, this principle is somewhat vague, and its applicability needs to be assessed case-by-case. Figure 7.22 illustrates the meaning: consider a beam of solid section, to which a traction $\bar{t}$ of a nonzero resultant is applied. If the traction is perturbed by a self-equilibrated load $\hat{t}$, the stresses will change significantly only in a region extending in all directions approximately by the characteristic dimension of the beam d.
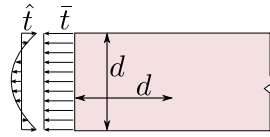
Fig. 7.22. Illustration of Saint-Venant's principle

Coming back to the shaft: relying on the Saint-Venant's principle, we apply the resultant torque by any convenient distribution of shear tractions. Again, this is a pure-traction problem, so essential boundary conditions to prevent rigid body motion should be added to make the solution unique. In this case, for instance a plane of anti-symmetry exists, or point supports could be added at convenient locations (for instance on the axis of the shaft).

7.9 Supports applied to nodes

Oftentimes the modeling abstraction of a structure does not lead to sufficient number of constraints on displacement, and the structure will have one or more rigid-body displacement options. In order to suppress the rigid body displacements we could apply constraints on displacements at the nodes.

The displacement constraints needs to be applied in some coordinate system, which will assume will be always described by Cartesian triple of vectors. Unless the restraints are defined in the global Cartesian coordinate system, we must always say in which coordinate system the support conditions are defined.

1. If we wish to force a node k to **_move on a surface_**, we need to establish a coordinate system with two vectors tangent to the surface at the node k. The third vector is then normal to the surface, and the desired constraint is then expressed as $u_n = 0$. Here u_n is the displacement components normal to the surface.

2. If we wish to force a node k to **move on along a curve**, we need to establish a coordinate system with one vector tangent to the curve at the node k, let us say we call that vector $\boldsymbol{t}$. The remaining two basis vectors of the coordinate system are normal to the curve, we could call them $\boldsymbol{n}$ and $\boldsymbol{m}$. The desired constraint is then expressed as $u_n = u_m = 0$. Here u_n and u_m are the displacement components normal to the curve.

3. If we wish to force a node k to be **immovable**, we can express the constraint in any Cartesian basis at the node: $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, and $\boldsymbol{e}_3$. The desired constraint is then expressed as $u_1 = u_2 = u_3 = 0$. Here u_1, u_2, and u_3 are the displacement components in the basis.

Provided the constraints are expressed in the global Cartesian coordinate system, we don't need to introduce any ad hoc coordinate systems. In the global Cartesian coordinate system we have for instance these constraints:

- Apply $u_x = u_z = 0$ at node k: force the node to move only in the direction of the y-coordinate (along the coordinate curve).
- Apply $u_y = 0$ at node k: force the node to move only in a plane parallel to x, z.
- Apply $u_x = u_y = u_z = 0$ at node k: the node becomes immovable.

Furthermore, note that we are here concerned with deformable bodies. For the purpose of the application of the restraints against rigid body displacements we will consider all bodies to be rigid. However, we will have to make allowance for the possibility of the supported body deforming under the loads. Therefore, the applied constraints must not prevent the deformation from occurring. As a whole, the system of applied restraints must allow for unhampered deformation of the analyzed structure.

> Always be mindful of the inadmissibility of single-point constraint should the associated reaction be nonzero.

7.9.1 Metal strip: example of free-floating structure

The metal strip with hemispherical dimples shown in Figure 7.23 is subjected to self-equilibrated tensile load in the x-direction. Note that there is no plane of symmetry in this problem. The goal is to suppress rigid body motion by applying node restraints. A fully free three-dimensional rigid body has six rigid body modes (six DOFs). To prevent the body from moving, six force reactions need to be generated, which means we need to apply six point constraints.
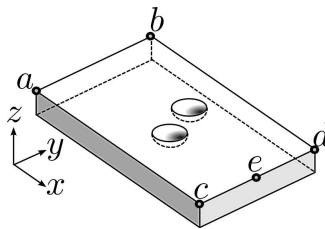
Fig. 7.23. Strip with dimples under tensile loads.

There is no unique solution. There are several possibilities. Here is one: call for Figure 7.24(A). We will force the nodes a and b to move along the curve in the y direction only. Introducing four constraints in this way, two DOFs are still left. We will suppress those by applying the restraint at the node e by setting the y and z displacements at node e to zero: this node can now only move on a coordinate curve parallel to the x direction.

Here is another possibility. We can enforce a restraint at the nodes a, b, and d to move on a plane parallel to the x, y plane. Consequently we suppress the z displacements at these three nodes.

Three DOFs are therefore left. We can suppress those by constraining the b and d nodes to move on a plane with the normal in the y-direction. One more restraint is needed, to prevent sliding in the direction of the x- axis. Forcing node e not to be movable in the x-direction completes the set of sufficient restraints. See Figure 7.24(B).

(A)   (B)
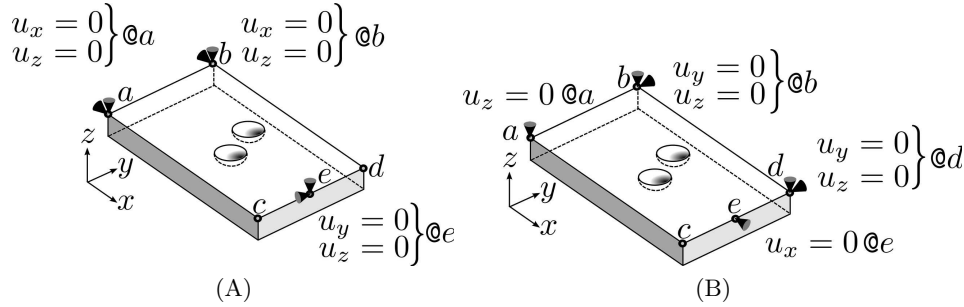
Fig. 7.24. Strip with dimples under tensile loads. Two possible configurations of supports at the nodes

Note that in Figure 7.24(B) we should not include the node c or e in the constraint to move only on a single plane parallel to x, y in addition to the three nodes a, b, and d: there is no guarantee that all of these nodes should stay in a single plane after the deformation occurs. If we added all of these nodes to the constraint, we couldn't guarantee the reactions to be all zero, and that would be against our rules.

Until we have mastered applying constraints at the nodes to free-floating bodies it is a good idea to check that the reactions will indeed be all zero. For this we need to write down conditions of equilibrium of the free-floating body: the sum of forces and the sum of torques must both vanish.

We will consider again the metal strip with hemispherical dimples from Figure 7.23. This time the supports will be applied as shown in Figure 7.25. The goal is to check that all the supports are admissible, i.e. check that the corresponding reactions are all guaranteed to be zero.
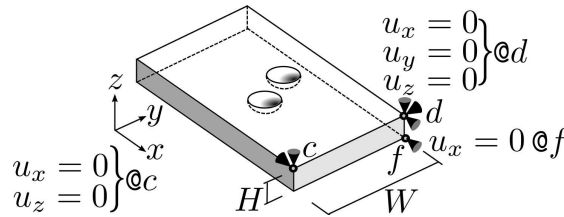
Fig. 7.25. Strip with dimples under tensile loads. Restraints applied at nodes.

Since the assumption is that the applied tensile load on the two faces orthogonal to the x direction is self-equilibrated, we can totally discount the applied load from the consideration of equilibrium and focus simply on the reactions. The reactions that correspond to the applied constraints have now been added to our sketch in Figure 7.26. The sum of the reactions in the three coordinate directions is

$$R_{d,x} + R_{c,x} + R_{f,x} = 0$$
$$R_{d,y} = 0$$
$$R_{d,z} + R_{c,z} = 0$$

Similarly, the torques of the reactions about the coordinate direction vectors, for simplicity all figured with respect to node d as that eliminates from the equations the largest number of reactions, are

$$-R_{c,z}W = 0$$
$$-R_{f,x}H = 0$$
$$R_{c,x}W = 0$$

It is a simple matter to verify that this system of six equations leads to the solution $R_{d,x} = R_{c,x} = R_{f,x} = R_{d,y} = R_{d,z} = R_{c,z} = 0$. Hence, all the reactions are guaranteed to be zero: the applied constraints are admissible.
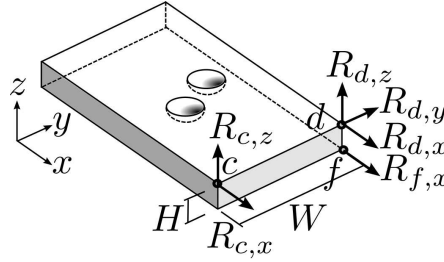
Fig. 7.26. Strip with dimples under tensile loads. Reactions.

Valuable insight can be gained by writing out the above relationships in terms of the matrix equation:

$$
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & -W & 0 \\
0 & 0 & 0 & 0 & 0 & -H \\
0 & 0 & 0 & W & 0 & 0
\end{bmatrix}
\begin{bmatrix}
R_{d,x} \\
R_{d,y} \\
R_{d,z} \\
R_{c,x} \\
R_{c,z} \\
R_{f,x}
\end{bmatrix}
=
\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}
\tag{7.28}
$$

The coefficient matrix on the left is clearly non-singular, it is easy to see how we could get this matrix in upper-triangular form with non-zero numbers on the diagonal by swapping rows six and five and five and four, and therefore the only solution for a zero right-hand side is the homogeneous one (all unknowns zero). If the coefficient matrix was singular, the solution could consist of some non-zero reactions, and we would have to conclude that the support conditions were inadmissible.

Furthermore, if the coefficient matrix was non-square (rectangular), it might be possible for the solution of the equilibrium equations to include some nonzero reactions. For instance, let us say there are more reactions then there are equations (the matrix is wider than it is tall). Then the equations are insufficient to determine the solution uniquely, and the support conditions may be inadmissible since some of the reactions may be non-zero. On the other hand, if there are fewer reactions than there are equations of equilibrium, the matrix is singular (there are more rows than there are columns), and we cannot guarantee the stability of the supported body: we need to add an appropriate number of additional supports.

7.10 Advanced three-dimensional elements

In professional finite element analysis work one straightaway perceives the need for using tools well-suited to the purpose. It is difficult to be efficient when the tools are flawed. Considering stress analysis of three-dimensional solids with hexahedral elements, the flaws of plain-vanilla finite elements are inescapable: the isoparametric hexahedron is too stiff in bending, much too susceptible to deterioration of performance in distorted configurations, especially when used with high aspect ratio (such as for plates or shells), and finally, using isoparametric hexahedra for analyses with little compressibility, such as for elastomeric structures or when plasticity of metals is of concern, is a frustrating exercise as the element "locks" (there is much too little of the correct deformation).

To design a hexahedral element capable of good performance under all these requirements is not a trivial proposition. Abaqus implements at least two decent candidates, the element with incompatible modes and the mean-strain element. The standard isoparametric element is too stiff because it reacts with too much resistance to some strains. These two approaches attempt to improve the response by either

- enhancing the representation of the strains so that the strains are correct, or
- making the troublesome strains invisible to the element.

In the first approach, additional possible deformation patterns are added to the element as internal degrees of freedom: If we make it possible for the element to represent bending deformation by incorporating the possibility of curving, it will not need to produce shear strains to deform. However, this will be in addition to the usual straight-edge mode of deforming. Since each element now has its own way of curving that is not shared with its neighbors, the resulting displacement pattern in the mesh is incompatible (the elements don't fit together perfectly after deformation). The element in Abaqus is called C3D8I, where the "I" stands for incompatible.

In the second approach, instead of the $2 \times 2 \times 2$ Gauss quadrature, which is the logical choice for the standard element, an alternative, reduced, integration scheme is used. For instance, if only a single integration point was used, the element would be blind to its distortion in the form of a bending pattern (see Figure 7.8). Unavoidably, not seeing this distortion means that "too much resisting force" is replaced with "no resisting force at all". Consequently the needed resistance must be added in the form of stabilization. Since when the element is blind to distortion strains, the unresisted deformations visually resemble the shape of an hourglass, the element is labeled "susceptible to hourglassing". The stabilization should make it impossible for the element to deform by hourglassing without offering some resistance. There are several possibilities how to prevent hourglassing, of which perhaps the "enhanced" assumed strain stabilization is the most successful. The resulting element is called C3D8RH (with enhanced hourglass control).

> The name of the element (C3D8RH) does not reflect the hourglass control. When interpreting results obtained with an hourglass-controlled element, check how it is stabilized: some forms are better than others. Choosing the right one may be tricky. The manuals have a lot more detail, but experience is essential.

In addition, the tetrahedral and hexahedral elements in Abaqus can also be formulated by *independently* approximating *displacements* and hydrostatic *pressure*. This can be effective in dealing with deformations with limited compressibility. The element is then employed in the so-called hybrid formulation. The hybrid formulation is indicated with an "H" appended to the element name. For instance, C3D8IH is the hybrid formulation of the incompatible-mode hexahedron C3D8I.

7.11 Analyzing shell structures with hexahedra and tetrahedra

The cylindrical shell roof of Figure 7.27 was chosen as here both membrane and bending stresses are of importance. The model was originally solved by Scordelis and co-workers in 1964 and it has been used since frequently for assessment of finite element performance (i.e. as a benchmark).

The shell is relatively thin (radius to thickness ratio of 100), it is supported by circular rigid diaphragms along the curved edges, loaded by distributed body load (self weight), and the target quantity is the deflection of the free straight edge at point A in the direction of the gravity vector $\mathbf{g}$. Because of the two planes of symmetry, only a quarter of the shell is modeled.

We use three finite elements, each with three progressively finer meshes. Each mesh has a single element through the thickness, but $n \times n$ elements along the supported arc and the free edge of the shell. Figure 7.28 shows two such meshes, for the hexahedral shapes on the right and the tetrahedral element on the left.

The best performer is the hexahedron H20 with the so-called reduced integration (Gauss rule with $2 \times 2 \times 2$ points), i.e. the Abaqus C3D20R. It delivers exceptional accuracy even with a mesh
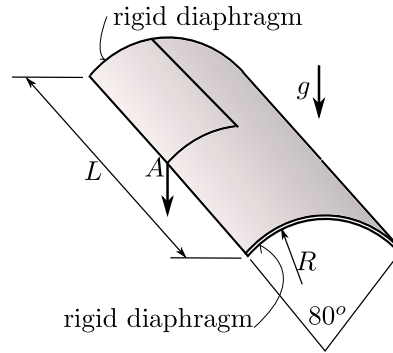
Fig. 7.27. Scordelis-Lo cylindrical shell. Quarter of the shell is modeled.
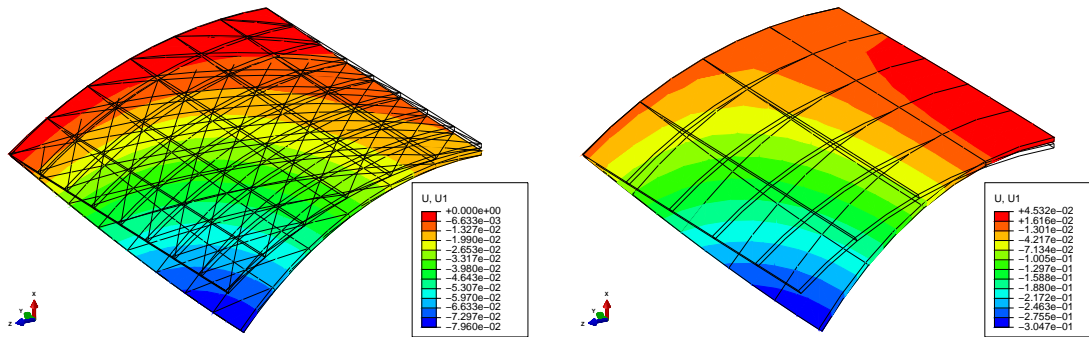
Fig. 7.28. Scordelis-Lo cylindrical shell. Magnified deflected shape shown for the T10 tetrahedron on the left, and H20 hexahedron on the right.

of two-by-two elements. The fully-integrated version of the H20 element, with the $3 \times 3 \times 3$ Gauss rule, is much stiffer (C3D20). With the coarse meshes ($n = 2, 5$) we get only a fraction of the correct deflection. The element H20 with full integration eventually converges, but clearly here we have an example of getting *more for doing less*: for the fully-integrated element we need to evaluate the finite element expressions at 27 points, while with the reduced-integrated element we only need 8. So we get much better performance for one third of the effort.

