Petr Krysl
pkrysl@ucsd.edu

A Pragmatic Introduction to the Finite Element Method for Thermal and Stress Analysis

With the Matlab toolbox SOFEA

December 13, 2005

Pressure Cooker Press
San Diego

Contents

Part I Introducing the Galerkin method

1 Model of a Taut Wire . . . 3
- 1.1 Deriving the PDE model . . . 3
- 1.2 Balance equation . . . 3
- 1.3 Boundary conditions . . . 4
- 1.4 Boundary conditions (in space) . . . 5
- 1.5 Initial conditions . . . 5
- 1.6 Anything else? . . . 6

2 The method of Mr. Galerkin . . . 7
- 2.1 Residual of the balance equation . . . 7
- 2.2 Integral test of the residual . . . 8
- 2.3 Test function . . . 8
- 2.4 Trial function . . . 9
- 2.5 Manipulation of the residuals . . . 10
- 2.6 Stiffness and mass matrix . . . 12
- 2.7 Piecewise linear basis functions . . . 13
- 2.8 Numerical quadrature . . . 15
- 2.9 Putting it together: system of ODE's . . . 18

3 Introducing the Matlab code . . . 19
- 3.1 Statics . . . 19
- 3.2 Statics: uniform load . . . 19
- 3.3 Free vibration . . . 22
- 3.4 Integration of transient motion . . . 24
  - 3.4.1 Using built-in Matlab solver . . . 25
  - 3.4.2 Using the Trapezoidal integrator . . . 25

4 The boundary conditions for the model of a taut wire . . . 29
- 4.1 Essential and natural boundary conditions at separate end-points . . . 30
- 4.2 Essential boundary conditions only . . . 30

4.3 Natural boundary conditions only . . . 31
4.4 Overspecified boundary conditions . . . 32

Part II Thermal analysis

5 Model of Heat Diffusion . . . 37
5.1 Balance equation . . . 37
5.2 Constitutive equation . . . 39
5.3 Boundary conditions . . . 40
5.4 Initial condition . . . 43
5.5 Summary of the PDE model of heat conduction . . . 43

6 Galerkin method for the model of heat conduction . . . 45
6.1 Weighted residual formulation . . . 45
6.2 Reducing the model dimension . . . 46
6.3 Test and trial functions: basis functions on triangulations . . . 48
6.4 Basis functions on the standard triangle . . . 50
6.5 Discretizing the weighted residual equation . . . 52
6.6 Derivatives of the basis functions; Jacobian . . . 55
6.7 Numerical integration . . . 58
6.8 Conductivity matrix . . . 60
6.9 Surface heat transfer matrix and load . . . 64

7 Steady-state heat diffusion solutions . . . 69
7.1 Steady-state diffusion equation . . . 69
7.2 Thick-walled tube . . . 69
7.3 Orthotropic insert . . . 72
7.4 The T4 NAFEMS Benchmark . . . 75

8 Transient heat diffusion solutions . . . 79
8.1 Discretization in time for transient heat diffusion . . . 79
8.2 Transient diffusion: The T3 NAFEMS Benchmark . . . 81
8.3 Transient cooling in a shrink-fitting application . . . 84

9 Expanding the library of element types . . . 87
9.1 Quadratic triangle T6 . . . 87
9.2 Quadratic 1-D element L3 . . . 89
9.3 Point element P1 . . . 89
9.4 Measuring (integrating) over domains . . . 90
9.5 Tetrahedron T4 . . . 94
9.5.1 Example: helical geometry . . . 95
9.6 On the simplex elements . . . 96
9.7 Quadrilateral Q4 . . . 96
9.8 Hexahedron H8 . . . 96

10 Convergence and error control . . . 97
10.1 First look at errors . . . 97
10.2 Richardson extrapolation . . . 98
10.3 The T4 NAFEMS Benchmark revisited . . . 98
10.4 Shrink fitting revisited . . . 98

Part III Stress analysis

11 Model of elastodynamics . . . 103
11.1 Balance of linear momentum . . . 103
11.2 Stress . . . 105
11.3 Local equilibrium . . . 109
11.3.1 Change of linear momentum . . . 109
11.3.2 Stress divergence . . . 109
11.3.3 All together now . . . 112
11.4 Strains and displacements . . . 113
11.5 Constitutive equation . . . 115
11.6 Initial conditions . . . 117
11.7 Boundary conditions . . . 117
11.7.1 Example: concrete dam . . . 117
11.7.2 Example: rigid punch . . . 118
11.7.3 Formal definition of boundary conditions . . . 118
11.7.4 Inadmissible "concentrated" boundary conditions . . . 119
11.7.5 Symmetry and anti-symmetry . . . 121
11.7.6 Example: pure-traction problem . . . 123
11.7.7 Example: shaft under torsion . . . 124
11.7.8 Example: overspecified boundary conditions . . . 125
11.8 Comparing the Thermal and Deformation models . . . 125

12 Galerkin formulation for elastodynamics . . . 127
12.1 Manipulation of the residuals . . . 127
12.1.1 The first two steps . . . 127
12.1.2 Step 3: Preliminaries . . . 128
12.1.3 Step 3: The glorious conclusion . . . 129
12.2 Method of weighted residuals as the Principle of Virtual Work . . . 130
12.3 Discretizing . . . 130
12.3.1 The trial function . . . 130
12.3.2 The test function . . . 132
12.3.3 Producing the requisite equations . . . 133
12.4 The discrete equations: system of ODE's . . . 134
12.4.1 Inertial term: Mass matrix . . . 134
12.4.2 Body loads and traction loads . . . 135
12.4.3 Resisting forces: Stiffness matrix . . . 136

12.4.4 Summary of the elastodynamics ODE's . . . 136
12.5 Constitutive equations of linearly elastic materials . . . 137
12.5.1 Orthotropic material. . . . 137
12.5.2 Transversely isotropic material. . . . 138
12.5.3 Isotropic material. . . . 139
12.6 Imposed (thermal) strains . . . 139
12.7 Strain-displacement matrix . . . 141
12.7.1 Transformation of basis . . . 142
12.8 Stiffness matrix . . . 144

13 Examples and further developments . . . 147
13.1 Modal analysis with the tetrahedron T4: the drum . . . 147
13.2 Modal analysis with the tetrahedron T4: the composite rod . . . 149
13.3 Tetrahedron T10 . . . 152
13.3.1 Example: the drum revisited . . . 153
13.4 The composite rod with the tetrahedron T10 . . . 154
13.5 Static analysis with hexahedron H8 . . . 154
13.6 Analyzing the effects of thermal strains . . . 154

14 Reduced-dimension models: plane strain, plane stress, axisymmetric 157
14.1 Plane strain model reduction . . . 157
14.1.1 Discretization . . . 159
14.2 Plane stress model reduction . . . 160
14.3 Model reduction for axial symmetry . . . 161
14.4 Material stiffness for two-dimensional models . . . 163
14.5 Strain-displacement matrices for two-dimensional models . . . 164
14.6 Integration for two-dimensional models . . . 165
14.7 Thermal strains in two-dimensional models . . . 167
14.8 Examples . . . 168
14.8.1 Thermal strains in a bimetallic assembly . . . 168
14.8.2 Orthotropic balloon . . . 172

A Appendix: GNU Free Documentation License . . . 175

References . . . 181

Index . . . 183

Part I

Introducing the Galerkin method

1

Model of a Taut Wire

This chapter will formulate a relatively simple model of a taut string. In the next chapter, we will seek approximate solutions to this model that are obtained with the Galerkin method.

1.1 Deriving the PDE model

Figure 1.1 illustrates an idealization of a taught wire. The wire is under prestressing force, P, assumed to be uniform along the length of the wire. The left hand end is immovably fixed, while the right hand side end is held in a fixture which can slide perpendicularly to the axis of the wire. A transverse force F_L is applied at the movable end. In addition, there may be some distributed force q acting along the length (but we shall ignore gravity). The transverse displacement is a function of both the axial coordinate x and the time t, $w = w(x, t)$. The transverse displacement is assumed to be very small compared to the length of the wire.

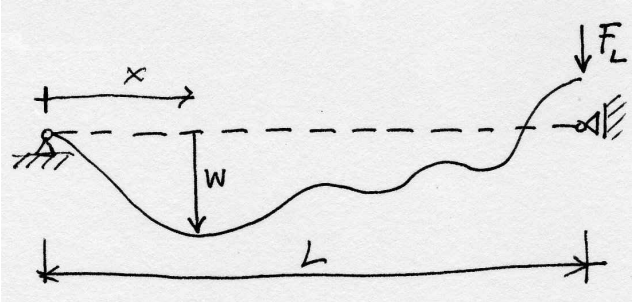

Fig. 1.1. Schematic of taut wire

1.2 Balance equation

Taking a section of length Δx of the wire (see Figure 1.2, collecting all the forces, and equating them to the inertial force (Newton's law), leads to a ***balance equation*** for

the taut wire

$$P\frac{\partial^2 w}{\partial x^2} + q = \mu\ddot{w} \ , \tag{1.1}$$

where $\ddot{w} = \frac{\partial^2 w}{\partial t^2}$ is the acceleration.

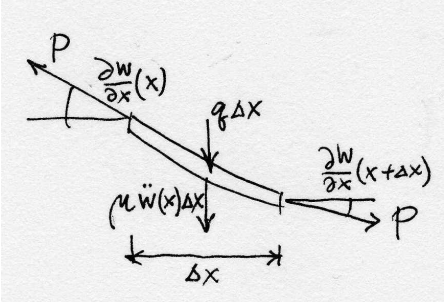

Fig. 1.2. The forces acting on a segment of the taut wire

1.3 Boundary conditions

The function w that describes the transverse deflection takes two arguments, x, and t. It is defined on a rectangle shown in Figure 1.3: $0 \le x \le L$, and $0 \le t \le \bar{t}$. It needs to be determined to satisfy the balance equation (1.1), but that would not completely nail the answer down. We also know something about the solution, namely at the ***boundaries*** of the domain rectangle.

How many pieces of information do we need to know? A reasonable answer is, 'Enough to make the solution unique.' To find the deflection w is going to involve integration, because the balance equation refers to space and time derivatives of w. Using the definitions

$$v = \frac{\partial w}{\partial t}$$
$$\theta = \frac{\partial w}{\partial x}$$

we may rewrite all the balance equation that involves the second derivatives of the function w as a system of first order differential equations

$$\begin{aligned} \frac{\partial \theta}{\partial t} &= \frac{\partial v}{\partial x} \\ P\frac{\partial \theta}{\partial x} + q - \mu\frac{\partial v}{\partial t} &= 0 \end{aligned}$$

For each derivative $\frac{\partial v}{\partial x}$, $\frac{\partial \theta}{\partial x}$, one boundary condition (integration constant) will be needed. Similarly, for each of the time derivatives $\frac{\partial v}{\partial t}$, and $\frac{\partial \theta}{\partial t}$ one boundary condition along the time axis will be required.

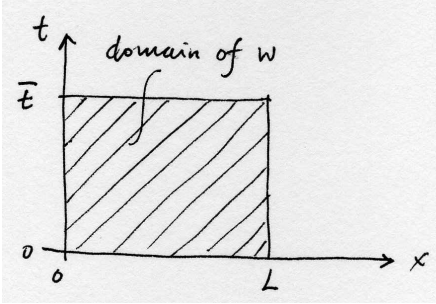

Fig. 1.3. The domain of the deflection function w

1.4 Boundary conditions (in space)

The conditions on w along the edges of the domain rectangle parallel to the time axis are known (for historical reasons) as *the* ***boundary conditions***. (Perhaps also because they are applied along the *physical boundaries* of the structure.)

It needs to be realized that the domain of the wire, that is the interval $0 \leq x \leq L$, has only one boundary, namely the two endpoints. Since these two points are disjoint, the boundary of the interval consists of two disjoint sets. As discussed in more detail in Chapter 4, we are really prescribing a single boundary condition. Since it happens to be applied at two disjoint points, we talk loosely of boundary conditions, even the though we should be saying boundary condition specifications.

For the sake of definiteness and this example, at the left-hand side end of the wire we are prescribing in general nonzero displacement,

$$w(0,t) = \bar{w}_0.and \tag{1.2}$$

As we shall find out, there is a good reason why this kind of condition is commonly called the ***essential boundary condition***.

At the other end the boundary condition is of a different nature. It is also a bit more interesting, as we have to derive it. Again, we take a short section of the wire of length Δx (see Figure 1.4). This time there are terms that are multiplied by Δx, but there are also others which are not. Only the latter survive when we make Δx go to zero.

$$-P\frac{\partial w}{\partial x}(L,t) + F_L(t) = 0\ . \tag{1.3}$$

This boundary condition is simply the balance of forces at the end of the wire. Boundary conditions of this kind are called ***natural boundary conditions***, and we'll find out presently why.

1.5 Initial conditions

Along the edges of the domain rectangle that are parallel to the space axis we also apply two conditions. However, as we all are aware, the time direction is special.

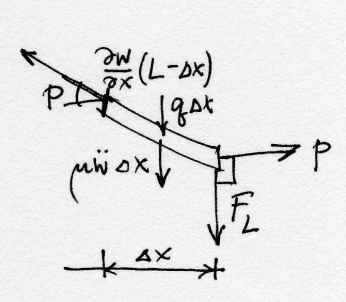

Fig. 1.4. The forces acting on a right hand side segment of the taut wire

Therefore, it will probably come to us naturally to expect to know something about the deflection at one point in time, typically at $t = 0$. Because this is the initial point along the time axis, these conditions are known as the ***initial conditions*** (and we need two of them):

$$w(x,0) = \bar{W}(x), \quad \frac{\partial w}{\partial t}(x,0) = \bar{V}(x) \ , \tag{1.4}$$

where $\bar{W}(x)$ (the initial deflection) and $\bar{V}(x)$ (the initial velocity) are known functions.

1.6 Anything else?

The balance equation (1.1), the boundary conditions (1.2) and (1.3), and the initial conditions (1.4) are all we need to fully define what model it is we are trying to find solutions to. It is an ***initial boundary value problem***, and as such it is quite typical of the models with which structural engineers have to deal. In what follows, we shall find out how to formulate an algorithm, the so-called Galerkin finite element method, which will supply an approximate solution to this problem.

2

The method of Mr. Galerkin

We will have to come to grips with the impossibility of satisfying the governing equation exactly with an approximate method. There's going to be an error in the balance equation (which we shall call a residual; another appropriate label might be imbalance). Similarly, the natural (force) boundary condition may not be satisfied exactly, and there is going to be a residual there too.[1]

2.1 Residual of the balance equation

The balance equation (1.1) may be written in the residual form as

$$P\frac{\partial^2 w}{\partial x^2} + q - \mu\ddot{w} = r_B(x,t) \ , \tag{2.1}$$

by simply moving the inertial force on the other side of the equals sign. The residual r_B is identically zero if w is the exact solution. For an approximate solution, the residual r_B varies from point to point, and from time to time, and is in general nonzero.

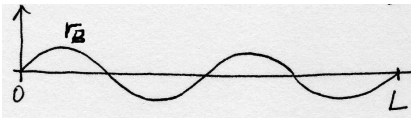

Fig. 2.1. Residual that integrates to zero, but is not identically zero

Checking that the balance residual is identically zero at each point x and each time t does not provide us with anything we can use to talk about approximate solutions: the residual is either zero or it isn't, but how do we measure whether the approximate solution for which the residual is not zero is good?

[1] Boris Grigoryevich Galerkin became a teacher of structural mechanics in St. Petersburg Polytechnical Institute in in 1908. Among his contemporaries, also active in St. Petersburg, were I. G. Bubnov, A. N. Krylov, and S. P. Timoshenko, well-known names in the profession. In 1915 Galerkin published an article, in which he put forward an idea of an approximate method to solve differential boundary value problems (he was working on plate and shell models at that time). Around that time Bubnov developed similar variational approach, hence this method is also known as the Bubnov-Galerkin method.

2.2 Integral test of the residual

One possible choice of a quality measure is to integrate the residual over the domain (length of the wire). We could think of the integral

$$\int_0^L r_B(x,t)\,\mathrm{d}x\ . \tag{2.2}$$

as a test: if the residual is identically zero, this integral will also come out zero. However, (2.2) may be zero even when the residual is not identically zero. In other words, if we wanted to prove that the residual corresponds to an exact solution, this would be an incomplete and flawed test. Consider Figure 2.1: the integral (2.2) is zero, but the residual itself may be very large (for instance, when $r_B = A\sin(2\pi nx/L)$, with $n = 1, 2, ...$).

2.3 Test function

A remedy that addresses this blindness of (2.2) to the shape of the residual may be to use a "window" (test) function $\eta(x)$

$$\int_0^L \eta(x) r_B(x,t)\,\mathrm{d}x\ . \tag{2.3}$$

Note that $\eta(x)$ is an arbitrary function. In particular, it could be a function of the shape shown in Figure 2.2, which is certainly going to give a nonzero value for (2.3) (the hatched area at the bottom). Therefore, it correctly indicates that the residual does not correspond to the exact solution. Equation (2.3) is known as the ***weighted residual statement***. Approximate approaches that start from the weighted residual statement are known as weighted residual methods.

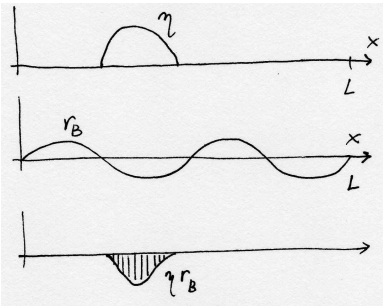

Fig. 2.2. Nonzero a residual which is detected in the integral (2.3)

Equation (2.3) is a reliable way of testing the residual, but computationally it seems hardly less difficult than testing the residual at each point of the domain: equation (2.3) needs to be evaluated for an infinite number of functions η in order to make sure there are no bumps in the residual. The job will still take an infinite time.

Let us contemplate a tangible analogy of what we're trying to do in equation (2.3). Imagine our job is to hold an inflatable balloon in a box, so that it does not jut out anywhere. Use the fingers of one hand to press down on the balloon, so that the balloon is at the top of the box in the spot where it is being held by the finger. If we put down all five fingers, the situation is as shown on the left in Figure 2.3. Each of the fingers may be thought of as a single test function η that pushes down the residual in some spots.

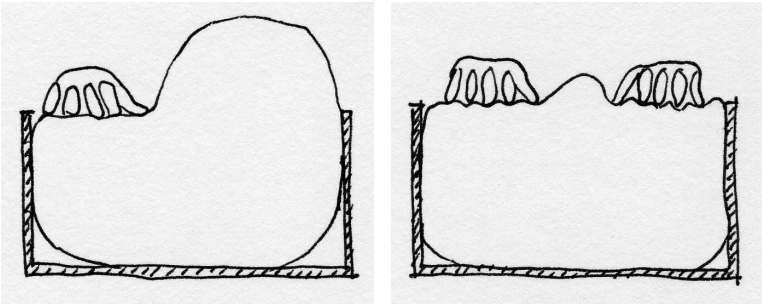

Fig. 2.3. Stuffing a balloon into a box

Evidently, the balloon bulges out a little bit in between the fingers, and a lot everywhere else. However, we have the option of pressing down on the balloon with the fingers of our other hand, and if we enrol our friends and relatives, and the chance passersby, we will manage to do a better and better job of stuffing the balloon into the box and holding it so that it does not protrude very much. Indeed, with an infinite number of fingers, we can hold the balloon so that it does not protrude at all.

In this way, we may begin to see how an trial-and-test approximate method may be formulated. Selecting a finite number of suitable functions η_j (fingers), we may be able to keep the residual small (but in general nonzero). By applying larger numbers of test functions, we may hope to be able to reduce the error in the residual. Also, for each η_j, $j = 1, ..., N$, we will make the integral (2.3) vanish

$$\int_0^L \eta_j(x) r_B(x,t)\, \mathrm{d}x = 0 \ , \tag{2.4}$$

which provides us with the means of calculating N coefficients (numbers) from these N equations.

2.4 Trial function

The task of formulating the approximate solution consists really of describing the shape of the deflection w. This can be done in a variety of ways, but for reasons that we shall give later, a piecewise linear representation is a good choice. Figure 2.4 illustrates this concept by showing how the shape may be defined by the N coefficients w_j. The attentive reader will at this point fidget: the piecewise linear shape of the deflection

curve is not going to allow us to express the second order derivatives $\partial^2 w/\partial x^2$. At the corners, the first derivatives will be discontinuous, and hence the second derivative will be a spike (so-called Dirac delta function). We can choose either to abandon the piecewise linear shape, and pass a smooth curve through the filled-circle points, or, we could change the rules of the game by getting rid of the second-order derivatives. As we shall presently see, the latter choice is commonly preferred.

In any case, the equations (2.4) may be used to calculate the values of w_j, $j = 1, ..., N$. The function that describes the shape of the approximate solution (with the N free parameters) is known as the ***trial function***. It describes a possible (candidate, trial) shape of the approximate solution; which becomes *the* solution once the values of the free parameters are known.

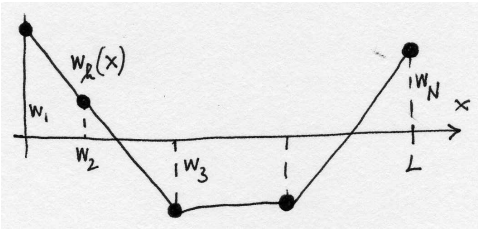

Fig. 2.4. Piecewise linear trial function

2.5 Manipulation of the residuals

We will seek the approximate solution w to satisfy the balance equation in the residual form (2.4), and we'll incorporate the boundary conditions and residual form too. The displacement boundary condition (1.2) will be included in the form of the residual

$$r_w(t) = w(0,t) - \bar{w}_0(t). \tag{2.5}$$

and the natural boundary condition (1.3) will be included as the residual

$$r_F(t) = -P\frac{\partial w}{\partial x}(L,0) + F_L \ . \tag{2.6}$$

Therefore, the approximate solution will be sought from the conditions

$$\begin{array}{ll} \int_0^L \eta_j(x) r_B(x,t)\ \mathrm{d}x = 0 & \\ \xi_w r_w(t) & = 0 \\ \xi_F r_F(t) & = 0 \end{array} \tag{2.7}$$

There are no conditions at this point on the trial function w other than smoothness that will guarantee the existence of the integrals in the first equation (2.7).

To reduce the complexity of (2.7), we may immediately realize that at the left-hand side end of the wire, we can quite simply design the trial function to make the

residual (2.5) identically zero. This will put another condition on w, and (2.7) may be cast as

$$\begin{array}{ll} \int_0^L \eta_j(x) r_B(x,t)\,\mathrm{d}x = 0 \\ \xi_F r_F(t) \qquad\quad = 0 \end{array} \tag{2.8}$$

where $w(0,t) = \bar{w}_0(t)$, and $w(x,t)$ sufficiently smooth in x.

In this way we managed to reduce the number of residuals, but we will do even better now. By applying integration by parts to the first equation in (2.8), we will be able to reduce the number of residuals further, and furthermore, we will be able to make it much easier to design a trial function by allowing for less smooth functions.

Substituting for the balance residual, we get three terms

$$\begin{array}{l} \int_0^L \eta_j(x) r_B(x,t)\,\mathrm{d}x = \\ \int_0^L \eta_j(x) P \dfrac{\partial^2 w}{\partial x^2}(x)\,\mathrm{d}x + \int_0^L \eta_j(x) q(x)\,\mathrm{d}x - \int_0^L \eta_j(x)\mu(x)\ddot{w}(x,t)\,\mathrm{d}x \end{array} \tag{2.9}$$

Integration per partes will not affect the second and third term on the right hand side, but for the first term we obtain

$$\int_0^L \eta_j P \frac{\partial^2 w}{\partial x^2}\,\mathrm{d}x = \left[\eta_j P \frac{\partial w}{\partial x}\right]_0^L - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\,\mathrm{d}x \tag{2.10}$$

The bracketed term is fraught with possibilities. Number one, we may recognize part of the bracket in equation (2.6). In fact, if we propose to satisfy $r_F = 0$ at the right hand side end of the wire ($x = L$) identically, we may simply replace $P\frac{\partial w}{\partial x}$ (which is known there) with F_L. That takes care of the force residual (2.6). Number two, at the left-hand side end of the wire the value of $P\frac{\partial w}{\partial x}$ is unknown, but we have the option of making η_j vanish at $x = 0$. This will burden all the η_j's with a condition, $\eta_j(x = 0) = 0$, but that is something we can afford.

We are in a position to summarize: We have been able to avoid the need to carry the displacement residual (2.5) [eliminated by design of the trial function] and the force residual (2.6) [incorporated into the balance residual– hence, "natural" boundary condition]. Therefore, we will try to find the approximate solution w to satisfy the balance equation in the residual form

$$\eta_j(L)F_L - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\,\mathrm{d}x + \int_0^L \eta_j q\,\mathrm{d}x - \int_0^L \eta_j \mu \ddot{w}\,\mathrm{d}x = 0, \quad j = 1, ..., N \tag{2.11}$$

where

$$\begin{array}{l} \eta_j(x=0) = 0, \quad \eta_j \in C^0, \quad j = 1, ..., N \\ w(x=0,t) = \bar{w}_0(t), \quad w \in C^0, \\ w(x, t=0) \approx \bar{W}(x), \quad \dfrac{\partial w}{\partial t}(x, t=0) \approx \bar{V}(x). \end{array} \tag{2.12}$$

We write for the trial function $w \in C^0$ and similarly for the test functions. This literally means that the functions are continuous, which is a substitute here for a more

precise mathematical statement, but which nevertheless ensures that the integrals in (2.11) exist.

The initial conditions need to be suitably approximated, in general we will not be able to satisfy them exactly (which is why we write $\approx$). Typically, interpolation is used.

2.6 Stiffness and mass matrix

It is time to come back to the choice of the test and trial functions. As advertised in Section 2.4, we have been able to change the requirements on the test and trial function: Their derivatives are now balanced– only the first-order derivatives are needed for both. Therefore, the piecewise linear interpolation function of Figure 2.4 is now an possibility. However, we can still forge ahead while keeping our options open.

Let us make the assumption that the time is fixed $t = \bar{t}$ ($\bar{t}$ some given number). To describe the trial function, we will resort to a common technique in interpolation which is to write the interpolant as a linear combination of basis functions. Therefore, let us assume that the trial function is written as

$$w(x, \bar{t}) = \sum_{i=1}^{N} N_i(x) w_i(\bar{t}) \tag{2.13}$$

where by $w_i(\bar{t})$ we simply mean that the coefficients of the linear combination w_i are actually functions of time, evaluated at the particular time $\bar{t}$. Substituting into (2.12), we obtain

$$\begin{aligned} &\eta_j(L) F_L - \int_0^L \frac{\partial \eta_j}{\partial x} P \sum_{i=1}^{N} \frac{\partial N_i}{\partial x} w_i(\bar{t}) \, \mathrm{d}x + \\ &\int_0^L \eta_j q \, \mathrm{d}x - \int_0^L \eta_j \mu \sum_{i=1}^{N} N_i \ddot{w}_i(\bar{t}) \, \mathrm{d}x = 0, \quad j = 1, ..., N \ , \end{aligned} \tag{2.14}$$

which may be simplified to

$$\begin{aligned} &\eta_j(L) F_L - \sum_{i=1}^{N} \left(\int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial N_i}{\partial x} \, \mathrm{d}x \right) w_i(\bar{t}) + \\ &\int_0^L \eta_j q \, \mathrm{d}x - \sum_{i=1}^{N} \left(\int_0^L \eta_j \mu N_i \, \mathrm{d}x \right) \ddot{w}_i(\bar{t}) = 0, \quad j = 1, ..., N \ , \end{aligned} \tag{2.15}$$

With the definitions

$$K_{ji} = \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial N_i}{\partial x} \, \mathrm{d}x \ , \tag{2.16}$$

where K_{ji} is usually referred to as the ***stiffness matrix***, and

$$M_{ji} = \int_0^L \eta_j \mu N_i \, \mathrm{d}x \ , \tag{2.17}$$

where M_{ji} is the (consistent) ***mass matrix***, we may write (2.15) as

$$\eta_j(L)F_L - \sum_{i=1}^{N} K_{ji}w_i(\bar{t}) + \int_0^L \eta_j q \, \mathrm{d}x - \sum_{i=1}^{N} M_{ji}\ddot{w}_i(\bar{t}) = 0, \quad j = 1, ..., N \ , \tag{2.18}$$

The matrix equation (2.29) is a system of coupled ordinary differential equations (evaluated at time $\bar{t}$), where the coupling is effected by the matrices K_{ji} and M_{ji}. The linear algebra is going to be much more efficient if the two matrices are *symmetric* and *sparse*.

The first property will follow if we take as the test functions η_j the basis functions themselves, $\eta_j \equiv N_i$. The second property may be achieved if the basis functions N_i are nonzero only on a small subset of the interval $0 \leq x \leq L$.

2.7 Piecewise linear basis functions

Lets us get back to the piecewise linear approximation we advertised for the trial function in Section 2.4. The broken line cannot be represented as a linear combination of linear functions that are all defined on the whole interval $0 \leq x \leq L$ (only two such functions are linearly independent, and these functions cannot represent the corners in the broken line). Therefore, we have to describe the piecewise linear curve interval-by-interval.

A common technique in interpolation is to write the interpolant as a linear combination of basis functions. In one dimension, the piecewise linear basis function is called the ***hat functions***. The six functions that are shown in Figure 2.5, all are examples of hat functions. For reasons that will be discussed later, we would want the hat functions as the constituent parts of a linear combination to be able to reproduce an arbitrary linear function over the whole interval. Because of the way in which we construct the hat functions in Figure 2.5, this property is automatically available.

Let us describe the construction of the piecewise linear basis functions. (In this book, the one-dimensional elements with two nodes at the end points are going to be referred to as L2.) First, the length of the wire is divided into disjoint subintervals. These subintervals are the ***finite elements*** for the one-dimensional problem. The end-points of the finite elements are called ***nodes***. Together, the finite elements and the nodes are known as the ***finite element mesh***: see Figure 2.6 (the element numbers are in the boxes; nodes are indicated by filled circles). Since all basis functions are constructed in the same way, we show the procedure for basis function N_3: as shown in the Figure 2.5, it is nonzero over two elements, 2 and 3; zero everywhere else. To be able to write it down over the two adjacent elements, we have to agree on the value of N_3 at node 3 (i.e. $N_3(x_3)$), which is shared by elements 2 and 3. Choosing $N_3(x_3) = 1$ has certain advantages, which will be introduced momentarily. Using the concept of Lagrange interpolation polynomials, we may write the function N_3 within element 2 as

$$N_3(x) = \frac{x - x_2}{x_3 - x_2}, \quad x_2 \leq x \leq x_3$$

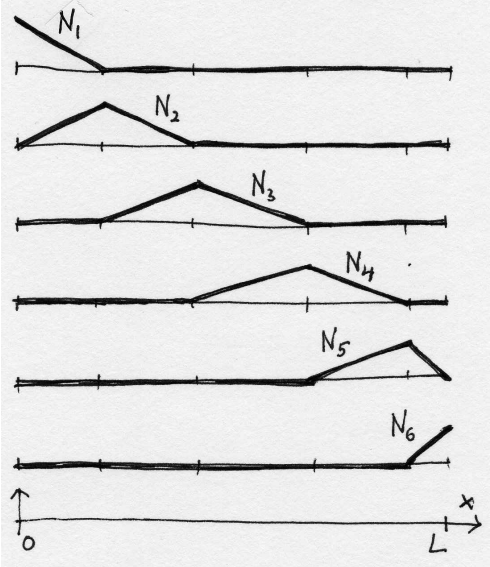

Fig. 2.5. Piecewise linear basis functions

and within element 3 as

$$N_3(x) = \frac{x - x_4}{x_3 - x_4}, \quad x_3 \leq x \leq x_4 \ .$$

All the other functions are expressed analogously. Putting them together in a linear combination for the trial function, we write

$$w(x) = \sum_{i=1}^{N} N_i(x) w_i$$

(for simplicity, we omit the time argument). Evaluating $w(x)$ at node k, we obtain

$$w(x_k) = \sum_{i=1}^{N} N_i(x_k) w_i$$

where the crucial expression is $N_i(x_k)$: by definition, the basis function N_k has value $+1$ at x_k, while all other functions $N_i, i \neq k$ are zero at x_k. This property is usually expressed mathematically as

$$N_i(x_k) = \delta_{ik} \ , \tag{2.19}$$

where the symbol δ_{ik} is known as the Kronecker delta

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

Because of this property, the value of $w(x_k)$ is

$$w(x_k) = \sum_{i=1}^{N} N_i(x_k) w_i = \sum_{i=1}^{N} \delta_{ik} w_i = w_k \ ,$$

and the parameters w_i have the physical meaning of the value of the interpolated function at the node i. The w_i's are usually called the ***degrees of freedom***, since being the control parameters of the trial function, they determine the shape of the actual solution from all the possible shapes of the trial function. They are the objects that our numerical method solves for.