The tetrahedral element is even stiffer (less accurate) than the fully-integrated H20, and improves slowly. The fault lies with the inability of this element to handle very large aspect ratios (the element is short in the thickness direction but elongated along the surface of the shell). In addition, the elements are also curved. That also goes down poorly.

Best to avoid using tetrahedral T10 elements for shell-like structures.

7.12 Harmonic Forced Vibration Analysis

In Harmonic Vibration Analysis (HVA) — a.k.a. forced harmonic vibration, or Steady-State Dynamics (SSD) — we assume that there are possibly three sources of loading, distributed body load, distributed traction load on the surface, or loading by moving supports. The weighted residual equation for this case is written as (refer to See Box 28)
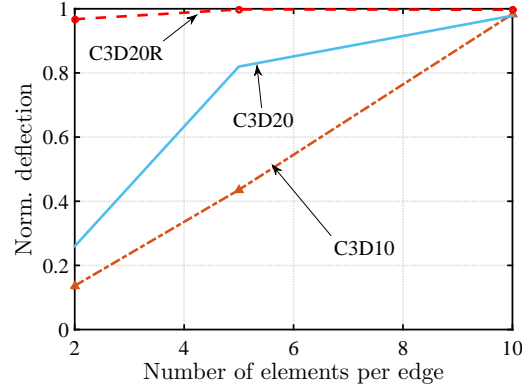
Fig. 7.29. Scordelis-Lo cylindrical shell. Comparison of the deflection at point A for three different element types. Convergence with the respect to the number of elements along the edges of the shell.

$$\sum_{p=1}^{N_{\mathrm{f}}} \int_V N_{j(q)}\rho N_{i(p)} \,\mathrm{d}V\,\delta_{r(q)s(p)}\,\ddot{U}_p + \sum_{p=1}^{N_{\mathrm{f}}} C_{qp}\dot{U}_p + \sum_{p=1}^{N_{\mathrm{f}}} \int_V \left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} \,\mathrm{d}V U_p$$

$$+ \sum_{p=N_{\mathrm{f}}+1}^{N} \int_V N_{j(q)}\rho N_{j(q)} \,\mathrm{d}V\,\delta_{r(q)s(p)}\,\ddot{U}_p + \sum_{p=N_{\mathrm{f}}+1}^{N} C_{qp}\dot{U}_p + \sum_{p=N_{\mathrm{f}}+1}^{N} \int_V \left[[\boldsymbol{B}_{j(q)}]^T[\boldsymbol{D}][\boldsymbol{B}_{i(p)}]\right]_{r(q)s(p)} \,\mathrm{d}V U_p$$

$$- \int_V N_{j(q)}[\overline{\boldsymbol{b}}]_{r(q)} \,\mathrm{d}V - \int_{S_{t,r(q)}} N_{j(q)}\overline{t}_{r(q)} \,\mathrm{d}S = 0$$

$$(7.29)$$

where we still keep the general time dependence. Notice that we have also added the possibility of a damping (velocity-proportional or a viscous) force. The above weighted residual equation results in the matrix expression, which is a system of ordinary differential equations,

$$[\boldsymbol{M}][\ddot{\boldsymbol{U}}] + [\boldsymbol{C}][\dot{\boldsymbol{U}}] + [\boldsymbol{K}][\boldsymbol{U}] = [\boldsymbol{L}] \,, \tag{7.30}$$

where $[\boldsymbol{M}]$ is the mass matrix, $[\ddot{\boldsymbol{U}}]$ is the vector of accelerations (second derivatives of the degrees of freedom with respect to time), $[\boldsymbol{C}]$ is the damping matrix, $[\dot{\boldsymbol{U}}]$ is the vector of velocities (first derivatives of the degrees of freedom with respect to time), $[\boldsymbol{K}]$ is the stiffness matrix, $[\boldsymbol{U}]$ is the vector of displacement degrees of freedom, and $[\boldsymbol{L}]$ is the vector of the time-dependent loadings of all kinds.

Now we introduce the harmonic time dependence. We do this with a complex exponential in order to account for the possibility of the loading and the response occurring at different phases of the cycle. So we assume the vector of loads is harmonic, i.e. $\boldsymbol{L} = \exp(\mathrm{i}\omega t)\widetilde{\boldsymbol{L}}$, where ω is the frequency of forcing, and $\widetilde{\boldsymbol{L}}$ is a time-independent vector that describes the distribution of the load over the structure, and analogously the response is assumed as $\boldsymbol{U} = \exp(\mathrm{i}\omega t)\widetilde{\boldsymbol{U}}$. Differentiating and factoring out the time dependence $\exp(\mathrm{i}\omega t)$ leads to

$$-\omega^2 \boldsymbol{M}\widetilde{\boldsymbol{U}} + \mathrm{i}\omega \boldsymbol{C}\widetilde{\boldsymbol{U}} + \boldsymbol{K}\widetilde{\boldsymbol{U}} = \widetilde{\boldsymbol{L}} \,, \tag{7.31}$$

or

$$\left(-\omega^2 \boldsymbol{M} + \mathrm{i}\omega \boldsymbol{C} + \boldsymbol{K}\right)\widetilde{\boldsymbol{U}} = \widetilde{\boldsymbol{L}} \,. \tag{7.32}$$

The matrix $-\omega^2 \boldsymbol{M} + \mathrm{i}\omega \boldsymbol{C} + \boldsymbol{K}$ is the dynamic stiffness (frequency response function matrix), and the time-independent vector $\widetilde{\boldsymbol{U}}$ is a complex quantity which encodes both the amplitude and the phase shift of the components of displacement of the structure under the harmonic loads.

The damping matrix C_{qp} is often modeled with the so-called Rayleigh stiffness- and mass-proportional damping. Then we take

$$C_{qp} = \alpha_R M_{qp} + \beta_R K_{qp} , \tag{7.33}$$

i.e. the damping matrix is taken as a linear combination of the mass matrix and the stiffness matrix. The coefficients of the linear combination can be determined by matching the damping ratio ζ at one or two selected frequencies. For instance, if we wish to apply a given damping ratio ζ at a single frequency ω_0, we may assume that the two mechanisms contribute equally and solve for the coefficients from

$$\frac{1}{2}\zeta = \alpha_R/\omega_0 \quad \text{and} \quad \frac{1}{2}\zeta = \beta_R\omega_0 \quad . \tag{7.34}$$

On the other hand, if we wish to apply a given amount of damping, with damping ratios ζ_1 and ζ_2 at two separate frequencies ω_1 and ω_2, we need to solve the system of coupled equations

$$\zeta_1 = \alpha_R/\omega_1 + \beta_R\omega_1 , \quad \zeta_2 = \alpha_R/\omega_2 + \beta_R\omega_2 , \tag{7.35}$$

for the combination of the damping coefficients α_R, β_R.

7.12.1 Response of a thin plate

In this section we will consider a comprehensive example that will consider vibration of a square simply-supported plate under transverse dynamic loads. The test is recommended by the National Agency for Finite Element Methods and Standards (U.K.): Test 13 from NAFEMS "Selected Benchmarks for Forced Vibration," R0016, March 1993.

The steel plate has dimensions of 10×10 m with the thickness of 0.05 m. The domain is discretized with 20-node hexahedral solid elements, 8×8 elements in the plane of the plate and a single element through the thickness . The simple support condition is approximated by distributed rollers on the boundary to enforce zero transverse displacement (but allowing in-plane displacements). Because only the out of plane displacements are prevented, the structure has three rigid body modes in the plane of the plate. The non-zero transverse modes of vibration are illustrated in Figure 7.30. The nonzero benchmark frequencies are 2.377 (fundamental), 5.961, 5.961, 9.483, 12.133, 12.133, 15.468, and 15.468 [Hz]. The finite element model is sufficiently accurate to deliver natural frequencies within a fraction of a percent, with the maximum error for the eight frequencies just over 2%.

Abaqus
- CAE file
- tutorial

Step: Step-frequency
Mode   4: Value = 221.26   Freq = 2.3674   (cycles/time)
Primary Var: U, Magnitude

Step: Step-frequency
Mode   6: Value = 1392.2   Freq = 5.9384   (cycles/time)
Primary Var: U, Magnitude

Step: Step-frequency
Mode   9: Value = 5611.2   Freq = 11.922   (cycles/time)
Primary Var: U, Magnitude

(a)   (b)   (c)
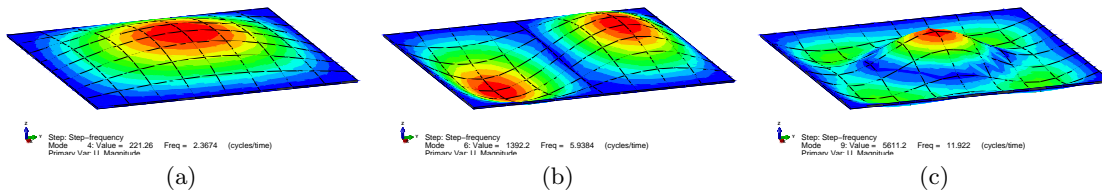
Fig. 7.30. Thin simply-supported plate. Mode shapes: (a) Fundamental frequency at 2.367 (2.377) Hz, (b) second natural frequency at 5.938 (5.961) Hz, (c) sixth natural frequency at 11.922 (12.133) Hz. The reference values are shown in parentheses.

The harmonic forced vibration is studied for uniform transverse load on the top surface of the plate $F = 100\sin(\omega t)$ N/m^2, in the range of 0 to 15 Hz. By varying the frequency of excitation ω we can sweep through the frequencies with equation (7.32). If we focus on a single quantity, as for instance in this example where we are interested in the transverse displacement of the center point of the plate, we get a curve that expresses how the quantity changes with frequency. Figure 7.31(a,b) shows the real and imaginary parts of the complex frequency response function $H_u(\omega)$. As expected,

the real component goes through zero at the peak response of the plate (the resonance), while the imaginary part peaks at the resonance. Figure 7.31(c) demonstrates the large peak in the modulus of the transfer function at the fundamental frequency which is due to the resonance of the plate when the forcing passes through the fundamental frequency. The reference NAFEMS solution for the peak displacement is 45.42 mm, which is almost equal to the thickness of the plate. The relatively small peak around 12 Hz is due to the sixth mode (Figure 7.30(c)). Finally, Figure 7.31(d) shows the phase of the complex frequency response function $H_u(\omega)$.
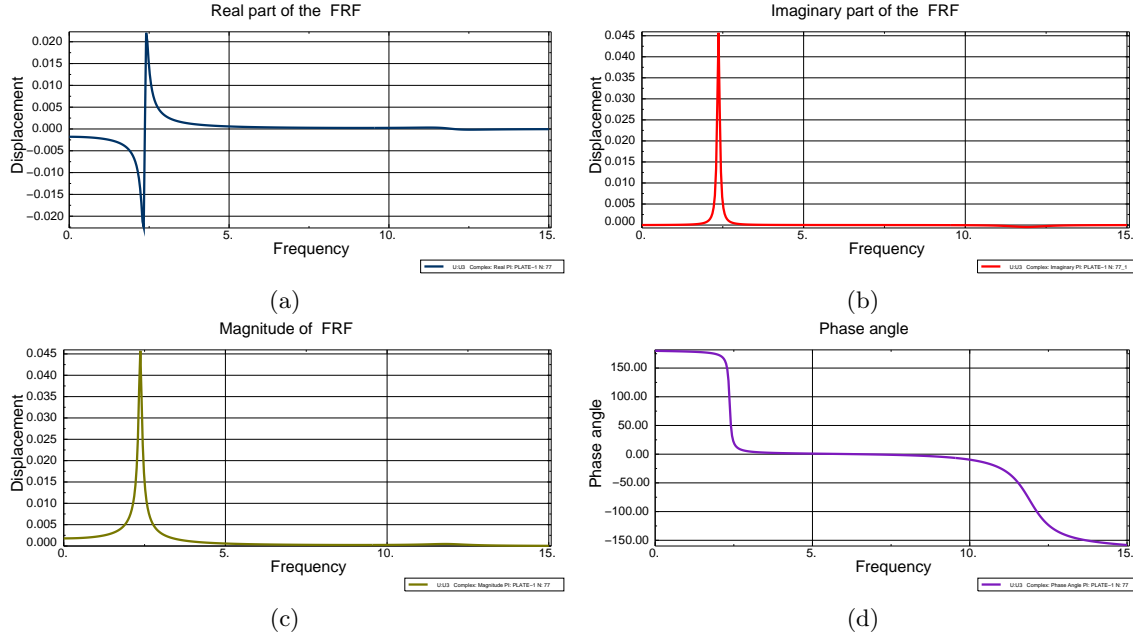
Fig. 7.31. Thin simply-supported plate. (a) Real part of the frequency response function of the transverse displacement of the center of the plate $\Re H_u(\omega)$. (b) Imaginary part of the frequency response function of the transverse displacement of the center of the plate $\Im H_u(\omega)$. (c) Magnitude (modulus) of the frequency response function of the transverse displacement of the center of the plate $|H_u(\omega)|$. (d) Phase angle of the frequency response function of the transverse displacement of the center of the plate.

Resonance phenomena are especially significant in stress analysis. Mechanical resonance is the tendency of a mechanical system to respond with increased flexibility when the frequency of the forcing matches the system's natural frequency of vibration. This may cause violent vibrations and hence strong stress oscillations. In extreme cases this may result in catastrophic failure in improperly constructed structures such as bridges, trains, and aircraft. However, even when individual peaks of the stress do not lead to immediate failure, large-amplitude stress cycles may cause fatigue of the material, which over the course of the life of the structure causes damage. Figure 7.32 shows the amplitude of the frequency response function of the tensile stress at the bottom surface of the plate $|H_{\sigma_x}(\omega)|$. As the displacement in Figure 7.31, this curve also has a peak at the first natural frequency of the structure. Clearly, compared to the static value of the stress (for $\omega = 0$) of around 1.12 MPa, the resonant value of the stress of over 30 MPa is a very significant increase.

7.13 Analysis of a composite plate

In this example we will investigate a cross-ply laminated plate. It consists of three homogeneous layers (lamina), where each layer consists of a matrix reinforced with parallel fibers. The material stiffness matrix corresponds to a homogenization of the matrix + fibers system in the form of an orthotropic elasticity. For an in-depth discussion of non-isotropic material models: See Box 18 .
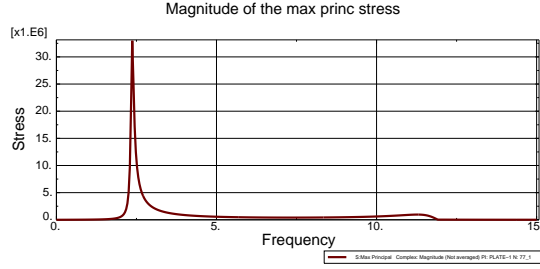
Fig. 7.32. Thin simply-supported plate. Frequency response function of the maximum of the stress component σ_x at the center of the plate $|H_{\sigma_x}(\omega)|$.