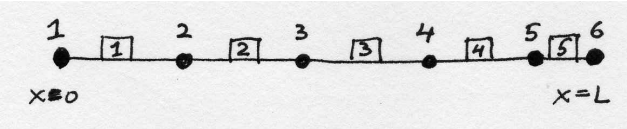

Fig. 2.6. The finite element mesh

2.8 Numerical quadrature

While in the preceding section we described how to compute the basis function N_3 by visiting the adjacent finite elements on which the function was nonzero, the task in an actual finite element program is different. The algorithm there is designed to facilitate numerical evaluation of the integrals in the residual equations. The integrals are calculated element-by-element (the integrands are in general discontinuous from element to element). Therefore, instead of being interested in a single basis function at any point within the mesh, we will rather be striving to calculate the values (and their derivatives) of *all the nonzero basis functions* at a particular point (the quadrature point) within a *single finite element*; see Figure 2.7. In the element-centric view,

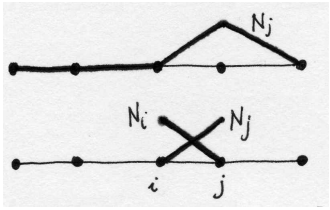

Fig. 2.7. Two different views on how to evaluate basis functions in the finite element mesh: top – compute a single basis function over the whole mesh; bottom – compute all nonzero basis functions over a single element.

we would evaluate the functions associated with the nodes at the endpoints of the element. Say for element connecting nodes i, and j, the functions N_i and N_j would be expressed as

$$N_i(x) = \frac{x - x_j}{x_i - x_j}, \quad N_j(x) = \frac{x - x_i}{x_j - x_i} \tag{2.20}$$

It is common practice to develop numerical integration rules on *standard intervals*. Often that will be $-1 \leq \xi \leq +1$ (line elements in one dimension, quadrilaterals in two

dimensions, and bricks in three dimensions all use this interval definition in so-called tensor-product forms). For instance, Simpson's 1/3 rule is given on this interval as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \frac{1}{3}f(\xi=-1)+\frac{4}{3}f(\xi=0)+\frac{1}{3}f(\xi=+1)$$

In general, a numerical quadrature rule would be written on the standard interval $-1 \leq \xi \leq +1$ as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \sum_{k=1}^{M} f(\xi_k)W_k \tag{2.21}$$

where ξ_k are the locations of the integration points, and W_k are their weights. Integrating numerically arbitrary functions over arbitrary intervals is then made possible by a map from the standard interval $-1 \leq \xi \leq +1$ to the arbitrary interval $a \leq x \leq b$

$$x = \frac{1}{2}(a+b)+\frac{1}{2}(b-a)\xi \tag{2.22}$$

(the first part is the midpoint of the interval $a \leq x \leq b$, the second part is the departure from the midpoint to either side). Because this map is linear, the relationship between the differentials is constant,

$$\mathrm{d}x = \frac{1}{2}(b-a)\mathrm{d}\xi$$

where the factor $\frac{1}{2}(b-a)$ is called the ***Jacobian determinant***. The Simpson's 1/3 rule is for an arbitrary interval $a \leq x \leq b$ expressed as

$$\int_{a}^{b} f(x)\mathrm{d}x \approx \frac{1}{2}(b-a)\left[\frac{1}{3}f(\xi=-1)+\frac{4}{3}f(\xi=0)+\frac{1}{3}f(\xi=+1)\right]$$

In general, if the map is

$$x = g(\xi) \tag{2.23}$$

and the numerical quadrature over an arbitrary interval may be written as

$$\int_{a}^{b} f(x)\mathrm{d}x \approx \sum_{k=1}^{M} f(\xi_k)\frac{\partial g}{\partial \xi}(\xi_k)W_k \tag{2.24}$$

where $\frac{\partial g}{\partial \xi}(\xi_k)$ is the Jacobian determinant evaluated at the quadrature point ξ_k.

Lets us now look at the integrals in the mass matrix (2.17). The integral will be evaluated over each element individually, and these contributions will be summed together. Therefore, let us consider the integral (2.17)over a single element, connecting nodes i and j

$$\int_{x_i}^{x_j} N_j(x)\mu(x)N_i(x)\ \mathrm{d}x\ .$$

Notice that we are not getting the indexes mixed up: N_j and N_i are the only two basis functions which are nonzero over the interval $x_i \leq x \leq x_j$. Clearly, the function $f(x)$ in equation (2.24) is

$$N_j(x)\mu(x)N_i(x)$$

which means that we have to express the basis functions in terms of ξ ($\mu(x)$ is typically constant over an element). However, we have the map (2.22), which upon substitution into (2.20) yields

$$N_i(\xi) = \frac{\xi - 1}{-2}, \quad N_j(\xi) = \frac{\xi + 1}{+2} \tag{2.25}$$

Basis functions expressed on the standard interval (2.25) are sometimes referred to as being expressed in the parametric coordinates. It is noteworthy that (2.25) are just the Lagrange interpolation polynomials on the standard interval. As we shall see later, writing down the basis functions over a standard shape – a square for general quadrilaterals, a cube for general brick elements, standards triangles or standard tetrahedra for general triangles or general tetrahedra, and so on– is not only convenient, but also highly advisable from the point of view of computer implementation: most of the code for different element shapes and types is then shared, and does not have to be repeated. However, it does mean that we have to express the derivative of the basis functions using a chain rule:

$$\frac{\partial N_i(\xi)}{\partial x} = \frac{\partial N_i(\xi)}{\partial \xi}\frac{\partial \xi}{\partial x} \tag{2.26}$$

The partial derivative $\frac{\partial \xi}{\partial x}$ is readily available from (2.22), and may be identified as the inverse of the Jacobian. To state the integral of the term corresponding to the mass matrix

$$\int_{x_i}^{x_j} N_j(x)\mu(x)N_i(x)\,\mathrm{d}x \approx \frac{1}{2}(x_j - x_i)\sum_{k=1}^{M}\frac{\partial N_j}{\partial x}(\xi_k)\frac{\partial N_i}{\partial x}(\xi_k)W_k.$$

where (2.26) is used to evaluate the derivatives of the basis functions.

At a first sight, the integrals in the stiffness matrix (2.16), and in the mass matrix (2.17), the latter will seem to require a more accurate numerical quadrature rule: the stiffness matrix involves products of the derivatives of the basis functions, which for linear basis functions are constants; the mass matrix, on the other hand, requires products of the basis functions themselves, which are linear functions of x. Therefore, the stiffness matrix will result from integrals of constants, while the mass matrix is the result of integrals of quadratic functions. However, at times increased efficiency and even accuracy may be achieved if the mass matrix is not integrated exactly, in particular diagonal mass matrices are often used to achieve both benefits.

The numerical quadratures that are in common use with polynomial finite elements are the Gaussian rules. They are well described in a number of textbooks, see for instance Reference [CC2005], and we are going to introduce them later.

2.9 Putting it together: system of ODE's

Applying the piecewise linear basis functions derived in Section 2.7 to equation (2.29) is a straightforward. The unknown degrees of freedom are $w_2(t)$, $w_3(t)$, ..., $w_N(t)$; the function $w_1(t)$ is given by the boundary conditions: the condition $w(x=0,t)=\bar{w}_0(t)$ becomes, as a result of the Kronecker delta property, $w_1(t)=\bar{w}_0(t)$.

The stiffness and mass matrices will be symmetric, tri-diagonal, i.e.

$$K_{ji}=\int_0^L \frac{\partial N_j}{\partial x} P \frac{\partial N_i}{\partial x}\,\mathrm{d}x \begin{cases} \neq 0, \text{ if } |i-j|\leq 1; \\ = 0, \text{ otherwise.} \end{cases} , \tag{2.27}$$

and

$$M_{ji}=\int_0^L N_j \mu N_i\,\mathrm{d}x \begin{cases} \neq 0, \text{ if } |i-j|\leq 1; \\ = 0, \text{ otherwise.} \end{cases} , \tag{2.28}$$

Equation (2.15) is trivially modified to read

$$N_j(L)F_L-\sum_{i=1}^{N} K_{ji}w_i(t)+\int_0^L N_j q\,\mathrm{d}x-\sum_{i=1}^{N} M_{ji}\ddot{w}_i(t)=0, \quad j=2,...,N\ , \tag{2.29}$$

Note that j runs from 2 to N (as explained above), but i ranges over all nodes, i.e. also the first degree of freedom is included, even though it is determined from the boundary condition. In effect, nonzero displacement $w_1(t)$ generates an external force with the jth component

$$-K_{j1}w_1(t)-M_{j1}\ddot{w}_1(t)\ .$$

The second order differential equations (2.29) may be integrated for instance by converting them to first order form and using an off-the-shelf Matlab integrator. However, because of their special form, there are excellent custom-tailored algorithms for this purpose: for example the Newmark explicit algorithm.

3

Introducing the Matlab code

In this chapter we will introduce a finite element library (or toolbox, if you prefer), `SOFEA`[1]. It will be used to produce finite element solutions using the results of the previous chapter for the Galerkin method. `SOFEA` implementation is in Matlab, and its design is based on the object oriented support in Matlab (release 14 and later). In particular, all the methods and algorithms are present in the library as classes, or methods defined for classes.

3.1 Statics

When the inertial forces may be neglected in the balance equation, we have the case of statics (static equilibrium). The Galerkin formulation simply drops the terms with the accelerations, and reads

$$N_j(L)F_L - \sum_{i=1}^{N} K_{ji}w_i + \int_0^L N_j q \, \mathrm{d}x = 0, \quad j = 2, ..., N \ , \tag{3.1}$$

which may be arranged in matrix form as

$$\boldsymbol{K}\boldsymbol{d} = \boldsymbol{L} \tag{3.2}$$

where $\boldsymbol{K}$ is a square $(N-1) \times (N-1)$ matrix collecting K_{ji}, $i, j = 2, ..., N$. The column matrix $\boldsymbol{d}$ collects the degrees of freedom $d_k = w_{k+1}$, $k = 1, ..., N-1$. The column matrix $\boldsymbol{L}$ is the load vector, with components

$$L_k = N_{k+1}(L)F_L - K_{k+1,1}w_1 + \int_0^L N_{k+1} q \, \mathrm{d}x = 0, \quad k = 1, ..., N-1 \ , \tag{3.3}$$

3.2 Statics: uniform load

Furthermore, we assume the transverse load q is uniform, and the transverse force is absent, $F_L = 0$. The Matlab script implementing the solution is `w1` [2]. It starts with the definition of the variables.

[1] `SOFEA` is ©2005, Petr Krysl

[2] Folder: `SOFEA/examples/taut_wire`

```
0001 disp('Taut wire: example 1-- statics, uniform load');
0002 L=6;
0003 P=4;
0004 q =-0.1;
```

Next, the mesh is defined: an array of nodes is created, with node 1 at $x = 0$ and so on. The function `fenode` is the constructor of the class `fenode`, and the attributes are being passed as fields of a `struct`, as pairs "name, value" (for instance, `'id',j`): this approach is uniformly adopted for all constructors. The array `gcells` collects finite elements of the type `gcell_L2`. The attribute `conn` is the connectivity: the numbers of nodes that are connected by the element. The finite elements are referred to as ***geometric cells***. The main reason is that the finite elements have rather limited responsibilities in `SOFEA`, namely calculation of the basis functions (and their derivatives), and drawing of the shape of the cell are essentially all that is required.

```
0005 n=2; % number of elements
0006 % Mesh
0007 x=0;
0008 fens=[];
0009 for j= 1:n+1
0010     fens=[fens fenode(struct ('id',j,'xyz',[x]));];
0011     x = x+(L/n);
0012 end
0013 gcells = [];
0014 for j= 1:n
0015     gcells = [gcells gcell_l2(struct('id',j,'conn',[j j+1]))];
0016 end
```

The operations on the mesh that reflect the particular problem that is being solved are encapsulated in the class descended from the finite element block, `feblock`. In particular, the prestressed wire stiffness and mass matrix, the effect of the distributed load, q, and the effect of the nonzero displacement at $x = 0$, are computed by the methods of the class `feblock_defor_taut_wire`. Note that Simpson's 1/3 rule is being used for the numerical quadrature. The finite element block consists of all the finite elements.

```
0017 % Finite element block
0018 feb = feblock_defor_taut_wire(struct ('mater',mater_defor,...
0019     'gcells',gcells,...
0020     'integration_rule',simpson_1_3_rule,...
0021     'P',P));
```

The quantities that are interpolated on the finite element mesh, such as the transverse displacement of the wire, w, or the geometry of the mesh, are represented in `SOFEA` as instances of the class `field`. The field `geom` records the geometry of the mesh. The constructor on line 0023 retrieves this information from the array of the nodes. The dimension of the field is 1 because each degree of freedom is just a single

displacement. On line 0025 we define the field of the transverse displacements, w. For convenience, it is defined by cloning the geom field, and then zeroing out all the degrees of freedom.

```
0022 % Geometry
0023 geom = field(struct ('name',['geom'], 'dim', 1, 'fens',fens));
0024 % Define the displacement field
0025 w    = 0*clone(geom,'w');
```

Next, the displacement (essential) boundary conditions are defined, and applied to the displacement field. The method set_ebc simply records which components of the degrees of freedom, at which node, are being prescribed (or released), and to which value are they being prescribed. The method apply_ebc is then used to transfer this information to the actual degrees of freedom. Finally, the method numbereqns numbers the degrees of freedom that are not being prescribed, effectively assigning each one a global equation number.

An important remark should be made here: Lines 0028, 0029, and 0030 illustrate a design feature of Matlab, where all arguments are passed by value . Therefore, no matter what we do with the arguments inside the functions, the values that were passed by the caller into those functions do not change at all. The functions work with copies, not the actual variables that the caller passed. If the caller wishes to change the variables, the method must return the changed value, and the caller must assign this value. Example: on line 0031 the method numbereqns will number the equations in a copy of the field w, and then will return the copy. Since we assign back to the field w, the computed numbering of the equations will be now available in w; if we did not assign back to w, all the work done by the method numbereqns would be forgotten.

```
0026 % Apply EBC's
0027 fenids=[1]; prescribed=[1]; component=[1]; val=0;
0028 w    = set_ebc(w, fenids, prescribed, component, val);
0029 w    = apply_ebc (w);
0030 % Number equations
0031 w    = numbereqns (w);
```

Next we create the global system of equations. The stiffness matrix is an object, K, which gets created in line 0033, and initialized to represent a dense neqns×neqns matrix. In line 0034, the stiffness matrices calculated for each finite element by the block feb are assembled into the global matrix K (class dense_sysmat). The number of equations is being retrieved from the displacement field (the globally equation numbers have been placed there above) using the get method. The get method is available for all SOFEA objects, and just typing w at the command line produces a list of all the properties that can be obtained from the object. A great way in which objects may be explored is the graphical user interface of the object browser, OBgui (part of SOFEA). Its operation is supported by the output of get(w) (notice that only

the object itself is passed as argument): using `get` in this way returns a cell array, with the name and the description of each attribute.

```
0032 % Assemble the system matrix
0033 K = start (dense_sysmat, get(w, 'neqns'));
0034 K = assemble (K, stiffness(feb, geom, w));
```

The load q is in this case represented by the `body_load` class. The global load object `sysvec` is assembled from element load vectors computed by the finite element block.

```
0035 % Load
0036 bl = body_load(struct ('magn',inline(num2str(q))));
0037 F = start (sysvec, get(w, 'neqns'));
0038 F = assemble (F, body_loads(feb, geom, w, bl));
```

Finally, the global stiffness object is asked to produce the actual stiffness array, and the global load object is asked to supply the actual load vector. The standard backslash Matlab operator than produces the solution, which is then stored in the proper places in the displacement field `w`. The method `scatter_sysvec` distributes the system vector (the solution of the system of linear equations) to the proper degrees of freedom, and we should note that again the result is assigned to `w`.

```
0039 % Solve
0040 w = scatter_sysvec(w, get(K,'mat')\get(F,'vec'));
```

A graphical representation is generated by plotting the linear coordinate (`gather`ed from the geometry field `geom`) versus the linear interpolation of the approximate displacements (`gather`ed from `w`). The analytical solution is also plotted (line `0043`). It is noteworthy that the approximate solution interpolates the analytical solution.

```
0041 % Plot
0042 xs= (0:0.01:L);
0043 plot (xs, -q/P*xs.*(xs/2-L),'r-','linewidth', 3);
0044 hold on
0045 plot (gather (geom, (1:n+ 1),'values'), ...
0046     gather (w, (1:n+ 1),'values'),'bo-','linewidth', 3);
0047 figure (gcf)
```

3.3 Free vibration

If we remove all external loads, and prescribe homogeneous (zero) displacements, the Galerkin formulation reads

$$-\sum_{i=2}^{N} K_{ji} w_i(t) - \sum_{i=2}^{N} M_{ji} \ddot{w}_i(t) = 0, \quad j = 2, ..., N \ , \tag{3.4}$$

which may be arranged in matrix form as

$$\boldsymbol{K}\boldsymbol{w} + \boldsymbol{M}\ddot{\boldsymbol{w}} = \boldsymbol{0} \tag{3.5}$$

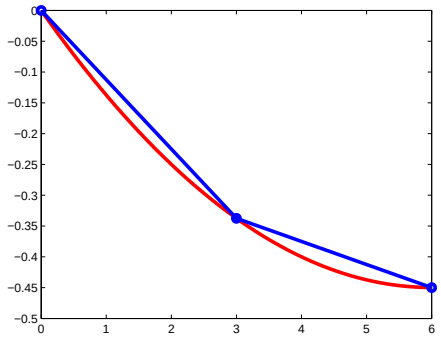

Fig. 3.1. The displacements of the taut wire

where $\boldsymbol{K}$ is a square $(N-1)\times(N-1)$ matrix collecting K_{ji}, $i,j=2,...,N$, and analogously for the mass matrix. The column matrix $\boldsymbol{w}$ collects the degrees of freedom $w_i(t)$. Equations (3.5) represent the so-called ***free vibration*** response. The solution is sought in the form $\boldsymbol{w}(t)=\boldsymbol{\phi}\exp(\omega t)$, which leads to the ***generalized eigenvalue problem***

$$\boldsymbol{K}\boldsymbol{\phi}-\omega^2\boldsymbol{M}\boldsymbol{\phi}=\boldsymbol{0} \tag{3.6}$$

where ω is the circular frequency, and ϕ is the eigenmode.

As an example, the present model will be used to calculate the first five natural frequencies of a simply-supported taut string of constant mass density. The Matlab script `wvib`[3] obtains the solution for a series of progressively finer and finer meshes (from 8 elements to 1024 elements). The mass matrix is either computed from the formula (2.17) (this is the so-called ***consistent mass matrix***), or or it is diagonalized (lumped) by assigning each node half the mass of the adjacent finite elements

$$M_{ji}=\mu h\delta_{ji}\ , \tag{3.7}$$

which results in the so-called ***lumped mass matrix***.

The analytical formula for the natural frequency [Gra91]

$$\omega^2=\frac{P}{\mu}\left(\frac{n\pi}{L}\right)^2\ ,\quad n=1,2,3,...$$

produces reference values which are compared with the frequencies solved for from (3.6). The results are summarized in Figure 3.2. The progressive reduction of error due to the use of more and more elements is called ***convergence***. It may be observed that the two mass matrix formulations leads to convergence from different sides: consistent mass matrix overestimates the natural frequencies, while the lumped mass matrix underestimates them. Concerning accuracy: clearly, for quite reasonable engineering tolerance of 5% error, it takes 32 elements for either formulation of the mass matrix to compute all five natural frequencies within the tolerance.

[3] Folder: `SOFEA/examples/taut_wire`

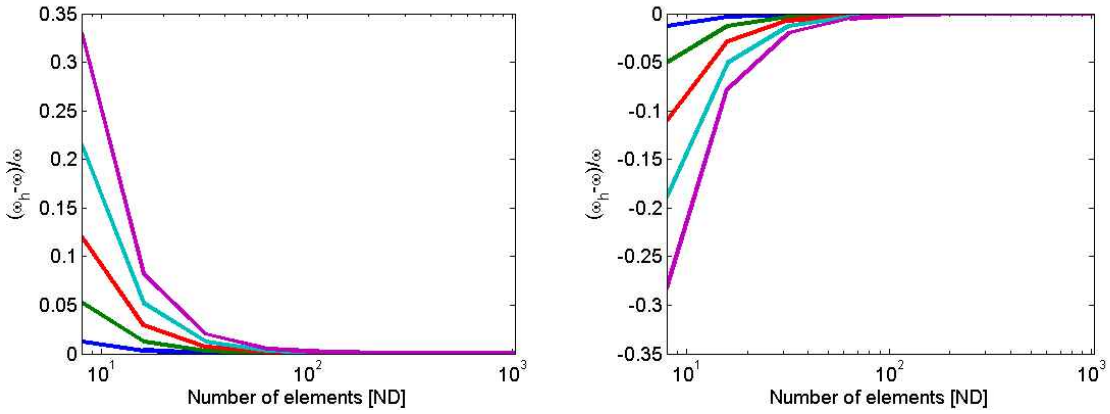

Fig. 3.2. Convergence of the first five natural frequencies of vibration; left: consistent mass matrix, right: lumped mass matrix. Vertical axis: normalized error $(\omega_h - \omega)/\omega$.

3.4 Integration of transient motion

In addition to analytical approaches, the system of ordinary differential equations (2.29) may be integrated numerically. In this section, we will apply an off-the-shelf Matlab integrator.

The Matlab integrators work with a system of first order differential equations. Therefore, (2.29) will be first converted to this form. For convenience, equation (2.29) will be cast in matrix form

$$\boldsymbol{M}\ddot{\boldsymbol{w}} + \boldsymbol{K}\boldsymbol{w} = \boldsymbol{F} \tag{3.8}$$

The stiffness matrix entries are defined in (2.16). The mass matrix may be either of the consistent variety (2.17), or a special-purpose matrix such as the lumped mass (3.7). Note that $\boldsymbol{w}$ is the vector function of time-dependent deflections.

Defining $\boldsymbol{v} = \dot{\boldsymbol{w}}$, the first order system may be written as

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{M} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{w}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ -\boldsymbol{K}\boldsymbol{w} + \boldsymbol{F} \end{bmatrix} , \tag{3.9}$$

which is in the "mass matrix" form

$$\widetilde{\boldsymbol{M}}\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}) \ . \tag{3.10}$$

In what follows we will consider a particular example of transient vibrations: unforced oscillation due to a particular set of initial conditions. In this example we consider initial zero deflection, and initial velocity which corresponds to a singular perturbation: a single node at the midspan is given a nonzero velocity. This is representative of a physical situation in which a stationary taut string is rapped sharply with a hammer. Waves propagate away from the midpoint, hit the fixed end-points, and complex interference pattern develops (see Figure 3.3).

3.4.1 Using built-in Matlab solver

The Matlab script `wtransient1`[4] computes the solution to the transient vibration problem using a built-in Matlab solver.

As expected, the initial conditions need to be supplied.

```
0024      w0 = gather_sysvec (w);
0025      v0 = w0; v0(round(n/2)) = 1;

0027      K = start (dense_sysmat, get(w, 'neqns'));
0028      K = assemble (K, stiffness(feb, geom, w));
0029      Kmat =get(K,'mat');
0030      % Assemble the consistent mass matrix
0031      M = start (dense_sysmat, get(w, 'neqns'));
0032      % Assemble the lumped mass matrix
0033      M = assemble(M, elemat( struct('eqnums',...
                  (1:n-1),'mat', eye(n-1) * mu*L/n)));
0034      Mmat=get(M,'mat');
0035      Mattilda= [eye(get(w, 'neqns')),zeros(get(w, 'neqns'));...
0036          zeros(get(w, 'neqns')),Mmat];
```

Matlab provides a suite of several ODE solvers. `ode23` is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine [BS89]. These solvers use a standard argument list.

```
0037      % Solve
0038      function rhs =f(t,y)
0039          neq=length(y)/2;
0040          w=y(1:neq); v=y(neq+1:end);
0041          rhs= [v;-Kmat*w];
0042      end
```

The invocation of the solver is unremarkable, except that we use `odeset` to supply the mass matrix, $\widetilde{\boldsymbol{M}}$. The output arguments collect an array of output times, and an array of calculated deflections at all internal nodes.

```
0043      [ts,ys]=ode23(@f,[0, 1500],[w0;v0],...
                    odeset('Mass',Mattilda,'RelTol',1e-3));
```

3.4.2 Using the Trapezoidal integrator

The Runge-Kutta of the previous section seems to be doing an adequate job. However, integrating the first order system is sub-optimal for the taut string problem in the first place: The dimension of the ODE system is doubled. Secondly, the Runge-Kutta `ode23` integrator is not actually doing very well.

[4] Folder: `SOFEA/examples/taut_wire`

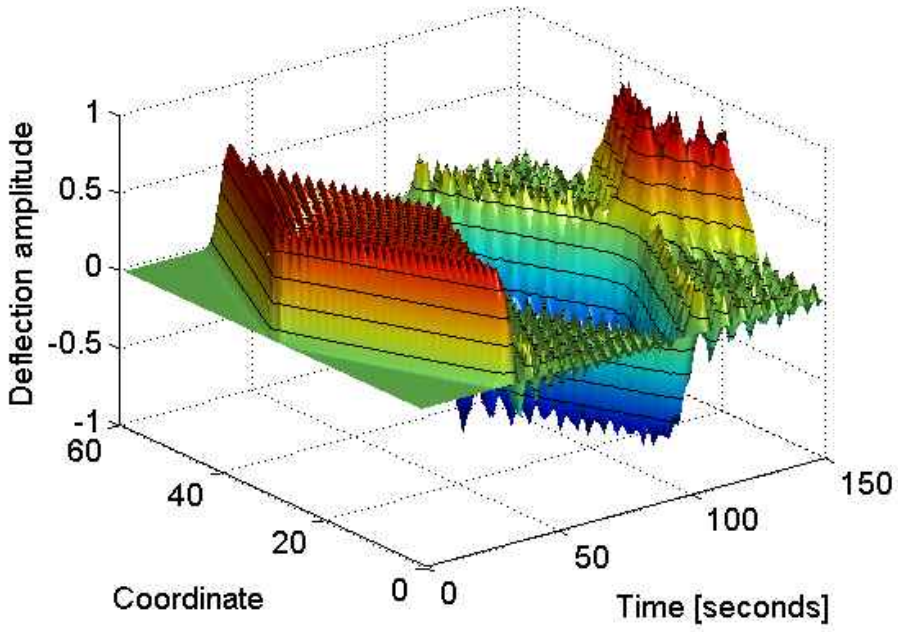

Fig. 3.3. Amplitude of the vertical deflection as a function of the coordinate x and the time.

Energy conservation is a very important indicator of the quality of the numerical solution, since energy *should* be conserved in the exact solution to this particular problem. For the taut string problem, the total energy may be defined as

$$TE = \frac{1}{2}\left(\dot{\boldsymbol{w}} \cdot \boldsymbol{M} \cdot \dot{\boldsymbol{w}} + \boldsymbol{w} \cdot \boldsymbol{K} \cdot \boldsymbol{w}\right) \ .$$

It may be observed that the solution produced by the `ode23` integrator does not conserve this quantity (refer to Figure 3.4), and that provides motivation for the development of algorithms specialized to mechanical systems.

The algorithm we are going to develop next is a special form of the well-known Newmark integrator which is very popular and well-respected in the computational mechanics community for a number of reasons [Hug00]. The starting point is the first order system (3.10). Integrating in time

$$\int_{t_0}^{t} \widetilde{\boldsymbol{M}}\dot{\boldsymbol{y}} \, \mathrm{d}\tau = \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}) \, \mathrm{d}\tau \ , \tag{3.11}$$

yields

$$\widetilde{\boldsymbol{M}}(\boldsymbol{y}_t - \boldsymbol{y}_{t_0}) = \int_{t_0}^{t} \boldsymbol{f}(\tau, \boldsymbol{y}) \, \mathrm{d}\tau \ . \tag{3.12}$$

The right hand side may be integrated in time using the trapezoidal rule, which leads to the algorithm

$$\widetilde{\boldsymbol{M}}(\boldsymbol{y}_t - \boldsymbol{y}_{t_0}) = \frac{t - t_0}{2}\left(\boldsymbol{f}(t, \boldsymbol{y}_t) + \boldsymbol{f}(t_0, \boldsymbol{y}_{t_0})\right) \ . \tag{3.13}$$

Instead of the first-order form, we may multiply through (3.13), obtaining a coupled system

$$\boldsymbol{w}_t = \boldsymbol{w}_{t_0} + \frac{t-t_0}{2}\left(\boldsymbol{v}_t + \boldsymbol{v}_{t_0}\right) \tag{3.14}$$

$$\boldsymbol{M}\boldsymbol{v}_t = \boldsymbol{M}\boldsymbol{v}_{t_0} - \frac{t-t_0}{2}\boldsymbol{K}\left(\boldsymbol{w}_t + \boldsymbol{w}_{t_0}\right) + \frac{t-t_0}{2}\left(\boldsymbol{F}_t + \boldsymbol{F}_{t_0}\right)\ , \tag{3.15}$$

which may be marched forward by substituting the first equation into the second, solving for $\boldsymbol{v}_t$, and then updating $\boldsymbol{w}_t$ from the first equation. The integrator obtained in this way is a special case of the general Newmark integrator [Hug00]. The Newmark algorithm has two free parameters, and for the special choice $\gamma = 1/2$ and $\beta = 1/4$ (the so-called average acceleration method) one obtains the trapezoidal integrator (3.14).

The Matlab script `wtransient2`[5] computes the solution to the transient vibration problem using a trapezoidal integrator (Newmark average-acceleration integrator). Note that the integrator algorithm is implemented as a nested function, which has access to variables defined in the defining environment, the stiffness matrix `Kmat` and the mass matrix `Mmat`.

```
0033      Mmat=get(M,'mat');
0034      % Solve
0035      function [ts,ys] = trapezoidal(nsteps,tspan,w0,v0)
0036          ts= zeros(nsteps, 1);
0037          nu =length(w0);
0038          ys = zeros(2*nu,nsteps);
0039          dt = (tspan(2)-tspan(1))/nsteps;
0040          t =tspan(1);
0041          for i=1:nsteps
0042              ts(i) =t;
0043              ys(1:nu,i) =w0; ys(nu+1:end,i) =v0;
0044              v1=(Mmat+((dt/2)^2)*Kmat)\...
                          ((Mmat-((dt/2)^2)*Kmat)*v0-dt*Kmat*w0);
0045              w1=w0+dt/2*(v0+v1);
0046              w0=w1;v0=v1;
0047              t=t+dt;
0048          end
0049          ys=ys';
0050      end
0051      %  Now call the integrator
0052      [ts,ys]=trapezoidal(3000,[0, 1500],w0,v0);
```

Figure 3.4 compares the computed total energy for the two integrators introduced in this section. Evidently, the nominally less accurate trapezoidal (Newmark average acceleration) integrator does a much better job than the Runge-Kutta `ode23` integrator.

[5] Folder: `SOFEA/examples/taut_wire`

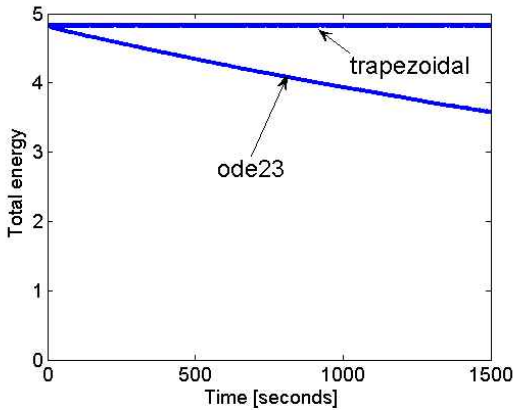

Fig. 3.4. Total energy obtained with two different time integrators.

4

The boundary conditions for the model of a taut wire

In this chapter we will explore the effect the boundary conditions have on the solution, both its existence and its computability with the Galerkin model.

We will consider only statics, so that the balance equation (1.1) drops the inertial term

$$P\frac{\partial^2 w}{\partial x^2} + q = 0 \ , \tag{4.1}$$

and furthermore we will assume that the transverse load q is a constant. With a definition $k = -q/P$, the task is to integrate

$$\frac{\partial^2 w}{\partial x^2} = k = \text{constant} \ , \tag{4.2}$$

which is easily accomplished as

$$w(x) = k\frac{x^2}{2} + Cx + D \ , \tag{4.3}$$

where C and D are integration constants to be determined from the boundary condition.

As already mentioned (Section 1.4), there is only one condition on the boundary (which for the problem of the wire consists of two disjoint sets, each consisting of a single point at the either end) that we can use to determine the solution. Since the balance equation is of second order, and the unknown is a single function, a single boundary condition is all that is needed (with caveats, however). A simple explanation for this need in one dimension is that a second order equation needs two integration constants; we will talk about this issue in higher dimensional domains when we deal with the heat conduction equation and also, in a lot more detail, in the part dedicated to the elasticity problem.

The boundary condition may be of two distinct types, either one of the two we introduced earlier: either prescribed deflection (essential boundary condition), or prescribed slope (derivative of the deflection, which is a natural boundary condition). Since we may prescribe only one boundary condition, but the boundary consists of two disjoint sets, we may in fact prescribe one or the other type at either of the two endpoints. It does matter which type of boundary condition is applied, and the details are discussed in this chapter.

4.1 Essential and natural boundary conditions at separate end-points

This selection of boundary condition specifications has been treated in the previous few chapters. The natural condition may be expressed in terms of end-point forces. At $x = L$ the relation is equation (1.3), and it may be derived in the same way at $x = 0$ as

$$P\frac{\partial w}{\partial x}(0) + F_0 = 0 \ . \tag{4.4}$$

The Galerkin algebraic equations for an essential boundary condition at $x = 0$ and a natural boundary condition at $x = L$ is given by equation (2.29), with proper allowance made for vanishing accelerations, and when the points of application of these boundary conditions are switched, the changes are limited to the boundary term and the conditions for the test and trial functions only

$$N_j(0)F_0 - \sum_{i=1}^{N} K_{ji}w_i + \int_0^L N_j q \ \mathrm{d}x = 0, \quad j = 1, ..., N-1 \ , \tag{4.5}$$

where

$$\begin{aligned} &N_j(x = L) = 0, \quad N_j \in C^0, \quad j = 1, ..., N-1 \\ &w_N = \bar{w}_L \ . \end{aligned} \tag{4.6}$$

After the solution for the deflection has been obtained, the slope at the end with the essential boundary condition may be computed, yielding the associated reaction force. For instance, for the problem (4.5), (4.6), the analytical solution of (4.3), with the boundary conditions

$$\frac{\partial w}{\partial x}(0) = -\frac{F_0}{P} = \bar{w}_0', \quad w(L) = \bar{w}_L \ ,$$

is

$$w(x) = \frac{k}{2}(x^2 - L^2) + \bar{w}_0'(x - L) + \bar{w}_L \ .$$

This yields the reaction force $F_L = P(kL + \bar{w}_0')$.

4.2 Essential boundary conditions only

The boundary conditions are in this case the deflections given at both ends,

$$w(0) = \bar{w}_0, \quad w(L) = \bar{w}_L \ ,$$

which may be substituted into (4.3) for $x = 0$ and $x = L$ to yield two equations for C and D. Upon substitution, the slopes at the end points are available as

$$\frac{\partial w}{\partial x}(0) = \frac{\bar{w}_L - \bar{w}_0}{L} - k\frac{L}{2}, \quad \frac{\partial w}{\partial x}(L) = \frac{\bar{w}_L - \bar{w}_0}{L} + k\frac{L}{2} \ .$$

From the slopes, the forces F_0 and F_L are available from (4.4) and (1.3). The physical meaning of these two forces is that of reactions at the supports.

The algebraic equation for this type of boundary conditions becomes

$$-\sum_{i=1}^{N} K_{ji} w_i + \int_0^L N_j q \, \mathrm{d}x = 0, \quad j = 2, ..., N-1 \ , \tag{4.7}$$

where

$$\begin{aligned} &N_j(x=0) = 0, N_j(x=L) = 0, \quad N_j \in C^0, \quad j = 1, ..., N-1 \\ &w_1 = \bar{w}_0, \quad w_N = \bar{w}_L \ . \end{aligned} \tag{4.8}$$

Note that the degrees of freedom to be computed are only at the interior nodes.

4.3 Natural boundary conditions only

The boundary conditions are in this case the slopes given at both ends,

$$\frac{\partial w}{\partial x}(0) = \bar{w}_0', \quad , \frac{\partial w}{\partial x}(L) = \bar{w}_L', \quad ,$$

Integrating (4.2) once yields for the slope

$$\frac{\partial w}{\partial x}(x) = kx + C \ ,$$

which may be used in conjunction with the boundary conditions to express the constant C as either of the two expressions

$$C = \bar{w}_0' \ , \quad \text{or} \quad C = \bar{w}_L' - kL \ .$$

But clearly, this is only possible if

$$\bar{w}_0' = \bar{w}_L' - kL \ ,$$

which may be interpreted as a condition under which a solution exists: if the two slopes are linked by the previous equation, solution exists; otherwise no solution exists, because the boundary conditions are contradictory.

If the solution exists, and the two slopes are not independent, the boundary conditions are really not going to be sufficient to determine two constants of integration, but only one. Correspondingly, the deflection of the wire is then

$$w(x) = k\frac{x^2}{2} + \bar{w}_0' x + D \ ,$$

where the constant D remains undetermined.

An initial boundary value problem with natural boundary conditions only is called ***pure-traction problem***, or Neumann problem. We could see that the solution would

exist only under certain conditions. In this case, the condition is one of *static equilibrium*: the end-point forces must balance the transverse load. Provided equilibrium may be established, the solution still remains *non-unique*, as it is possible to translate the wire perpendicularly to its axis without affecting the equilibrium. In terms of the linear algebra of the solution this is manifested by a singular stiffness matrix. An obvious computational treatment is to force the displacement at one node to be some known a value, for instance zero. Adding this "superfluous boundary condition" makes the problem solvable uniquely. (We might wish to consider that one boundary condition specification was in fact missing because of the linear dependence between the slopes; so in this way we are really just filling a void.) Another possibility is to add an artificial spring at one node.

4.4 Overspecified boundary conditions

The boundary condition application consisted so far of one and only one condition specification at each of the two endpoints (i.e. at *the boundary*). In this section we will attempt to apply *two boundary conditions at one end* and none at the other. Since this will turn out to be too much prescribed information at one point, will call this the case of over-specified boundary conditions.

For instance, we could prescribe two pieces of information at $x = 0$

$$w(0) = \bar{w}_0 \ , \quad \frac{\partial w}{\partial x}(0) = \bar{w}_0' \ , \tag{4.9}$$

Integrating (4.2) once yields for the slope

$$\frac{\partial w}{\partial x}(x) = kx + C \ ,$$

which may be used in conjunction with the natural boundary condition to express the constant C as

$$C = \bar{w}_0' \ .$$

Integrating again yields

$$w(x) = k\frac{x^2}{2} + \bar{w}_0' x + w_0 \ ,$$

where all the constants are determined from the two boundary conditions at a single endpoint. Using this expression, we can calculate the deflection and slope at $x = L$. The interesting thing is that the slope is in general nonzero

$$\frac{\partial w}{\partial x}(L) = kL + \bar{w}_0' \neq 0 \ ,$$

which means that even though we did not assume this, there must be an applied force F_L at the end where no boundary conditions have been specified. For the same reasons, it is not possible to prescribe either deflection or slope at $x = L$ while the

boundary condition (4.9) is in force. Any mismatch between the prescribed values and the calculated values would make the existence of the solution an impossibility (the solution must satisfy the balance equation and all boundary conditions).

But the preceding discussion was based on the analytical integration of the balance equation. Is this way of prescribing boundary conditions compatible with the Galerkin technique? For the moment, we will go back all the way to the weighted residual equation (2.11), but we will keep all the boundary terms (and drop the inertial terms – statics)

$$\eta_j(L)F_L - \eta_j(0)F_0 - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\, \mathrm{d}x + \int_0^L \eta_j q\, \mathrm{d}x = 0, \quad j = 1, ..., N\ . \tag{4.10}$$

The functions η_j and w need to be just sufficiently smooth to make the integrals exist. But now we introduce the boundary condition (4.9). As discussed below equation (2.10), the condition $w(x = 0) = \bar{w}_0$ could be used to eliminate the associated reaction, F_0, by setting $\eta_j(0)$ for all j. The resulting weighted residual formulation is

$$\eta_j(L)F_L - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\, \mathrm{d}x + \int_0^L \eta_j q\, \mathrm{d}x = 0, \quad j = 1, ..., N\ . \tag{4.11}$$

where we must place one condition on w

$$w(x = 0) = \bar{w}_0, \quad w \in C^0\ . \tag{4.12}$$

However, we realize that the prescribed force F_0 now no longer affects the solution! Another difficulty is that F_L is not known, and hence there is not enough equations to solve for $w_i, i = 2, ..., N$ and F_L. What is needed is an additional equation that would link together F_0 and F_L. The equation of global equilibrium is such an equation

$$F_L = F_0 - qL\ ,$$

and may be added to the equations resulting from (4.11). The global equations are then of a blocked nature. The Matlab script `woverspec` [1] generates a numerical solution for the definition of boundary conditions discussed above. The code snippet below illustrates the system matrix, `A`, in line `0034`, which clearly displays how it is composed of distinct blocks

$$\boldsymbol{A} = \begin{bmatrix} & \boldsymbol{K} & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \\ \begin{bmatrix} 0 \cdots 0 \end{bmatrix} & & 1 \end{bmatrix}\ .$$

[1] Folder: `SOFEA/examples/taut_wire`

```
0028 K = start (dense_sysmat, get(w, 'neqns'));
0029 K = assemble (K, stiffness(feb, geom, w));
0030 % and now the special terms due to the boundary conditions
0032 z = zeros(N-1, 1); zt=z';
0033 z(end) = 1;
0034 A = [get(K,'mat'),z;zt,1];
```

The right hand side of the system equations is correspondingly expanded by a single element at the bottom.

```
0036 fi = force_intensity(struct ('magn',inline(num2str(q))));
0037 F = start (sysvec, get(w, 'neqns'));
0038 F = assemble (F, body_loads(feb, geom, w, fi));
0039 F = assemble (F, nz_ebc_loads(feb, geom, w));
0040 % and now the special terms
0041 b=get(F,'vec');
0042 b = [b; -(F_0-q*L)];
```

Figure 4.1 illustrates the correct solution versus an incorrect solution obtained when the F_L term is simply dropped from (4.11): the denotation "natural" is then fully operational, since that means that the natural boundary condition $F_L = 0$ is in effect. That can be verified visually as the slope at the right hand side is apparently approaching zero.

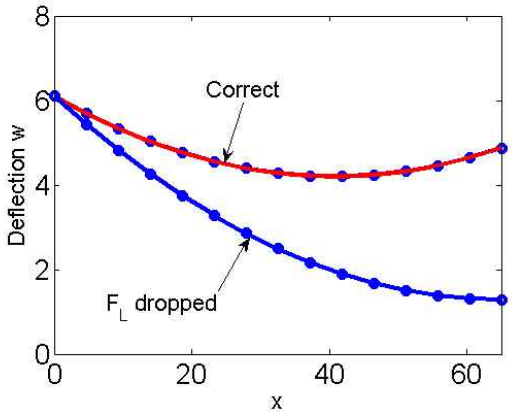

Fig. 4.1. The wire problem with overspecified boundary conditions.

Clearly, we can see that overspecification of boundary conditions does not fit the framework of the Galerkin method very well. Special treatment is required. However, there is worse news: the existence of the solution has to be demonstrated case-by-case, it does not follow automatically. The situation is much worse in higher dimensional problems.

Part II

Thermal analysis

5

Model of Heat Diffusion

In this chapter we will develop a finite element model for heat conduction problems. The excellent textbook by Lienhard and Lienhard [LL05] has all the details one might require to supplement the treatment that follows.

5.1 Balance equation

In this section, our goal is to derive the balance equation that governs heat conduction in solids as a partial differential expression. It will be converted to a residual form, which will then be treated with the Galerkin method.

To begin with 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 5.1). The amount of heat energy in the control volume U is expressed in terms of volume density of heat energy, u

$$U = \int_V u \, \mathrm{d}V \tag{5.1}$$

As the means of change of the heat energy within the control volume we consider

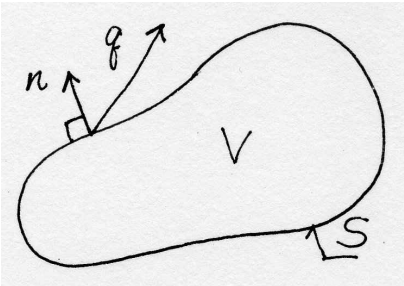

Fig. 5.1. The domain for the heat conduction problem

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

$$\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \ , \tag{5.2}$$

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

$$\int_V Q \, \mathrm{d}V \ . \tag{5.3}$$

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

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \ . \tag{5.4}$$

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

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\int_V u \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V \ , \tag{5.5}$$

and with the application of the chain rule, the relationship (5.5) is expressed as

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}T}\frac{\partial T}{\partial t} \, \mathrm{d}V \ , \tag{5.6}$$

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

Substituting, we write

$$\int_V c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \ . \tag{5.7}$$

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

$$\int_V \alpha \frac{\partial M}{\partial t} \, \mathrm{d}V = \int_V \mu \, \mathrm{d}V \ , \tag{5.8}$$

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

$$\alpha \frac{\partial M}{\partial t} = \mu \ , \tag{5.9}$$

which is a local version of (5.8). 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 (5.7), we have to convert the surface integral to a volume integral. We have the needed tool in the celebrated ***divergence theorem***

$$\int_V \mathrm{div}\boldsymbol{q} \, \mathrm{d}V = \int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \ , \tag{5.10}$$

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

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

Consequently, equation (5.7) may be rewritten

$$\int_V c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = -\int_V \mathrm{div}\boldsymbol{q} \, \mathrm{d}V + \int_V Q \, \mathrm{d}V \ . \tag{5.11}$$

and grouping the terms as

$$\int_V \left[c_V \frac{\partial T}{\partial t} + \mathrm{div}\boldsymbol{q} - Q \right] \mathrm{d}V = 0 \ . \tag{5.12}$$

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***

$$c_V \frac{\partial T}{\partial t} + \mathrm{div}\boldsymbol{q} - Q = 0 \ . \tag{5.13}$$

5.2 Constitutive equation

Equation (5.13) contains too many variables: both temperature and heat flux. Since it is a scalar equation, the logical next step is to express the heat flux in terms of temperature. That is the contents of the Fourier model: heat flows opposite to the gradient of the temperature (downhill). In matrix form

$$\boldsymbol{q} = -\boldsymbol{\kappa}(\mathrm{grad}T)^T \ . \tag{5.14}$$

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

$$\boldsymbol{\kappa} = \kappa \mathbf{1} \tag{5.15}$$

for the so-called thermally isotropic material, and

$$\boldsymbol{\kappa} = \begin{pmatrix} \kappa_x & 0 & 0 \\ 0 & \kappa_y & 0 \\ 0 & 0 & \kappa_z \end{pmatrix} \tag{5.16}$$

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. (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

$$\text{grad}T = \left(\frac{\partial T}{\partial x} \; \frac{\partial T}{\partial y} \; \frac{\partial T}{\partial z} \right) \tag{5.17}$$

With the constitutive equation, the balance equation (5.13) is now purely in terms of the absolute temperature,

$$c_V \frac{\partial T}{\partial t} - \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] - Q = 0 \; . \tag{5.18}$$

5.3 Boundary conditions

From now on, V is going to be the volume of the whole solid of interest. The most important fact about the boundary conditions is that we need to have a boundary condition *at each point* of the surface S. As we expect by now, the model is about temperature. Correspondingly, the boundary conditions are an expression of our knowledge of the temperature distribution in the solid.

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 5.2).Therefore,

$$T(\boldsymbol{x}, t) - \overline{T}(\boldsymbol{x}, t) = 0, \quad \boldsymbol{x} \text{ on } S_1 \; . \tag{5.19}$$

where by $\boldsymbol{x}$ we mean the position vector. This type of condition is known as the primary, or ***essential***, boundary condition.

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

$$\boldsymbol{n} \cdot \boldsymbol{q} - \overline{q}_n = 0, \quad \text{on } S_2 \; . \tag{5.20}$$

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.

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

$$\boldsymbol{n} \cdot \boldsymbol{q} - h(T - T_a) = 0, \quad \text{on} \quad S_3 \ . \tag{5.21}$$

where T_a is the known temperature of the surrounding medium (ambient temperature), and h is the heat transfer coefficient.

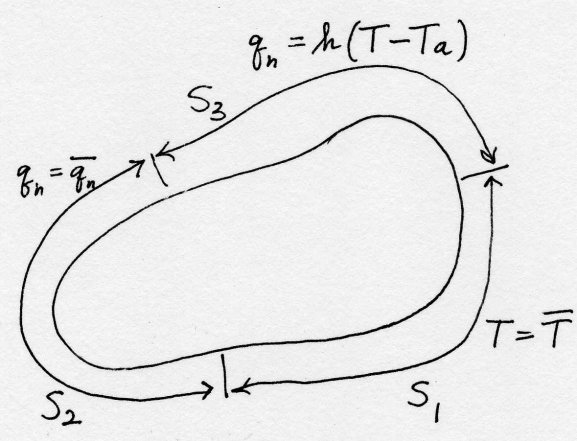

Fig. 5.2. The subdivision of the surface for the purpose of the boundary condition application

On the sufficiency of boundary conditions.

As pointed out earlier in this section, one boundary condition is needed at each point on the boundary. The precise mathematical statement of the necessity of having one boundary condition in place is somewhat involved, but we can build on intuition fairly easily.

Would it be possible to specify one boundary condition at only a subset of the complete boundary, leaving the behavior of the solution along part of the boundary unspecified? As a thought experiment, we consider a square domain, shown in Figure 5.3, with no source of heat generation, and zero temperature prescribed on the S_1 subset of the boundary. On the $\widehat{S}$ part of the boundary we assume nothing is known about the temperature distribution. Is it possible that the temperature field is completely determined by these boundary conditions?

If this was true, the variation of temperature along $\widehat{S}$ wouldn't affect the solution in the domain. However, if zero temperature was prescribed all around the circumference of the square, the solution to this problem would be zero temperature everywhere. Consequently, also the normal component of the flux (in fact all components of the flux) would vanish everywhere. Evidently, if the temperature along $\widehat{S}$ was nonzero, it

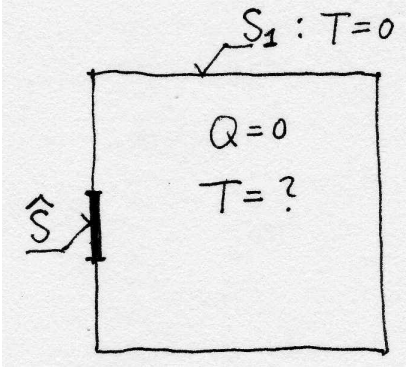

Fig. 5.3. The square domain with partially undefined boundary condition

would require transitioning to zero temperature on S_1 (and elsewhere within the domain), hence the solution within the domain *would* depend on the temperature along $\widehat{S}$; alternatively, if there was nonzero heat flux along $\widehat{S}$, the temperature distribution within the square domain would be affected. This is illustrated in Figure 5.4: varying the heat flux along $\widehat{S}$ (here shown for two different uniform distributions, positive and negative) changes the distribution of temperature. Therefore, we must conclude that *prescribing one boundary condition along the entire boundary* of the domain is a *necessary condition* to make the solution unique.

Is not a sufficient condition, however. In the so-called Neumann problem only flux is being prescribed along the entire boundary. This is equivalent to the pure-traction problem of Section 4.3. The solution is not unique, because any temperature distribution of the form $T(x, y) + \widetilde{T}$, where $T(x, y)$ satisfies the balance equation and the natural boundary conditions, and $\widetilde{T}$ is a constant, is also a solution (the constant term disappears with differentiation). Typically, the Neumann boundary conditions are supplemented with temperature being prescribed at one point to remove the constant $\widetilde{T}$ from consideration.

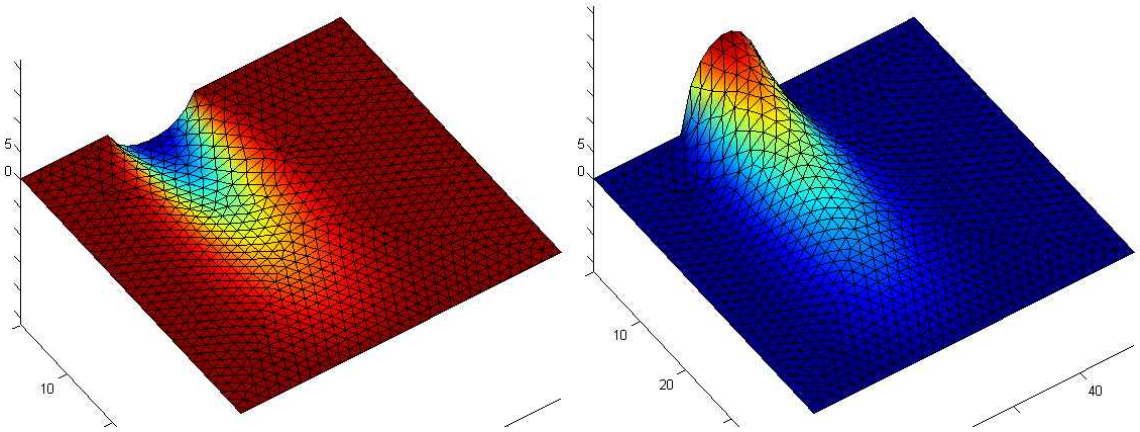

Fig. 5.4. The square with variable heat flux along part of its boundary

5.4 Initial condition

The primary variable in our problem is the temperature, T, and it is present in the balance equation (5.18) with the first order time derivative. Therefore, we will need one initial condition,

$$T(\boldsymbol{x}, 0) = \overline{T}_0(\boldsymbol{x}) \quad \text{in} \quad V \ . \tag{5.22}$$

The initial condition must be match with boundary conditions on S_1 at time $t = 0$:

$$\overline{T}_0(\boldsymbol{x}) = \overline{T}(\boldsymbol{x}, 0), \quad \boldsymbol{x} \text{ on } S_1 \ . \tag{5.23}$$

5.5 Summary of the PDE model of heat conduction

Figure 5.5 gives a diagrammatic overview of the terminology and the various equations of the model of heat conduction.

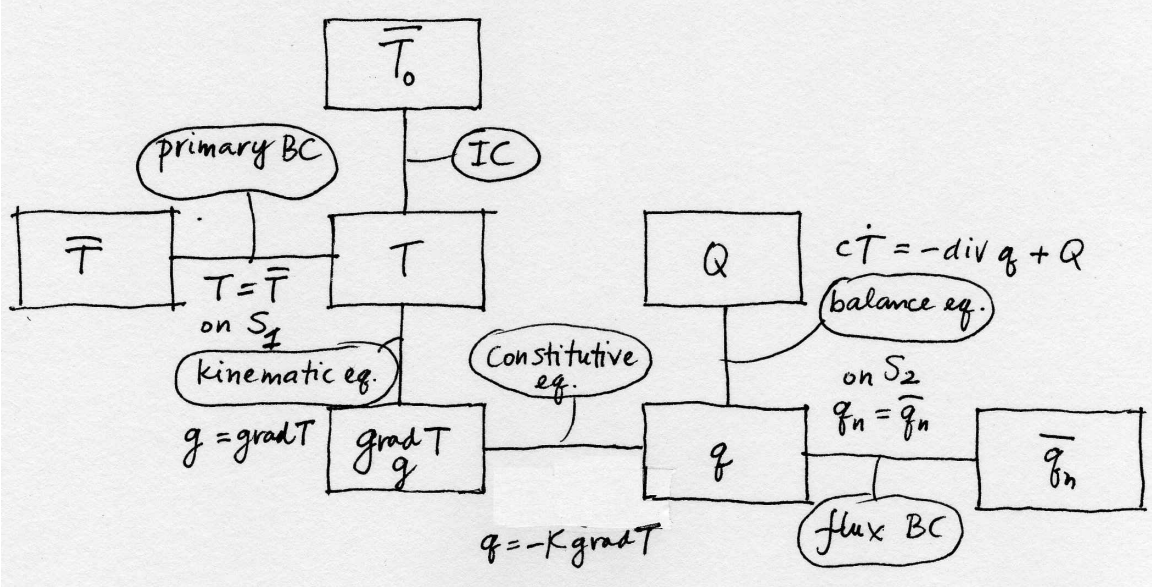

Fig. 5.5. Diagram of the heat conduction model

6

Galerkin method for the model of heat conduction

6.1 Weighted residual formulation

The balance equation (5.18) defines the balance residual as

$$r_B = c_V \frac{\partial T}{\partial t} - \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] - Q \ . \tag{6.1}$$

As explained in Chapter 2, the first step is to decide to satisfy the essential boundary condition by the restricting possible trial functions to only those that satisfy the essential boundary conditions a priori

$$T(\boldsymbol{x},t) - \overline{T}(\boldsymbol{x},t) = 0, \quad \boldsymbol{x} \text{ on } \ S_1 \ . \tag{6.2}$$

and to write the weighted residual equation for the local balance residual

$$\int_V \eta(\boldsymbol{x}) r_B(\boldsymbol{x},t) \, \mathrm{d}V \ . \tag{6.3}$$

The first and the third term are kept as they are, but the second term reminds us of a similar term in equation (2.3): the test function η multiplies an expression that contains the second derivatives of temperature (the div $\left[\boldsymbol{\kappa}(\text{grad}T)^T\right]$ term), which was why Section 2.5 was needed. Balancing the order of differentiation by shifting one derivative from the temperature to the test function η will be beneficial here too: similarly to Section 2.5, we will be able to satisfy the natural boundary conditions without having to include them as a residual (naturally!). As before, the price to pay is the need to place some restrictions on the test function.

Integration by parts was used in Section 2.5, and just a little bit more general tool will work here too. For the moment, it will be convenient to work with the expression

$$-\eta \ \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] = \eta \ \text{div}\boldsymbol{q} \ ,$$

that is, with the flux variable replacing $\boldsymbol{\kappa}(\text{grad}T)^T$.

The integration by parts in the case of a multidimensional integral is generalized in the divergence theorem (5.10). We may anticipate that η div$\boldsymbol{q}$ is the result of the chain rule applied to the vector $\eta \, \boldsymbol{q}$. That is indeed the case

$$\mathrm{div}\,(\eta\,\boldsymbol{q}) = \eta\,\mathrm{div}\boldsymbol{q} + (\mathrm{grad}\eta)\,\boldsymbol{q} \tag{6.4}$$

which is easily verified in components.

Therefore, we may start working on the integral

$$\int_V \eta\,\mathrm{div}\boldsymbol{q}\,\mathrm{d}V$$

where we substitute from (6.4)

$$\int_V \eta\,\mathrm{div}\boldsymbol{q}\,\mathrm{d}V = \int_V \mathrm{div}\,(\eta\,\boldsymbol{q})\;\mathrm{d}V - \int_V (\mathrm{grad}\eta)\,\boldsymbol{q}\,\mathrm{d}V \tag{6.5}$$

The divergence theorem may be applied to the first integral on the right

$$\int_V \eta\,\mathrm{div}\boldsymbol{q}\,\mathrm{d}V = \int_S \eta\,\boldsymbol{q}\cdot\boldsymbol{n}\,\mathrm{d}S - \int_V (\mathrm{grad}\eta)\,\boldsymbol{q}\,\mathrm{d}V \tag{6.6}$$

But $\boldsymbol{q}\cdot\boldsymbol{n}$ is known on parts of the boundary – see equations (5.20) and (5.21). Therefore, we may split the surface integral into one for each sub-surface,

$$\int_V \eta\,\mathrm{div}\boldsymbol{q}\,\mathrm{d}V = \int_{S_1} \eta\,\boldsymbol{q}\cdot\boldsymbol{n}\,\mathrm{d}S + \int_{S_2} \eta\,\overline{q}_n\,\mathrm{d}S + \int_{S_3} \eta\,h(T-T_a)\,\mathrm{d}S - \int_V (\mathrm{grad}\eta)\,\boldsymbol{q}\,\mathrm{d}V \tag{6.7}$$

We see that the situation is analogous to the one discussed below equation (2.10): 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 η vanish along S_1. In this way, we obtain

$$\int_V \eta\,\mathrm{div}\boldsymbol{q}\,\mathrm{d}V = \int_{S_2} \eta\,\overline{q}_n\,\mathrm{d}S + \int_{S_3} \eta\,h(T-T_a)\,\mathrm{d}S - \int_V (\mathrm{grad}\eta)\,\boldsymbol{q}\,\mathrm{d}V \tag{6.8}$$

where $\eta(\boldsymbol{x}) = 0$ for $\boldsymbol{x}\in S_1$.

Expanding the weighted residual equation (6.3) yields

$$\begin{aligned}&\int_V \eta\,r_B\,\mathrm{d}V = \\ &\quad\int_V \eta c_V\frac{\partial T}{\partial t}\,\mathrm{d}V + \int_V (\mathrm{grad}\eta)\,\boldsymbol{\kappa}(\mathrm{grad}T)^T\,\mathrm{d}V - \int_V \eta Q\,\mathrm{d}V \\ &\quad+\int_{S_2} \eta\,\overline{q}_n\,\mathrm{d}S + \int_{S_3} \eta\,h(T-T_a)\,\mathrm{d}S = 0, \qquad \eta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x}\in S_1\end{aligned} \tag{6.9}$$

6.2 Reducing the model dimension

In this Section we show how the original three-dimensional model can be reduced to just two active dimensions. For some structures we can make the observation that the temperature does not vary along one coordinate direction, say along the z direction.

Figure 6.1 shows a disk of thickness Δz. It is a slice of a structure of an unchanging cross-section which is very along in the z direction compared to the transverse dimensions (some authors call this "infinitely long", evidently with a bagfull of grains of salt). If the temperature distribution does not depend on the z direction, and if we can neglect what is happening near the end sections, the component of the temperature gradient along the z direction will be negligible, $\partial T/\partial z = 0$. This does not necessarily

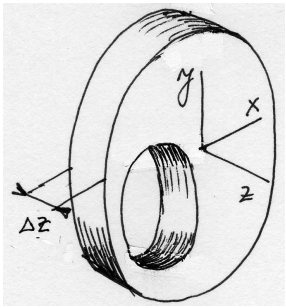

Fig. 6.1. Diagram of the heat conduction model

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 (5.14) to yield

$$q_z = \kappa_{zx}\partial T/\partial x + \kappa_{zy}\partial T/\partial y \ .$$

However, for the two classes of materials (5.15) and (5.16) 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.

Going back to the Figure 6.1: the heat flux through the cross sections is zero, and the temperature through 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 heat flux, $\overline{q}_n = 0$ (type S_2, equation (5.20))

$$\boldsymbol{n} \cdot \boldsymbol{q} = \pm q_z = 0, \quad \text{on the cross sections} \ . \tag{6.10}$$

Since the temperature does not vary with z, the integrals (6.8) may be simplified by pre-integrating in the thickness direction, $\mathrm{d}V = \Delta z \ \mathrm{d}S$. The volume integrals are then evaluated over the cross-sectional area, S_c, (see Figure 6.2); provided $\overline{q}_n$ and h are independent of z, the surface integrals are computed as integrals over the contour of the cross-section, C_c.

$$\begin{aligned} &\int_{S_c} \eta c_V \frac{\partial T}{\partial t} \ \Delta z \ \mathrm{d}S + \int_{S_c} (\mathrm{grad}\eta) \ \boldsymbol{\kappa} (\mathrm{grad}T)^T \ \Delta z \ \mathrm{d}S - \int_{S_c} \eta Q \ \Delta z \ \mathrm{d}S \\ &+ \int_{C_{c,2}} \eta \ \overline{q}_n \ \Delta z \ \mathrm{d}C + \int_{C_{c,3}} \eta \ h(T - T_a) \ \Delta z \ \mathrm{d}C = 0, \quad \eta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in C_{c,1} \end{aligned} \tag{6.11}$$

Note that the thickness Δz is a constant and could cancel without any effect on the solution. Nevertheless, equation (6.11) still applies to a fully three-dimensional body. To maintain this notion throughout the book, we shall not cancel the thickness.

Note that (6.11) does not refer to z, except in the term $\partial./\partial z$. We know that the temperature does not depend on z, and concerning the gradient of η: we simply assume that η does not depend on z: $\eta = \eta(x, y)$. The last assumption completes the reduction of the problem to two dimensions: all the functions depend on x and y only.