The geometry is shown in Figure 7.33, and the dimensions are $a = 200\text{mm}$, $b = 600\text{mm}$, $t = 20\text{mm}$. The plate is composed of three homogeneous layers of equal thickness, which are considered perfectly bonded together. The plate was assumed with simple-support boundary conditions and uniform transverse load. The solution for the deflection and the three-dimensional stresses was derived analytically (Table 2 [11]).
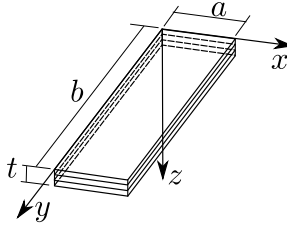
Fig. 7.33. Cross-ply laminated plate of three layers.

The material of each constituent layer is linearly elastic, and all three layers consist of the same material, they differ only in the orientation of the local material axes $\overline{x}, \overline{y}, \overline{z}$ (Cartesian coordinate system with basis vectors $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$, $\boldsymbol{e}_{\overline{z}}$). The entries of the material stiffness matrix (or the material compliance matrix) are to be understood as being expressed in the local (material) coordinate system attached to the material point in the form of the rotation matrix (5.80). where we have defined the transformation matrix

$$[\boldsymbol{R}_m] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} , \tag{7.36}$$

composed of the components of the vectors $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ on the basis vectors $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ in the columns of the material orientation matrix. This matrix is recognized as an orthogonal matrix (rotation matrix), whose inverse is its transpose

$$[\boldsymbol{R}_m]^{-1} = [\boldsymbol{R}_m]^T .$$

The material of all three layers is of the *same* orthotropic type in the material coordinate system, and it is described using the so-called "engineering constants": E_1, E_2 , and E_3 are Young's moduli corresponding to the tensile (compressive) loading in the x, y, and z directions, respectively; ν_{23} is Poisson's ratio that characterizes normal deformation in the z-direction when the load is applied in the y-direction, ν_{13} is Poisson's ratio that characterizes normal deformation in the z-direction when the load is applied in the x-direction, and ν_{12} is Poisson's ratio that characterizes normal deformation in the y-direction when the load is applied in the x-direction; and G_{23} , G_{13} , and G_{12} are shear moduli corresponding to the shear load applied in the $y - z$, $x - z$, and $x - y$ planes, respectively. Poisson's ratios and Young's moduli are related by the reciprocal relations as

$$\nu_{ij}E_j = \nu_{ji}E_i, \quad i,j = 1, 2, 3, i \neq j \ . \tag{7.37}$$

In the present case, each layer is homogeneous, and the material orientation matrix is defined by the angle used to derive the local coordinate axes from the global coordinate system by rotation about the global z- axis. For the three layers considered here, the angles are $0^o/90^o/0^o$.

7.13.1 Strain-displacement matrix for anisotropic materials

The entries of the material stiffness matrix are expressed on the local Cartesian basis of material orientation directions. The elementwise stiffness matrix links nodal displacements to nodal forces and they are typically expressed in the global Cartesian coordinate system. Hence, we see that there is a need to transform between the local material directions and the global Cartesian basis. Drawing on the example of the element stiffness matrix (7.16), the element's restoring force may be expressed as

$$\boldsymbol{F}_e = \boldsymbol{K}_e \boldsymbol{U}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \ \mathrm{d}V \ \boldsymbol{U}_e \ , \tag{7.38}$$

While the displacement components in $\boldsymbol{U}_e$ and the force components in $\boldsymbol{F}_e$ are in the global Cartesian basis, the material stiffness matrix is expressed on the basis of the local material directions. Evidently, the linear algebra operations between the material matrix and displacements/forces must incorporate the transformation from one basis to the other. One possibility is to have transformation built into the strain-displacement matrix, which then takes displacement in the global Cartesian basis and produces strains in the local material coordinate system. This can be accomplished by writing

$$[\boldsymbol{\epsilon}]^{(\overline{x})} = \mathcal{B}^{\overline{x}}[\boldsymbol{u}]^{(\overline{x})} = \mathcal{B}^{(\overline{x})}\left([\boldsymbol{R}_m]^T[\boldsymbol{u}]^{(x)}\right) = \left(\mathcal{B}^{(\overline{x})}[\boldsymbol{R}_m]^T\right)[\boldsymbol{u}]^{(x)} = \mathcal{B}^{(\overline{x},x)}[\boldsymbol{u}]^{(x)} \tag{7.39}$$

where we use the superscript $(\overline{x})$ or (x) to indicate in which coordinate system the components are expressed. The strain-displacement operator $\mathcal{B}^{(\overline{x},x)} = \mathcal{B}^{(\overline{x})}[\boldsymbol{R}_m]^T$ incorporates the geometric transformation $[\boldsymbol{R}_m]^T$ of the displacement vector components from the global basis into the local material directions basis. The global-to-local ***strain-displacement matrix*** for node k is therefore defined as

$$\boldsymbol{B}_k^{(\overline{x},x)} = \mathcal{B}^{(\overline{x},x)}\left(N_k(\boldsymbol{x})\right) = \mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)[\boldsymbol{R}_m]^T = \boldsymbol{B}_k^{(\overline{x})}[\boldsymbol{R}_m]^T \tag{7.40}$$

The matrix $\boldsymbol{B}_k^{(\overline{x})}$ requires gradients of the basis functions with respect to the *material* Cartesian directions. To obtain the gradients of the basis functions from (3.57) the matrix of node coordinates is simply transformed to the Cartesian basis of material directions as

$$[\overline{\boldsymbol{x}}] = [\boldsymbol{x}][\boldsymbol{R}_m] \tag{7.41}$$

so that the Jacobian matrix is computed in the local material coordinates

$$\left[\overline{\boldsymbol{J}}\right] = [\overline{\boldsymbol{x}}]^T \left[\mathrm{grad}_{(\xi,\eta,\zeta)}N\right] \ , \tag{7.42}$$

and

$$\left[\mathrm{grad}_{(\overline{x})}N\right] = \left[\mathrm{grad}_{(\xi,\eta,\zeta)}N\right]\left[\overline{\boldsymbol{J}}\right]^{-1} \ . \tag{7.43}$$

7.13.2 Cross-ply plate results

The plate is studied using the so-called stacked solid continuum modeling. This means that each layer is modeled with continuum (hexahedral) elements, possibly with multiple elements through the thickness, and the individual layers are bonded together through tie constraints.

The material constants of the orthotropic material on the local material coordinates system are taken as $E_1 = 172350\mathrm{MPa}$, $E_2 = E_3 = 6894\mathrm{MPa}$, $G_{12} = G_{13} = 3447\mathrm{MPa}$, $G_{23} = 1378.8\mathrm{MPa}$, $\nu_{23} = \nu_{13} = \nu_{12} = 0.25$.

For an isotropic material the (only?) reason we may wish to use different coordinate systems in which to express the stresses and strains has to do with the geometry of the structure. For instance if the structure is of a cylindrical shape or has an axis of symmetry, a natural choice would be a cylindrical coordinate system in which to express the results of the simulation.

For materials with fewer symmetries, such as transversely isotropic or orthotropic, there are usually two coordinate systems in which to express stresses and strains. The first coordinate system again has to do with the geometry of the structure. The default would be a global Cartesian coordinate system. The second coordinate system would be the local material coordinates. In this coordinate system, the response of the material is as easy to understand as possible (put another way: in any other coordinate system it is much harder to relate strains and stresses).

Figure 7.34 shows isosurface rendering of the two normal stresses σ_x, and σ_y, in the global coordinate system. The maxima occur in the first and third layer for σ_x (bending in the short dimension), and in the middle layer (bending in the long direction, concentrated near the supported faces of the plate).
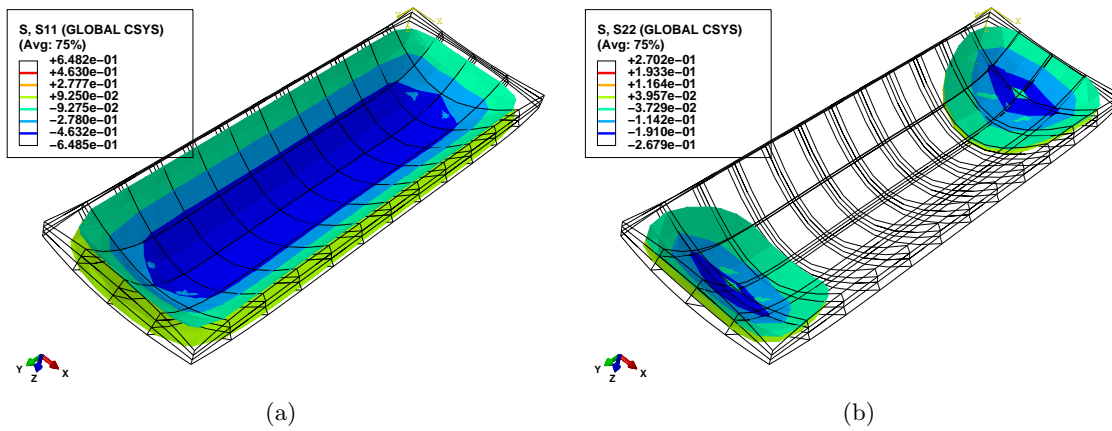
Fig. 7.34. Cross-ply laminated plate. Deformed shape with color-coded isosurfaces (a) of σ_x, (b) of σ_y.

Stress (and strain) data from laminated plates are often visualized with the help of graphs through the thickness. These are helpful to highlight features of stress that are invisible on the surface. Figure 7.35 shows the graphs of the components of stress in the global Cartesian coordinate system along two paths through the thickness of the plate: σ_x along the path through the center of the plate, and σ_y at the location of the maximum of that stress component. Note that even though the lines of the graphs seem to imply continuity of stress, the stress in fact changes discontinuously (jumps) between layers.

Figure 7.36 shows the shear stress σ_{xz} (in global coordinates). In order to visualize the variation of the shear stress at the interior of the plate, display groups have been applied to separately display the filled contours on the second and third layer. Since the second layer is soft in shear parallel to the fibers (global y direction), the shear deformation is mostly concentrated in that layer, and this fact can be used to interpret the deformation along the simply-supported edge of the plate shown in Figure 7.36.

7.14 Background, explanations, details

Box 28. Three-dimensional elastodynamics model

The weighted residual (WR) formulation for elastodynamics is derived using the following steps:

, S11 (Not averaged): Norm Dist. along 'Center' w/t Intersections

(a)

-1) (Not averaged): Norm Dist. along 'S22Max' w/t Intersections

(b)
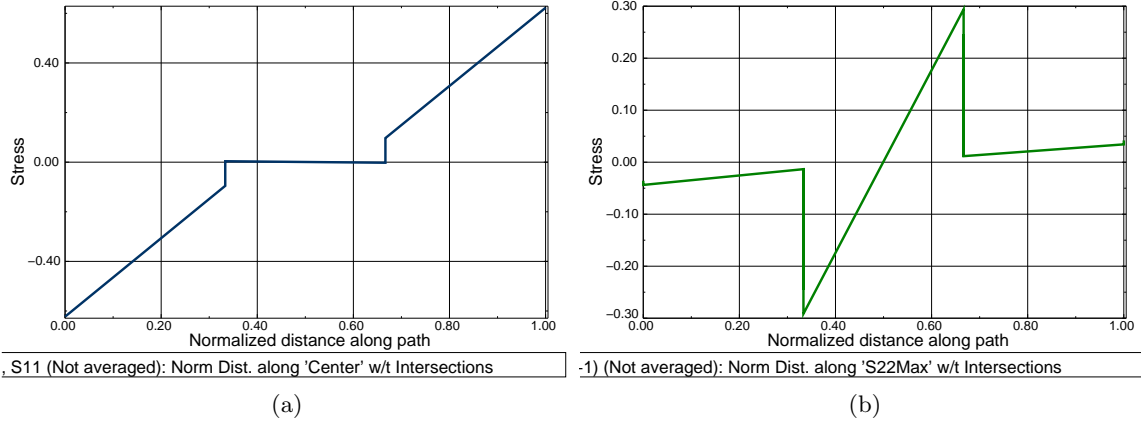
Fig. 7.35. Cross-ply laminated plate. Graphs of (a) of σ_x (at the center of the plate), (b) of σ_y (at the location of the maximum).

(a)

(b)
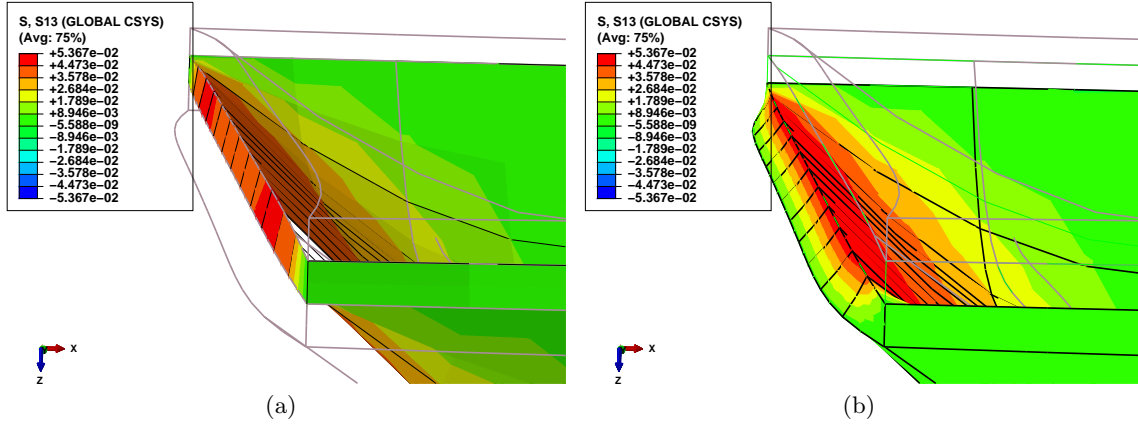
Fig. 7.36. Cross-ply laminated plate. Deformed shape with color-coded σ_{xz}. (a) Filled contour on layer 2. (b) Filled contour on layer 3.

1. Introduce the weighted residual equations for the balance partial differential equation and the force boundary condition.
2. Satisfy the displacement (essential) conditions by design of the trial functions: that will force the test functions to vanish along parts of the boundary.
3. Shift one derivative from the trial function to the test function. This will incorporate the natural boundary conditions in the balance residual equation (and eliminate the force boundary condition residual from further consideration).

The result is the WR in the form of a balance equation for each free degree of freedom q in the direction $r(q)$ at node $j(q)$

$$
\sum_{p=1}^{N_{\mathrm{f}}} \int_V N_{j(q)} \rho N_{i(p)} \,\mathrm{d}V \; \ddot{U}_p + \sum_{p=1}^{N_{\mathrm{f}}} C_{qp} \dot{U}_p + \sum_{p=1}^{N_{\mathrm{f}}} \int_V \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}] \right]_{r(q)s(p)} \,\mathrm{d}V \; U_p
$$

$$
+ \sum_{p=N_{\mathrm{f}}+1}^{N} \int_V N_{j(q)} \rho N_{j(q)} \,\mathrm{d}V \; \ddot{U}_p + \sum_{p=N_{\mathrm{f}}+1}^{N} C_{qp} \dot{U}_p + \sum_{p=N_{\mathrm{f}}+1}^{N} \int_V \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}] \right]_{r(q)s(p)} \,\mathrm{d}V \; U_p
$$

$$
- \int_V N_{j(q)} \bar{b}_{r(q)} \,\mathrm{d}V - \int_{S_{t,r(q)}} N_{j(q)} \bar{t}_{r(q)} \,\mathrm{d}S = 0
$$

$$(7.44)$$

The essential boundary conditions specify the fixed degrees of freedom (and the corresponding velocities and accelerations) at node $i(p)$ in the direction $s(p)$

$$U_p = \overline{U}_p, \text{ where node } i(p) \text{ is on } S_{u,s(p)} \quad \text{for } s(p) = x, y, z .$$

$$(7.45)$$

that hold for all times. In addition, the initial conditions hold for all nodes i at the initial time $t = 0$

$$U_p(t = 0) = \overline{U}_{0p} \quad \dot{U}_p(t = 0) = \overline{\dot{U}}_{0p} \qquad \text{for } s = x, y, z .$$

$$(7.46)$$

The first line in (7.44) represents the inertial, damping, and elastic forces due to free degree of freedom displacements. The integrals define entries of the mass, damping, and stiffness matrix. The second line defines corresponding loads due to motion along the prescribed (datum) degrees of freedom. The third line corresponds to the externally applied body load and traction load.