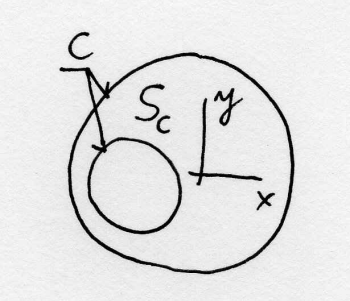

Fig. 6.2. Diagram of the heat conduction model

6.3 Test and trial functions: basis functions on triangulations

It is time to talk about the test and trial function. They are both functions of x and y only, $\eta = \eta(x, y)$ and $T = T(x, y, t)$ (and for the trial function, time). The only difference between them is the value they assume on part of the boundary (part of the cross-section contour, for our two-dimensional disk) where the temperature is being prescribed, $C_{c,1}$:

$$T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \quad \eta(\boldsymbol{x}) = 0 \quad \boldsymbol{x} \text{ on } C_{c,1} \ .$$

Let us consider first the test function. It needs to be defined as a function of x and y over arbitrarily shaped domains. 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)[1]. The domain of the disk with a hole (shown in Figure 6.2) is approximated as a collection of triangles (in other words, it is *tiled* with triangles, or *triangulated*), see Figure 6.3. 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*** (compare with Section 2.7), while the line segments connecting the nodes are called ***edges***. Evidently, the triangles are the finite elements.

[1]The so-called "linear triangle" made its first appearance in a lecture by Courant in 1943, applied to Poisson's equation, which is a time-independent version of the heat conduction equation of this chapter. 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. Clough then applied the triangle to problems in civil engineering, and he also coined the terminology "finite element".

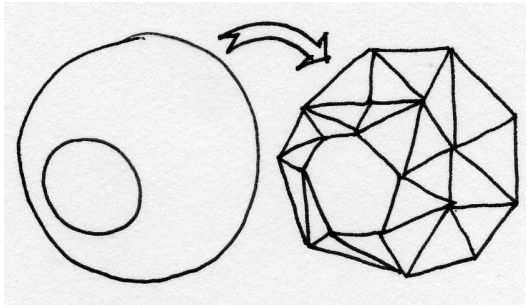

Fig. 6.3. Mesh of the disk domain

Interpolation on the triangle mesh will be treated as a linear combination of "tents". 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 basis function tent is shown in Figure 6.4. For those who do not like tents, the term *hat function* may be preferable.

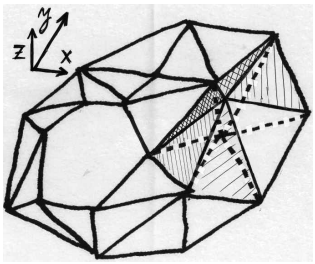

Fig. 6.4. Visual representation of one basis function on the mesh of the disk

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; evidently, it is defined to be zero everywhere else. (If you are inside the "tent", you are standing on the support of the function.)

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 separately. As discussed in Section 2.8, the alternative viewpoint would rather express all the nonzero pieces of all the basis functions over a single triangle (element). Referring to Figure 6.4, 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, we stand before the task of writing down the expressions for the three basis functions on a single triangle.

6.4 Basis functions on the standard triangle

Each of the three basis functions is zero along one edge of the triangle: again, refer to Figure 6.4. The task is accomplished most readily when the triangle is in a special position with respect to the coordinates: the ***standard triangle***; see Figure 6.5. The basis functions associated with nodes ② and ③ are simply

$$N_2(\xi, \eta) = \xi \ , \tag{6.12}$$

and

$$N_3(\xi, \eta) = \eta \ . \tag{6.13}$$

As is easily verified, N_2 is zero along the edge ①③, and assumes value $+1$ at node ②; analogous properties hold for N_3. If N_1 should be equal to $+1$ at the origin, it must be written as

$$N_1(\xi, \eta) = 1 - \xi - \eta \ . \tag{6.14}$$

Clearly, N_1 vanishes at the edge opposite node ①. Thus, we see that the three functions we just formulated satisfy the Kronecker delta property, equation (2.19). As in Section 2.7 this means the degree of freedom at each node of the triangle is the value of the interpolated function at the node.

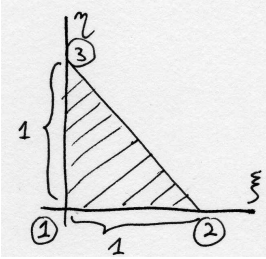

Fig. 6.5. Standard triangle

In this way, we formulate interpolation over the standard triangle. One quantity that we can interpolate on the standard triangle are the Cartesian coordinates.

$$\boldsymbol{x} = \sum_{i=1}^{3} N_i(\xi, \eta) \boldsymbol{x}_i \ , \tag{6.15}$$

where the result of the interpolation is a point in the Cartesian coordinates

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} \ ,$$

and

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad i = 1, 2, 3 \ ,$$

are the coordinates of the three points that are being interpolated. Equation (6.15) is a mapping from the pair ξ, η to the point x, y. Substituting for the basis functions, it may be written explicitly as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} . \tag{6.16}$$

This matrix equation is accompanied by the picture in Figure 6.16. The two vectors, $\boldsymbol{v}$ and $\boldsymbol{w}$, are the two columns of the square matrix in (6.16):

$$\boldsymbol{v} = \begin{bmatrix} (x_2 - x_1) \\ (y_2 - y_1) \end{bmatrix} ,$$

and

$$\boldsymbol{w} = \begin{bmatrix} (x_3 - x_1) \\ (y_3 - y_1) \end{bmatrix} .$$

If both ξ and η vary between zero and one, equation (6.16) adds the two vectors, $\xi\boldsymbol{v}$ and $\eta\boldsymbol{w}$ to the vector $[x_1, y_1]^T$, 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 (6.16) covers the area of the hatched triangle. To summarize, equation (6.16) is a *map* from the standard triangle to a triangle in the Cartesian coordinates with corners in given locations.

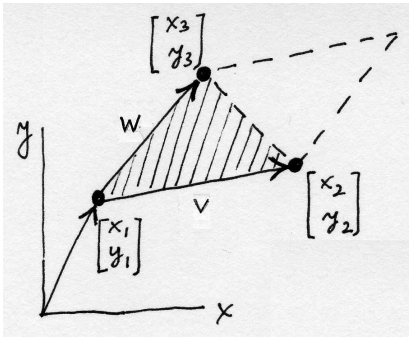

Fig. 6.6. Interpolating Cartesian coordinates on the standard triangle

Inverting (6.16) to express ξ and η, which could then be substituted into (6.12) – (6.14) to produce basis functions in terms of x and y, looks appealing but should be resisted. The reason is that numerical quadrature is available on the standard triangle, but is much harder on general triangles. This will become especially clear with quadratic elements later in the book.

However, since equation (6.16) is a invertible map from the standard triangle to a triangle in the Cartesian coordinates (invertibility follows if the triangle does not have its corners in a single straight-line: why?), we do get an approach to *evaluating basis functions on a general triangle*. Given a point $\bar{x}, \bar{y}$ in the Cartesian coordinates, and within the bounds of a triangle, we can use the inverse of the map (6.16) to

obtain point $\bar{\xi}, \bar{\eta}$ in the standard triangle (path 1 in Figure 6.7). Therefore, we may then evaluate $N_i(\bar{\xi}, \bar{\eta})$, which is the value $N_i(\bar{x}, \bar{y})$ (path 2 in Figure 6.7). That seems awkward, but normally we would want to evaluate the basis functions in order to perform numerical quadrature, that is at a particular point (quadrature point) within the triangle. In that case, $\bar{\xi}, \bar{\eta}$ would be *known* (and $\bar{x}, \bar{y}$ would be unknown), and calculation of the function value is easy. Evaluation of the *derivatives* of the basis functions is a little bit more complex, and will be discussed later in the section on numerical quadrature.

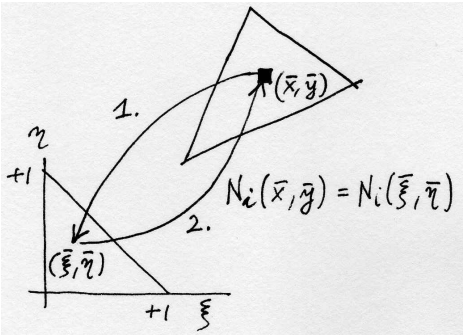

Fig. 6.7. Using the map from the standard triangle to evaluate basis functions over a general triangle

We understand now that each node in the mesh is associated with a single basis function. In the following, whenever we will write

$$N_i = N_i(x, y) \ ,$$

it has to be understood that within each triangle in the mesh, $x = x(\xi, \eta)$, $y = y(\xi, \eta)$, where ξ and η are coordinates on the standard triangle.

6.5 Discretizing the weighted residual equation

The trial function will be expressed using the basis functions as (compare with Section 2.7)

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

where the sum ranges over all the basis functions (i.e. over all the nodes in the mesh). Included are also nodes on the boundary where the temperature is being prescribed, $C_{c,1}$[2]. These nodes are on the other hand excluded from the set of possible test functions (which is expected to vanish along $C_{c,1}$), so that we will choose

$$\eta(x, y) = N_i(x, y), \quad i \text{ excluded when node } i \in C_{c,1}$$

[2] Apropos boundaries: Figure 6.8 clearly shows that with straight edges we are only approximating any boundaries that are curved. Some error is involved, but fortunately we are able to control this error by reducing the length of the edges.

The nodes whose basis functions are not part of the linear combination for the test function are shown as empty circles in Figure 6.8.

To simplify, we shall adopt the following notation:

$$\eta(x, y) = N_j(x, y), \quad \forall \text{ free } j,$$

where "free j" ranges over the nodes where the temperature is not being prescribed; and

$$T(x, y, t) = \sum_{\text{all } i} N_i(x, y) T_i(t)$$

where "all i" ranges over all the nodes, including those where the temperature is being prescribed.

As an aside, because the basis on the standard triangle satisfies the Kronecker delta property (2.19), the values of the degrees of freedom $T_i(t)$ at the nodes "prescribed i" (the nodes with the empty circles in Figure 6.8) are simply the values of the interpolated prescribed temperature at the nodes, $T_i(t) = \overline{T}(x_i, y_i, t)$.

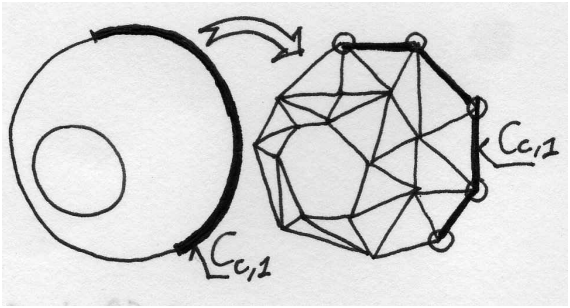

Fig. 6.8. Interpolating Cartesian coordinates on the standard triangle

The finite element expansions for the trial and test functions are now substituted into the weighted residual integral (6.11). For clarity, the substitution will be shown term-by-term (henceforth we will omit the arguments):

$$\int_{S_c} \eta c_V \frac{\partial T}{\partial t} \, \Delta z \, \mathrm{d}S = \int_{S_c} N_j c_V \sum_{\text{all } i} N_i \frac{\partial T_i}{\partial t} \, \Delta z \, \mathrm{d}S \ , \quad \forall \text{ free } j \ , \tag{6.17}$$

which simplifies to

$$\sum_{\text{all } i} \left[\int_{S_c} N_j c_V N_i \, \Delta z \, \mathrm{d}S \right] \frac{\partial T_i}{\partial t} \ , \quad \forall \text{ free } j \ , \tag{6.18}$$

The term in the bracket mixes together i and j from two different sets. However, some of the degrees of freedom $\partial T_i / \partial t$ are known. Therefore, separating the known and unknown may be a good idea:

$$\begin{aligned}
&\sum_{\text{all } i}\left[\int_{S_c} N_j c_V N_i \ \Delta z \ \mathrm{d}S\right]\frac{\partial T_i}{\partial t} = \\
&\sum_{\text{free } i}\left[\int_{S_c} N_j c_V N_i \ \Delta z \ \mathrm{d}S\right]\frac{\partial T_i}{\partial t} + \\
&\sum_{\text{prescribed } i}\left[\int_{S_c} N_j c_V N_i \ \Delta z \ \mathrm{d}S\right]\frac{\partial \overline{T}_i(t)}{\partial t} \ , \quad \forall \text{ free } j \ ,
\end{aligned} \tag{6.19}$$

The first integral on the right hand side of (6.19) suggests defining a matrix

$$C_{ji} = \int_{S_c} N_j c_V N_i \ \Delta z \ \mathrm{d}S \ , \quad \forall \text{ free } j, i, \tag{6.20}$$

the ***capacity matrix***. The integral in the second term will be given a different symbol, since the meaning is different from the first (the latter is a load-like term)

$$\overline{C}_{ji} = \int_{S_c} N_j c_V N_i \ \Delta z \ \mathrm{d}S \ , \quad \forall \text{ free } j, \ \forall \text{ prescribed } i. \tag{6.21}$$

Next, the second term in (6.11):

$$\begin{aligned}
&\int_{S_c} (\mathrm{grad}\eta)\ \boldsymbol{\kappa}(\mathrm{grad}T)^T \ \Delta z \ \mathrm{d}S = \int_{S_c} (\mathrm{grad}N_j)\ \boldsymbol{\kappa}(\mathrm{grad}\sum_{\text{all } i} N_i T_i)^T \ \Delta z \ \mathrm{d}S = \\
&\sum_{\text{free } i}\left[\int_{S_c} (\mathrm{grad}N_j)\ \boldsymbol{\kappa}(\mathrm{grad}N_i)^T \ \Delta z \ \mathrm{d}S\right] T_i + \\
&\sum_{\text{prescribed } i}\left[\int_{S_c} (\mathrm{grad}N_j)\ \boldsymbol{\kappa}(\mathrm{grad}N_i)^T \ \Delta z \ \mathrm{d}S\right] \overline{T}_i \quad \forall \text{ free } j,
\end{aligned} \tag{6.22}$$

and the ***conductivity matrix*** may be defined

$$K_{ij} = \int_{S_c} (\mathrm{grad}N_j)\ \boldsymbol{\kappa}(\mathrm{grad}N_i)^T \ \Delta z \ \mathrm{d}S \ , \quad \forall \text{ free } j, i, \tag{6.23}$$

and the elements to go with the load-like term

$$\overline{K}_{ij} = \int_{S_c} (\mathrm{grad}N_j)\ \boldsymbol{\kappa}(\mathrm{grad}N_i)^T \ \Delta z \ \mathrm{d}S \ , \quad \forall \text{ free } j, \ \forall \text{ prescribed } i. \tag{6.24}$$

Next, the load term corresponding to the internal heat generation:

$$L_{Q,j} = \int_{S_c} N_j Q \ \Delta z \ \mathrm{d}S \ , \quad \forall \text{ free } j, \tag{6.25}$$

Finally, the terms corresponding to natural boundary conditions. On the $C_{c,2}$ part of the boundary, only a load term results.

$$L_{q2,j} = -\int_{C_{c,2}} N_j \ \overline{q}_n \ \Delta z \ \mathrm{d}C \tag{6.26}$$

On the other hand, on the $C_{c,3}$ part of the boundary, where the heat flux is proportional to the difference between the ambient temperature and the surface temperature, we get a load term

$$L_{q3,j} = \int_{C_{c,3}} N_j \; hT_a \; \Delta z \; \mathrm{d}C \; , \quad \forall \text{ free } j, \tag{6.27}$$

and a ***surface heat transfer matrix***:

$$H_{ji} = \int_{C_{c,3}} N_j \; hN_i \; \Delta z \; \mathrm{d}C \; , \quad \forall \text{ free } j, i, \tag{6.28}$$

To summarize, using the definitions of the various matrices and load terms, the system of ordinary differential equations that results from the finite element discretization in space reads

$$\begin{aligned} &\sum_{\text{free } i} C_{ji} \frac{\partial T_i}{\partial t} + \sum_{\text{free } i} K_{ji} T_i + \sum_{\text{free } i} H_{ji} T_i \\ &+ \sum_{\text{prescribed } i} \overline{C}_{ji} \frac{\partial \overline{T}_i(t)}{\partial t} + \sum_{\text{prescribed } i} \overline{K}_{ji} \overline{T}_i - L_{Q,j} - L_{q2,j} - L_{q3,j} = 0 \end{aligned} \qquad \forall \text{ free } j, \tag{6.29}$$

6.6 Derivatives of the basis functions; Jacobian

The results of this section are much more general than may be expected. While we derive the formulas from which the derivatives of basis functions may be calculated for the linear triangles, the same implementation is used for all the so-called ***isoparametric*** elements in the `SOFEA` toolbox.

To evaluate the conductivity matrix, we need to be able to calculate the derivatives of the basis functions with respect to x and y. Equations (6.12–6.14) 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

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial x} \tag{6.30}$$

$$\frac{\partial N_i}{\partial y} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial y} \tag{6.31}$$

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

$$\left[\frac{\partial \heartsuit}{\partial x}, \; \frac{\partial \heartsuit}{\partial y}\right] = \left[\frac{\partial \heartsuit}{\partial \xi}, \; \frac{\partial \heartsuit}{\partial \eta}\right] \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} \; . \tag{6.32}$$

The derivatives are arranged in row matrices is that these objects are *gradients* of the ♡ function [compare with (5.17)]. The matrix

$$\left[\widetilde{J}\right] = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} , \tag{6.33}$$

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 equation (6.16). The question is how to evaluate the partial derivatives of the type $\partial\xi/\partial x$, since the inverse of the map (6.16) is not known (at least not in general). If we start the chain rule from the other end (switching the role of the variables), we obtain

$$\left[\frac{\partial \heartsuit}{\partial \xi}, \frac{\partial \heartsuit}{\partial \eta}\right] = \left[\frac{\partial \heartsuit}{\partial x}, \frac{\partial \heartsuit}{\partial y}\right] \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \tag{6.34}$$

and inverting the Jacobian matrix in equation (6.32) we get

$$\left[\frac{\partial \heartsuit}{\partial \xi}, \frac{\partial \heartsuit}{\partial \eta}\right] = \left[\frac{\partial \heartsuit}{\partial x}, \frac{\partial \heartsuit}{\partial y}\right] \left[\widetilde{J}\right]^{-1} . \tag{6.35}$$

Comparing (6.34) and (6.35) yields

$$[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \left[\widetilde{J}\right]^{-1} , \tag{6.36}$$

where $[J]$ is the ***Jacobian matrix*** of the map (6.16). The elements of $[J]$ are directly available from the matrix in (6.16). However, even more useful is to start from (6.15), and by definition the Jacobian matrix is then

$$[J] = \begin{bmatrix} \displaystyle\sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} x_i \ , \ \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} x_i \\ \displaystyle\sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} y_i \ , \ \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} y_i \end{bmatrix} , \tag{6.37}$$

Note that the Jacobian matrix may be expressed as the product of two matrices:

$$[J] = [\mathtt{x}]^T [\mathtt{Nder}] , \tag{6.38}$$

where [x] collects the coordinates of the nodes (three nodes, for the triangle)

$$[\mathtt{x}] = \begin{bmatrix} x_1 \ , y_1 \\ x_2 \ , y_2 \\ x_3 \ , y_3 \end{bmatrix} \ , \tag{6.39}$$

and [Nder] collects in each row the gradient of the basis function with respect to the parametric coordinates

$$[\mathtt{Nder}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} \ , \dfrac{\partial N_1}{\partial \eta} \\ \dfrac{\partial N_2}{\partial \xi} \ , \dfrac{\partial N_2}{\partial \eta} \\ \dfrac{\partial N_3}{\partial \xi} \ , \dfrac{\partial N_3}{\partial \eta} \end{bmatrix} \ , \tag{6.40}$$

The calculation of the spatial derivatives by an isoparametric geometric cell (recall that finite elements in SOFEA represent the calculation of basis functions and their derivatives in the gcell class) is a straightforward rewrite of the formulas above. The method Ndermat_spatial takes three arguments: a descendent of the class gcell, and the two arrays (6.40) and (6.39). The dimensions of the two arrays are (line 0013): nbfuns= number of basis functions (= 3 for the triangle), and dim= number of space dimensions (= 2 for the triangle).

```
0013 function Ndersp = Ndermat_spatial[3] (self, Nder, x)
0014     [nbfuns,dim] = size(Nder);
0015     if (size(Nder) ~= size(x))
0016         error('Wrong dimensions of arguments!');
0017     end
```

The Matlab code on line 0018 reflects literally the formula (6.38).

```
0018     J = x' * Nder;% Compute the Jacobian matrix
0019     detJ = det(J);% Compute the Jacobian
```

The Jacobian (determinant of the Jacobian matrix) should be positive. An error is reported when the Jacobian is non-positive; the generic case is treated in line 0023, which literally transcribes equation (6.32).

```
0020     if (detJ <= 0) % trouble
0021         error('Non-positive Jacobian');
0022     end
0023     Ndersp = Nder * inv(J);
0024 end
```

To round off the discussion in this section, we need to present the code that evaluates the basis functions (6.12–6.14) and the derivatives of the basis functions with respect to the parametric coordinates ξ, η. For the linear triangle T3 (class

[3] Folder: SOFEA/classes/gcell/@gcell

`gcell_T3`) the two methods are delightfully simple: method `Nmat` computes a column array of basis function values, N_j in row j, given the parametric coordinates $\xi \leftarrow$ `param_coords(1)`, $\eta \leftarrow$ `param_coords(2)`.

```
0008 function val = Nmat[4](self,param_coords)
0009     val = [(1 - param_coords(1) - param_coords(2));...
0010         param_coords(1); ...
0011         param_coords(2)];
0012     return;
0013 end
```

The method `Nder_param` returns an array with three rows (one for each basis function), with the gradient of the basis function j with respect to ξ, η in row j.

```
0010 function val = Nder_param[5](self, param_coords)
0011     val = [-1 -1; ...
0012             +1  0; ...
0013              0 +1];
0014     return;
0015 end
```

6.7 Numerical integration

Stepping closely along the lines of the discussion in Section 2.8, we formulate the numerical integration procedure for the linear triangle. We begin by highlighting the role of the Jacobian matrix.

Consider a map from ξ, η to x, y: a slight generalization of (6.15) in that the map is not necessarily linear (see Figure 6.9)

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{bmatrix} . \tag{6.41}$$

The parallelogram (rectangle) generated by the vectors $[\mathrm{d}\xi, 0]^T$ and $[0, \mathrm{d}\eta]^T$ (given in components in the Cartesian coordinate system ξ, η), has the area of ($\times$ is the cross product symbol)

$$\begin{bmatrix} \mathrm{d}\xi \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathrm{d}\eta \end{bmatrix} = \mathrm{d}\xi\mathrm{d}\eta \ .$$

Remember, this is happening in two dimensions: the cross product is a scalar.

The two vectors $[\mathrm{d}\xi, 0]^T$ and $[0, \mathrm{d}\eta]^T$ are mapped by the map (6.41) to vectors

$$\begin{bmatrix} \mathrm{d}\xi \\ 0 \end{bmatrix} \longrightarrow \mathrm{d}\xi \begin{bmatrix} \dfrac{\partial x}{\partial \xi} \\ \dfrac{\partial y}{\partial \xi} \end{bmatrix} , \quad \begin{bmatrix} 0 \\ \mathrm{d}\eta \end{bmatrix} \longrightarrow \mathrm{d}\eta \begin{bmatrix} \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \eta} \end{bmatrix} , \tag{6.42}$$

[4] Folder: `SOFEA/classes/gcell/@gcell_T3`
[5] Folder: `SOFEA/classes/gcell/@gcell_T3`

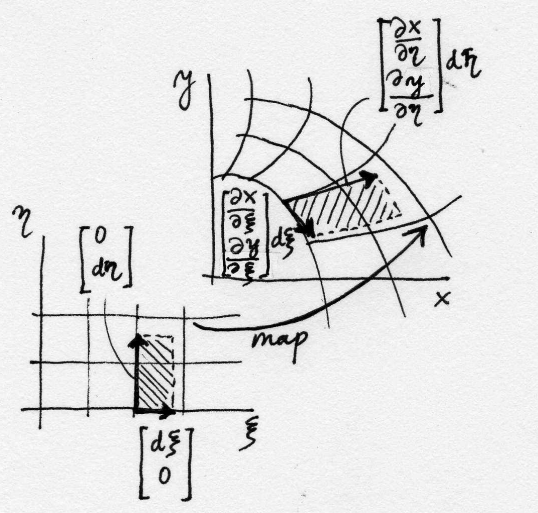

Fig. 6.9. Mapping of areas for a general map between coordinates

where the square brackets hold components in the standard Cartesian basis. Note that these vectors are ***tangent*** to the coordinate curves, which consist of the points in the physical space x, y that are maps of the curves $\xi = \text{const}$ and $\eta = \text{const}$. The area of the hatched parallelogram in Figure 6.9 is

$$\mathrm{d}\xi \begin{bmatrix} \dfrac{\partial x}{\partial \xi} \\ \dfrac{\partial y}{\partial \xi} \end{bmatrix} \times \mathrm{d}\eta \begin{bmatrix} \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \mathrm{d}\xi\mathrm{d}\eta \begin{bmatrix} \dfrac{\partial x}{\partial \xi} \\ \dfrac{\partial y}{\partial \xi} \end{bmatrix} \times \begin{bmatrix} \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \eta} \end{bmatrix} \quad . \tag{6.43}$$

Compare this equation with (6.36): the two vectors in the cross product are the columns of the Jacobian matrix from (6.36). In fact, the cross product of the columns is the determinant of the Jacobian matrix (or, as the determinant is known, the Jacobian). Therefore, the map (6.41) maps areas as

$$\mathrm{d}\xi\mathrm{d}\eta \longrightarrow \mathrm{d}\xi\mathrm{d}\eta \det[J] \quad . \tag{6.44}$$

As a consequence of (6.44), we have the following change of coordinates in integrals:

$$\int_{S_{[x,y]}} f(x,y)\mathrm{d}x\mathrm{d}y = \int_{S_{[\xi,\eta]}} f(\xi,\eta) \det[J(\xi,\eta)]\, \mathrm{d}\xi\mathrm{d}\eta \ . \tag{6.45}$$

Numerical quadrature rules take advantage of the relative ease with which these rules may be formulated on standard shapes, triangles, squares, cubes, etc. Thus, the integral on the left of (6.45) will be approximated as

$$\int_{S_{[x,y]}} f(x,y)\mathrm{d}x\mathrm{d}y \approx \sum_{k=1}^{M} f(\xi_k,\eta_k) \det[J(\xi_k,\eta_k)]\, W_k \ . \tag{6.46}$$

In the `SOFEA` toolbox, the surface Jacobian $\det[J(\xi,\eta)]$ is computed for two-dimensional manifold geometric cells by the method `Jacobian_surface`; it is discussed in Section 9.4.

We will introduce three integration rules for the standard triangle, one-point, three-point, and six-point quadrature, but many other rules are available: a number of authors have compiled tables, see for instance Hughes' book [Hug00]. The 1-point rule will be able to integrate linear polynomials in ξ,η exactly, and the 3-point does the job for up to quadratic polynomials in ξ,η. The six-point rule is good for fourth order polynomials, which may seem an overkill for applications with linear triangles, but its worth may be appreciated later. Table 6.1 gives the coordinates of the integration points, and their weights.

Rule	Coordinates ξ_j,η_j	Weights W_j	Integrates exactly
1-point	1/3, 1/3	1/2	linear polynomial
3-point	2/3, 1/6	1/6	quadratic polyn.
	1/6, 2/3	1/6	
	1/6, 1/6	1/6	
6-point	0.816847572980459, 0.091576213509771	0.109951743655322	quartic polyn.
	0.091576213509771, 0.816847572980459	0.109951743655322	
	0.091576213509771, 0.091576213509771	0.109951743655322	
	0.108103018168070, 0.445948490915965	0.223381589678011	
	0.445948490915965, 0.108103018168070	0.223381589678011	
	0.108103018168070, 0.108103018168070	0.223381589678011	

Table 6.1. Numerical integration rules on the standard triangle

6.8 Conductivity matrix

As already discussed in Chapter 3, all the problem dependent code is concentrated in a descendent of the `feblock` class. In particular, the two-dimensional heat diffusion model of this chapter is implemented in the `feblock_diffusion` finite element block class.

The conductivity (6.23) and other matrices are computed by evaluating the contributions from each element separately, storing these contributions element-by-element in a cell array, and then finally assembling all the element contributions into the overall system matrix. Therefore, the conductivity matrix would be computed element-by-element as

$$K_{ij} = \sum_e \int_{\triangle_e} (\mathrm{grad} N_j)\, \boldsymbol{\kappa} (\mathrm{grad} N_i)^T \, \Delta z \, \mathrm{d}S \,. \tag{6.47}$$

The procedure is illustrated in Figure 6.10: Only the gradients of the basis functions A, B, C are nonzero over the triangle e. The conductivity matrix *of this element* is calculated as if there were only three nodes in the whole mesh, all with free degrees of freedom (temperatures). It is therefore a 3×3 matrix K_e. However, only the unknown

degrees of freedom are given equation numbers on the global level. In this example, we assume that node A carries the unknown number 13, node C carries the unknown 61, and node B is associated with prescribed temperature, and correspondingly a zero (0) indicates that there is no equation for node B. The K_e matrix should be added to the global matrix as indicated in the sum (6.47), and the arrows pointing towards the appropriate elements ((13, 13), (13, 61), (61, 13), (61, 61)) of the global conductivity matrix K indicate the so-called ***assembly*** of the element matrix. This assembly process is executed in the `assemble`[6] method of the classes `dense_sysmat` and `sparse_sysmat`.