Inertial term: Mass matrix

Two terms emerge from (7.44): firstly, the **inertial force**

$$F_{a,q} = \sum_{p=1}^{N_f} M_{qp} \ddot{U}_p ,$$

$$(7.47)$$

produced by the free accelerations $\ddot{U}_p$ which are coupled together by the **consistent mass matrix**

$$M_{qp} = \int_V N_{j(q)} \rho N_{i(p)} \, \mathrm{d}V \, \delta_{r(q)s(p)} ,$$

$$(7.48)$$

where q means *degree of freedom number* corresponding to component $r(q)$ at node $j(q)$ (which is a free degree of freedom). Note well that M_{qp} is an entry of a matrix *matrix*, addressed by the row index q and the column index p. On the right-hand side we have also a matrix.

Secondly, there is the **inertial load** produced by the acceleration of the supported nodes

$$F_{\overline{a},q} = \sum_{p=N_f+1}^{N} M_{qp} \overline{\ddot{U}}_p .$$

$$(7.49)$$

Here we have substituted the prescribed values of the acceleration for the fixed degrees of freedom p.

Resisting forces: Stiffness matrix

Two contributions result: the first is the (elastic) **resisting force** produced by the deformed material.

$$F_{e,q} = \sum_{p=1}^{N_f} K_{qp} U_p .$$

$$(7.50)$$

The matrix generating the resisting force is the **stiffness matrix**

$$K_{qp} = \int_V \left[[\boldsymbol{B}_{j(q)}]^T [\boldsymbol{D}][\boldsymbol{B}_{i(p)}] \right]_{r(q)s(p)} \mathrm{d}V ,$$

$$(7.51)$$

where q and p means degree of freedom number corresponding to direction $r(q)$ at node $j(q)$ and to component $s(p)$ at node $i(p)$; both are free degrees of freedom.

The second is the **nonzero-displacement load** due to the deformation induced by fixed essential boundary conditions. In structural analysis, this kind of load is frequently associated with the loading condition called the *support settlement*.

$$F_{\overline{e},q} = \sum_{p=N_f+1}^{N} K_{qp} \overline{U}_p .$$

$$(7.52)$$

Here we have substituted the prescribed values of the displacement for the fixed degrees of freedom p.

Body loads and traction loads

The next two terms represent external loads. The **body load vector** component q corresponding to free component $r(q)$ at node $j(q)$ is

$$F_{b,q} = \int_V N_{j(q)} \bar{b}_{r(q)} \, \mathrm{d}V \ . \tag{7.53}$$

The **surface traction load vector** component q corresponding to free component $r(q)$ at node $j(q)$ is

$$F_{t,q} = \int_{S_{t,r(q)}} N_{j(q)} \bar{t}_{r(q)} \, \mathrm{d}S \ . \tag{7.54}$$

Note well that $F_{b,q}$ and $F_{t,q}$ are components of a *one-dimensional array* (column vector), addressed by the row index q.

Damping forces

Terms 2 and 5 in equation (7.44) corresponds to damping forces generated by the damping coupling between individual degrees of freedom. For instance, the **damping force** acting on one free degree of freedom due to motion in another free degree of freedom

$$F_{v,q} = \sum_{p=1}^{N_f} C_{qp} \dot{U}_p(t) \ . \tag{7.55}$$

The first term corresponds to the damping force on the degree of freedom q by the velocity of another free dof p coupled through the **damping matrix**. The damping matrix may be totally arbitrary, or it may be taken in the form of the proportional (Rayleigh) damping. Then we take

$$C_{qp} = \alpha_R M_{qp} + \beta_R K_{qp} \ , \tag{7.56}$$

i.e. the damping matrix is taken as a linear combination of the mass matrix and the stiffness matrix. The coefficients of the linear combination can be determined by matching the damping ratio ζ at one or two selected frequencies. For instance, if we wish to apply a given damping ratio ζ at a single frequency ω_0, we may assume that the two mechanisms contribute equally and solve for the coefficients from

$$\frac{1}{2}\zeta = \alpha_R/\omega_0 \quad \text{and} \quad \frac{1}{2}\zeta = \beta_R\omega_0 \quad . \tag{7.57}$$

On the other hand, if we wish to apply a given amount of damping, with damping ratios ζ_1 and ζ_2 at two separate frequencies ω_1 and ω_2, we need to solve the system of coupled equations

$$\zeta_1 = \alpha_R/\omega_1 + \beta_R\omega_1 \ , \quad \zeta_2 = \alpha_R/\omega_2 + \beta_R\omega_2 \ , \tag{7.58}$$

for the damping combination coefficients α_R, β_R.

The equations (7.44) could be directly integrated using an arbitrary ODE integrator, or, more suitably, with a specialized mechanical integrator such as the Newmark average-acceleration integrator. Other approaches, such as integration of the harmonic modal equations are often used.

End Box 28

Box 29. Mass matrix of the four-node tetrahedron T4

We compute here the mass matrix of the tetrahedron T4 element. Assume uniform mass density.

Node	x	y	z
1	1.4727	0.9193	-0.7713
2	1.4727	1.2000	0
3	1.4727	0.4543	0.0530
4	3.0000	0.5063	0.0290

Numerical quadrature will be used, both one-point and four-point rules (c.f. Table 7.1). The volume integral in the expression

$$M_{qp} = \int_V N_{j(q)}(\boldsymbol{x})\rho N_{i(p)}(\boldsymbol{x}) \ dV \ \delta_{r(q)s(p)} \ ,$$

will be approximated as

$$\int_V N_{j(q)}(\boldsymbol{x})\rho N_{i(p)}(\boldsymbol{x}) \ dV \approx \sum_k N_{j(q)}(\boldsymbol{\xi}_k)\rho N_{i(p)}(\boldsymbol{\xi}_k)W_k \det\left[J(\boldsymbol{\xi}_k)\right]$$

The basis functions for the tetrahedron are given in (7.12). Therefore, the (constant) gradients of the basis functions with respect to the parametric coordinates are obtained (analogously to the triangle T3; viz (3.58)) as

```
gradNpar = array([[-1, -1, -1],
                  [1, 0, 0],
                  [0, 1, 0],
                  [0, 1, 0]])
```

The array of the vertex locations given as

```
x = array([[1.4727, 0.9193, -0.7713],
           [1.4727, 1.2000, 0],
           [1.4727, 0.4543, 0.0530],
           [3.0000, 0.5063, 0.0290]])
```

the Jacobian matrix follows from (3.60), which is readily transcribed into Python code as `J = dot(x.T, gradNpar)`:

```
array([[ 0.    ,  0.    ,  1.5273],
       [ 0.2807, -0.465 , -0.413 ],
       [ 0.7713,  0.8243,  0.8003]])
```

The Jacobian is a constant, 0.90116, (i.e. independent of the integration point), computed as `detJ = linalg.det(J)`.

The one-point rule evaluates the basis functions of the centroid of the tetrahedron. Therefore all the basis functions assume the value of $1/4$ at the quadrature point. For instance we get

$$\int_V N_1(\boldsymbol{x})\rho N_1(\boldsymbol{x}) \ dV \approx N_1(\boldsymbol{\xi}_1)\rho N_1(\boldsymbol{\xi}_1)W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = (1/4)\rho(1/4)(1/6)0.9012 = 0.0094\rho$$

But a more descriptive expression will be obtained by keeping $W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = \det\left[J(\boldsymbol{\xi}_1)\right]/6 = V_{tet}$ instead of numbers, where V_{tet} is the volume of the tetrahedron element, so that the above may be put as

$$\int_V N_1(\boldsymbol{x})\rho N_1(\boldsymbol{x}) \ dV \approx \frac{1}{16}V_{tet}\rho = \frac{1}{16}m_{tet}$$

where m_{tet} is the mass of the tetrahedron. Since all basis functions have the same value of the quadrature point, the same expression is obtained for the one-point quadrature rule for all basis functions. The δ_{rs} represents a 3×3 identity matrix, and the mass matrix therefore results as

$$M = \frac{1}{16}m_{tet} \begin{bmatrix} \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \end{bmatrix}$$

Note that there are four "1"'s in each row, which means that for the four nodes accelerating in either of the three directions with the same acceleration a (i.e. the element accelerates as a rigid body) the total inertial force is $a\, m_{tet}$, as expected.

Now we will apply the four-point quadrature rule from Table 7.1. For instance we have

$$\int_V N_2(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \sum_k N_2(\boldsymbol{\xi}_k)\rho N_3(\boldsymbol{\xi}_k)W_k \det\left[J(\boldsymbol{\xi}_k)\right] \approx \rho W_k \det\left[J(\boldsymbol{\xi}_k)\right] \sum_k N_2(\boldsymbol{\xi}_k)N_3(\boldsymbol{\xi}_k)$$

because $\rho W_k \det\left[J(\boldsymbol{\xi}_k)\right]$ is constant. Substituting the definitions of the basis functions, $N_2 = \xi$ and $N_3 = \eta$, yields

$$\sum_k N_2(\boldsymbol{\xi}_k)N_3(\boldsymbol{\xi}_k) = aa + aa + ba + ab = 2a(a+b) = 1/5$$

and hence

$$\int_V N_2(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \frac{1}{20}V_{tet}\rho = \frac{1}{20}m_{tet}$$

Therefore, the mass matrix will link the nodes 2 and 3 by the 3×3 matrix

$$\frac{1}{20}m_{tet} \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix}$$

Similarly

$$\int_V N_3(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \sum_k N_3(\boldsymbol{\xi}_k)\rho N_3(\boldsymbol{\xi}_k)W_k \det\left[J(\boldsymbol{\xi}_k)\right] \approx \rho W_k \det\left[J(\boldsymbol{\xi}_k)\right] \sum_k N_3(\boldsymbol{\xi}_k)N_3(\boldsymbol{\xi}_k)$$

and

$$\sum_k N_3(\boldsymbol{\xi}_k)N_3(\boldsymbol{\xi}_k) = aa + aa + aa + bb = 2/5$$

and hence

$$\int_V N_3(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \frac{1}{10}V_{tet}\rho = \frac{1}{10}m_{tet}$$

$$M = \frac{1}{20}m_{tet} \begin{bmatrix} \begin{bmatrix} 2\ 0\ 0 \\ 0\ 2\ 0 \\ 0\ 0\ 2 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 2\ 0\ 0 \\ 0\ 2\ 0 \\ 0\ 0\ 2 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 2\ 0\ 0 \\ 0\ 2\ 0 \\ 0\ 0\ 2 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} \\ \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 1\ 0\ 0 \\ 0\ 1\ 0 \\ 0\ 0\ 1 \end{bmatrix} & \begin{bmatrix} 2\ 0\ 0 \\ 0\ 2\ 0 \\ 0\ 0\ 2 \end{bmatrix} \end{bmatrix}$$

Again, to check the mass matrix we can consider the element accelerating as a rigid body in any one of the three directions in the Cartesian coordinates. Since for any selected direction, for each node the sum of the row of the mass matrix is $m_{tet}/4$, the total mass obtained by adding the contributions from all four nodes is m_{tet}, as expected.

Both of the above mass matrices are the so-called consistent mass matrices, but only the second is integrated exactly. The first matrix is integrated inaccurately. As we have seen for the prestressed wire model, various tricks can be undertaken with integration rules for mass matrices to improve the accuracy of the computed frequencies, so we can't necessarily conclude that integrating the mass matrix inaccurately will produce a less accurate estimate of the natural frequencies (and vice versa).

End Box 29

Box 30. Body load on a T4 tetrahedron

Here we compute the nodal loads of the tetrahedron T4 element due to a uniform body load $\overline{b}$. Numerical quadrature with a one-point rule will be adequate for this task since it can exactly integrate linear functions over the domain of the tetrahedron. The volume integral

$$F_{b,q} = \int_V N_{j(q)}(\boldsymbol{x})\overline{b}_{r(q)}\ \mathrm{d}V$$

may be simplified for the uniform body load to read

$$F_{b,q} = \overline{b}_{r(q)} \int_V N_{j(q)}(\boldsymbol{x})\ \mathrm{d}V\ .$$

The integral over the volume of the element may be evaluated in the coordinates of the parametric domain of the standard shape

$$\int_V N_{j(q)}(\boldsymbol{x})\ \mathrm{d}V\ = \int_{V_{[\xi,\eta,\zeta]}} N_{j(q)}(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right]\ \mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta$$

The Jacobian of the T4 element is related to its volume V_{tet} as $\det\left[J(\boldsymbol{\xi})\right] = 6V_{tet}$. The one-point numerical quadrature therefore results in

$$\int_{V_{[\xi,\eta,\zeta]}} N_{j(q)}(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right]\ \mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta = N_{j(q)}(\boldsymbol{\xi}_1)W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = (1/4)(1/6)6V_{tet} = \frac{V_{tet}}{4}$$

and therefore the uniform body load contributes the force

$$F_{b,q} = \frac{\bar{b}_{r(q)} V_{tet}}{4}$$

to each of the four nodes of the element. In other words, the total force $\bar{b}_r V_{tet}$ acting on the entire element in the direction i is split equally among its four nodes.

End Box 30

Box 31. Nodal loads for traction applied to a T3 triangle

We compute here the nodal loads for the triangle T3 surface element generated by uniform traction load $\bar{t}$. The surface integral
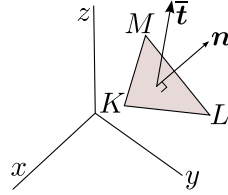
Fig. 7.37. Triangular element with uniform traction load

$$F_{t,q} = \int_{S_{t,r(q)}} N_{j(q)}(\boldsymbol{x})[\bar{t}]_{r(q)} \, dS = \int_{S_{tri}} N_{j(q)}(\boldsymbol{x})\bar{t}_{r(q)} \, dS \;,$$

where S_{tri} is the triangle KLM, may be simplified for uniform traction load to read

$$F_{t,q} = \bar{t}_{r(q)} \int_{S_{tri}} N_{j(q)}(\boldsymbol{x}) \, dS$$

The task therefore reduces to the integration of basis functions over the surface of the triangle, which may be evaluated in the coordinates of the parametric domain of the standard triangle

$$\int_{S_{tri}} N_{j(q)}(\boldsymbol{x}) \, dS = \int_{S_{[\xi,\eta]}} N_{j(q)}(\boldsymbol{\xi}) \det [J(\boldsymbol{\xi})] \, d\xi d\eta$$

The Jacobian for the triangle is obtained as the cross product of the two tangent vectors

$$\frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \xi}, \quad \text{and} \quad \frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \eta} \;,$$

and because the basis functions are linear in ξ, η the tangent vectors are constant (the same at all points of the triangle). It is also shown above that the Jacobian of the surface is equal to twice the area of the triangle. Hence we get

$$\int_{S_{[\xi,\eta]}} N_{j(q)}(\boldsymbol{\xi}) \det [J(\boldsymbol{\xi})] \, d\xi d\eta = \det [J(\boldsymbol{\xi})] \int_{S_{[\xi,\eta]}} N_{j(q)}(\boldsymbol{\xi}) \, d\xi d\eta = 2 S_{tri} \int_{S_{[\xi,\eta]}} N_{j(q)}(\boldsymbol{\xi}) \, d\xi d\eta$$

Numerical quadrature with a one-point rule will be adequate, as N_j is linear in its arguments

$$\int_{S_{[\xi,\eta]}} N_{j(q)}(\boldsymbol{\xi}) \, d\xi d\eta \approx N_{j(q)}(\boldsymbol{\xi}_1) W_1 = (1/3)(1/2) = 1/6$$

Thus we conclude that the uniform traction load contributes the force component

$$F_{t,q} = \bar{t}_{r(q)} \int_{S_{tri}} N_j(\boldsymbol{x}) \, dS = \bar{t}_{r(q)} \times 2S_{tri} \times (1/6) = \frac{\bar{t}_r S_{tri}}{3}$$

to each of the three nodes j of the element. In words, the total force $\bar{t}_{r(q)} S_{tri}$ acting on the entire element in the direction $r(q)$ is split equally among its three nodes.