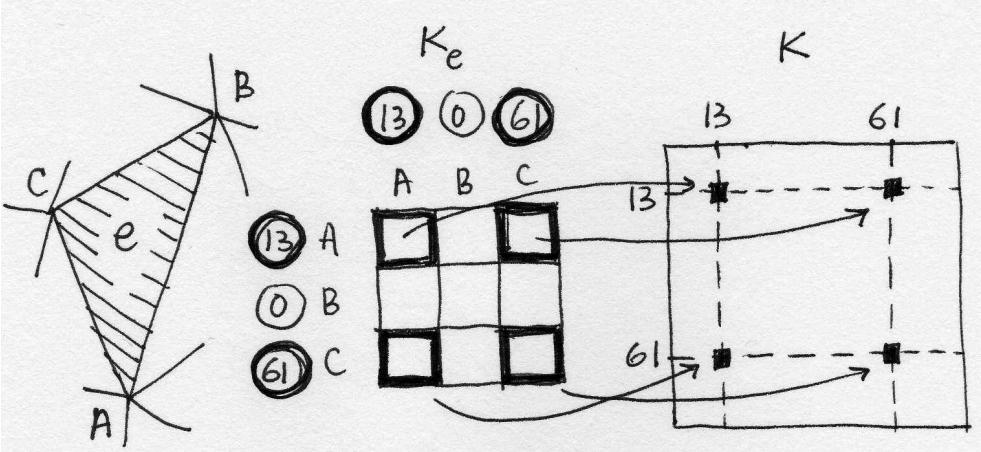

Fig. 6.10. Assembly of the element conductivity matrix

The method `conductivity` returns the array `ems` of element matrix objects (class `elemat`), each of which represents the conductivity matrix of a single element. The method begins by retrieving some information from the parent class, such as `gcells` (cell array of the geometric cells), `integration_rule`, and the material `mat`.

```
0009 function ems = conductivity[7](self, geom, theta)
0010     gcells = get(self.feblock,'gcells');
0011     ngcells = length(gcells);
0012     nfens = get(gcells(1),'nfens');
0013     dim = get(geom,'dim');
0014     % Pre-allocate the element matrices
0015     ems(1:ngcells) = deal(elemat);
0016     % Integration rule
0017     integration_rule = get(self.feblock, 'integration_rule');
0018     pc = get(integration_rule, 'param_coords');
0019     w  = get(integration_rule, 'weights');
0020     npts_per_gcell = get(integration_rule, 'npts');
```

[6] Folder: `SOFEA/classes/sysmat/@dense_sysmat`
[7] Folder: `SOFEA/classes/feblock/@feblock_diffusion`

```
0021      % Material
0022      mat = get(self.feblock, 'mater');
0023      kappa = get(get(mat,'property'),'conductivity');
```

The loop over all the geometric cells starts with the retrieval of the connectivity (i.e. the numbers of the nodes which are connected together by the cell), and of the array of the node coordinates, `x` (compare with (6.39)). Then, the element conductivity matrix `Ke` is initialized to zero, and the loop over all the quadrature points may begin.

```
0024      % Now loop over all gcells in the block
0025      for i=1:ngcells
0026          conn = get(gcells(i), 'conn'); % connectivity
0027          x = gather(geom,conn,'values','noreshape');% node coord
0028          Ke = zeros(nfens); % element matrix
```

The loop over the integration points starts with calculation of the gradients of the basis functions with respect to the parametric coordinates, see array (6.40). Then compute the spatial derivatives of the basis functions, `Nspder`, and the Jacobian, `detJ`. Note that the method used, `Jacobian_volume`, computes a *volume* Jacobian: even though the method works with representation of temperatures as functions of two variables, x, y, the problem that is being solved is still the heat conduction through a three-dimensional solid (there is no such thing as a two-dimensional body for an engineer; only mathematicians seem to know where to find them).

```
0029          % Loop over all integration points
0030          for j=1:npts_per_gcell
0031              N = Nmat (gcells(i), pc(j,:));
0032              Nder = Ndermat_param (gcells(i), pc(j,:));
0033              Nspder = Ndermat_spatial(gcells(i), Nder, x);
0034              detJ = Jacobian_volume(gcells(i),pc(j,:),x);
```

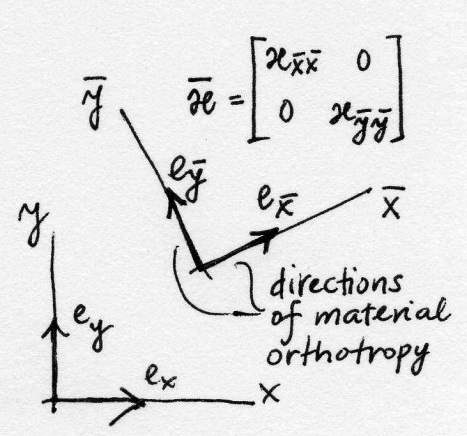

Fig. 6.11. Directions of material orthotropy

Often for orthotropic materials the axes of orthotropy vary from point-to-point. In that case it makes sense to describe the material properties in local Cartesian coordinates, and then allow the finite element block to define a transformation matrix between the local coordinate directions and the global Cartesian basis: refer to Figure 6.11. The attribute of a material parameter is thus the material conductivity in the rotated basis, $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$

$$[\overline{\kappa}] = \begin{bmatrix} \kappa_{\overline{x}} & 0 \\ 0 & \kappa_{\overline{y}} \end{bmatrix} \tag{6.48}$$

which is rotated using the transformation matrix (rotation matrix)

$$[R_m] = \left[[\boldsymbol{e}_{\overline{x}}] \, [\boldsymbol{e}_{\overline{y}}] \right] \tag{6.49}$$

whose columns are the components of the basis vectors $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$ in the global Cartesian coordinates. in The material conductivity matrix in the global basis is then expressed using the ordinary transformation rule

$$[\kappa] = [R_m][\overline{\kappa}][R_m]^T \tag{6.50}$$

Compute the local material directions.

```
0035              Rm = material_directions(self,gcells(i),pc(j,:),x);
```

Now exercise the integration rule. Note that the element conductivity matrix may be computed as a whole, as opposed to element-by-element, since the gradients of the basis functions are arranged as rows of the `Nspder` matrix.

```
0036              Ke = Ke + Nspder*Rm*kappa*Rm'*Nspder' * detJ * w(j);
```

Finally, the computed element conductivity matrix `Ke` is stored in the `ems(i)` object of class `elemat`, both the matrix itself and the equation numbers that go with each column and each row (compare with Figure 6.10: 13, 0, 61; zero means "no equation").

```
0037          end
0038          ems(i)=set(ems(i), 'mat', Ke);
0039          ems(i)=set(ems(i), 'eqnums',gather(temp,conn,'eqnums'));
0040      end
0041      return;
0042 end
```

Since the topic of the Jacobians has been brought up, we point out how the volume Jacobian is computed for the triangle element. We ignore everything that does not pertain to the present case, and we focus on line `0021`: the volume Jacobian is computed as the product of the surface Jacobian (determinant of (6.36)) and the "other dimension" (thickness).

```
0015 function detJ = Jacobian_volume[8](self, pc, x)
0016  ...
0021          detJ=Jacobian_surface(self,pc,x)*other_dimension(self,pc,x);
0022      end
0023 end
```

[8] Folder: `SOFEA/classes/gcell/@gcell_2_manifold`

6.9 Surface heat transfer matrix and load

While in the preceding section the required integrals were over the volume of the domain and executed over the area S_c (which together with a thickness gives volume), the surface heat transfer matrix (6.28) and the surface heats transfer load (6.27) (and also the prescribed heat flux load (6.26)) require integration over the bounding surface, evaluated over the curves, $C_{c,3}$ (or $C_{c,2}$), which together with a thickness gives a surface area. Since the volume integrals are being performed over the area of the triangles in the mesh, the surface integrals will be evaluated over the edges of these triangles.

Evaluating the basis functions (6.12–6.14) along the edges of the standard triangle, we may observe that the basis function associated with the opposite vertex is identically zero, and the other two at the end-points of the edge vary linearly along the edge. In fact, completely in agreement with the basis functions (2.20) defined on the line element L2. Therefore, integrating an expression along the edge of the triangle T3 that connects nodes i, j yields exactly the same result as integrating along the line element L2 that connects nodes i, j. However, the two approaches are not the same thing: if, for the purpose of numerical integration, we use the element L2, the design of the numerical integration code will be reusable: the same piece of code may be used to integrate quantities along a curve which is tiled with finite element edges with linear variation of basis functions – the triangle T3, the quadrilateral Q4, the hexahedron H8, the tetrahedron T4.

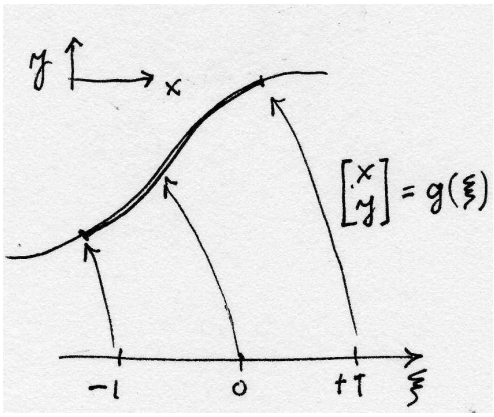

Fig. 6.12. Mapping of the standard interval to a Cartesian space

Let us take for instance equation (6.26). The goal is to evaluate an integral of the form

$$\int_C f(\boldsymbol{p})\,\mathrm{d}C \tag{6.51}$$

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 on the

curve $\boldsymbol{p}$ will have appropriate number of components. 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 the standard interval $-1 \leq \xi \leq +1$ (compare with the 1-D map (2.23), and refer to Figure 6.12, where the map is two-dimensional)

$$\boldsymbol{p} = \boldsymbol{g}(\xi) \ . \tag{6.52}$$

For two closely spaced points on the curve, $\boldsymbol{p}(\xi)$ and $\boldsymbol{p}(\xi + \Delta\xi)$, where $\Delta\xi$ is the distance between 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

$$\boldsymbol{p}(\xi + \Delta\xi) = \boldsymbol{p}(\xi) + \frac{\partial \boldsymbol{p}(\xi + \varepsilon\Delta\xi)}{\partial \xi}\Delta\xi \qquad 0 \leq \varepsilon \leq 1. \tag{6.53}$$

The two points may be connected with a vector approximately tracking the curve (see Figure 6.13),

$$\boldsymbol{p}(\xi + \Delta\xi) - \boldsymbol{p}(\xi) = \frac{\partial \boldsymbol{p}(\xi + \varepsilon\Delta\xi)}{\partial \xi}\Delta\xi \ ,$$

whose length (squared) is

$$(\Delta C)^2 = \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon\Delta\xi)}{\partial \xi}\Delta\xi\right) \cdot \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon\Delta\xi)}{\partial \xi}\Delta\xi\right) = \left|\left|\frac{\partial \boldsymbol{p}(\xi + \varepsilon\Delta\xi)}{\partial \xi}\right|\right|^2 (\Delta\xi)^2 \ .$$

Skipping over the details, we may conclude that for infinitesimally short intervals, $\Delta\xi \to \mathrm{d}\xi$, the following relationship is obtained

$$\mathrm{d}C = \left|\left|\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi}\right|\right| \mathrm{d}\xi \ , \tag{6.54}$$

where $\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi}$ is the vector ***tangent*** to the curve at ξ, and $\left|\left|\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi}\right|\right|$ is the Jacobian to be used in the change-of-variables device for the curve integral.

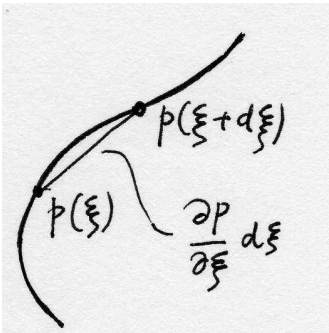

Fig. 6.13. Length of a curve

Let us now specialize these developments to the L2 element: the map (6.52) reads

$$\boldsymbol{p} = \boldsymbol{g}(\xi) = \sum_{i=1}^{2} N_i(\xi)\boldsymbol{x}_i \ . \tag{6.55}$$

where N_i are given by (2.25). Therefore, the tangent vector (see (6.54)) reads

$$\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} = \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{2}$$

and the Jacobian is $h/2$, where $h = ||\boldsymbol{x}_2 - \boldsymbol{x}_1||$ is the length of the element.

Of course, for general elements with n nodes, the implementation computes the tangent as

$$\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} = \texttt{x'*Nder} \ , \tag{6.56}$$

using the following two matrices ,

$$[\texttt{x}] = \begin{bmatrix} x_1 \ , \ y_1 \\ x_2 \ , \ y_2 \\ \ldots, \ \ldots \\ x_n \ , \ y_n \end{bmatrix} \ , \tag{6.57}$$

where the number of columns is equal to the number of spatial dimensions, 1, 2 (which is assumed in (6.57)), or 3, and [Nder] collects in each row the gradient of the basis function with respect to the parametric coordinate

$$[\texttt{Nder}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} \\ \dfrac{\partial N_2}{\partial \xi} \\ \ldots \\ \dfrac{\partial N_n}{\partial \xi} \end{bmatrix} \ , \tag{6.58}$$

These matrices should be compared with those defined for the triangle T3, equations (6.39) and (6.40). The only difference is the number of space dimensions, the number of basis functions, and the number of parametric dimensions; all of these are taken account of by the Matlab code automatically. For the one-dimensional manifold geometric cells, this computation is implemented in the method `Jacobian_curve` of the class `gcell_1_manifold`. Compare formula (6.56) with line 0015.

```
0013 function detJ = Jacobian_curve[9](self, pc, x)
0014     Nder = Ndermat_param (self, pc);
0015     tangents =x'*Nder;
0016     [sdim, ntan] = size(tangents);
```

[9] Folder: `SOFEA/classes/gcell/@gcell_1_manifold`

```
0017      if      ntan==1 % 1-D gcell
0018          detJ = norm(tangents);
0019      else
0020          error('Got an incorrect size of tangents');
0021      end
0022  end
```

The method `surface_transfer` computes the heat transfer matrix (6.28) by integrating over the appropriate part of the surface. Note that the surface needs to be discretized by compatible geometric cells: when the domain (volume) is covered by three-node triangles, the boundary (surface) is covered by two-node line segments, and so on. Most of the preparation steps are straightforward, and are therefore omitted. The loop over the geometric cells computes the heat transfer matrix for each element. The Jacobian is computed in a way that is appropriate for the geometric cells (yet another example of dynamic method dispatch): the method `Jacobian_surface`.

```
0009 function ems = surface_transfer[10](self, geom, temp)
...
0021      h = self.surface_transfer;
0022      % Now loop over all gcells in the block
0023      for i=1:ngcells
0024          conn = get(gcells(i), 'conn'); % connectivity
0025          x = gather(geom, conn, 'values', 'noreshape'); % coord
0026          He = zeros(nfens); % element matrix
0027          for j=1:npts_per_gcell % Loop over integration points
0028              N = Nmat(gcells(i), pc(j,:));
0029              detJ = Jacobian_surface(gcells(i),pc(j,:),x);
0030              He = He + h*N*N' * detJ * w(j);
0031          end
0032          ems(i)=set(ems(i),'mat', He);
0033          ems(i)=set(ems(i),'eqnums',gather(temp,conn,'eqnums'));
0034      end
0035      return;
0036  end
```

The surface Jacobian is computed for the L2 (line) element following the same principle as before for the volume Jacobian for the element T3. Focusing just on this case, we see that the surface Jacobian is computed as the product of the curve Jacobian and the "other dimension" (thickness).

```
0015 function detJ = Jacobian_surface[11](self, pc, x)
0016   ...
0021          detJ=Jacobian_curve(self,pc,x)*other_dimension(self,pc,x);
0022      end
```

[10] Folder: `SOFEA/classes/feblock/@feblock_diffusion`
[11] Folder: `SOFEA/classes/gcell/@gcell_1_manifold`

```
0023 end
```

Computation of the surface transfer loads could duplicate the work done in the previous method (the same kind of integral), but in order to avoid this duplication, we use the following trick: compute an array of the element heat surface transfer matrices, and multiply by the vector of ambient temperatures (whenever they are nonzero). The method `surface_transfer_loads` is therefore quite straightforward: compute the element surface heat transfer matrices (line `0010`), then loop over the geometric cells and retrieve the ambient temperatures from the field `amb`. Provided this vector is nonzero, the product `He*pT` is stored in the element vector object.

```
0009 function evs = surface_transfer_loads[12] (self, geom, temp, amb)
0010     ems = surface_transfer(self, geom, temp);
0011     gcells = get(self.feblock,'gcells');
0012     % Pre-allocate the element matrices
0013     evs(1:length(gcells)) = deal(elevec);
0014     % Now loop over all gcells in the block
0015     for i=1:length(gcells)
0016         conn = get(gcells(i), 'conn'); % connectivity
0017         pT = gather(amb, conn, 'prescribed_values');
0018         if norm (pT) ~= 0
0019             He =  get(ems(i),'mat'); % element matrix
0020             evs(i) = set(evs(i), 'vec', He*pT);
0021             evs(i) = set(evs(i), 'eqnums', ...
0022                 gather(temp, conn, 'eqnums'));
0023         end
0024     end
0025     return;
0026 end
```

[12] Folder: `SOFEA/classes/feblock/@feblock_diffusion`

7

Steady-state heat diffusion solutions

7.1 Steady-state diffusion equation

The ordinary differential equations that result from discretization in space, equations (6.29), lead to steady-state solutions when $\partial T_i(t)/\partial t = 0$, and $\partial \overline{T}_i(t)/\partial t = 0$. The latter are a sine qua non condition, while the former follows when all the transients in the solution decay (in infinite time, in general). Substituting into (6.29), we obtain

$$\sum_{\text{free } i} K_{ji} T_i + \sum_{\text{free } i} H_{ji} T_i = - \sum_{\text{prescribed } i} \overline{K}_{ji} \overline{T}_i + L_{Q,j} + L_{q2,j} + L_{q3,j} \quad \forall \text{ free } j, \quad (7.1)$$

which is a system of linear equations for the unknown nodal temperatures. The nodal temperatures are now just numbers, not functions.

7.2 Thick-walled tube

The first example is a thick-walled rectangular tube, with the outside temperature being prescribed as zero, and the interior surface (perfectly) insulated. The material is isotropic. As shown in Figure 7.1, the planes of symmetry may be used to reduce the size of the problem. Therefore, only one quarter is discretized, and perfect insulation is applied at the symmetry planes (no heat flow through the symmetry planes). There is a distributed heat source in the material (for instance, heat released by curing cement paste).

The Matlab script is `lshape1`[1]. The first few lines define some ancillary variables, and then the two-dimensional mesh generator of triangle meshes is invoked. The generator is thoroughly described in the user's guide `targe2_users_guide.pdf`[2], but we will describe a little bit the Matlab interface. The first argument is a cell array, each element a string (character array), with one command for the mesh generator. Thus, the first six strings define curves that bound the domain (straight-line segments), the line `0011` defines a subregion (chunk of area to be covered with triangles), and the last

[1] Folder: `SOFEA/examples/heat_diffusion`
[2] Folder: `SOFEA/meshes/targe2`

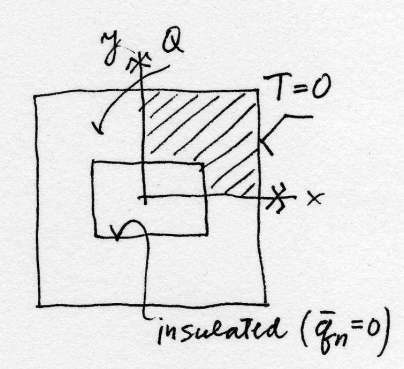

Fig. 7.1. Heat diffusion in a thick-walled rectangular tube

line defines the mesh size. The second argument to `targe2_mesher` is the thickness of the slab (default value of 1.0).

```
0001 kappa=[0.2 0; 0 0.2]; % conductivity matrix
0002 Q=0.01100; % uniform heat source
0003 num_integ_pts=1; % 1-point quadrature
0004 [fens,gcells] = targe2_mesher({...textcolorcomment
0005         ['curve 1 line 20 0 48 0'],...
0006         ['curve 2 line 48 0 48 48'],...
0007         ['curve 3 line 48 48 0 48'],...
0008         ['curve 4 line 0 48 0 13'],...
0009         ['curve 5 line 0 13 20 13'],...
0010         ['curve 6 line 20 13 20 0'],...
0011         'subregion 1  property 1 boundary 1 2 3 4 5 6',...
0012         ['m-ctl-point constant 3.5']
0013         }, 1.0);
```

Next, the property object appropriate for heat diffusion is created, `property_diffusion`, and supplied the material conductivity matrix $\boldsymbol{\kappa}$, and the heat source Q. The material object acts as a mediator between the property object in the finite element block. The finite element block of class `feblock_diffusion` is created, with attributes: the material, the array of geometric cells, and an integration rule (class `tri_rule` is used for triangles).

```
0014 prop=property_diffusion(struct('conductivity',kappa,'source',Q));
0015 mater=mater_diffusion (struct('property',prop));
0016 feb = feblock_diffusion (struct ('mater',mater,...
0017     'gcells',gcells,...
0018     'integration_rule',tri_rule(num_integ_pts)));
```

Two fields are created: `geom` represents the geometry (i.e. the locations of the nodes), and it is therefore initialized from the finite element node array, `fens`; and `theta`

represents the temperatures at the nodes, and it is initially undefined, except for the number of nodes `nfens`.

```
0019 geom = field(struct('name',['geom'], 'dim', 2, 'fens',fens));
0020 theta=field(struct('name',['theta'], 'dim', 1, 'nfens',...
0021     get(geom,'nfens')));
```

The essential boundary conditions are next applied to the temperature field. The utility function `fenode_select` is used to select nodes from the `fens` array based on their location: nodes which fall into given bounding boxes are selected ($[x_{lo}\ x_{hi}\ y_{lo}\ y_{hi}]$ =`[48 48 0 48]` and so on for the other box); to avoid problems with precision, the boxes are for the purpose of the "in"-test inflated by 0.01. The array `prescribed` is filled with ones to indicate that all degrees of freedom are to be prescribed, the components to be prescribed are passed as an empty array (line 0026), which simply means all components are meant. The values to which the temperatures are being prescribed are all zeros. The data defining the essential boundary conditions are set in the field, and then applied (line 0029). The free node parameters are then assigned global equation numbers.

```
0022 fenids=[fenode_select(fens,struct('box',[48 48 0 48],...
0023     'inflate', 0.01)),...
0024     fenode_select(fens,struct('box',[0 48 48 48],...
0025     'inflate', 0.01))];
0026 prescribed=ones(length(fenids),1);
0027 comp=[];
0028 val=zeros(length(fenids),1);
0029 theta = set_ebc(theta, fenids, prescribed, comp, val);
0030 theta = apply_ebc (theta);
0031 theta = numbereqns (theta);
```

The conductivity matrix is sparse (the linear system to be solved is going to be moderately large, and the efficiency afforded by a sparse matrix is not to be sneezed at), and it is assembled from element conductivity matrices in line 0032. The heat load vector is assembled from element load vectors, and the solution of the linear system of equations is scattered into the `theta` field.

```
0032 K = start (sparse_sysmat, get(theta, 'neqns'));
0033 K = assemble (K, conductivity(feb, geom, theta));
0034 F = start (sysvec, get(theta, 'neqns'));
0035 F = assemble (F, source_loads(feb, geom, theta));
0036 theta = scatter_sysvec(theta, get(K,'mat')\get(F,'vec'));
```

The last fragment of code takes care of the graphic presentation of the results. The field `colorfield` holds one color (a triple of floating-point numbers) per node, and those colors are obtained from the temperature field by mapping node temperatures to colors (line 0042). The geometric cells of individual finite elements are plotted twice. Once as a raised colored surface (line 0047), and the second time as a wireframe in the x, y plane (line 0049). The resulting graphic is shown in Figure 7.2.

```
0038 gv=graphic_viewer;
0039 gv=reset (gv,[]);
0040 T=get(theta,'values');
0041 dcm=data_colormap(struct('range',[min(T),max(T)],'colormap',jet));
0042 colorfield=field(struct ('name', ['colorfield'], 'data',...
0043     map_data(dcm, T)));
0044 geomT=field(struct ('name', ['geomT'], ...
0045     'data',[get(geom,'values'), get(theta,'values')]));
0046 for i=1:length (gcells)
0047     draw(gcells(i), gv, struct ('x',geomT, 'u',0*geomT,...
0048         'colorfield',colorfield, 'shrink',0.9));
0049     draw(gcells(i), gv, struct ('x',geom, 'u',0*geom, ...
0050         'facecolor','none'));
0051 end
```

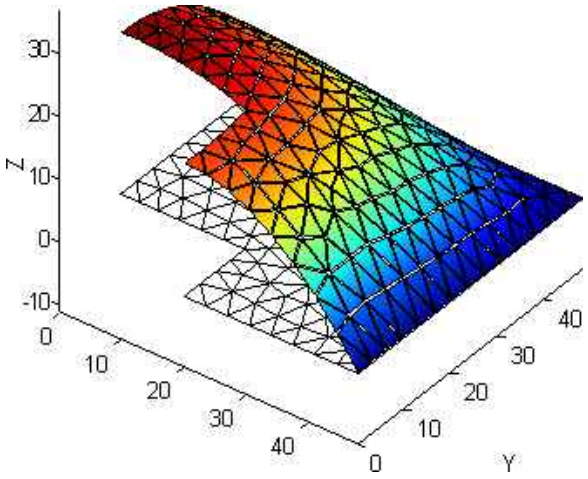

Fig. 7.2. Heat diffusion in a thick-walled rectangular tube: graphic presentation of results

7.3 Orthotropic insert

The next example introduces nonzero essential boundary conditions, and orthotropic material properties. A square block of isotropic material is insulated on the vertical edges, and two different temperatures our applied on the horizontal edges. There is a square insert of orthotropic material within the larger square. The orientation of the material axes is indicated in Figure 7.3. Physically the insert could be made of parallel fibers (for instance carbon), embedded in a polymer matrix. The fibers conduct heat well, while in the transverse direction the polymer matrix hampers heat conduction. The problem is solved with script `squareinsquare`[3]. The domain consists

[3] Folder: `SOFEA/examples/heat_diffusion`

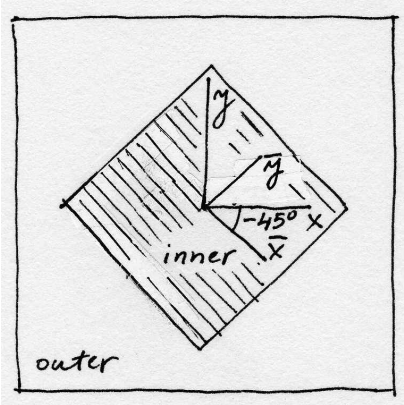

Fig. 7.3. Heat diffusion in inhomogeneous domain with orthotropic material properties

of two materials, and consequently we define two material conductivity matrices: the inner material has strongly orthotropic properties; the outer material is isotropic. The rotation matrix that defines the local material properties of the insert is setup in line 0005.

```
0001 kappainner=[2.25 0; 0 0.06]; % orthotropic conductivity matrix
0002 kappaouter=[0.25 0; 0 0.25]; % isotropic conductivity matrix
0003 alpha =-45;% local material orientation angle
0004 ca=cos(2*pi/360*alpha); sa=sin(2*pi/360*alpha);
0005 Rm = [ca, -sa;sa, ca];% local material directions
```

The mesh generator defines the eight boundary segments, and then sets up two subregions: note that the two subregions are assigned different numerical identifiers (1 and 2) to distinguish elements belonging to different subregions.

```
0007 [fens,gcells, groups] = targe2_mesher({...
0008         ['curve 1 line -48 -48 48 -48'],...
0009         ['curve 2 line 48 -48 48 48'],...
0010         ['curve 3 line 48 48 -48 48'],...
0011         ['curve 4 line -48 48 -48 -48'],...
0012         ['curve 5 line 0 -31 31 0'],...
0013         ['curve 6 line 31 0 0 31'],...
0014         ['curve 7 line 0 31 -31 0'],...
0015         ['curve 8 line -31 0 0 -31'],...
0016         ['subregion 1  property 1 ' ...
0017         '   boundary 1 2 3 4 -8 -7 -6 -5'],...
0018         ['subregion 2  property 2 '...
0019         '   boundary 5 6 7 8'],...
0020         ['m-ctl-point constant 4.75']
0021         }, 1.0);
```

The inner subregion consists of the geometric cells `gcells(groups{2})` (`groups{2}` is a list of indexes of the cells that belong to the subregion 2). Note that the local

material directions matrix is being supplied to the finite element block constructor (line 0027).

```
0022 propinner=property_diffusion(struct('conductivity',kappainner,...
0023     'source',0));
0024 materinner=mater_diffusion(struct('property',propinner));
0025 febinner = feblock_diffusion(struct ('mater',materinner,...
0026     'gcells',gcells(groups{2}),...
0027     'integration_rule',tri_rule(num_integ_pts),'Rm',Rm));
0028 propouter=property_diffusion(struct('conductivity',kappaouter,...
0029     'source',0));
0030 materouter=mater_diffusion(struct('property',propouter));
0031 febouter = feblock_diffusion(struct ('mater',materouter,...
0032     'gcells',gcells(groups{1}),...
0033     'integration_rule',tri_rule(num_integ_pts)));
```

The boundary conditions are straightforward, but notice that the two horizontal edges are being assigned different, nonzero, temperatures.

```
0036 fenids=[fenode_select(fens,struct('box',[-48 48 -48 -48],...
0037     'inflate', 0.01))];
0038 prescribed=ones(length(fenids),1);
0039 comp=[];
0040 val=zeros(length(fenids),1)+20;% ambient temperature
0041 theta = set_ebc(theta, fenids, prescribed, comp, val);
0042 fenids=[fenode_select(fens,struct('box',[-48 48 48 48],...
0043     'inflate', 0.01))];
0044 prescribed=ones(length(fenids),1);
0045 comp=[];
0046 val=zeros(length(fenids),1)+57;% hot inner surface
0047 theta = set_ebc(theta, fenids, prescribed, comp, val);
0048 theta = apply_ebc (theta);
```

When assembling the conductivity matrix, the contributions from the two blocks are assembled separately. The thermal loads corresponding to nonzero essential boundary conditions (conductivity only: recall that this is steady-state) are assembled only for the outer subregion block, since there are no boundary conditions on the boundary of the inner block.

```
0050 K = start (sparse_sysmat, get(theta, 'neqns'));
0051 K = assemble (K, conductivity(febinner, geom, theta));
0052 K = assemble (K, conductivity(febouter, geom, theta));
0053 F = start (sysvec, get(theta, 'neqns'));
0054 F = assemble (F, nz_ebc_loads_conductivity(febouter, geom, theta));
```

The results are presented in Figure 7.4. The distorting effect of the insert is noteworthy: in one direction the insert acts as a heat sink/source, in the perpendicular direction it is an insulator.

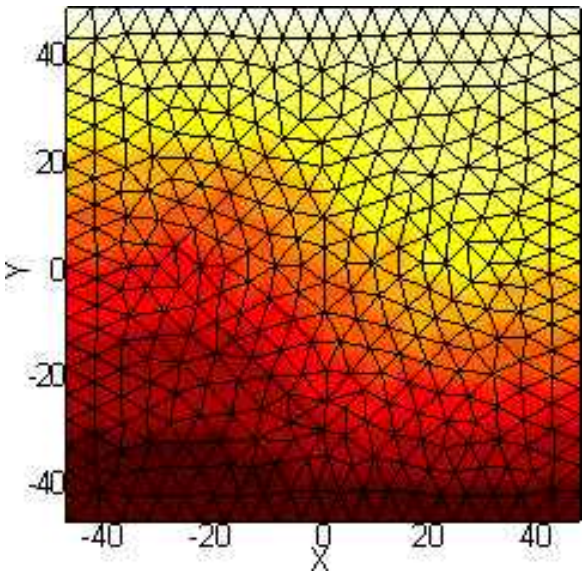

Fig. 7.4. Heat diffusion in inhomogeneous domain with orthotropic material properties: temperature distribution. Notice the distorting effect of the insert

7.4 The T4 NAFEMS Benchmark

This problem is one of the NAFEMS (National Agency for Finite Element Methods and Standards (UK)) benchmark tests for thermal analyses. It consists of 2d region 0.6 meter wide by 1 meter high, with a fixed temperature of 100°C100 on the lower boundary, perfect insulator on the left boundary, and a heat transfer at 750W/m^2°C on the other two boundaries (see Figure 7.5). The material in the region has a thermal conductivity of 52W/m°C. The problem is to calculate the steady-state temperature distribution. A complete description of this problem is given in the paper by Cameron, Casey, and Simpson [CCS].

The `SOFEA` solution is in the Matlab script `t4nafems`[4]. Note that also

```
0001 kappa=[52 0; 0 52]; % conductivity matrix
0002 Q=0.0; % uniform heat source
0003 h=750;
0004 num_integ_pts=1; % 1-point quadrature
0005 [fens,gcells,groups,edge_gcells,edge_groups]=targe2_mesher({...
0006         ['curve 1 line 0 0 0.6 0'],...
0007         ['curve 2 line 0.6 0 0.6 0.2'],...
0008         ['curve 3 line 0.6 0.2 0.6 1.0'],...
0009         ['curve 4 line 0.6 1.0 0 1.0'],...
0010         ['curve 5 line 0 1.0 0 0'],...
0011         'subregion 1  property 1 boundary 1 2 3 4 5',...
0012         ['m-ctl-point constant 0.05']
0013         }, 1.0);
```

[4]Folder: `SOFEA/examples/heat_diffusion`

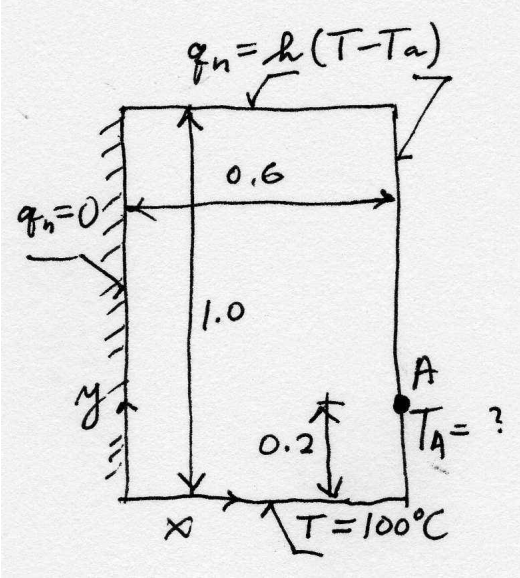

Fig. 7.5. The T4 NAFEMS benchmark geometry and boundary conditions.

...

Nota bene that two blocks are being created: the first for the triangular elements in the interior of the domain, and the second, `edgefeb`, for the edge elements (line segment elements with two nodes) along the two boundary edges of the domain with the convective boundary condition.

```
0019 edgefeb = feblock_diffusion (struct ('mater',mater,...
0020     'gcells',edge_gcells([edge_groups{[2, 3, 4]}]),...
0021     'integration_rule',gauss_rule(1,num_integ_pts),...
0022     'surface_transfer', h));
...
```

Create a field to represent the prescribed ambient temperature along the boundary. The interior values are never used, only the ones on the boundary. They happen to be all equal to zero, but we will not ignore them in the interest of clarity.

```
0026 amb = clone(theta, ['amb']);
0027 fenids=[
0028     fenode_select(fens,struct('box',[0.6 0.6 0 1],...
0029     'inflate', 0.01)),...
0030     fenode_select(fens,struct('box',[0 1 1 1],...
0031     'inflate', 0.01))]   ;
0032 prescribed=ones(length(fenids),1);
0033 comp=[];
0034 val=zeros(length(fenids),1)+0.0;
0035 amb = set_ebc(amb, fenids, prescribed, comp, val);
0036 amb = apply_ebc (amb);
```

The essential boundary condition on the temperature field is applied.

```
0037 fenids=[
0038     fenode_select(fens,struct('box',[0. 0.6 0 0],...
0039     'inflate', 0.01))]    ;
0040 prescribed=ones(length(fenids),1);
0041 comp=[];
0042 val=zeros(length(fenids),1)+100.0;
0043 theta = set_ebc(theta, fenids, prescribed, comp, val);
0044 theta = apply_ebc (theta);
```

The system matrix and the system load vector are assembled, including the surface heats transfer contribution (line 0047), and the surface heat transfer load (line 0051). Note that these are computed on the edge element block `edgefeb`. The contribution of the nonzero prescribed temperature is also added in.

```
0046 K = start (sparse_sysmat, get(theta, 'neqns'));
0047 K = assemble (K, conductivity(feb, geom, theta));
0048 K = assemble (K, surface_transfer(edgefeb, geom, theta));
0049 F = start (sysvec, get(theta, 'neqns'));
0050 F = assemble(F, source_loads(feb, geom, theta));
0051 F = assemble(F, nz_ebc_loads_conductivity(feb, geom, theta));
0052 F = assemble(F, surface_transfer_loads(edgefeb,geom,theta,amb));
```

After the solution, the temperature at the node $x = 0.6$, $y = 0.2$ (label A in Figure 7.6), is retrieved with `gather` and printed. The calculated value of 18.2481°C agrees well with the reference solution of 18.3°C. The singularity near the corner where the two kinds of boundary conditions meet (prescribed temperature with convective surface heat transfer) is clearly visible.

```
0053 theta = scatter_sysvec(theta, get(K,'mat')\get(F,'vec'));
0054 gather(theta,fenode_select(fens,...
0055     struct('box',[0.6 0.6 0.2 0.2],'inflate', 0.01)),'values')
```

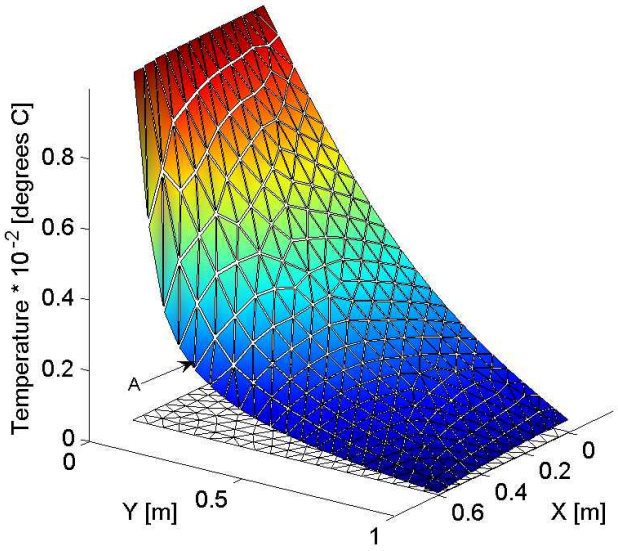

Fig. 7.6. Temperature distribution for the T4 NAFEMS benchmark.

8

Transient heat diffusion solutions

8.1 Discretization in time for transient heat diffusion

The ordinary differential equations (6.29) need to be numerically integrated in time as analytical solutions are not possible in general. Hughes (2000) describes a finite difference method, the ***generalized trapezoidal method*** , including its accuracy and stability properties (Chapter 8, Reference [Hug00]). To unclutter the equations, we will use a matrix notation, with the following definitions:

$$\widetilde{\boldsymbol{K}} = [K_{ji} + H_{ji}], \quad \text{free } j, i, \tag{8.1}$$

for the effective conductivity matrix, which bundles the bulk conductivity with the surface heat transfer matrix,

$$\overline{\boldsymbol{K}} = [\overline{K}_{ji}], \quad \text{free } j, \text{ prescribed } i, \tag{8.2}$$

for the rectangular conductivity matrix that relates the prescribed temperatures to the heat fluxes,

$$\boldsymbol{C} = [C_{ji}], \quad \text{free } j, i, \tag{8.3}$$

for the capacity matrix,

$$\overline{\boldsymbol{C}} = [\overline{C}_{ji}], \quad \text{free } j, \text{ prescribed } i, \tag{8.4}$$

for the rectangular capacity matrix that relates the prescribed temperature rates to the heat fluxes, and

$$\boldsymbol{L} = [L_{Q,j} + L_{q2,j} + L_{q3,j}], \quad \text{free } j \ . \tag{8.5}$$

The free temperatures and their rates are collected in column matrices

$$\boldsymbol{T} = [T_j], \quad \dot{\boldsymbol{T}} = \left[\frac{\partial T_i}{\partial t}\right], \quad \text{free } j \ . \tag{8.6}$$

and the prescribed temperatures and their rates

$$\overline{\boldsymbol{T}} = [\overline{T}_j], \quad \dot{\overline{\boldsymbol{T}}} = \left[\frac{\partial \overline{T}_i}{\partial t}\right], \quad \text{prescribed } j \ . \tag{8.7}$$

Therefore, equation (6.29) may be recast as

$$\boldsymbol{C}\dot{\boldsymbol{T}} + \widetilde{\boldsymbol{K}}\boldsymbol{T} + \overline{\boldsymbol{C}}\dot{\overline{\boldsymbol{T}}} + \overline{\boldsymbol{K}}\overline{\boldsymbol{T}} - \boldsymbol{L} = \boldsymbol{0} \ . \tag{8.8}$$

The generalized trapezoidal method proposes to express the relationship between the temperatures and the rates of temperatures at two different time instants, t_n and t_{n+1}, as

$$\theta\dot{\boldsymbol{T}}_{n+1} + (1-\theta)\dot{\boldsymbol{T}}_n = \frac{\boldsymbol{T}_{n+1} - \boldsymbol{T}_n}{\Delta t} \ , \tag{8.9}$$

where a quantity expressed at time t_n is given a subscript n, and $\Delta t = t_{n+1} - t_n$. The free parameter θ is used to control accuracy and stability of the scheme.

Equation (8.9) is applied to the time stepping of (8.8) by writing (8.8) at the two time instants, t_n and t_{n+1}, and then mixing together these two equations. Thus, we add together

$$\theta\left[\boldsymbol{C}\dot{\boldsymbol{T}}_{n+1} + \widetilde{\boldsymbol{K}}\boldsymbol{T}_{n+1} + \overline{\boldsymbol{C}}\dot{\overline{\boldsymbol{T}}}_{n+1} + \overline{\boldsymbol{K}}\overline{\boldsymbol{T}}_{n+1} - \boldsymbol{L}_{n+1}\right] = \boldsymbol{0} \ , \tag{8.10}$$

and

$$(1-\theta)\left[\boldsymbol{C}\dot{\boldsymbol{T}}_n + \widetilde{\boldsymbol{K}}\boldsymbol{T}_n + \overline{\boldsymbol{C}}\dot{\overline{\boldsymbol{T}}}_n + \overline{\boldsymbol{K}}\overline{\boldsymbol{T}}_n - \boldsymbol{L}_n\right] = \boldsymbol{0} \ , \tag{8.11}$$

and if we assume that equation (8.9) applies not only to the free temperatures, but also to the prescribed temperatures, the mixture of rates (left-hand side of (8.9)) may be replaced with the difference of the temperatures (right hand side of (8.9)). The resulting equation refers only to temperatures at two times, and may be solved to yield $\boldsymbol{T}_{n+1}$, provided $\boldsymbol{T}_n$ is known.

$$\begin{aligned}\left[\frac{1}{\Delta t}\boldsymbol{C} + \theta\widetilde{\boldsymbol{K}}\right]\boldsymbol{T}_{n+1} = &\left[\frac{1}{\Delta t}\boldsymbol{C} - (1-\theta)\widetilde{\boldsymbol{K}}\right]\boldsymbol{T}_n + \theta\boldsymbol{L}_{n+1} + (1-\theta)\boldsymbol{L}_n \\ &- \left[\frac{1}{\Delta t}\overline{\boldsymbol{C}} + \theta\overline{\boldsymbol{K}}\right]\overline{\boldsymbol{T}}_{n+1} + \left[\frac{1}{\Delta t}\overline{\boldsymbol{C}} - (1-\theta)\overline{\boldsymbol{K}}\right]\overline{\boldsymbol{T}}_n\end{aligned} \tag{8.12}$$

The form of equation (8.12) is pleasingly symmetric, fully reflective of the blocked nature of these equations. However, for implementation the following form is going to be more profitable:

$$\begin{aligned}\left[\frac{1}{\Delta t}\boldsymbol{C} + \theta\widetilde{\boldsymbol{K}}\right]\boldsymbol{T}_{n+1} = &\left[\frac{1}{\Delta t}\boldsymbol{C} - (1-\theta)\widetilde{\boldsymbol{K}}\right]\boldsymbol{T}_n + \theta\boldsymbol{L}_{n+1} + (1-\theta)\boldsymbol{L}_n \\ &- \overline{\boldsymbol{C}}\frac{\overline{\boldsymbol{T}}_{n+1} - \overline{\boldsymbol{T}}_n}{\Delta t} - \overline{\boldsymbol{K}}\left[\theta\overline{\boldsymbol{T}}_{n+1} + (1-\theta)\overline{\boldsymbol{T}}_n\right]\end{aligned} \tag{8.13}$$

The last line in this equation indicates how the contributions from prescribed temperatures (and hence also prescribed temperature rates) may be calculated: the term

$$-\overline{\boldsymbol{C}}\frac{\overline{\boldsymbol{T}}_{n+1} - \overline{\boldsymbol{T}}_n}{\Delta t}$$

introduces the contributions of the temperature rates, since the fraction on the right is an approximation of the temperature rate, and the term

$$-\overline{\boldsymbol{K}}\left[\theta\overline{\boldsymbol{T}}_{n+1}+(1-\theta)\overline{\boldsymbol{T}}_{n}\right]$$

contributes the effect of prescribed temperatures (in the form of a mixture of temperatures at time n and $n+1$).

Now to the question of how to choose the value of θ: upon closer inspection of equation (8.9) we may conclude that the two choices, $\theta = 0$ and $\theta = 1$, will lead to Euler methods – the ***forward*** (explicit) ***Euler*** for the former, and the ***backward*** (implicit) ***Euler*** for the latter. The value of $\theta = 1/2$ is known as the ***Crank-Nicholson*** method. The explicit Euler method has the limitation of conditional stability, which leads to severe restrictions on the time step. On the other hand, the backward Euler and the Crank-Nicholson are for equations (8.13) unconditionally stable. While the Crank-Nicholson is nominally more accurate than the backward Euler, the latter is often given preference because it tends to eliminate oscillations in the solution.

8.2 Transient diffusion: The T3 NAFEMS Benchmark

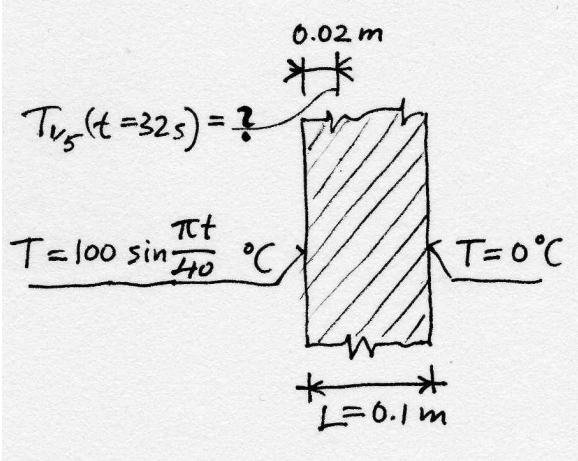

Fig. 8.1. Heat diffusion through a plate (one-dimensional problem), with time-dependent boundary conditions

This test is recommended by the National Agency for Finite Element Methods and Standards (UK), and it is surprisingly exacting. The domain shown in Figure 8.1. One face is held at 0°C, the other face experiences sinusoidal variations in temperature. The temperature at $t = 32$ seconds 0.02 m under the heated face is sought. It is assumed that the plate is very large compared to its thickness, and the problem may therefore be reduced to one dimension, along the thickness. The implementation of transient heat conduction in `SOFEA` is in fact dimension independent, and we simply take care to define the various objects properly for a 1-D problem and the rest follows.

The solution is presented in the Matlab script `t3nafems`[1]. First, the various parameters are defined. Note that the backward Euler method ($\theta = 1$) is selected for the time discretization.

```
0001 kappa=[35.0]; % conductivity matrix
0002 cm = 440.5;% specific heat per unit mass
0003 rho=7200;% mass density
0004 cv =cm* rho;% specific heat per unit volume
0005 Q=0; % uniform heat source
0006 Tampl=100;
0007 Tamb=0;
0008 Tbar =@(t)(Tampl*sin(pi*t/40)+ Tamb);%hot face temperature
0009 num_integ_pts=2; % quadrature
0010 L=0.1;% thickness
0011 dt=0.5; % time step
0012 tend= 32; % length of the time interval
0013 t=0;
0014 theta = 1.0; % generalized trapezoidal method
0015 online_graphics= ~true;% plot the solution as it is computed?
0016 n=3*5;% needs to be multiple of five
```

The mesh is created by `block1d`[2], a simple utility which produces a uniformly spaced mesh on the interval $0 \leq x \leq L$. Note that not only the essential boundary conditions are applied to the temperature field, but also the initial condition (which happens to be 0°C).

```
0017 [fens,gcells] = block1d(L,n,1.0); % Mesh
0018 prop=property_diffusion(struct('conductivity',kappa,...
0019     'specific_heat',cv,'rho',rho,'source',Q));
0020 mater=mater_diffusion (struct('property',prop));
0021 feb = feblock_diffusion (struct (...
0022     'mater',mater,...
0023     'gcells',gcells,...
0024     'integration_rule',gauss_rule(1,num_integ_pts)));
0025 geom = field(struct ('name',['geom'], 'dim', 1, 'fens',fens));
0026 tempn = field(struct ('name',['temp'], 'dim', 1,...
0027     'nfens',get(geom,'nfens')));
0028 tempn = set_ebc(tempn, 1, 1, 1, Tbar(t));
0029 tempn = set_ebc(tempn, n+1, 1, 1, Tamb);
0030 tempn = apply_ebc (tempn);
0031 tempn = numbereqns (tempn);
0032 tempn = scatter_sysvec(tempn,gather_sysvec(tempn)*0+Tamb);
```

[1] Folder: `SOFEA/examples/diffusion`
[2] Folder: `SOFEA/meshes`

The conductivity and capacity matrix are time independent; we compute them once, and henceforth work only with the arrays `Km` and `Cm`.

```
0033 K = start (dense_sysmat, get(tempn, 'neqns'));
0034 K = assemble (K, conductivity(feb, geom, tempn));
0035 Km = get(K,'mat');
0036 C = start (dense_sysmat, get(tempn, 'neqns'));
0037 C = assemble (C, capacity(feb, geom, tempn));
0038 Cm = get(C,'mat');
```

The time stepping begins. First, the temperature boundary conditions are time-dependent, which means they have to be set for each pass through the time loop (i.e. for each time instant).

```
0039 Tfifth = [];
0040 while t<tend+0.1*dt % Time stepping
0041     if online_graphics
0042         plot([0, 0],[0,Tampl],'.'); hold on
0043         plot(get(geom,'values'),get(tempn,'values'));
0044         figure(gcf);
0045         title (['Time =' num2str(t)]); hold off; pause(0.1);
0046     end
0047     tempn1 = tempn;
0048     tempn1 = set_ebc(tempn1, 1, 1, 1, Tbar(t+dt));
0049     tempn1 = set_ebc(tempn1, n+1, 1, 1, Tamb);
0050     tempn1 = apply_ebc (tempn1);
```

The thermal loads corresponding to nonzero temperatures and temperature rates are applied next. We may compare the fields that are being passed on lines `0052` and `0054` with (8.13) and the discussion below that equation: The Matlab code is a literal transcription of the formulas. Note that we are directly working with objects of the class `field`, using operator overload (adding and multiplying fields).

```
0051     F = start (sysvec, get(tempn, 'neqns'));
0052     F = assemble (F, nz_ebc_loads_conductivity(feb, geom, ...
0053         theta*tempn1 + (1-theta)*tempn));
0054     F = assemble (F, nz_ebc_loads_capacity(feb, geom, ...
0055         (tempn1-tempn)*(1/dt)));
0056     Tn=gather_sysvec(tempn);
0057     Tfifth = [Tfifth Tn(n/5+1)];
```

The individual objects in this system of linear equations for `Tn1` are again directly recognizable in formula (8.13).

```
0058     Tn1 = (1/dt*Cm+theta*Km) \ ((1/dt*Cm-(1-theta)*Km)*Tn+...
0059         get(F,'vec'));
0060     tempn = scatter_sysvec(tempn1,Tn1);
0061     t=t+dt;
```

```
0062 end
```

The results are summarized in Figure 8.2. The reference solution is 36.6°C at the time $t = 32$ seconds, and the curve shown in the figure has been obtained with 500 elements through the thickness (yielding 36.16°C). The solution with 15 elements is seen to be in considerable error. This is somewhat surprising, but a closer look at the behavior of the solution during the time interval of interest shows significant temperature gradients near the hot surface, which perhaps explains why it is so expensive to get an accurate solution.

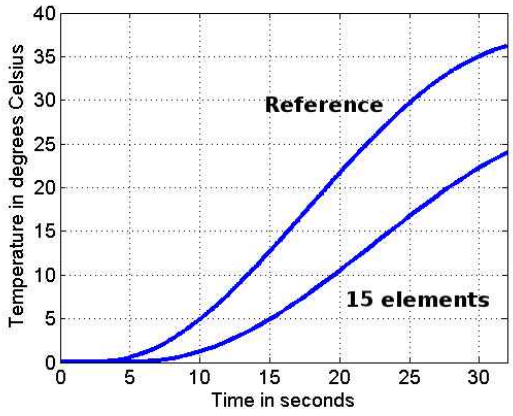

Fig. 8.2. Heat diffusion through a plate (one-dimensional problem): temperature 0.02 m under the heated face

8.3 Transient cooling in a shrink-fitting application

Shrink fitting is a common manufacturing process used to assemble two parts: Figure 8.3. In our case, the cold part is maintained at −10°C prior to the assembly, while the hot part is at 84°C. The temperature of the ambient air is 17°C. The task is to determine how long it will take before the temperature of the hot part drops below 70°C (which is given as a manufacturing constraint).

The problem is solved by the script `shrinkfit`[3]. Let us jump directly into the mesh generation: Figure 8.3 shows the two regions, and the boundary edges (note the numbers next to the edges). The mesh is relatively coarse considering the thickness of some of the geometry – only around three elements through the thickness of the hot part. Even so the mesh has almost 2300 triangular elements, and the transient solution takes a couple of minutes.

```
0017 [fens,gcells,groups,edge_gcells,edge_groups]=targe2_mesher({...
0018     'curve 1 line 0 0 50 0',...
0019     'curve 2 arc 50 0 80 0 center 65 -0.001 ',...
```

[3]Folder: `SOFEA/examples/diffusion`

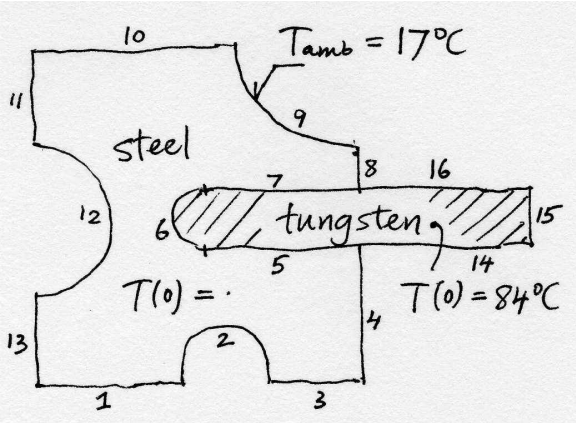

Fig. 8.3. Transient cooling of a shrink-fitted assembly: schematic

```
0020      'curve 3 line 80 0 110 0',...
0021      'curve 4 line 110 0 110 50',...
0022      'curve 5 line 110 50 65 50 ',...
0023      'curve 6 arc 65 50 65 70 center 65.001 60  ',...
0024      'curve 7 line 65 70 110 70',...
0025      'curve 8 line 110 70 110 85',...
0026      'curve 9 arc 110 85 65 120 center 110 120 ',...
0027      'curve 10 line 65 120 0 120',...
0028      'curve 11 line 0 120 0 85',...
0029      'curve 12 arc 0 85 0 35 center -0.001 60 rev',...
0030      'curve 13 line 0 35 0 0',...
0031      'curve 14 line 110, 50, 160, 50',...
0032      'curve 15 line 160, 50, 160, 70',...
0033      'curve 16 line 160, 70, 110, 70',...
0034      ['subregion 1  property 1 boundary '...
0035      ' 1 2 3 4 5 6 7 8 9 10 11 12 13'],...
0036      ['subregion 2  property 2 boundary '...
0037      ' -5 -6 -7 14 15 16'],...
0038      ['m-ctl-point constant 3']
0039      }, 1.0);
```

For the edges that separate the metal from the air, elements to be used in the surface heat transfer needs to be generated (finite element block `efeb`). Note that the interior edges are omitted.

```
0040 prop_steel=...
0041      property_diffusion (struct('conductivity',kappa_steel,...
0042      'specific_heat',cv_steel,'source',0.0));
0043 mater_steel=mater_diffusion (struct('property',prop_steel));
0044 feb_steel = feblock_diffusion (struct ('mater',mater_steel,...
0045      'gcells',gcells(groups{1}),...
0046      'integration_rule',tri_rule(num_integ_pts)));
```

```
0047 prop_tungsten=...
0048     property_diffusion (struct('conductivity',kappa_tungsten,...
0049     'specific_heat',cv_tungsten,'source',0.0));
0050 mater_tungsten=mater_diffusion (struct('property',prop_tungsten));
0051 feb_tungsten = feblock_diffusion (struct ('mater',mater_tungsten,...
0052     'gcells',gcells(groups{2}),...
0053     'integration_rule',tri_rule(num_integ_pts)));
```

The ambient temperature is defined in the field `amb`. The temperature is applied at the nodes associated with the boundary edges in the loop (line 0063).

```
0062 amb = clone(tempn, ['amb']);
0063 for i= 1:length(edge_gcells)
0064     conn = get(edge_gcells(i),'conn');
0065     amb = set_ebc(amb, conn, conn*0+1, [], conn*0+Ta);
0066 end
0067 amb = apply_ebc (amb);
```

The time stepping loop is almost identical to the example in Section 8.2, except the thermal load vector is based on the convective surface heat transfer.

```
0115     F=assemble(F, surface_transfer_loads(efeb,geom,tempn,amb));
```

The evolution of the lowest and highest temperature in the entire assembly is shown in Figure 8.4: the highest temperature drops below 70°C after around 13 seconds. Obviously, we are not addressing the issue of accuracy, neither in the resolution afforded by the mesh, nor in the selection of the time step. However, the time measurements in an actual manufacturing process are not likely to be accurate to more than half a second.

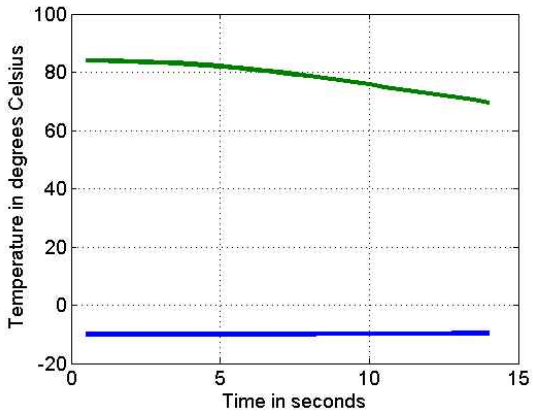

Fig. 8.4. Transient cooling of a shrink-fitted assembly: time evolution of the lowest and highest temperature in the assembly

9

Expanding the library of element types

The linear triangle T3 is not particularly accurate, but for the linear heat diffusion problem it is quite adequate. Nevertheless, we will introduce another couple of elements to expand the scope of the approximation methods discussed so far. This is desirable from a couple of different viewpoints. Firstly, with the linear triangle we have been able to construct basis functions which allow for linear variations in temperature to be represented exactly. Hence, if the exact solution is a constant gradient of temperature, the approximate solution does not involve any discretization error. Unfortunately, the usefulness of this is limited, since constant gradients of temperature are not commonly encountered in applications. If the basis functions can represent higher-order polynomials, for instance quadratic, the resulting method will be able to represent more complex gradients of temperature: linear, in the case of the quadratic variation of temperature. To oversimplify a little bit, the more complex the temperature variations that are reproduced without error, the higher the overall accuracy of the scheme.

Secondly, introducing different element types may enable us to play games with different quadrature schemes. One view of the finite element method could put the basis function above all: the elements are there only to integrate all the expressions that involved the basis functions and their derivatives as accurately as possible (exactly?). However,

9.1 Quadratic triangle T6

The triangle T6 makes it possible to design basis functions that can reproduce quadratic variations of the temperature. More precisely, it will do that in the terms of the coordinates on the standard triangle. As we shall see, the map from the standard triangle will also allow for quadratic temperature variation in the physical space, but more generally it will lead to rational expressions.

The first task will be to formulate the basis functions on the standard triangle, Figure 9.1. To be able 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 common device of equipping the basis functions with the

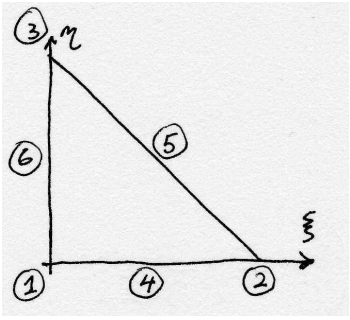

Fig. 9.1. Standard quadratic triangle.

Kronecker delta property (2.19). Let us start with the basis function $N_2 = a_0 + a_1\xi + a_2\eta + a_3\xi\eta + a_4\xi^2 + a_5\eta^2$: writing

$$N_2(\xi_k, \eta_k) = \delta_{2k}, \qquad \text{for } k = 1, , 6$$

at all six nodes (see Table 9.1), provides us with six equations from which the six coefficients may be determined. That is however tedious and boring: let us use commonsense and guesswork instead. Drawing the standard triangle plane while looking along the η axis we see that three and two nodes respectively align, which obviously makes it possible to make the function N_2 equal to zero in these two locations, and equal to one at node 2 with a Lagrange polynomial

$$N_2 = \frac{(\xi - 0)(\xi - 1/2)}{(1 - 0)(1 - 1/2)} = \xi(2\xi - 1)$$

Similarly, in the other direction we have for $N_3 = \eta(2\eta - 1)$.

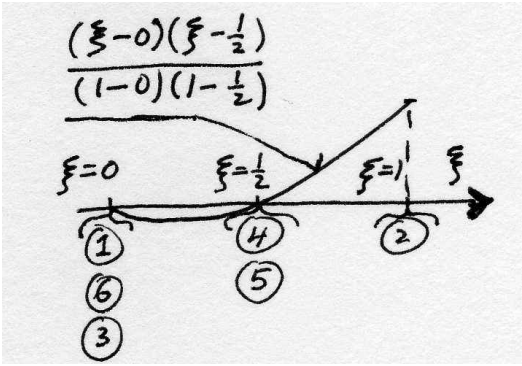

Fig. 9.2. Standard quadratic triangle: one-dimensional view of basis function N_2.

To approach the construction of the other basis functions, we note that both N_2 and N_3 may be written as the normalized product of planes: for N_2 the two planes are $\widehat{p}_2(\xi, \eta) = \xi$ and $\widetilde{p}_2(\xi, \eta) = \xi - 1/2$, and N_2 is written as

$$N_2 = \frac{\widehat{p}_2(\xi,\eta)\widetilde{p}_2(\xi,\eta)}{\widehat{p}_2(1,0)\widetilde{p}_2(1,0)} = \xi(2\xi - 1) \ .$$

Similarly for N_3 and N_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_1 the planes are $\widehat{p}_1(\xi,\eta) = 1 - \xi - \eta$ (this is the same N_1 as in (6.14)) and $\widetilde{p}_1(\xi,\eta) = 1 - 2\xi - 2\eta$ (compare with Figure 9.3)

$$N_1 = (1 - \xi - \eta)(1 - 2\xi - 2\eta) \ .$$

Coordinate	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
ξ	0	1	0	1/2	1/2	0
η	0	0	1	0	1/2	1/2

Table 9.1. Standard quadratic triangle: locations of the nodes

For the mid-edge nodes, 4, 5, 6, we find planes that pass through two triples of nodes. For instance, for node 6 (see Figure 9.3), the two planes are $\widehat{p}_6(\xi,\eta) = 1-\xi-\eta$ and $\widetilde{p}_6(\xi,\eta) = \eta$ (same as N_3 as in (6.13)

$$N_6 = 4(1 - \xi - \eta)\eta \ .$$

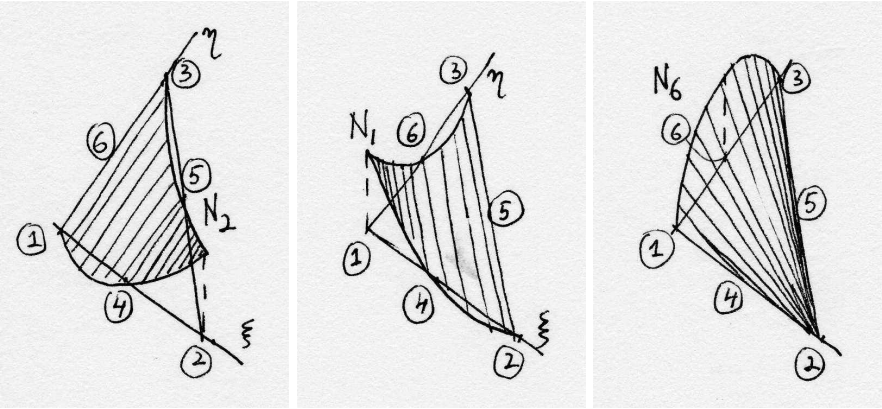

Fig. 9.3. Standard quadratic triangle: Basis functions N_2, N_1, and N_6.

9.2 Quadratic 1-D element L3

9.3 Point element P1

Browsing the `classes/gcell` folder, one may notice the `gcell_X_manifold` class folders with `X`= 0, 1, 2, 3. All `SOFEA` geometric cells are of certain so-called manifold

dimension: solids are of manifold dimension 3, surfaces are of dimension 2, while curves and points are of dimensions 1 and 0. Since we commonly solve heat diffusion (and other problems) in domains that are solids, surfaces, and curves, we also have to deal with integration over the boundaries of these domains; these are, correspondingly, surfaces, curves, and points.

When the heat diffusion problem was being formulated in two-dimensional domains in Section 6, the discrete domain consisted of triangles (elements T3), and the discrete boundary consisted of line segments (elements L2). Analogously, when the heat diffusion is solved in a one-dimensional domain (interval of the real line) which is covered by elements L2, the boundary consists of two points: hence the need for elements of type P1.

Evaluating the integrals of the surface heat transfer matrix (6.28) and the surface heats transfer load (6.27) (and also the prescribed heat flux load (6.26)) over the boundary on interval on the real line simply means taking the values of the integrants at the end points. In terms of a quadrature formula applied at the boundary point a (analogous to (2.24)),

$$f(a) \approx f(\xi_1)J(\xi_1)W_1$$

which is going to give the expected results with $\xi_1 = 0$, $f(\xi_1) = f(a)$, $W_1 = 1$, and the Jacobian $J(\xi_1) = 1$. The quadrature rule with these properties is `point_rule`, and a sample script to use this type of evaluation of boundary integrals for one-dimensional heat diffusion problems is `transcool`[1].

Programming remark: With the introduction of the element P1, a closure is achieved: the same code will now work for heat diffusion problems solved on one-dimensional, two-dimensional, and three-dimensional domains. The programming principles of object-oriented design that are in action here are polymorphism (the methods operate on objects in different types uniformly), and dynamic dispatch (an appropriate method is selected based on the class of the object on which it is invoked).

9.4 Measuring (integrating) over domains

The uniform treatment of the manifold dimension of the domain allows us to produce dimension-independent code. Therefore, integration of any scalar function over any domain or subdomain is carried out by a single method of the class `feblock`. Consider as an example the geometry of a cylinder, the volume tiled with tetrahedra, the bounding surface covered with triangles, the edges of the cylindrical faces approximated with straight two-node segments, and one node at each vertex of the mesh. We may integrate over the volume of the 3-D mesh to find and approximation of the volume of the original cylinder; or over the length of a single edge to approximate the circumference; or over the area of one circular face to find an approximation of the cross-sectional area; or to count all the nodes on the cylindrical surface when we integrate over all the vertices of the triangles on that surface.

[1] Folder: `SOFEA/examples/diffusion`

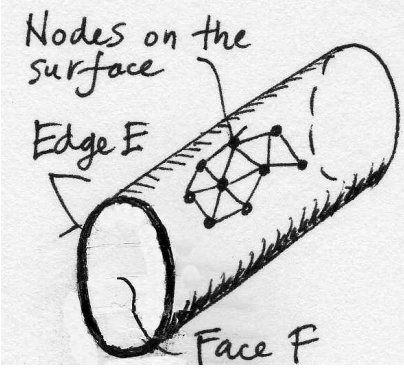

Fig. 9.4. Geometry of a cylinder.