End Box 31

Box 32. Imposed (thermal) strains

Often the material from which a structure is built up reacts to the environment by deformation. The material experiences the environment in possibly different ways or different measures in different locations, and stresses are produced.

To get started, think about a very small piece of material that is exposed to the environment so that we can assume that the resultant relative deformation is homogeneous and no stress is produced. For the sake of this argument, let us consider one particular environmental effect: ***thermal expansion***. However, similar effects may be produced by shrinkage, swelling, piezoelectric effects, and so on.

When the small sample of material is at a *reference temperature* it is unstressed, and we define its displacements and the associated strains to be zero in this state: the *reference state*. Then the temperature is increased by ΔT, and the material responds by displacement, and because by assumption the deformation is homogeneous, the entire sample experiences uniform strains. Based on experimental evidence, the following model is adopted for orthotropic materials to describe the strains in coordinates aligned with the material directions

$$[^{\theta}\boldsymbol{\epsilon}] = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \Delta T \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \\ 0 \\ 0 \end{bmatrix} . \tag{7.59}$$

Evidently, the thermal expansion is assumed not to cause any shear strains. The factors $\alpha_x, \alpha_y, \alpha_z$ are the so-called ***coefficients of thermal expansion***; different in different directions, in general.

In addition to the imposed strain $\boldsymbol{\epsilon}_{\Theta}$, the sample is also exposed to stresses on its boundary, again such that they result in uniform strain. The total strains that the sample experiences consist of the imposed strains to which the mechanical strains are added. Since the mechanical strains are available from the stresses through the constitutive equation, we write

$$\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_{\text{total}} = {}^{\theta}\boldsymbol{\epsilon} + \boldsymbol{D}^{-1}\boldsymbol{\sigma} , \tag{7.60}$$

Therefore, we have a modification of the constitutive equation

$$\boldsymbol{\sigma} = \boldsymbol{D} \left(\boldsymbol{\epsilon} - {}^{\theta}\boldsymbol{\epsilon} \right) . \tag{7.61}$$

The total strain is related to the displacement via the strain-displacement relation (5.107), and thus we may write

$$\boldsymbol{\sigma} = \boldsymbol{D} \left(\mathcal{B}\boldsymbol{u} - {}^{\theta}\boldsymbol{\epsilon} \right) .$$

As expected, the residual equation form that enters the discretization process needs to be augmented with respect to (5.130) to become

$$\int_V \boldsymbol{\eta} \cdot \rho \ddot{\boldsymbol{u}} \, \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \bar{\boldsymbol{b}} \, \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \bar{t}_i \, \mathrm{d}S \quad + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D} \left(\mathcal{B}\boldsymbol{u} - {}^\theta\boldsymbol{\epsilon} \right) \, \mathrm{d}V = 0 \quad (7.62)$$

$$u_i = \bar{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

Clearly, there will be one more term to discretize

$$- \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D} \, {}^\theta\boldsymbol{\epsilon} \, \mathrm{d}V \;, \tag{7.63}$$

yielding the **_thermal strain load_**

$$F_{\Theta,q} = \left[\int_V \mathcal{B}^T \left(N_{j(q)}(\boldsymbol{x}) \right) \boldsymbol{D} \, {}^\theta\boldsymbol{\epsilon} \, \mathrm{d}V \right]_{r(q)} \;. \tag{7.64}$$

End Box 32

8

Interpretation of FEA Results

In this chapter we will look at the issue of interpretation of the results. Not the actual calculation of the results, we assume the results have been computed using the methods described previously. The matter at hand is rather what to do with the results, how to understand what they're telling us. By now we all understand that the results are always "wrong", meaning they are always in error. The critical questions are: knowing they are in error, how do we process these results, and how do we use them?

8.1 Singularities

Stresses are often of primary interest when designing structures. However, there are some inherent limitations to what is computable and how.

For example, some features in structures when analyzed with the elasticity model result in stresses that are infinite at some points. Obviously, there is little merit in attempts aimed at computing the stress at such points. Even away from these points, the stress is often very inaccurate unless special models are employed. Therefore, it behooves us to understand which features lead to infinite stresses so that we can formulate solution strategies appropriately.
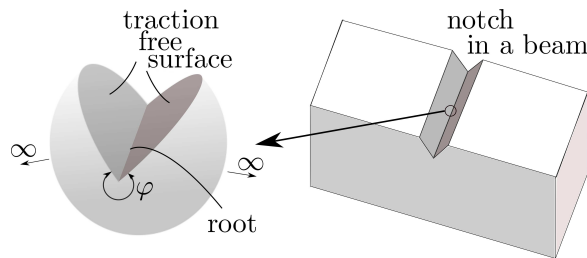
Fig. 8.1. Notch and the idealized geometry of an infinite wedge

Some geometric features resemble notches, scratches, cracks, or put more generally, concave corners with a sharp root. Such geometries lead to states of stress which can be analyzed with two dimensional models (the displacements are functions of two variables). The idealization of such geometries is the ***wedge*** as a locus of points where two bounding surfaces meet at an angle. Figure 8.1 shows a geometry with a notch. Away from the locations where the notch root runs out into the side surfaces, the state of stress may be analyzed using the idealization shown on the left, the *wedge* [1]. The material surrounding the edge of the root extends to infinity in all directions, but the solution is of interest only in the immediate vicinity of the root edge. The analytical solution possesses a singularity in the stress components for angles $\varphi > 180°$ for symmetric and for $\varphi > 260°$ for anti-symmetric loadings of the form

$$\sigma \sim r^{\alpha} \ , \tag{8.1}$$

where r is the radial distance from the edge, and $\alpha \leq 0$ measures the strength of the singularity. Figure 8.2 provides a sketch of the dependence of the exponent α on the angle φ. For symmetric loads, even a very slight notch will lead to singular stresses, and the strongest singularity occurs for $\varphi = 360°$ (infinitely sharp crack). Similarly, anti-symmetric loading will produce the same strength of singularity for a crack configuration ($\varphi = 360°$, $\alpha = -1/2$). In the same figure on the right, the graph shows that the stronger the singularity, the wider the general area around the wedge root where the stresses are high (they are infinite at the edge $r = 0$ for all strengths).
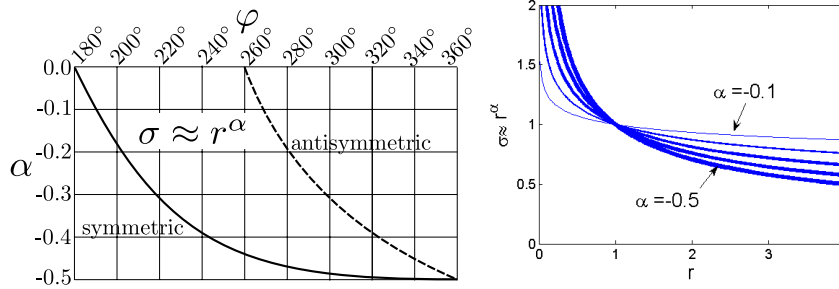
Fig. 8.2. On the left: Strength of the singularity in the stress. Symmetric (lower, solid, curve) and anti-symmetric (upper, dashed, curve) loading. On the right: Variation of the stress in the radial direction as it depends on the strength of the singularity. The stronger the singularity (i.e. the larger $|\alpha|$), the larger the volume of material that experiences high stress values

There are also other sources of singularities in the elasticity model. Singular stresses accompany edges along ***multi-material interfaces***, as shown in Figure 8.3, or in Figure 8.5. Singular stresses are also generated at sudden transitions from fixed displacement to prescribed traction (***discontinuity in boundary conditions***), see Figure 8.4 .
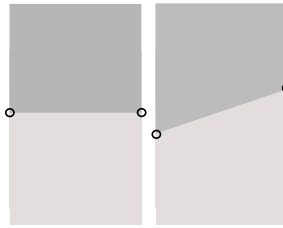
Fig. 8.3. Singularity at a multi-material interface: butt joint and scarf joint
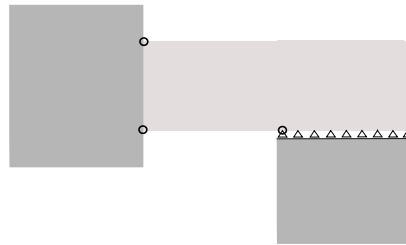
Fig. 8.4. Singularities due to sudden change in displacement boundary conditions. Corner between a clamped edge and the free surface. Location where the roller support ceases to apply.

We also pointed out in Section 6.8 (and in detail: See Box 24) that ***concentrated loads*** at points or along curves generate infinite stresses. As discussed in detail there, the energy in the model is infinite in the limit. Therefore, this kind of singularity is "even worse" than the singular stresses near the tip of a wedge. Also, point supports and supports along curves leads to infinite stresses with similar characteristics as concentrated loads (unless the associated reactions are identically zero).

The most complex singularities to analyze are the true ***3-D singularities at corners***. Examples include (call for Figure 8.5) the intersections of wedge fronts (crack fronts) with free surfaces (marked 1), corners of multi-material wedges (2), corners on multi-material interfaces (3), or corners in homogeneous geometries (4,6), conical points (marked 5), or corners in wedges (on crack fronts) (marked 7).
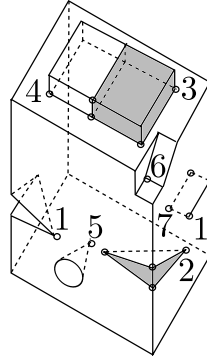
Fig. 8.5. 3-D singular points

8.2 Interpretation of stresses

In stress analysis the stress distribution is linked to the strains. The strains are defined using the spatial derivatives of the displacements. The displacements are continuous within each element (because the basis functions are continuous), and their derivatives are also continuous within each element (because the derivatives of the basis functions are continuous). For instance, across an element the strains may be constant (T4 tetrahedra), or vary linearly (T10 tetrahedra). At the interfaces between the elements the displacements are continuous (because along the finite element interfaces the basis functions guarantee continuity), but the strains are in general discontinuous.

It is important to realize that, as a consequence of the strains being discontinuous across inter-element boundaries, the finite element solution for the stresses will be discontinuous across element interfaces. That is unphysical since we know that the traction vector (5.89) changes continuously in dependence on the normal to the surface on which it acts and the stress. Observe Figure 8.6: We have separated a body consisting of homogeneous material with an arbitrary cut. Here, one half of the material is on the bottom, the other have is on the top. The colors distinguish between the parts separated by the cut, but the material constitution is the same. The point A is on the cut surface between the two materials. We can separate the two halfs along this interface, and write the traction vector on the side of the bottom material as

$$\boldsymbol{t}(A_-) = \mathcal{P}_{\boldsymbol{n}(A_-)}\boldsymbol{\sigma}(A_-)$$

and the traction vector on the side of the top material

$$\boldsymbol{t}(A_+) = \mathcal{P}_{\boldsymbol{n}(A_+)}\boldsymbol{\sigma}(A_+) \ .$$

Since the two normals are related as $\boldsymbol{n}(A_-) = -\boldsymbol{n}(A_+)$, and since by principle of action and reaction we have $\boldsymbol{t}(A_-) = -\boldsymbol{t}(A_+)$, we must have

$$t(A_-) = \mathcal{P}_{\boldsymbol{n}(A_-)}\boldsymbol{\sigma}(A_-) = -t(A_+) = -\mathcal{P}_{\boldsymbol{n}(A_+)}\boldsymbol{\sigma}(A_+)$$

and therefore because

$$-\mathcal{P}_{\boldsymbol{n}(A_+)} = \mathcal{P}_{-\boldsymbol{n}(A_+)} = \mathcal{P}_{\boldsymbol{n}(A_-)}$$

and, importantly, because the normal $\boldsymbol{n}$ is really arbitrary in the sense that in doesn't have to be vertical, it can point in any direction, we can conclude

$$\boldsymbol{\sigma}(A_-) = \boldsymbol{\sigma}(A_+) \,.$$

In words, the stress vector is continuous across the (imaginary) interface between the two halves separated by the cut. Unfortunately this holds for the mathematical model of elasticity, the finite element quantities will behave differently! In particular, even though the inter-element interfaces are in a way similar to this imaginary cut, the stress components are not at all guaranteed to be continuous across the interface: the element to one side of the interface will have a different value of stress than the element on the other side.
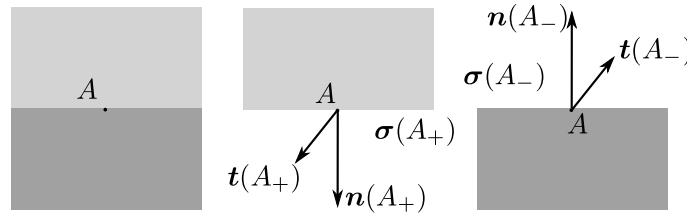
Fig. 8.6. Continuity of stress across an imaginary interface produced by cutting.

In what follows, unless we say otherwise, when we talk about displacements, strains, and stresses we mean the quantities computed by the finite element model. Here is what we know about the finite element stress: The stress is related to the strain through the constitutive equation. The constitutive equation is invoked only at the quadrature points. Therefore, the stress is consistent with the strains (and with the displacements from which the strains are derived) only at the quadrature points. In the present model of elasticity, it may seem attractive to use the constitutive equation at *any* point within the element to compute the stresses. However, it needs to be realized that such stresses are an *interpretation* of the solution, not the solution itself. In other commonly used models in stress analysis, for instance in plasticity, computing the stresses from the displacements at other points than the quadrature points would be ill defined (and ill advised).

> In the finite element method, because of everything being set up by evaluating integrals, displacements, strains, and stresses are all consistent at the integration (quadrature) points. Everything else is an interpretation of the computed results.

As an example, consider the calculation of the stresses in the cantilevered bracket in Figures 8.7–8.8. The finite element model is based on the incompatible hexahedron C3D8I with $2 \times 2 \times 2$ Gaussian quadrature.

Principal stress vectors

As the base case consider Figure 8.7: the stresses consistent with the finite elements solution are displayed at the $2 \times 2 \times 2$ quadrature points as the principal stress vectors that are visual representations of the principal stresses and directions of the principal stresses. The directions of the vectors are the directions of the principal stresses, the lengths of the vectors are the magnitudes of the principal stresses. The arrows also include the information on whether the principal stress is compressive or

tensile: when the arrows emanate from the integration point, the stress is tensile; otherwise, when the arrows converge onto the integration point, the principal stresses compressive. Each vector is also colored to represent the magnitude of the principal stress. This is the honest representation of our knowledge of stress in the finite element model.