The method `measure` of the class `feblock` evaluates the integral

$$\int_{V_n} f(\boldsymbol{x})\, \mathrm{d}V_n \ , \tag{9.1}$$

where V_n is the volume of an n-dimensional manifold ($n = 0, 1, 2, 3$). The method takes as arguments the geometry field (evidently, the volume of any discrete manifold is going to depend on the locations of its vertices), and a function handle.

```
0013 function result = measure[2] (self, geom, varargin)
0014     gcells =self.gcells;
0015     ngcells = length(self.gcells);
0016     % Integration rule
0017     integration_rule = get(self, 'integration_rule');
0018     pc = get(integration_rule, 'param_coords');
0019     w  = get(integration_rule, 'weights');
0020     npts_per_gcell = get(integration_rule, 'npts');
```

When the function handle is not supplied, it is assumed that the function to be integrated over the manifold is $f(\boldsymbol{x}) = +1$; otherwise, it can be for instance the location-dependent mass density. There are a number of uses to which this method could be applied: as an example consider the calculation of the moments of inertia, or calculation of the centroid.

```
0021     if nargin >=3
0022         fh =varargin{1};
0023     else
0024         fh =@(x) (1);
0025     end
```

Loop over all geometric cells: collect the geometry from the supplied field.

```
0026     result = 0;
0027     % Now loop over all gcells in the block
```

[2] Folder: `SOFEA/classes/feblock/@feblock`

```
0028      for i=1:ngcells
0029          conn = get(gcells(i), 'conn'); % connectivity
0030          x = gather(geom, conn, 'values', 'noreshape');
```

Each type of a geometric cell must provide functions for calculating the basis functions and the derivatives of the basis functions with respect to the parametric coordinates in order to evaluate the Jacobian. It needs to be realized that again the method `Jacobian` is dispatched dynamically, to be treated differently in dependence on the dimension of the manifold. The result of `Jacobian` may depend on the location of the point in the parametric or spatial coordinates.

```
0031          % Loop over all integration points
0032          for j=1:npts_per_gcell
0033              detJ = Jacobian(gcells(i),pc(j,:),x);
```

The array of basis functions is computed so that the spatial location may be evaluated, and supplied to the function `fh`. The result is accumulated with numerical quadrature.

```
0034              N = Nmat(gcells(i), pc(j,:));
0035              result = result + fh(N'*x)*detJ*w(j);
0036          end
0037      end
0038  end
except that
```

As an example, here is the `Jacobian` method for a two-dimensional manifold (a surface), which in this case invokes the method `Jacobian_surface` to do the work. The number of space dimensions of the space `sdim` in which the manifold is embedded could be 2 (the manifold is just a piece of the Euclidean plane), or 3 (the manifold is then a piece of surface). The number of tangents must be 2 (compare with (6.42), and refer to Figure 9.5): they are

$$\frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \xi}, \quad \text{and} \quad \frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \eta}, \ .$$

The Jacobian is the length of the cross product of the two tangents (refer to the Figure 6.9). Here, the cross product is expressed through a skew-symmetric matrix.

```
0013 function detJ = Jacobian_surface[3](self, pc, x)
0014      Nder = Ndermat_param (self, pc);
0015      tangents =x'*Nder;
0016      [sdim, ntan] = size(tangents);
0017      if     ntan==2 % 2-D gcell
0018          if sdim==ntan
0019              detJ = det(tangents);% Compute the Jacobian
0020          else
```

[3] Folder: `SOFEA/classes/gcell/@gcell_2_manifold`

```
0021             detJ = norm(skewmat(tangents(:,1))*tangents(:,2));
0022         end
0023     else
0024         error('Got an incorrect size of tangents');
0025     end
0026 end
```

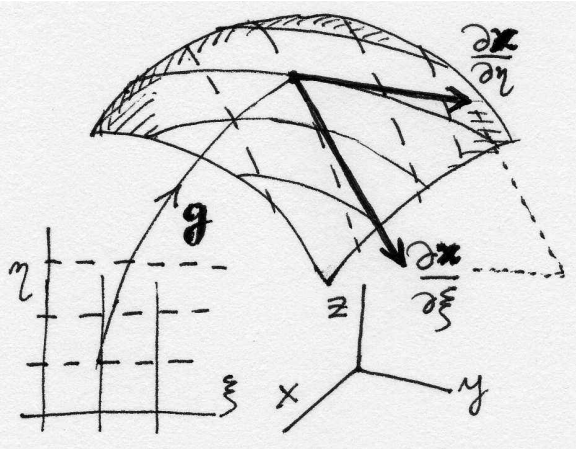

Fig. 9.5. Surface with the coordinate curves and tangents

Finally, we give an example of the use of the `measure` method: The Matlab script `test_measure`[4] measures the volume and the surface area of a rectangular block tiled with tetrahedra T4.

```
0001 a=2.5*pi;
0002 b=2.95;
0003 c=6.1313;
0004 [fens,gcells] = t4block(a,b,c, 5, 4, 7);
0005 bg=mesh_bdry(gcells);
0006 geom = field(struct ('name',['geom'], 'dim', 3, 'fens',fens));
0007 feb = feblock (struct ('mater',[], 'gcells',gcells,...
0008     'integration_rule', tet_rule (1)));
0009 disp([' The volume is = ' num2str(measure(feb,geom,1))
                 ', to be compared with ' num2str(a*b*c)])
0010 feb = feblock (struct ('mater',[], 'gcells',bg,...
0011     'integration_rule', tri_rule (1)));
0012 disp([' The surface is = ' num2str(measure(feb,geom,1))
                 ', to be compared with ' num2str(2*(a*b+a*c+b*c))])
0013 drawmesh({fens,bg},'gcells','facecolor','red','shrink', 0.8);
```

It might seem tempting to evaluate all the objects of the computational methods used in this book, the conductivity matrix, the mass matrix, the load terms, with a

[4] Folder: `SOFEA/examples/miscellaneous`

generalization of the `measure` method. Unfortunately, Matlab passes arguments by value, which means that to accumulate as the result, for instance, a 20000×20000 stiffness matrix would be prohibitively expensive and wasteful. (The correct result would be produced, if that's any consolation.)

9.5 Tetrahedron T4

The tetrahedron with four nodes at the corners (element T4) is a straightforward extension of the triangle T3. The standard tetrahedron is shown in Figure 9.6. 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

$$N_1(\xi,\eta,\zeta) = 1-\xi-\eta-\zeta \ , \ N_2(\xi,\eta,\zeta) = \xi \ , \ N_3(\xi,\eta,\zeta) = \eta \ , \ N_4(\xi,\eta,\zeta) = \zeta \ . \quad (9.2)$$

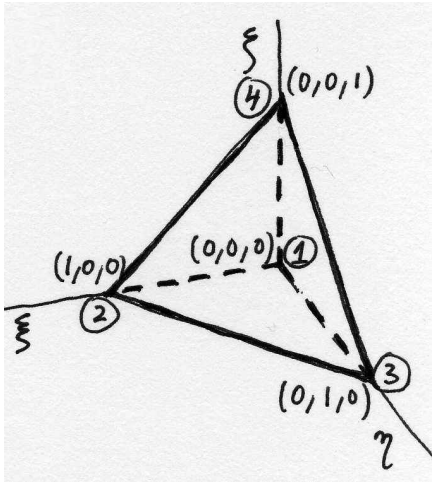

Fig. 9.6. Standard tetrahedron

Table 9.2 defines two integration rules for tetrahedra [Hug00]. The one-point rule is adequate for conductivity matrix evaluation, while the four-point rule could handle the capacity matrix terms.

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	0.1381966, 0.1381966, 0.1381966	1/24	quadratic polynomial
	0.5854102, 0.1381966, 0.1381966	1/24	
	0.1381966, 0.5854102, 0.1381966	1/24	
	0.1381966, 0.1381966, 0.5854102	1/24	

Table 9.2. Numerical integration rules on the standard tetrahedron

The four basis functions of the tetrahedron each vanish along the opposite face (basis function N_i on the face opposite node i and so on). The remaining three vary along this face exactly as if it was a triangle T3. The situation is entirely analogous to the one discussed in Section 6.9 for the triangle T3 and the line segment L2. Therefore, evaluation of the surface heat transfer contributions is simply performed using geometric cells of type T3.

9.5.1 Example: helical geometry

The script `helixcooled`[5] illustrates a solution with a full 3-D geometry discretized with the T4 tetrahedra. The problem is to determine steady state surface temperature for a helical spring, with variable cross-section– see Figure 9.7. The thick end is maintained at constant temperature, and on the rest of the surface there's convection cooling.

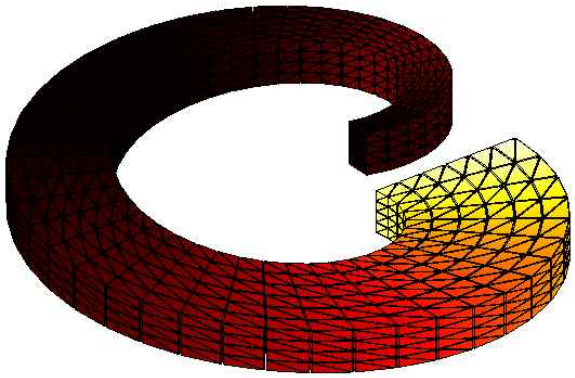

Fig. 9.7. The cooling of a helical spring.

The mesh is a simple regular block tiled with tetrahedra, but it is then shaped by moving nodes to different locations using the utility `transform_apply`, first by changing its cross-section, and then by shifting all nodes in the y-direction. Finally, the shape is twisted into a helix using `transform_2_helix`.

```
0008 [fens,gcells] = t4block(Angle,Width,Height, 50, 6, 4);
0009 Radius = 1.2;
0010 fens=transform_apply(fens,...
                    @(x,data)(x.*[1,(1-x(1)/Angle/1.2),1]),[]);
0011 fens=transform_apply(fens,@(x,data)(x+ [0,Radius,0]),[]);
0012 climbPerRevolution= 1.3;
0013 fens = transform_2_helix(fens,climbPerRevolution);
```

The surface mesh consists of triangles T3, and is extracted from the tetrahedral mesh using the utility `mesh_bdry`. The surface mesh is immediately drawn with `drawmesh`.

[5] Folder: `SOFEA/examples/diffusion`

```
0014 bgcells=mesh_bdry(gcells);
0015 drawmesh({fens,bgcells},'gcells','facecolor','red')
```

Next, the finite element blocks for the tetrahedral elements in the volume and the triangular elements on the surface are created. Note that the two blocks use different quadrature rules, `tet_rule` for the tetrahedra, and `tri_rule` for the triangles; both use just one integration point.

```
0017 feb = feblock_diffusion (struct ('mater',mater,...
0018     'gcells',gcells,...
0019     'integration_rule',tet_rule(num_integ_pts)));
0020 bfeb = feblock_diffusion (struct ('mater',mater,...
0021     'gcells',bgcells,...
0022     'integration_rule',tri_rule(num_integ_pts),...
0023     'surface_transfer', h));
```

From this point on, the script does not depend on the element types, be it the calculation of the system matrices, or graphics output.

9.6 On the 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. Tiling domains with simplex elements is attractive, because a number of mathematical properties guarantees 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, where mesh generators often fail to produce good-quality meshes, or where they often just fail.

While the simplex elements perform adequately in the heat conduction models, in other types of analyses their inherent simplicity tends to work against them. For instance, as we shall see in linear elasticity the response of meshes composed of simplex elements is quite poorly represented – they are "too stiff".

9.7 Quadrilateral Q4

9.8 Hexahedron H8

10

Convergence and error control

10.1 First look at errors

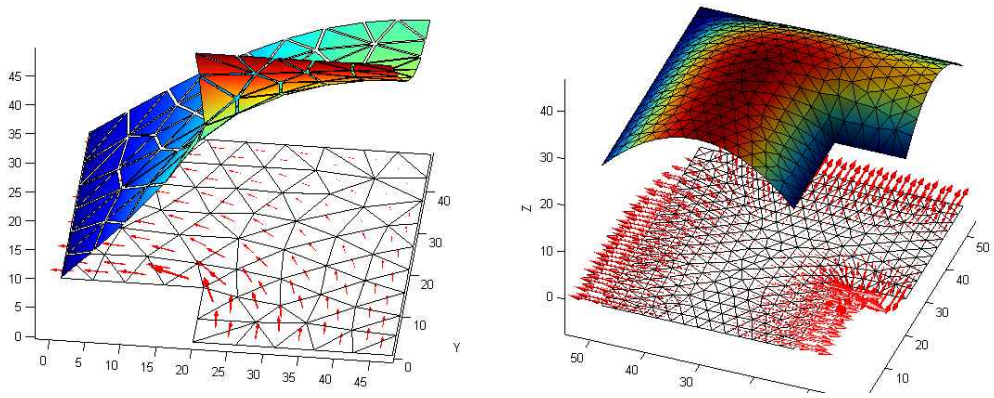

Fig. 10.1. The effect of a reentrant corner on the flux.

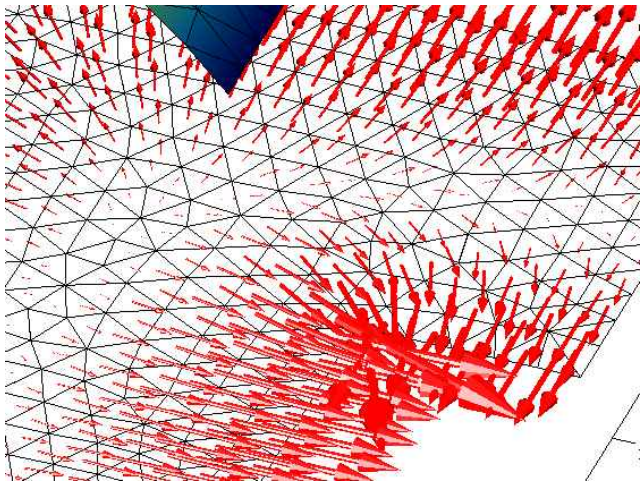

Fig. 10.2. The effect of a reentrant corner on the flux: close-up.

10.2 Richardson extrapolation

Richardson extrapolation is a way of extracting an asymptotic estimate of some quantity of interest from a series of computed values for it. If we assume that the error in the quantity q may be expanded in a Taylor series at mesh size $h = 0$, we may write the error in the remainder form as

$$E_q(h) = \tag{10.1}$$

10.3 The T4 NAFEMS Benchmark revisited

This problem has been discussed in Section 7.4. Cameron, Casey, and Simpson [CCS] cite the reference value for the temperature at the point indicated in Figure 7.5 of 18.3°C. However, more recent investigations of this benchmark indicate that value of 18.25°C should be expected [IL05]. Let us check these numbers.

Two models will be used, the first using elements T3, and the second using the more accurate quadratic elements T6.

18.25396

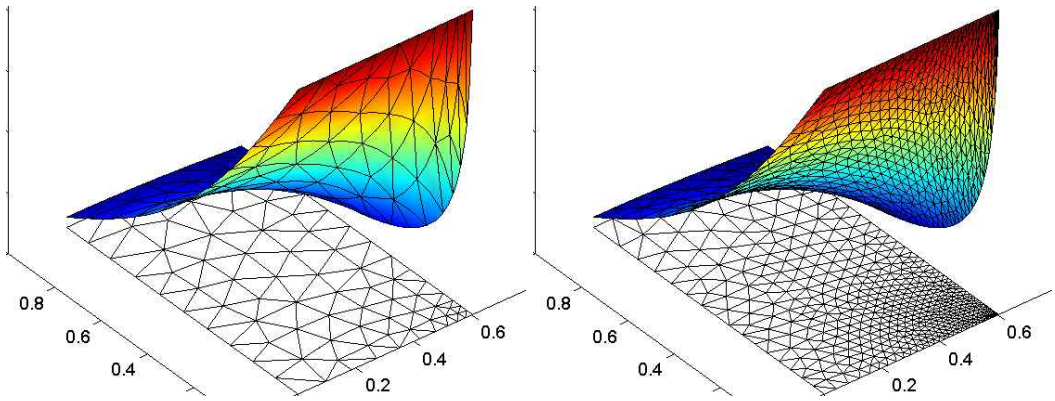

Fig. 10.3. T4 NAFEMS Benchmark: solution with quadratic elements, initial and final mesh.

10.4 Shrink fitting revisited

Figure 10.5 shows the temperature distribution at three time instants. The extremely high gradient at the beginning is evident, but in fact high temperature gradients exist even at the end of the process.

As you recall, the heat flux is derived from the temperature (equation (5.14)). The finite element approximation with the triangles (T3) and with the line elements (L2) will be able to reproduce linearly varying temperatures, hence constant temperature gradients (i.e. heat flux). Therefore, we will conclude that where the heat flux changes, the finite element approximation will be in error. To control the error, we can reduce

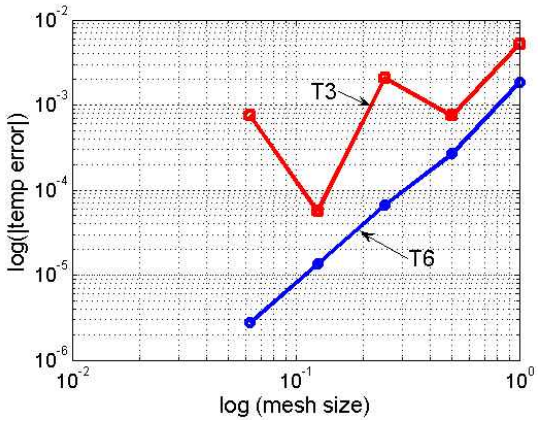

Fig. 10.4. T4 NAFEMS Benchmark: solution with quadratic elements, initial and final mesh.

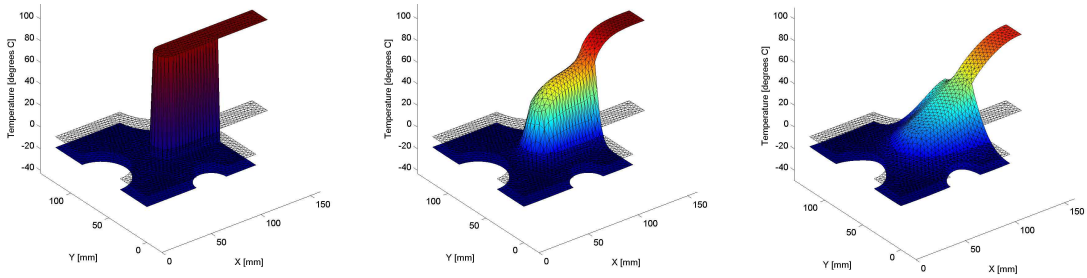

Fig. 10.5. Transient cooling of a shrink-fitted assembly; left to right: temperature distribution for time $t = 0, 2, 13$

the element dimensions. Doing so in areas of steep changes in the heat flux, while keeping areas with approximately uniform heat flux tiled with coarse elements, is known as ***adaptive mesh control***.

Figure 10.6 shows the heat flux on two meshes as arrows centered at the barycenters of the elements (barycenter here means average of the vertex locations). The first mesh is quite coarse (script `shrinkfitad1`[1]), but it is possible to identify regions in which the gradient changes strongly (next to the tungsten inset); the adaptive mesh is generated to reflect the demand for finer (smaller) elements (script `shrinkfitad2`[2]).

The temperature evolution obtained with the two meshes, the coarse one, and the adaptively refined one, is illustrated in Figure 10.7, and the higher-quality of the adaptive results should be noted: especially striking is the spurious oscillation of the lowest temperature for the coarse mesh.

[1]Folder: `SOFEA/examples/diffusion`
[2]Folder: `SOFEA/examples/diffusion`

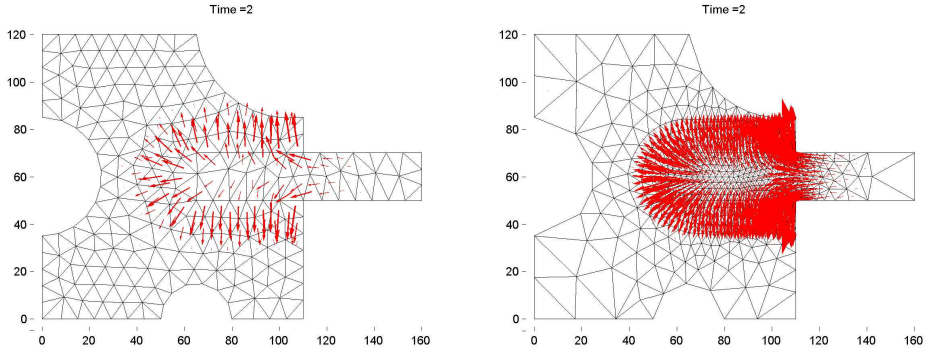

Fig. 10.6. Transient cooling of a shrink-fitted assembly; left: coarse mesh, right: adaptive mesh. Heat flux for time $t = 2$

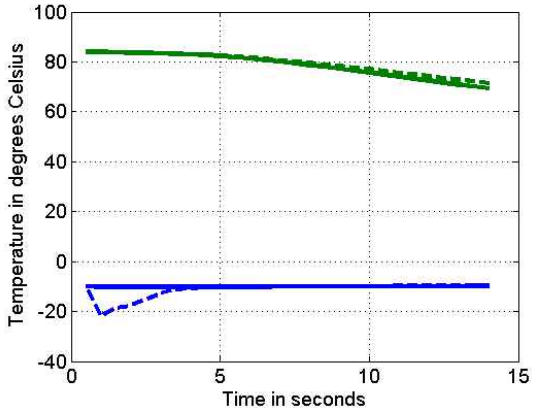

Fig. 10.7. Transient cooling of a shrink-fitted assembly: time evolution of the lowest and highest temperature in the assembly. Comparing temperatures obtained with a coarse model (dashed lines) and with an adaptively refined model (solid lines).

Part III

Stress analysis

11

Model of elastodynamics

11.1 Balance of linear momentum

From elementary dynamics we can apply Newton's equation of motion for a particle, $\dot{\boldsymbol{v}} = m\boldsymbol{F}$, 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. Evidently, our goal is to formulate a continuum model rather than deal with the discrete collection of particles.

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 Figure 11.1. 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***.

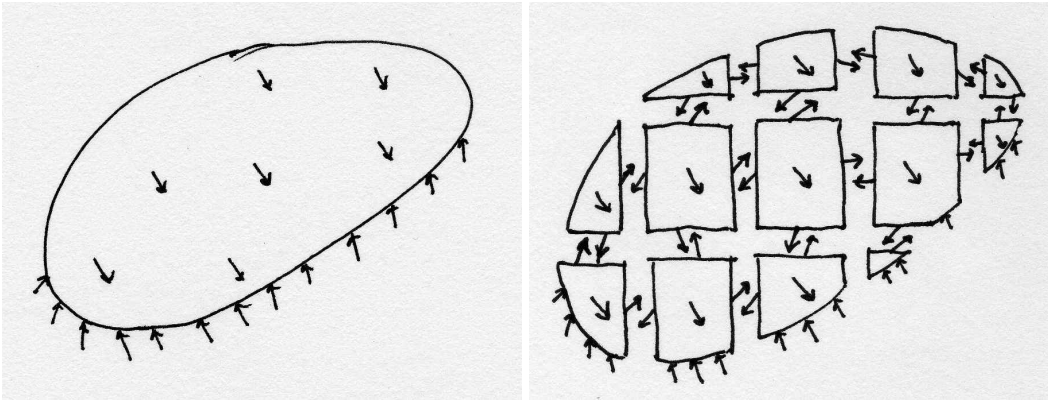

Fig. 11.1. A continuous body with applied distributed force on the boundary, and within the volume (on the left). The same body cut up into many small volumes (particles), with their interaction represented by distributed forces along the cuts (on the right).

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 occurring through 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 of 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

$$\frac{\mathrm{d}}{\mathrm{d}t}(m_j\boldsymbol{v}_j) = \boldsymbol{F}_j \tag{11.1}$$

where we may write for the mass of the particle $m_j = \rho V_j$, with V_j the volume of the particle, and ρ the mass density, $\boldsymbol{x}_j$ 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 $\overline{\boldsymbol{b}}$ and the tractions $\boldsymbol{t}$ on the surface of the particle volume

$$\boldsymbol{F}_j = \overline{\boldsymbol{b}}V_j + \int_{S_{\mathtt{int}}} \boldsymbol{t}\mathrm{d}S + \int_{S_{\mathtt{ext}}} \boldsymbol{t}\mathrm{d}S \tag{11.2}$$

where the surface integral is split into two parts (see Figure 11.2): the interior surfaces $S_{\mathtt{int}}$, where two particle volumes are separated, and the exterior surfaces $S_{\mathtt{ext}}$.

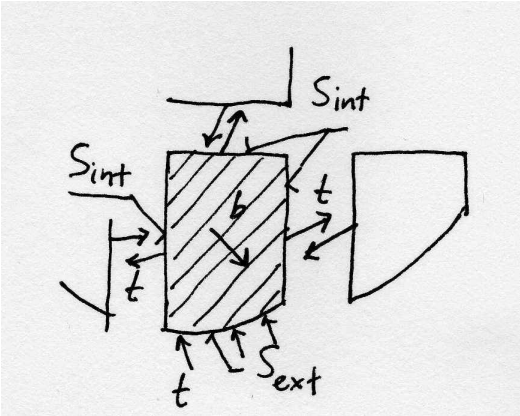

Fig. 11.2. Isolated particle volume.

Now we will collect the contributions of equation (11.1) by summing over all the particles

$$\sum_{j=1}^{N} \frac{\mathrm{d}}{\mathrm{d}t}(m_j \boldsymbol{v}_j) = \sum_{j=1}^{N} \boldsymbol{F}_j \tag{11.3}$$

which may be rewritten of the limit of infinitely many particles as integrals

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\mathrm{d}m = \int_V \boldsymbol{b}\mathrm{d}V + \int_{S_{\mathtt{ext}}} \boldsymbol{t}\mathrm{d}S + \sum_{j=1}^{\infty}\int_{S_{\mathtt{int},j}} \boldsymbol{t}\mathrm{d}S \ , \tag{11.4}$$

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 the equation (11.5) cancels, and the final statement of the ***balance of linear momentum*** of the material in the volume V reads

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\ \mathrm{d}m = \int_V \boldsymbol{b}\ \mathrm{d}V + \int_S \boldsymbol{t}\ \mathrm{d}S \ , \tag{11.5}$$

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).

11.2 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_k are the shear components. 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 Figure 11.3. The normal component is obtained as

$$t_n = \boldsymbol{n}\cdot\boldsymbol{t} \ . \tag{11.6}$$

The shear part of the traction $\boldsymbol{t}_s$ is obtained by subtracting the normal part of the traction from the traction vector $\boldsymbol{t}$

$$\boldsymbol{t}_s = \boldsymbol{t} - t_n\boldsymbol{n} \ . \tag{11.7}$$

The task before us now is 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 to the tractions on

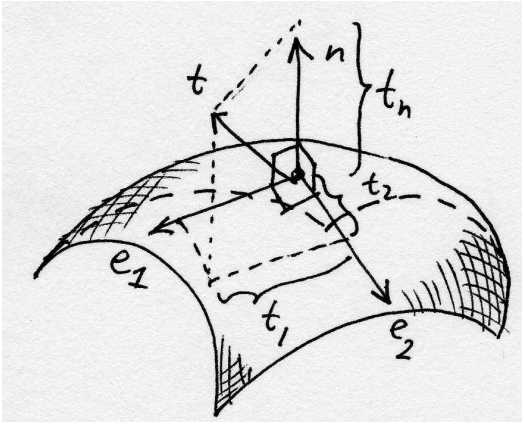

Fig. 11.3. Components of traction.

the surface (and any body loads, if present) through the mathematical device of the stress.

First, inspect Figure 11.4: 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 still containing 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 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 representative of the deformation of the material in the volume.

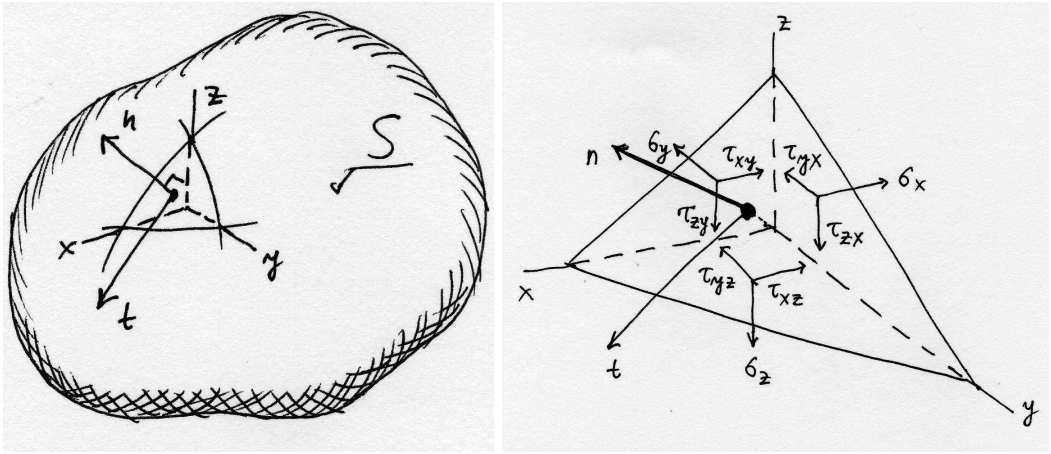

Fig. 11.4. Relating the components of traction to stress.

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 perpendicular to the x-axis and so on; this can be deduced from the volume of the tetrahedron in Figure 11.5.

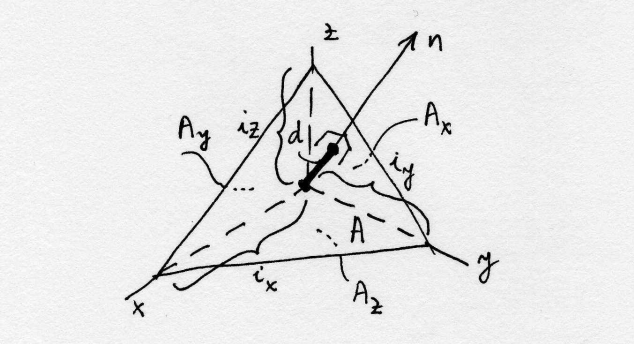

Fig. 11.5. Components of traction.

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

$$\begin{aligned} t_x &= \sigma_x n_x + \tau_{xy} n_y + \tau_{xz} n_z \\ t_y &= \tau_{yx} n_x + \sigma_y n_y + \tau_{yz} n_z \\ t_z &= \tau_{zx} n_x + \tau_{zy} n_y + \sigma_z n_z \end{aligned} \tag{11.8}$$

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 equation (11.8) suggests the matrix expression

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} , \tag{11.9}$$

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. The principal direction components and the principle stress σ are solved for from the two equations

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \sigma \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} , \tag{11.10}$$

and

$$\det \left(\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} - \sigma \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = 0 . \tag{11.11}$$

Balance of angular momentum and stress symmetry.

It would appear that there are nine components of the stress that need to be related to the deformation, but it is straightforward to show that in the matrix (11.9) the off-diagonal elements must be equal: Consider a rectangular volume of material (again, for convenience the drawing in Figure 11.6 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 perpendicular to the plane of the paper, the normal stresses and any body forces will turn out to be negligible compared to to the effect of the shear stresses, and from the resultant equation we obtain the symmetries

$$\tau_{xy} = \tau_{yx} , \quad \tau_{xz} = \tau_{zx} , \quad \tau_{yz} = \tau_{zy} . \tag{11.12}$$

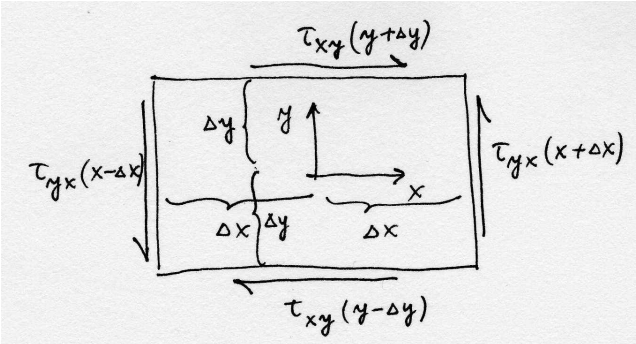

Fig. 11.6. Components of traction.

Consequently, there is 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)

$$[\boldsymbol{\sigma}] = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{xz}, \tau_{yz}]^T . \tag{11.13}$$

Equation (11.8) may be rewritten in terms of the stress vector $\boldsymbol{\sigma}$ as

$$\boldsymbol{t} = \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} , \tag{11.14}$$

where the matrix "***vector-stress vector dot product***" operator is defined as

$$\mathcal{P}_{\boldsymbol{n}} = \begin{bmatrix} n_x & 0 & 0 & n_y & n_z & 0 \\ 0 & n_y & 0 & n_x & 0 & n_z \\ 0 & 0 & n_z & 0 & n_x & n_y \end{bmatrix} . \tag{11.15}$$

Equation (11.14) 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 the 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.