(a)   (b)   (c)
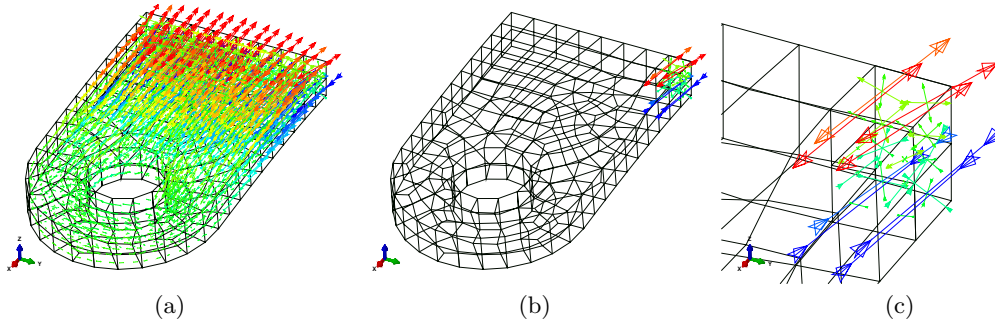
Fig. 8.7. Interpretations of the calculated stresses. Principal stress vectors at integration points. (a) All integration points in all elements. (b) All integration points in two elements. (c) Close-up of (b).

(a)   (b)
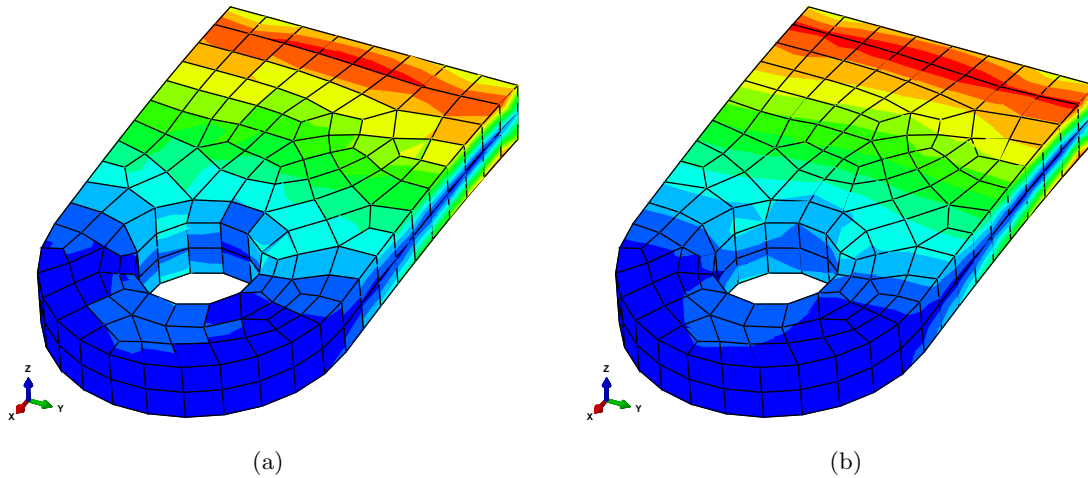
Fig. 8.8. Interpretations of the calculated stresses. Von Mises stress color-coded with a contour plot. (a) Element stresses without averaging. (b) Element stresses averaged at the nodes.

Element stress without averaging

Figure 8.8(a) presents the stresses element-by-element. In some elements the stresses are shown with some variation within the element, in others the reported stresses are uniform within the element. In the present case, for the incompatible hexahedra C3D8I, the stresses have something resembling a linear variation across the element.

Averaged (nodal) stress

Stress recovery (or "averaging") is a procedure in which values of the stress at the nodes are computed from the values of the stress in the elements around the node. Various principles are used to arrive at the nodal stress, for instance simple average may be used, or a more sophisticated inverse-distance-weighted average, or the highly successful Superconvergent Patch Recovery (SPR) [12]. In

Figure 8.8(b) we show the interpretation of the stress which is the result of the Abaqus average recovery: The magnitude of the stress at a node is computed by extrapolating from the magnitude of the stress at the quadrature points nearby. The nodal stress magnitudes are then interpolated using the element basis functions and smooth coloring is applied. The usefulness of such a smooth visual representation of the stress distribution for the evaluation of the results is evident. Nevertheless such a continuous stress field is an interpretation of the finite element results (often performed in a so-called *postprocessing* step). It should be realized that *extrapolating* the stress may be fraught with significant errors: the stresses are typically *smoothed* in these postprocessing procedures and spikes in stress may get wiped out. On the other hand, the recovered stress may be an improvement of the stress representation, if for no other reason than because it is continuous across element interfaces. It is not usually clear which of these two has occurred in the problem at hand, and therefore extra caution is always in order.

Convergence in stress

Often one can use the plots of element stress without the nodal averaging for quick visual assessment of the convergence of the stress. When it is no longer possible to detect discontinuous change in color coding between two adjacent finite elements, we can interpret that as evidence of the stress being of quality acceptable to make judgments about accuracy. For instance, consider Figure 8.9 where we compare the von Mises stress plots for six progressively finer meshes. The blocky nature of the element stress plot recedes as we apply more elements. It is interesting that even though the deflection is clearly very accurate even for the coarsest mesh ad (a) (refer to Figure 7.29), it takes a lot more refinement to render the stresses in acceptable quality.

8.3 Stress concentrations

The finite element formulations in this book may be applied to the solution of the so-called stress concentration problems. These are produced by stress raisers, which are features that cause local or global increases in the stress, but which do not lead to infinite stresses. The book [6] is an extremely useful resource; a mode of operation where the solution of a finite element model is cross-checked with this book, or vice versa, is to be generally recommended.

The presence of stress raisers has a significant effect on the fatigue strength of structures [6]. To be aware of stress raisers is highly advisable for all modelers, since stress concentrations where the stress changes rapidly are generally locations of high error in the finite element solution: compare with Section 6.4.1. Therefore, an appropriately constructed mesh will reflect the presence of stress raisers by suitable refinement (reduction in mesh size) around the stress concentrations. Figure 8.10 presents examples of stress raisers: the small-radius regions near the elliptical hole, the vicinity of the stiffened hole, through-hole in a shaft, and the mass-reduction holes in a flywheel, the groove underneath the snap-on ring, the shoulder fillet, and the fillets on the T-head suspension, the rounded portion of the stress relief groove, the cylindrical surfaces of the holes in the clevis and lug joint, the large-curvature portion of the curved hook, the surface of the depression in a plate, and the narrow cross-section reduction in a shaft. These are just a few selected examples. The main lesson is to be on the lookout for similar features in the geometries of analyzed structures.

8.4 Errors, validation, and verification

Modeling physical events (for instance, the deflection of an airplane wing when the aircraft is turning is a physical event) on the computer may be described as shown in Figure 8.11. In the first step, a **physical event** is idealized into a **mathematical model**. The mathematical model has rarely exact (analytical) solutions, which gives rise to the need for a **discrete model** (the Galerkin finite element method in this book, but there are many others: finite difference and finite volume methods, boundary element methods, and so on). The unknowns in the discrete model are solved
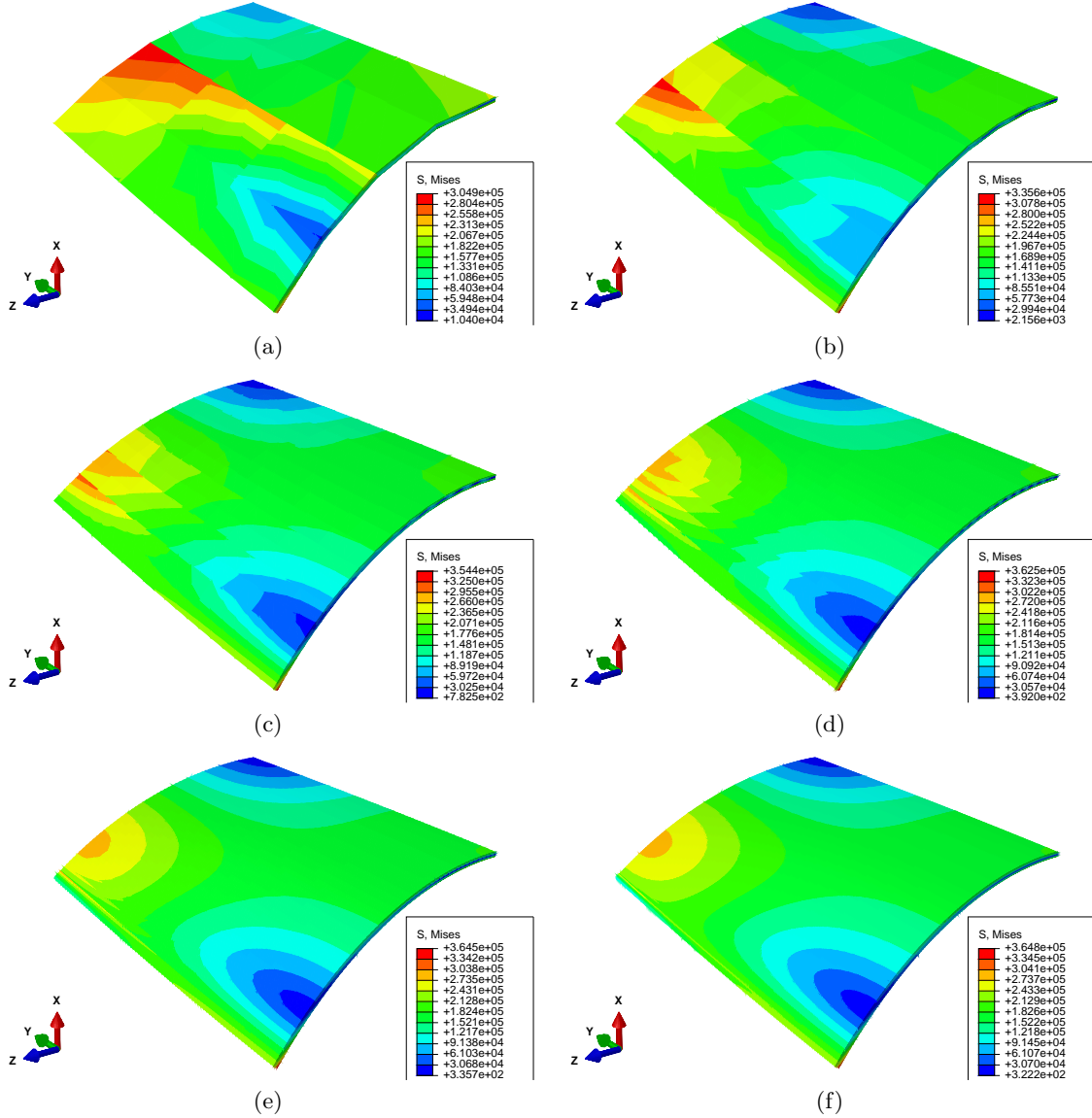
Fig. 8.9. Scordelis-Lo cylindrical shell. Magnified deflected shape shown, with color-coded von Mises stress. Stress displayed without nodal averaging. H20 hexahedron: (a) 3 elements/side, maximum 3.049e5, (b) 5 elements/side, maximum 3.356e5, (c) 10 elements/side, maximum 3.544e5, (d) 20 elements/side, maximum 3.625e5, (e) 40 elements/side, maximum 3.645e5, (f) 80 elements/side, maximum 3.648e5.

for using various numerical methods – solvers for systems of linear and nonlinear algebraic equations, eigenvalue solvers, integrators for systems of ordinary differential equations, ... – resulting in the ***solution*** of the discrete model. The accuracy of the solution to the discrete model may be then assessed with respect to the mathematical model with various methods of ***error estimation*** (Richardson extrapolation, for instance). If the accuracy is not sufficient, a better discrete model is produced by ***adaptation***, else the solution is deemed acceptable for either one of two actions: prediction or verification.
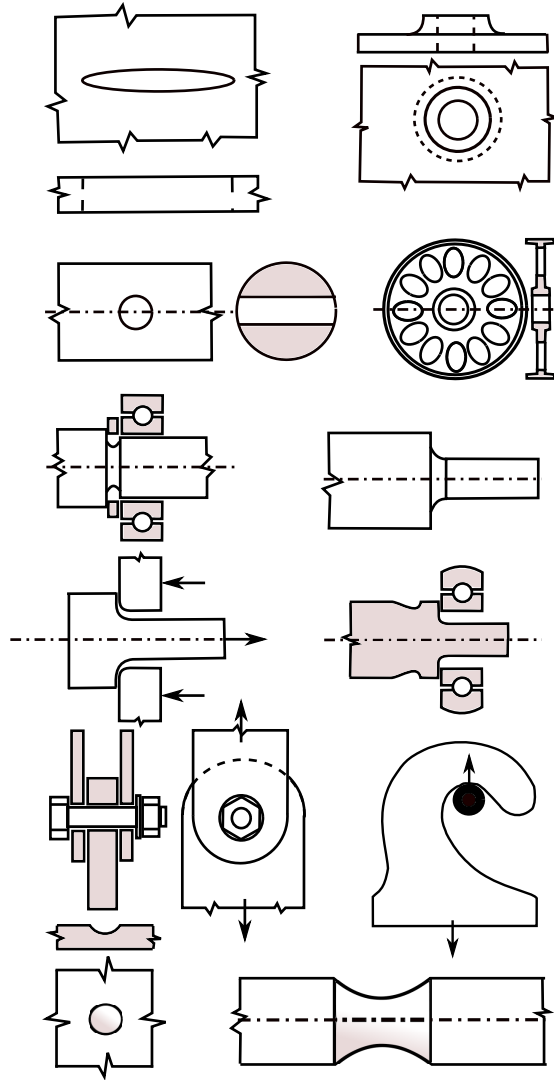
Fig. 8.10. Stress raisers. Left to right, top to bottom: Elliptical hole, stiffened hole, through-hole in a shaft, mass-reduction holes in a flywheel, snap-on ring groove, shoulder fillet, T-head suspension, stress relief groove, clevis and lug joint, curved hook, depression in a plate, cross-section reduction in a shaft

8.4.1 Verification and Prediction

There are two uses for the solution of the discrete model, depending on whether a reference (exact, or very accurate) solution of the mathematical model is available or not. If it is available, we obtain a ***verification*** of the way the discrete model is implemented in the computer: the ingredients that go into the discrete model are correct, and the equations are solved right, therefore we observe convergence towards (closeness to) the reference solution of the mathematical model. The verification should be addressed thoroughly by the designers of the computational software, but users also tend to perform verifications to gain confidence in the software. Physical events are typically very idealized to produce mathematical models that can be solved very accurately or analytically. Such mathematical models are called ***benchmarks***.

On the contrary, if the reference solution of the mathematical model is not known, the solution to the discrete model will make ***predictions*** of various quantities possible (deflections, natural frequencies, strains or stresses, energy, and so on).
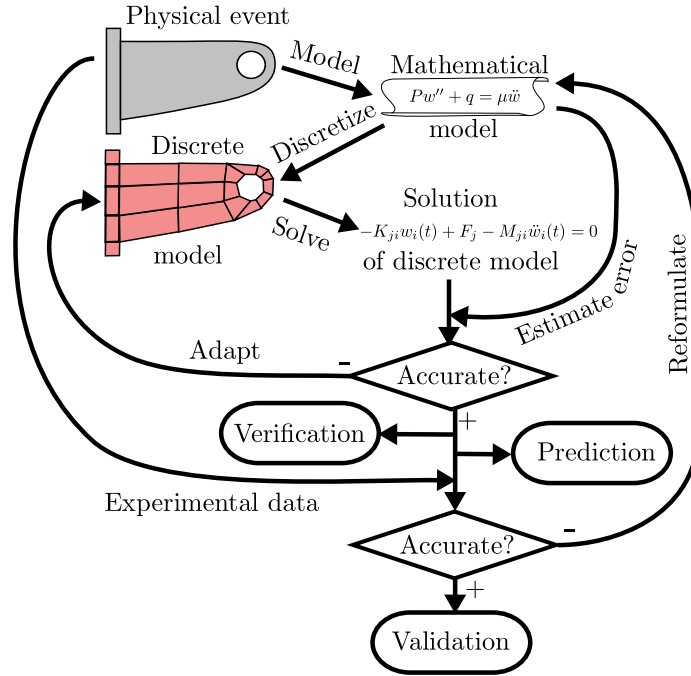
Fig. 8.11. Diagram of the modeling pipeline.

8.4.2 Validation

Finally, if comparison of some quantities from the solution with **observations** of the corresponding quantities in the physical event is possible, and provided the agreement is good, we call the modeling pipeline **validated** for this particular physical event; otherwise, if an improvement to the mathematical model may be made, for instance by including aspects of the physical event that have been deliberately neglected before, we perform yet another adaptation and another pass through the modeling pipeline. If there is a significant mismatch, we may decide that a completely different mathematical model needs to be formulated, perhaps even requiring a new theory. In this way, we get the chance to **falsify** a theory.

8.4.3 Errors

The contributions to the detected mismatch are associated with the arrows in the graph of Figure 8.11: there's the **modeling error** that accompanies the idealization of the physical event into a mathematical model, the **discretization error** when converting the mathematical model into a discrete model, the **solution error** produced by the numerical algorithms, and there may also be an **observation error** due to inaccurate or erroneous measurements. Finally, an important source of mismatch may also be **data uncertainty**, as all the input quantities will only be known with a certain margin of error (material parameters, geometric dimensions, and so on).

8.4.4 Using modeling to make predictions

Comparing the solution of the discrete model with experimental observations allows us to call the solution validated with respect to the specific physical event. If we needed to observe the physical event each time we ran a simulation so that it could be validated, there would not be many incentives for using the simulation in the first place. However, in practice we use validations for a series of specific events to sample the "event space" to establish a **range of validity**. If a particular physical event $\mathcal{E}$ resembles other events for which the validity of the model has been established, we assume that the
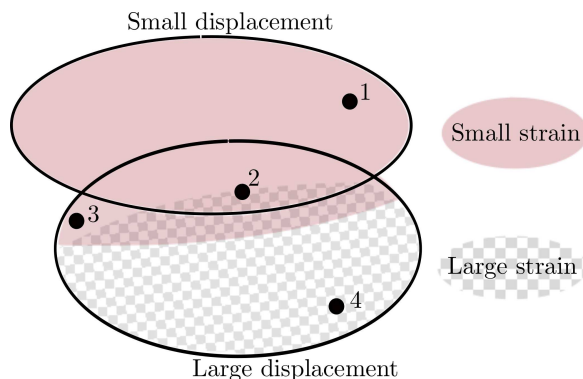
Fig. 8.12. Classes of stress analysis problems based on the magnitude of the displacement, and the magnitude of the strain. Examples: 1: concrete dam; 2: thin clamped plate under a transverse load; 3: steel measuring tape; 4: penetration of a steel slab by a high-velocity tungsten projectile.

model will be likely validated also for $\mathcal{E}$. As an example, consider the classification of the physical events with respect to the magnitude of displacements and strains (Figure 8.12). If the physical event is the deformation of a structure that works roughly as a steel measuring tape, we assume that it is in the class of events for which models based on large displacements but small strains are validated. Note that the two model classes, small and large displacement, overlap. That is because there's no sharp division of the physical events that can be modeled with either depending on the required accuracy and the importance of capturing the effect of large displacements. For instance, deflections of clamped plates under transverse loads depend to a certain degree on the tension that develops in the structure when it deforms; if the contribution is negligible, small-displacement model could be adequate, otherwise a large-deflection model may be required.

8.4.5 Using benchmarks

The value of benchmarks for the verification of numerical models is clear, but there is another reason analysts should take advantage of benchmark problems: they need to build up a feel for the relative accuracy and robustness of finite elements, and such data is readily accessible in convergence studies performed on benchmark problems because of the availability of the reference solution. The convergence graphs displayed in the previous sections of this chapter are readily available sources of valuable insights.

Eventually, analysts develop intuition to help them assess the suitability of a particular discretization for a particular physical event. For example, performing a series of convergence analyses for different values of the Poisson ratio will help us create a map of the performance of a particular element as it depends on this parameter. Figure 8.13 shows such a map for the fully-integrated brick element H8.
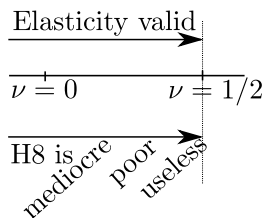
Fig. 8.13. The effect of the Poisson ratio on the accuracy of the fully-integrated brick element H8.

Figure 8.14 illustrates on the example of plate-like structures how their thickness would affect the choice of the mathematical model as well as the choice of the discretization. The 3-D elasticity

theory on which the solid elements H20 and T10 are based is valid for all thickness to span ratios. We see that for relatively thicker structures the solid elements might be an effective discretization, while for really thin plates a specialized plate element (based either on the shear-agnostic Kirchhoff theory for very thin plates, or based on the Mindlin theory that incorporates transverse shear for moderately thick plates) would be more a better choice. Finite element analysts construct maps to effective modeling strategies and techniques, such as the one given in Figure 8.14, by experience: running benchmarks, reading manuals and technical papers, and so on.
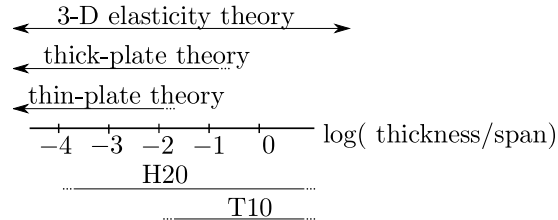
Fig. 8.14. Schematic classification of stress and deflection analysis finite elements for plate-like structures. Ranges for which the finite elements H20 and T10 could be effective.

8.5 Writing FEA reports

When writing FEA reports we can organize our thinking using the following questions:

Why did we do it? Describe the objectives, the scope, and the setting.
What did we study? Describe the circumstances, the studied object(s), the environmental and loading conditions.
What did we assume? Describe the simplifications and assumptions about the structure and its loading conditions.
How did we do it? Describe the tools used, the models, the settings of the tools, and the limitations of the models.
What did we find out? List and summarize the results.
What did we conclude? Formulate conclusions by interpreting the results.

Reports typically consist of the following constituent parts

1. Title page + Table of contents
2. Executive summary
3. Introduction
4. Material properties and allowables summary
5. Geometry, simplifications, coordinate systems
6. Boundary condition summary (supports, loads, environmental conditions)
7. Details (calculations, explanations, ...)
8. Conclusions (interpretation, prediction, ...)
9. References

The executive summary typically summarizes the major results and conclusions in the report so that the reader may determine the substance of the material and decide whether to read the report in detail. This section should clearly state whether or not the structure meets the design requirements, and if it fails it should explain in non-technical language how. The executive summary must also include any recommendations based on the analysis. This section usually includes visual overview of the system being analyzed, along with figures or diagrams showing a summary of loading and results. This is often the most read section of the report.

> The FE analysis report is often *the* document that makes the study ***reproducible***. In other words, the report should allow someone else to perform the same study to verify our own results. This aspect is very important indeed: if for no other reason, then to meet our professional responsibilities.

An example of a report from a finite element analysis is provided in the next section. See Box 33 The section also includes a list of reporting dos and don'ts, an incomplete list, but useful nonetheless. See Box 34

8.6 Background, explanations, details

Box 33. Strap analysis: Sample FEA report

The following pages show a sample report from an FE analysis. (The report pages are framed for visual clarity.)

Investigation of the Tresca Stress in a Steel Strap

Petr Krysl

February 18, 2017

Contents

1 Executive Summary

The objective was to calculate accurately the Tresca stress, as a measure of stress intensity, in a steel strap under tensile loads. The part was modeled using the plane-stress idealization. Figure 1 shows graphically the boundary conditions (clamped edge at the left-hand side, edge with uniformly distributed pressure on the right-hand side), and the six points of interest where the Tresca stress was to be computed.
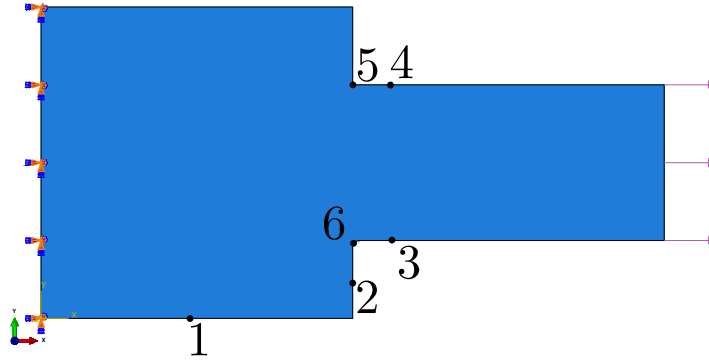
Figure 1: The strap is clamped at the left and loaded with uniform tensile traction on the right. The output locations for the Tresca stress are indicated by dots and numerical labels.

The geometry was partitioned to introduce the points of interest explicitly into the definition of the geometry. Five finite element meshes of quadrilateral elements of uniform element size, taken as $h = 1, 1/2, 1/4, 1/8, 1/16$ [in], were generated. The problem was solved using the incompatible-mode quadrilateral element type. The Tresca stress was obtained from the stress displays with nodal averaging, and Richardson extrapolation was attempted using the solution on the three finest meshes for each point of interest. Except for point of interest 5, which is associated with a concave corner and therefore the stress there is theoretically infinite, the extrapolation could be performed and the stress converged with rates between 1.12 and 2.8. The estimated true Tresca stress was presented in a table.

The results of the study allow us to conclude that

- The incompatible-mode quadrilateral from Abaqus performs well under the studied conditions; and

- the Richardson extrapolation is a valuable technique to estimate the true answers from the finite element computations.

2

2 Introduction

The goal of the present study is to illustrate calculation of accurate stress results under nontrivial conditions. In the present case the geometry incorporates several stress raisers: the concave corners where the narrower part of the strap transitions into the wide part, one with a fillet, and one with sharp interior angle; further, where the wide part is clamped along the left-hand side edge, the corners of that side are also locations of stress concentrations.

The procedure is to compute several solutions with uniform meshes of different resolutions (i.e. built up of elements of different sizes), and then to attempt extrapolation to the limit of the element size vanishing ($h \to 0$).

The Abaqus/Standard, version 6.14.1, finite element program was adopted in this study.

3 Material properties

The material was assumed to be steel, with Young's modulus of 30×10^6 psi and Poisson ratio of 0.3.

4 Geometry

The geometry of the strap is shown in Figure 2. Note that one corner is left sharp with a 90° interior angle, and one corner is rounded with a fillet feature.

Figure 3 illustrates the partitioning of the part into eight regions in order to incorporate the points of interest into the geometry, and to make it possible for the mesher to succeed in building quadrilateral meshes. The six points of interest are indicated in the figure. Note that point 5 is located at the root of the concave corner, and point 6 is located at the midpoint of the edge that represents the fillet.

The strap is of uniform thickness of 0.25 in, which is a sufficiently small thickness to allow for the plane-stress modeling to be used.
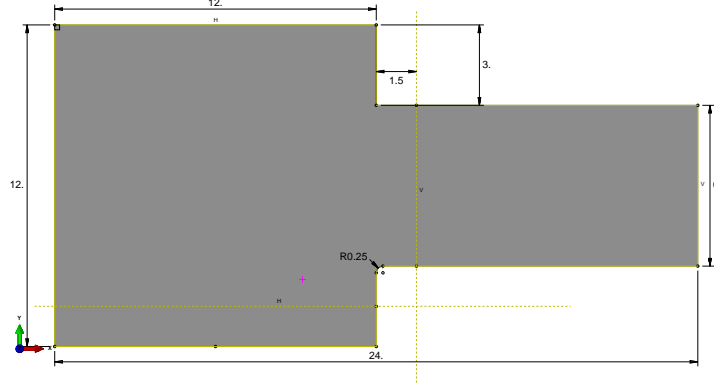
Figure 2: Sketch of the strap. The dimensions are in inches. The thickness is uniform, equal to 0.25".

5 Boundary conditions

The plane-stress model is adopted. As shown in Figure 1, the left-hand-side edge of the wide portion of the strap is clamped and the narrow edge on the right-hand side is loaded by uniform pressure 333.3333 [psi] to generate total force of 500 [lbf] as applied to the strap. The applied supports are sufficient to remove all rigid body modes.

6 Detailed calculations

The meshes were generated of uniform element size, with the element size taken as $h = 1, 1/2, 1/4, 1/8, 1/16$ [in]. The advancing-front mesh generator produced meshes of reasonable quality, meaning that no error messages were generated. Table 1 shows the numbers of nodes and elements for the five meshes used. Figure 1 demonstrates mesh 1 for element size $h = 1$in.

The element type adopted for all models was the incompatible-mode quadrilateral, CPS4I.

The results are illustrated in Figure 4 for the coarsest mesh employed. Note well that the coarsest mesh does not even register the singularity of the sharp corner (point of interest 5).

Figure 5 shows the distribution of the Tresca stress obtained with the

4

Figure 3: The partitioning of the strap domain. The eight regions shown above. The dimensions of the partitioning shown below. The six points of interest are also shown relative to the partitioning of the domain.

NUMBER OF	$h = 1$	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$
ELEMENTS	267	996	3911	15607	63169
NODES	841	3062	11879	47110	190083
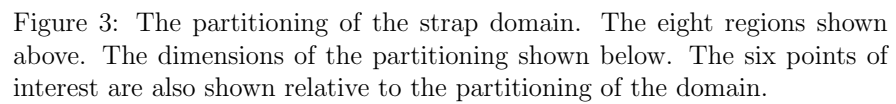
Table 1: Uniform meshes for the five element sizes, $h = 1, 1/2, 1/4, 1/8, 1/16$ [in].

5
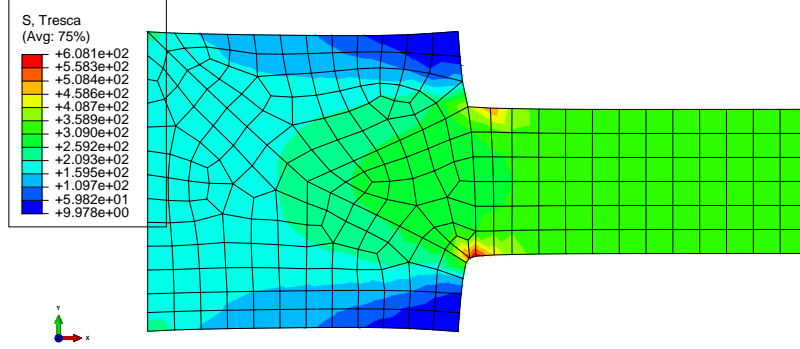
Figure 4: Tresca stress shown using the contour plot for mesh 1 (element size $h = 1[\text{in}]$), on the deformed geometry (deformations magnified with a factor of ≈ 11000).

finest mesh employed (for element size $h = 1/16$ [in]). The stress at the sharp corner (point of interest 5) has at this point exceeded the stress at the fillet (point of interest 6).

The computed values of the Tresca stress are summarized in Table 2. The computed data is also summarized in Figure 6. For each point the computed Tresca stress values are plotted on a linear-linear scale as a function of the element size. Except for points of interest 2 and 6 the data is well behaved for all five meshes; for the two exceptions the first two simulations yield a different trend then the last three.

The true value of the Tresca stress was estimated with Richardson extrapolation [1]. The results for the three finest meshes (mesh 3, 4, and 5) were used to estimate the true solution, and also as a byproduct the convergence rate. Table 3 presents the estimates of the true Tresca stress and the convergence rate at the six points of interest. The convergence rate is between 1.0 and 2.0 (with 1.0 the expected value; the value converges faster at point 2, which may be related to the low value of stress at that location). At point 5 the presence of a singularity makes extrapolation meaningless as indicated by the *negative* convergence rate.

The slope of the approximate error (difference of successive solutions) needs to be matched to the slope of the estimated true error on a log-log

6
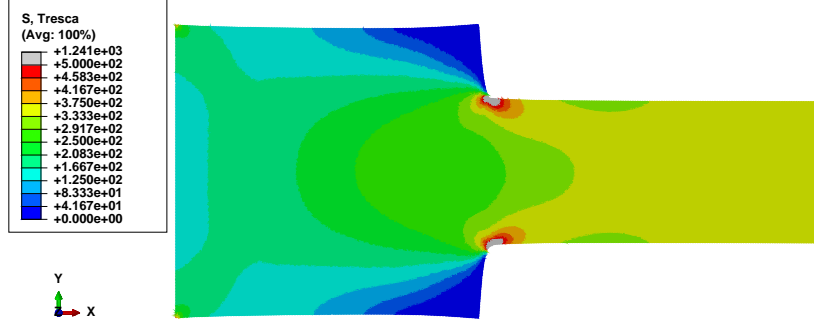
Figure 5: Tresca elementwise (not averaged) stress shown using the contour plot for mesh 5 (element size $h = 1/16$ [in]), on the deformed geometry (deformations magnified with a factor of ≈ 11000). The element edges are not displayed for clarity.

Point	$h = 1$	$h = 1/2$	$h = 1/4$	$h = 1/8$	$h = 1/16$
1	117.64	116.02	115.12	114.48	114.24
2	50.96	38.192	32.017	32.341	32.387
3	376.76	364.59	359.63	357.18	356.38
4	382.56	362.35	356.55	354.61	353.72
5	408.45	528.82	699.41	924.15	1240.5
6	539.4	685.79	672.19	920.24	999.8

Table 2: Computed values of the Tresca stress in psi, for the five uniform meshes of element sizes, $h = 1, 1/2, 1/4, 1/8, 1/16$ [in], for which the solution was obtained.

Point of interest	Estimated Tresca stress [psi]	Convergence rate β
1	114.09	1.43
2	32.40	2.80
3	356.00	1.62
4	352.97	1.12
5	148.09	-0.49
6	1037.40	1.64

Table 3: True values of the Tresca stress estimated with extrapolation, with the convergence rate indicated

7
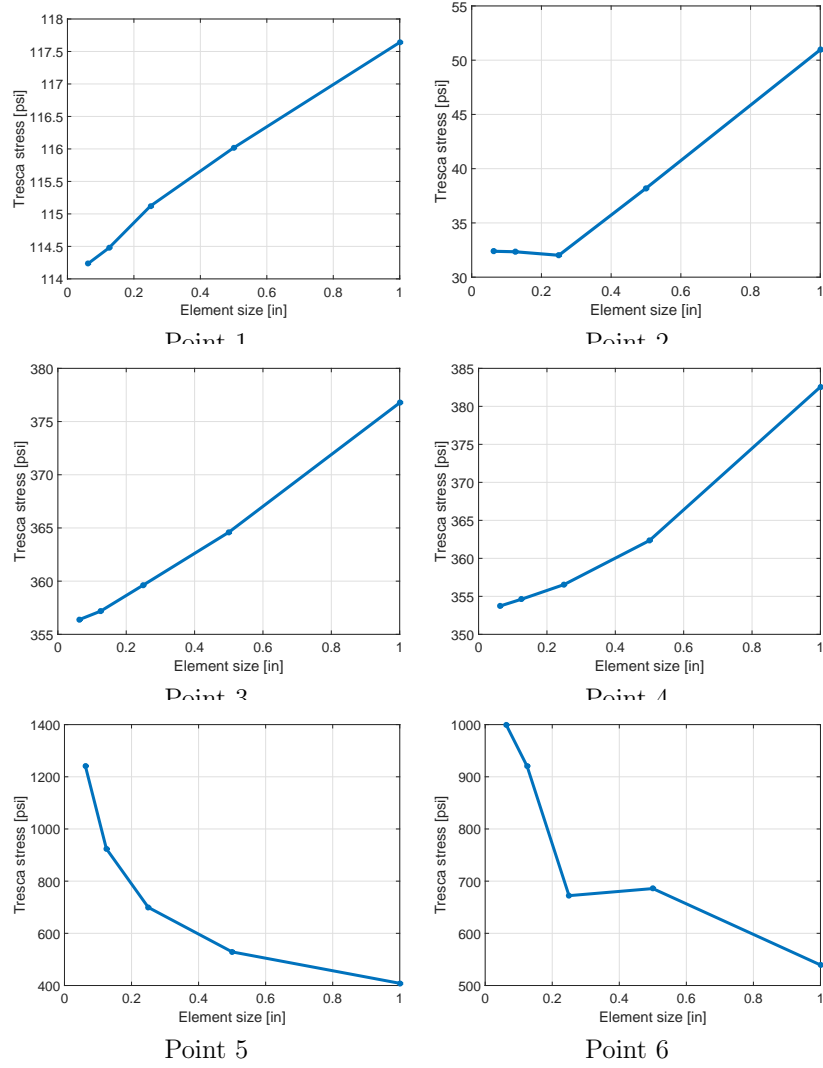
Figure 6: Computed values of the Tresca stress [psi] for the five element sizes, $h = 1, 1/2, 1/4, 1/8, 1/16$ [in], for which the solution was obtained.

8

Point of interest	Tresca stress [psi]
1	114.09
2	32.40
3	356.00
4	352.97
5	(*)
6	1037.40

Table 4: True values of the Tresca stress. (*): extrapolation at point of interest 5 is not applicable, the true value is theoretically infinite.

plot. Figure 7 presents the approximate error (dashed line) computed from the Tresca stresses for meshes 3,4,5, and the estimated true error (solid line) of these three solutions. The convergence rates of these two kinds of error curve are in visual agreement for all points of interest. This is a confirmation of the suitability of the computed data for extrapolation.

7   Conclusions

The sequence of computed solutions could be used to extrapolate to obtain estimates of the true values at all points of interest shown in Figure 1, with the exception of point of interest 5 (the root of a concave corner), where extrapolation is not applicable. The estimated true values of the Tresca stress are listed in Table 4.

References

[1] P. Krysl, *Finite Element Modeling with Abaqus and Python for Thermal and Stress Analysis*, San Diego, Pressure Cooker Press, 2017.
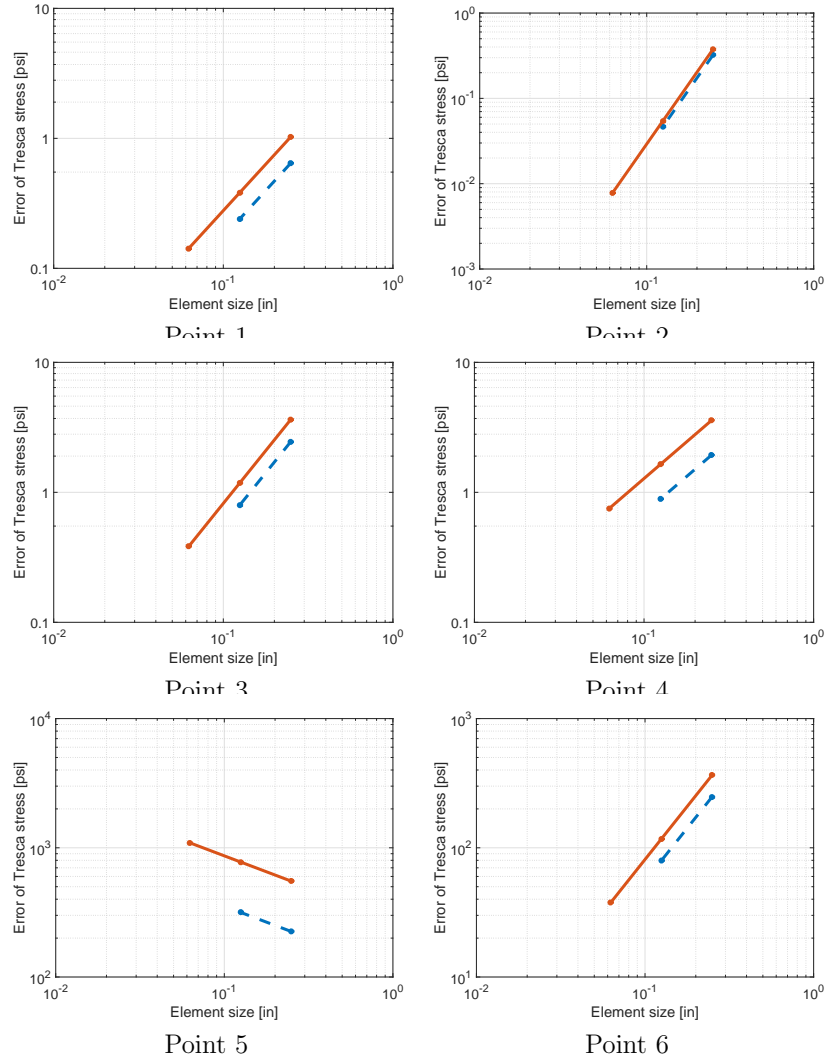
Figure 7: Approximate errors (dashed line) and estimated true errors (solid line) of the Tresca stress [psi]. The negative convergence rate for point 5 corresponds to the blowup of the stress value due to the presence of the singularity.

10

End Box 33

Box 34. Dos and Don'ts of FEA reporting

Dos

Here is an incomplete list of things to remember concerning the content of the report:

1. Do write the report with the reproducibility of your work in mind: someone else should be able to come along later and verify or reproduce your analysis from the information provided in the report (report writers are not necessarily on hand later when they are needed: that's why the information should be in the report).
2. Do build the analysis report as you go, that way no important detail will be forgotten.
3. Do supply clear and convincing evidence for the correctness of the analysis.
4. Do describe main assumptions unambiguously, and justify all major idealizations.
5. Do provide units for input parameters and results.
6. Do describe the meshes used, any mesh controls and element size information, the element types (with all relevant settings: for instance for the reduced-integration hexahedra we must state the type of hourglassing stabilization, as that is not available from the element type, such as C3D8R).
7. Do state the type of the solver used: for instance steady-state dynamics can be direct or modal. This could make a difference.

The following is a list of reminders concerning the delivery of information: text, figures, tables, and so on.

1. Do provide a caption to every figure and table.
2. Do use lettering of figures of sufficient size: a good guide is to make sure the lettering of the figure elements (tick marks, labels, legends) when placed in the text is of roughly the same size as the text itself.
3. Do label figure axes.
4. For multiple curves in graphs do include legends or tag curves in figures with arrows so that the reader can distinguish between individual curves. Alternatively, use markers (or line styles) and describe the curves using the markers (or line styles) in the caption. Refer to Figure 7.20 for an example.

Don'ts

1. When only a single simulation was performed, don't represent the results as having quantitative meaning. Don't imply that such results can be used for design checks and similar purposes. The results from a single computation do not come with any estimate of the error. Without a guess of the error, such results may be interpreted qualitatively at best. This is not to say that a single simulation cannot produce an accurate result, it may or it may not. The problem is we don't know which of the two we have before us. A possible exception is a single simulation that was performed for a set of circumstances that have been studied thoroughly before. Then we may know something about the likelihood of the FE results being within a certain error of the true solution. As an example, take an analysis of a strap with one bolt hole as configuration A and two bolt holes as configuration B. If configuration A was studied and the error of the FE solution was estimated, let us say with the most accurate FE solution being within 5% of the true answer, then if the same type of FE model is used with a similar mesh we can guess with some confidence that the FE answer will be produced with similar error for configuration B.

2. When an adaptive simulation is performed, the results are obtained on a sequence of meshes where no simple relationship exists between the element sizes employed at a particular location from simulation to simulation. Therefore this data is not amenable to extrapolation. Do not attempt to extrapolate from adaptive-simulation data.

3. When an adaptive simulation is performed, the errors are characterized with the so-called error indicators. Error *indicators* are not error *estimators*: the error is not quantified, only relative contributions to the error from the mesh are calculated to guide the construction of the next mesh. Therefore, the final result of adaptive simulation does not come with an estimate of error, and really must be treated as if only a single simulation was performed. Do not consider such results to be firm quantitative answers. See Box 35

4. Don't provide only nodal-average stress results. The elementwise (unaveraged) stresses should be documented because they are an honest and deeply meaningful representation of the accuracy of the model.

5. Do not limit the stress output illustrated in the report to only stress invariants (von Mises or Tresca stress). In addition to these we often need to show which parts of the structure are in compression or tension, which are loaded by bending or torsion. For this purpose we need to provide plots of the stress components, in suitable coordinate systems. Other useful plots include principal-stress plots which show not only the magnitude but also the direction of the principal axes.

The following is a list of no-nos concerning the delivery of information: text, figures, tables, and so on.

1. Don't include in the report figures or tables that are never referenced in the text.

2. Don't discuss in the report how to operate the software, which buttons to press, which dialogues to fill in, and so on. Summarize the settings, perhaps with tables, but the document is not a software tutorial or a software manual.

End Box 34

Box 35. Strap Adaptive FEA

The problem of finding the Tresca stress in a metal strap was studied with an h-adaptive mesh refinement in Abaqus. (For the report describing the extrapolation to the limit: See Box 33 .) The solution documented in Table 8.1 are the results of the final mesh of a three-step adaptive process. Compared to the estimated true values the adaptive procedure yielded numbers which vary between very accurate to the barely okay (refer to the column that shows the relative error of the adaptive solution). The problem is we do not know which numbers are good and which numbers are bad without some effort to go to the limit of an infinitely refined mesh. Extrapolation allows us to do that, but for extrapolation we need at least two, better three, successive solutions with progressively refined meshes. The adaptive analysis produces a sequence of solution steps, but the meshes are not related in a way that allows for extrapolation.

End Box 35

Point of interest	Estimated true stress	Adaptive simulation stress	Relative error of the adaptive solution
1	114.09	113.70	0.30%
2	32.40	40.84	21.66%
3	356.00	368.02	3.27%
4	352.97	367.47	3.95%
5	(*)	2284.33	NA
6	1037.40	1082.39	4.20%

Table 8.1. The Tresca stress in psi. Limit value prediction: (*) means extrapolation at point of interest 5 is not applicable, the true value is theoretically infinite. Comparison with the final result from an adaptive simulation.

References

1. J.R. Barber. *Elasticity.* Solid Mechanics and Its Applications. Springer, 2009.
2. R. D. Blevins. *Formulas for Natural Frequency and Mode Shape.* Krieger publishing company, reprint edition, 2001.
3. T. J. R. Hughes. *The Finite Element Method - Linear Static and Dynamic Finite Element Analysis.* Dover Publications, Inc., 2000.
4. M. W. Hyer. *Stress Analysis of Fiber-reinforced Composite Materials.* WCB/McGraw-Hill, 1998.
5. John H. Lienhard IV and John H. Lienhard V. *A Heat Transfer Textbook.* http://web.mit.edu/lienhard/www/ahtt.html, 2005.
6. W. D. Pilkey. *Peterson's Stress Concentration Factor.* John Wiley & Sons, 2nd edition, 1997.
7. I.S. Sokolnikoff. *Mathematical Theory of Elasticity.* Robert E. Krieger publishing Company, 1983.
8. B. Szabó and I. Babuška. *Introduction to Finite Element Analysis: Formulation, Verification and Validation.* Wiley Series in Computational Mechanics. Wiley, 2011.
9. S. P. Timoshenko. *History of Strength of Materials.* Dover, 1983.
10. W. Young and R. Budynas. *Roark's Formulas for Stress and Strain.* McGraw Hill professional, 2001.
11. Ashraf M. Zenkour. Three-dimensional elasticity solution for uniformly loaded cross-ply laminates and sandwich plates. *Journal of Sandwich Structures and Materials*, 9(3):213–238, 2007.
12. O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method: The Basis.* Oxford [etc.] : Butterworth Heinemann, 2000.

Index