11.3 Local equilibrium

In complete analogy to the model of heat conduction, the global balance equation (11.5) (in this case, balance of linear momentum, for the heat conduction it was balance of heat energy (5.4)) needs to be converted to a local form. The local form would express dynamic equilibrium of an infinitesimal particle as an equation that holds at a point.

11.3.1 Change of linear momentum

There are three terms in the global balance (11.5), 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,

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_m \boldsymbol{v} \, \mathrm{d}m = \int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}m \, . \tag{11.16}$$

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

$$\int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}m = \int_V \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \rho \, \mathrm{d}V \, . \tag{11.17}$$

11.3.2 Stress divergence

The divergence theorem may be now applied to the third term in (11.5), that is to the surface integral. However, introducing the abstract symbol for the divergence of stress generates more questions than answers. We will get to the form of the divergence theorem that will work for us in this book in a roundabout way.

Consider a small volume (parallelepiped) with faces parallel to coordinate planes of the global Cartesian basis (Figure 11.7, and refer also to Figure 11.1); 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 $-1 \le \xi \le +1$, $-1 \le \eta \le +1$, and $-1 \le \zeta \le +1$.

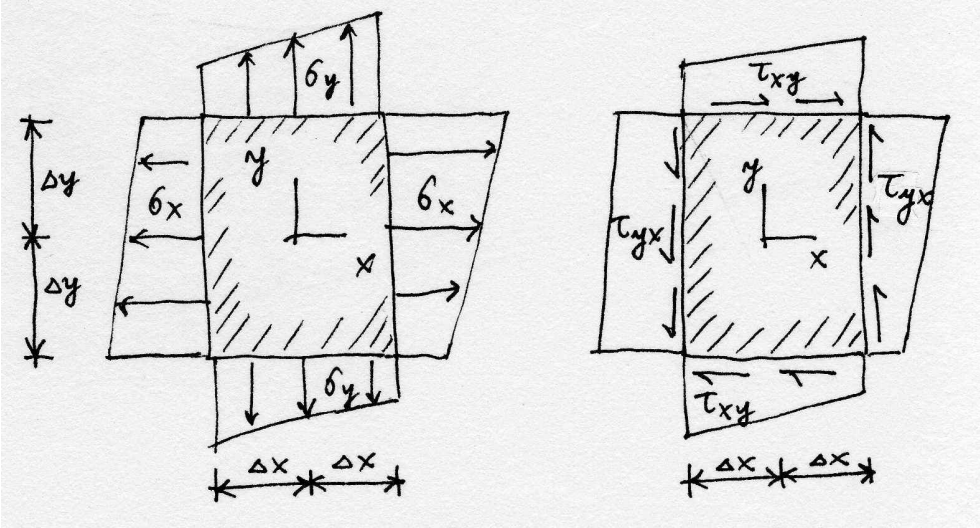

Fig. 11.7. Components of traction.

As you can see, 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 particle 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

$$\Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \sigma_x(x+\Delta x, y+\eta\Delta y, z+\zeta\Delta z)\,\mathrm{d}\eta\mathrm{d}\zeta \approx \Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \left[\sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\Delta x + \frac{\partial\sigma_x(x,y,z)}{\partial y}\eta\Delta y + \frac{\partial\sigma_x(x,y,z)}{\partial z}\zeta\Delta z\right]\,\mathrm{d}\eta\mathrm{d}\zeta$$

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 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

$$\Delta b_x^* = \Delta V\left[\frac{\partial\sigma_x(x,y,z)}{\partial x} + \frac{\partial\tau_{xy}(x,y,z)}{\partial y} + \frac{\partial\tau_{xz}(x,y,z)}{\partial z}\right]\ , \tag{11.18}$$

and analogously in the other two directions

$$\Delta b_y^* = \Delta V\left[\frac{\partial\tau_{yx}(x,y,z)}{\partial x} + \frac{\partial\sigma_y(x,y,z)}{\partial y} + \frac{\partial\tau_{yz}(x,y,z)}{\partial z}\right]\ , \tag{11.19}$$

and

$$\Delta b_z^* = \Delta V\left[\frac{\partial\tau_{zx}(x,y,z)}{\partial x} + \frac{\partial\tau_{zy}(x,y,z)}{\partial y} + \frac{\partial\sigma_z(x,y,z)}{\partial z}\right]\ . \tag{11.20}$$

Now the same argument that was established around equation (11.2) 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 forces (11.18-11.20), 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

$$[\boldsymbol{b}^*] = \begin{bmatrix} \dfrac{\partial \sigma_x(x,y,z)}{\partial x} + \dfrac{\partial \tau_{xy}(x,y,z)}{\partial y} + \dfrac{\partial \tau_{xz}(x,y,z)}{\partial z} \\ \dfrac{\partial \tau_{yx}(x,y,z)}{\partial x} + \dfrac{\partial \sigma_y(x,y,z)}{\partial y} + \dfrac{\partial \tau_{yz}(x,y,z)}{\partial z} \\ \dfrac{\partial \tau_{zx}(x,y,z)}{\partial x} + \dfrac{\partial \tau_{zy}(x,y,z)}{\partial y} + \dfrac{\partial \sigma_z(x,y,z)}{\partial z} \end{bmatrix} \tag{11.21}$$

and may be recognized as the ***stress divergence***.

These two forces must be 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 (11.15), we may write the stress divergence as

$$\boldsymbol{b}^* = \mathcal{B}^T \boldsymbol{\sigma} \tag{11.22}$$

where the ***stress-divergence operator*** $\mathcal{B}^T$ is defined as

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & 0 & \partial/\partial y & \partial/\partial z & 0 \\ 0 & \partial/\partial y & 0 & \partial/\partial x & 0 & \partial/\partial z \\ 0 & 0 & \partial/\partial z & 0 & \partial/\partial x & \partial/\partial y \end{bmatrix} \ . \tag{11.23}$$

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 these useful operators, the ***divergence theorem*** may be written in terms of stress as

$$\int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} \, \mathrm{d}S \ . \tag{11.24}$$

11.3.3 All together now

Putting the three integrals from (11.5) into the volume-integral form leads to a pointwise expression of local equilibrium (following exactly the same argument as in Section 5.1):

$$\int_V \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}V = \int_V \overline{\boldsymbol{b}} \, \mathrm{d}V + \int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V \quad \Rightarrow \quad \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \overline{\boldsymbol{b}} + \mathcal{B}^T \boldsymbol{\sigma} \ . \tag{11.25}$$

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.

11.4 Strains and displacements

The measurable quantities in this problem are the strains (the *relative* deformation). The strains are divided into two groups, based on the effect the strains display when expressed in Cartesian coordinates: the normal strains (stretches), and the shear strains.

The strains are an expression of local variations in the positions of points after deformation. The deformation (motion) is expressed as displacements. The ***displacement 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

$$[\boldsymbol{u}(A,t)] = \begin{bmatrix} x(A,t) \\ y(A,t) \\ z(A,t) \end{bmatrix} - \begin{bmatrix} x(A,0) \\ y(A,0) \\ z(A,0) \end{bmatrix} .E \tag{11.26}$$

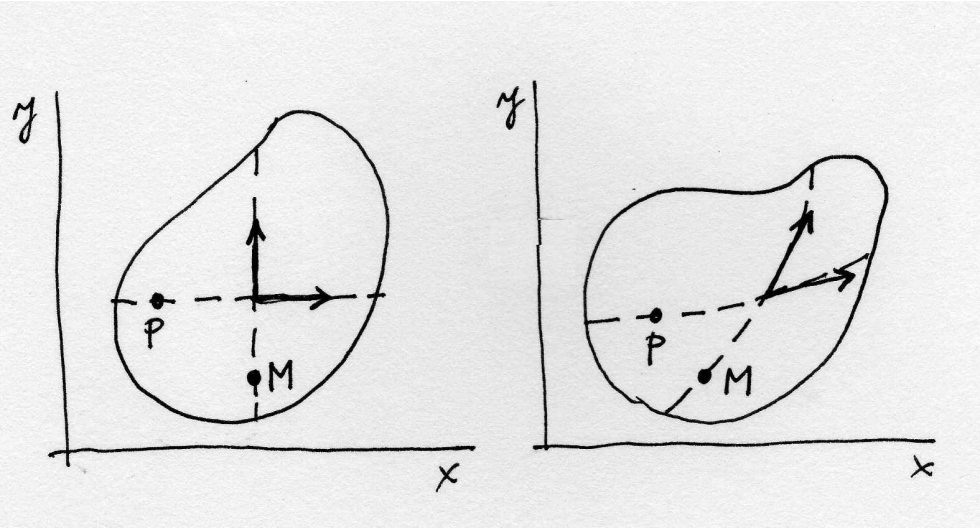

Fig. 11.8. Material curves, and tangents to material curves. Left: before deformation, right: after deformation.

It will be useful to approach the meaning of strains from the point of view of what happens to tangents to material curves. A ***material curve*** consists of the same material points (particles) during the deformation of the material. A visual picture may be useful: recall that some specimens have a grid etched upon them before they are being mechanically tested (deformed). The etching curves that go in one direction may be thought of as points whose one coordinate changes and the other is being held fixed. Figure 11.8 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

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 \\ 0 \end{bmatrix} . \tag{11.27}$$

The parameter that varies along the curve through the point M is y. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} . \tag{11.28}$$

The tangent vectors (11.27) and (11.30) 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

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 + \dfrac{\partial u_x}{\partial x} \\ \dfrac{\partial u_y}{\partial x} \end{bmatrix} . \tag{11.29}$$

The parameter that varies along the curve through the point M is y. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} \dfrac{\partial u_x}{\partial y} \\ 1 + \dfrac{\partial u_y}{\partial y} \end{bmatrix} . \tag{11.30}$$

The ***stretches*** measure the relative change in length of the tangent vectors at the same point before and after deformation. For instance, the tangent vector (11.27) is of unit length before deformation, and the vector (11.29) 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,

$$|\frac{\partial u_k}{\partial j}|\ll 1,\quad k,j=x,y,z\ , \tag{11.31}$$

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

$$\frac{1+\frac{\partial u_x}{\partial x}-1}{1}=\epsilon_x\ .$$

The ***shears*** measure the change in angle between originally perpendicular 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 Figure 11.8, 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 (11.31), gives for the change of 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 (11.22)

$$\boldsymbol{\epsilon}=\mathcal{B}\boldsymbol{u}\ , \tag{11.32}$$

where $\mathcal{B}$ is now called the ***symmetric gradient*** (or strain-displacement) operator.

11.5 Constitutive equation

The stress may now be replaced in the balance equation (11.25) by reference to the primary variable, the displacement. However, first we need to discuss the link between

the measurable quantities, the strains, and the mathematical device in the balance equation, the stress. As for the thermal model, this link is the constitutive equation.

Since the angular momentum balance (11.12) reduces the number of stress components to six, correspondingly there is six components of strain. Therefore, the energy of deformation may be defined as the work of each stress component on the corresponding strain component. Let us consider some pre-existing stressed state in a very small neighborhood of a given point. So that we don't have to specify the volume, we will refer to ***energy density*** (the energy in a certain volume may be obtained by integrating the energy density over this volume). The state of stress is described by the stress vector $\boldsymbol{\sigma}$. Let us superimpose an infinitesimal strain variation $\mathrm{d}\boldsymbol{\epsilon}$ upon the extant strains. The density of the work of the current stress on the strain change is expressed as

$$\mathrm{d}\boldsymbol{\epsilon}^T\boldsymbol{\sigma} \ . \tag{11.33}$$

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,***constitutive equation***, is expressed as a matrix product

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon} \ , \tag{11.34}$$

where $\boldsymbol{D}$ is a constant 6×6 matrix of the elastic coefficients (also known as the elasticities); $\boldsymbol{D}$ may be also referred to as the ***material stiffness*** matrix. Clearly, when there is no strain, the stress is zero. Let us now increase strain from zero to its final value, $\boldsymbol{\epsilon}$, by scaling with a number $0 \leq \theta \leq 1$

$$\widehat{\boldsymbol{\epsilon}} = \theta\boldsymbol{\epsilon} \ ,$$

and furthermore use the linear elasticity (11.34). The expression for the change of the energy of deformation density (11.35) will become

$$\mathrm{d}\widehat{\boldsymbol{\epsilon}}^T\widehat{\boldsymbol{\sigma}} = \mathrm{d}\widehat{\boldsymbol{\epsilon}}^T\boldsymbol{D}\widehat{\boldsymbol{\epsilon}} = \mathrm{d}\theta\boldsymbol{\epsilon}^T\boldsymbol{D}\theta\boldsymbol{\epsilon} \ . \tag{11.35}$$

The deformation process starts at $\theta = 0$ and reaches its final stage at $\theta = 1$. In this process, the total energy density stored in the material is

$$\phi(\boldsymbol{\epsilon}) = \int_0^1 \mathrm{d}\theta\boldsymbol{\epsilon}^T\boldsymbol{D}\theta\boldsymbol{\epsilon} = \frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{D}\boldsymbol{\epsilon} \ . \tag{11.36}$$

Mathematically, the expression $\frac{1}{2}\boldsymbol{\epsilon}^T\boldsymbol{D}\boldsymbol{\epsilon}$ is known as a ***quadratic form***. One interesting the 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, $\boldsymbol{D} = \boldsymbol{D}^T$***.

At the moment, we will leave the material stiffness matrix unspecified, since a detailed discussion follows in Section 12.5.

11.6 Initial conditions

11.7 Boundary conditions

Similarly to the heat conduction problem, at each point on the bounding surface a boundary condition is required. The boundary conditions may be in terms of the primary variable, the displacement, or in terms of the flux variable, the stress. For heat conduction, the boundary condition in terms of flux referred to the *normal* flux only, since the flux parallel to the surface is essentially uncontrollable. Similarly, for elasticity the flux boundary condition will not attempt to prescribe all six components of stress, but rather the "projection" of stress, the traction.

A complicating circumstance is that the primary variable and the traction both have three components. Therefore, the surface of the solid needs to be considered three times as to the appropriate boundary condition, once for each component.

Selection of the appropriate boundary conditions is critical to successful modeling. Typically, the boundary conditions that are applied to our models are only approximations of the physical reality. Thus, the first guidelines for the application of boundary conditions will be based on *physical considerations*.

11.7.1 Example: concrete dam

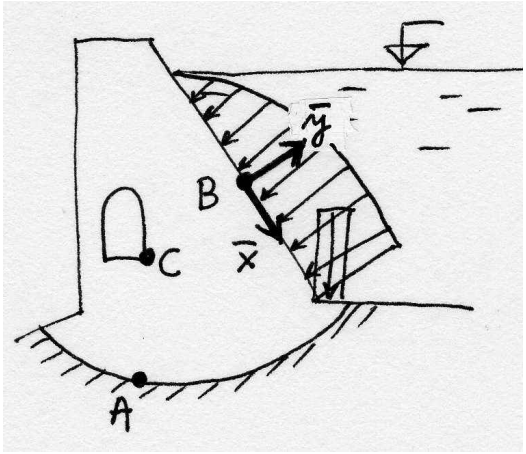

Fig. 11.9. Example of boundary conditions: concrete dam with a tunnel.

For instance, Figure 11.9 shows the cross-section of a dam, and of interest is the stress near point C in the corner of the tunnel. Therefore, we could decide to neglect the deformation of the soil near the base of the dam, and prescribe along the surface A zero magnitude for all displacement components. In reality, this is not strictly true, and a so-called *modeling error* is being introduced by making this choice. In a careful analysis, the influence of this error would be assessed, for instance by varying the boundary condition, or including the soil in the analysis.

On the surfaces exposed to the water behind the dam, including the one with point A, the structure is loaded by water *pressure*, which is a special kind of traction: using an ad hoc Cartesian coordinate system as indicated in the figure, the traction components are

$$t_{\bar{x}} = 0, \quad t_{\bar{y}} = p, \quad t_{\bar{z}} = 0$$

where p is the water pressure at the particular location.

All the other surfaces in the model that show up as curves, are assigned so-called ***traction-free*** boundary condition: there are no known loads applied there. Assuming the model is two-dimensional, of the so-called plane strain type, the remaining surfaces are parallel to the plane of the paper, and are assigned zero displacement normal to the paper, and zero shear components of traction in the plane of the paper. This type of model is discussed later in the textbook.

Let us now discuss the associated variable (so-called work-conjugate variable)– traction – along the surfaces where we prescribed displacements, for instance at point A. The physical meaning of such tractions, which are generated in the soil by the stress in the bulk of the dam near the surface, is clear: they are the ***reactions***. They are initially unknown, but as soon as the displacements are available from the solution, the reactions may be calculated.

The work-conjugate variable along the traction-free surfaces is displacement, which is initially unknown, but which will be produced during the solution process. Similarly, displacement is unknown on the surfaces exposed to the water behind the dam.

11.7.2 Example: rigid punch

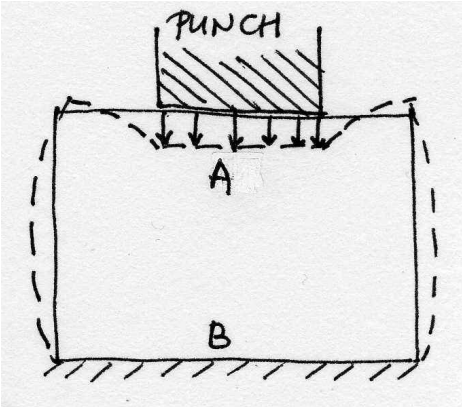

Fig. 11.10. Example of boundary conditions: rigid punch.

11.7.3 Formal definition of boundary conditions

At each point of the boundary we define a Cartesian coordinate system. It could be the surface-aligned system of Figure 11.3, it could be the global system, or an arbitrarily

oriented system. For instance, refer to Figure 11.11. Both the traction vector and the displacement vector at a given point may be written in such a coordinate system in terms of the components as

$$[\boldsymbol{t}] = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad , \quad [\boldsymbol{u}] = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \quad .$$

For each direction $i = x, y, z$ we separate the surface S into two disjoint parts:

1. $S_{t,i}$ where the traction component t_i is being prescribed;
2. $S_{u,i}$ where the displacement component u_i is being prescribed.

It holds that $S = S_{t,i} \bigcup S_{u,i}$, and it could be that either $S_{t,i} = \emptyset$ or $S_{u,i} = \emptyset$.

The boundary conditions may now be expressed as

$$t_i = (\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})_i = \bar{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \tag{11.37}$$

as the ***traction*** (natural) ***boundary condition***, and

$$u_i = \overline{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \tag{11.38}$$

as the ***displacement*** (essential) ***boundary condition***.

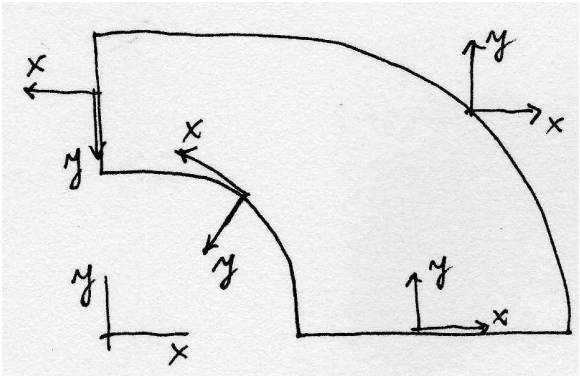

Fig. 11.11. Local coordinate systems used for boundary condition definitions.

11.7.4 Inadmissible "concentrated" boundary conditions

Consider that a resultant force of magnitude F is to be applied along the z-direction, in other words perpendicularly to the surface shown in Figure 11.13, 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 $\Delta x \to 0, \Delta y \to 0$, it must hold $t_z \to \infty$. However, since from the boundary conditions we have $\sigma_z = t_z$, we must conclude that in the immediate vicinity of the

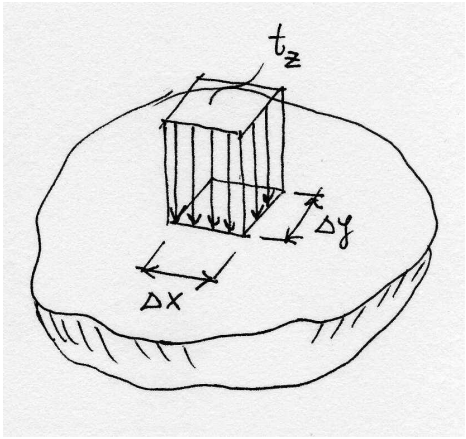

Fig. 11.12. Concentrated force as the limit of traction on infinitesimally small area.

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 [Sok], 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 are too lazy to think the problem through.

As a corollary, we must conclude that if we apply a *displacement boundary condition at a point*, the associated reaction will be infinite in the limit, unless the limit value for the reaction is zero.

Very similar analysis may be performed for a *distributed load along a curve*: Figure 11.13. 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.

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.

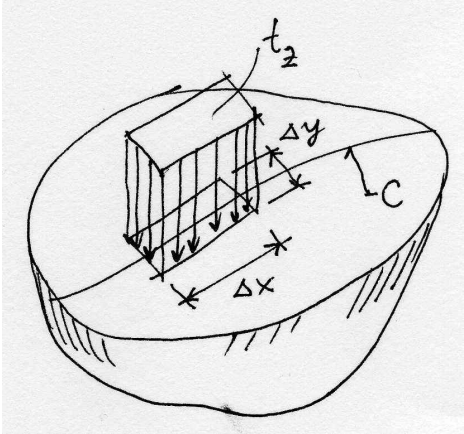

Fig. 11.13. Distributed load along a curve as the limit of traction on infinitesimally small area.

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.

11.7.5 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 be 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 11.14. 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) perpendicular 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 perpendicular component of displacement $u_\perp$ at a point on the plane of symmetry must be zero

$$u_\perp = 0 \; . \tag{11.39}$$

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.

$$\boldsymbol{t}_{s(-n)} = \boldsymbol{t}_{s(n)} \; .$$

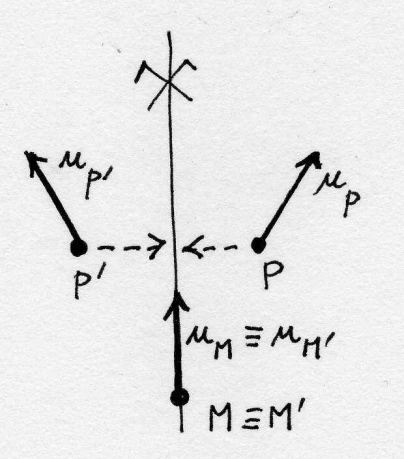

Fig. 11.14. Symmetric displacement pattern.

On the other hand, using equation (11.7) we have

$$\boldsymbol{t}_{s(n)} = \boldsymbol{t}_{(n)} - \big(\boldsymbol{n} \cdot \boldsymbol{t}_{(n)}\big)\, \boldsymbol{n} \ ,$$

and substituting from (11.15) 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,

$$\boldsymbol{t}_{s(n)} = \boldsymbol{0} \ , \tag{11.40}$$

must hold.

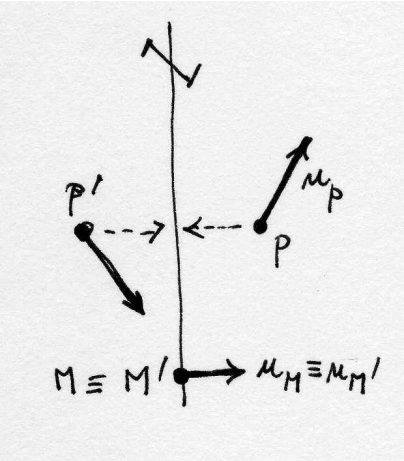

Fig. 11.15. Anti-symmetric displacement pattern.

The state of ***anti-symmetry with respect to reflection in a plane*** is defined by the conditions that specify it as the opposite of symmetry. The situation is illustrated in Figure 11.15. 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 perpendicular 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

$$u_{\parallel,i} = 0 \quad \text{for the two in-plane directions } i \ . \tag{11.41}$$

For the tractions, an analysis quite similar to that leading to equation (11.40) but applied to the normal component of the traction leads to the condition

$$t_\perp = 0 \ . \tag{11.42}$$

Therefore, we can summarize the boundary conditions on the plane of symmetry or on the plane of anti-symmetry with the delightfully simple Table 11.1. The boundary

Quantity	Symmetry	Anti-symmetry
Tractions $\parallel$	0	unknown
Displacements $\parallel$	unknown	$\boxed{0}$
Traction $\perp$	unknown	0
Displacement $\perp$	$\boxed{0}$	unknown

Table 11.1. Boundary conditions on the plane of symmetry or anti-symmetry

conditions that needs to be explicitly prescribed in finite element analyses are boxed in the Table 11.1: zero tractions are incorporated automatically (natural boundary conditions!) and need not be explicitly specified. Note that the unknown tractions are the reactions.

11.7.6 Example: pure-traction problem

Often static problems are posed where initially only a statically equilibriated set of traction and/or body loads is given. As an example, we consider the dog bone tensile specimen of Figure 11.16 (slice through the axis of symmetry is shown). Uniform tractions are applied at the opposite cross-sections, equal in magnitude, but of opposite sign, so that specimen is in static equilibrium: the so-called ***pure-traction problem***. As such, this set of boundary conditions does not allow for the finite element solution to be computed without additional devices: the entire specimen may be translated or rotated as a rigid body without any change in the stress state. Therefore, the stiffness matrix would be singular.

The rigid body motion may be described by displacements of the form

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} 0 & -\Theta_z & \Theta_y \\ \Theta_z & 0 & -\Theta_x \\ -\Theta_y & \Theta_x & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \ , \tag{11.43}$$

where $\Theta_x, \Theta_y, \Theta_z$ and a_x, a_y, a_z are constants describing ***rotation*** (through the skew-symmetric matrix – another way of writing a cross product of two vectors), and ***translation*** of the points of a rigid body.

It doesn't take too much effort to verify that strains computed from displacements (11.43) are all identically zero. Therefore, also the deformation energy induced by the rigid body motion is zero. Using (11.36) for the energy density, the energy of deformation produced by strains $\boldsymbol{\epsilon}$ is

$$\begin{aligned} \Phi(\boldsymbol{\epsilon}) = \int_V \phi(\boldsymbol{\epsilon}) \, \mathrm{d}V = \sum_e \int_{V_e} \phi(\boldsymbol{\epsilon}) \, \mathrm{d}V = \\ \sum_e \frac{1}{2} \boldsymbol{U}_e^T \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \, \mathrm{d}V \, \boldsymbol{U}_e = \\ \sum_e \frac{1}{2} \boldsymbol{U}_e^T \boldsymbol{K}_e \boldsymbol{U}_e = \frac{1}{2} \boldsymbol{U}^T \boldsymbol{K} \boldsymbol{U} \, . \end{aligned} \tag{11.44}$$

Therefore, since this energy vanishes for a nonzero displacement $\boldsymbol{U} \neq \boldsymbol{0}$, the global matrix $\boldsymbol{K}$ must be singular to produce a positive semi-definite quadratic form $\frac{1}{2}\boldsymbol{U}^T \boldsymbol{K} \boldsymbol{U} \geq 0$. The stiffness matrix falls short of the full rank by the number of possible rigid body modes: six, when all three rotations and translations are possible, or less.

To restore the full rank of the stiffness matrix, all possible rigid body modes must be prevented by an additional supports (displacement boundary conditions).

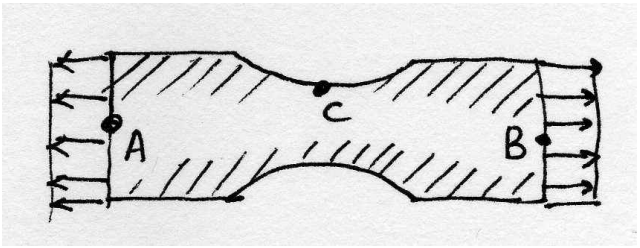

Fig. 11.16. Example of boundary conditions: dog bone specimen under tension.

11.7.7 Example: shaft under torsion

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 define shear tractions is when we formulate simplified models in which the effect of the omitted part of a structure is introduced as resultants into the model, both pressure resultant, and shear resultant.

As an example, we will consider a shaft of circular cross-section, with two through-holes. Of interest is the local stress concentration around the holes when the shaft is

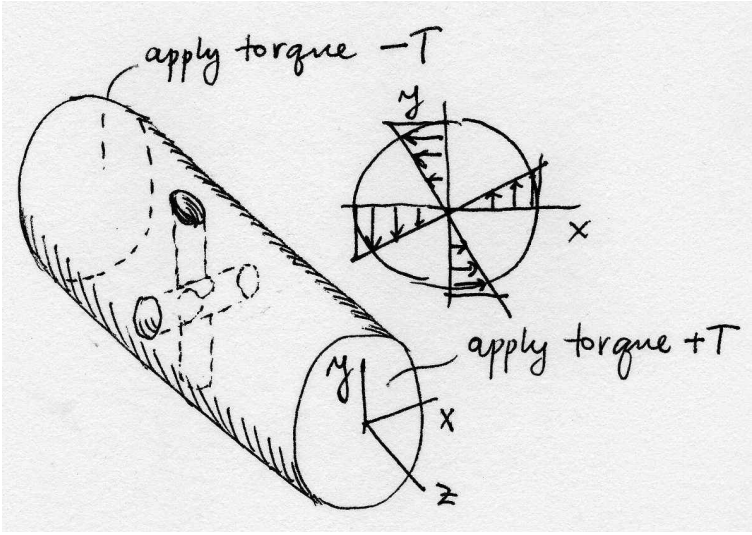

Fig. 11.17. Example of boundary conditions: shaft loaded by torque.

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 midsection with the holes, and the torque is applied as prescribed shear tractions.

The way in which we distribute the shear tractions is only an approximation of the stress distribution that would exist in the complete part. However, that so-called ***Saint-Vénant's principle***

11.7.8 Example: overspecified boundary conditions

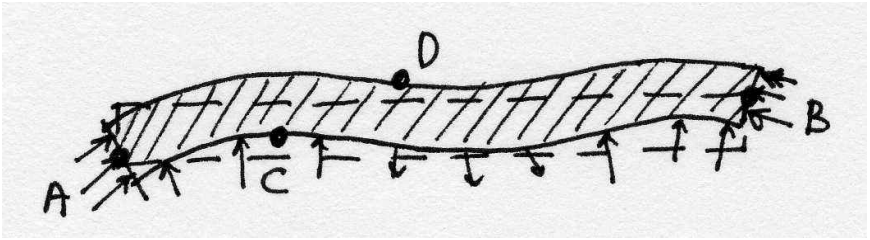

Fig. 11.18. Example of boundary conditions: plate with unknown boundary conditions.

11.8 Comparing the Thermal and Deformation models

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\,\mathrm{d}m = \int_V \boldsymbol{b}\,\mathrm{d}V + \int_S \boldsymbol{\Sigma}\cdot\boldsymbol{n}\,\mathrm{d}S\ . \tag{11.45}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_V u\,\mathrm{d}V = \int_V Q\,\mathrm{d}V - \int_S \boldsymbol{q}\cdot\boldsymbol{n}\,\mathrm{d}S\ . \tag{11.46}$$

12

Galerkin formulation for elastodynamics

12.1 Manipulation of the residuals

Rearranging the balance equation (11.25) into the residual form leads to

$$\boldsymbol{r}_B = \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \overline{\boldsymbol{b}} - \mathcal{B}^T \boldsymbol{\sigma} \tag{12.1}$$

which is a statement of imbalance when the force residual is nonzero. This is in complete analogy to the model of taut wire, or the model of heat conduction. The plan of action is the same:

1. Formulate weighted residual equations for the balance, force boundary condition, and displacement boundary condition.
2. Satisfy the displacement condition by design, which will place a condition on the form of the trial functions.
3. Shift a derivative from the stress to the test function, which will incorporate the natural boundary conditions in the balance residual equation (and eliminate the force boundary condition residual), and which will place a condition on the form of the test functions.

12.1.1 The first two steps

We begin with step 1: The natural boundary condition (11.37) leads to the residual

$$r_{t,i} = \left(\mathcal{P}_n \boldsymbol{\sigma}\right)_i - \bar{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \tag{12.2}$$

which will be incorporated into the balance residual equation, and the displacement boundary condition (11.38)

$$r_{u,i} = u_i - \overline{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \tag{12.3}$$

which will be satisfied by the choice of the trial functions (which takes care of the step 2).

The weighted residual equations are simply 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

$$\int_{S_{t,i}} r_{t,i}\eta_i \, \mathrm{d}S = 0 \, ,$$

or, expanded,

$$\int_{S_{t,i}} r_{t,i}\eta_i \, \mathrm{d}S = \int_{S_{t,i}} \left[(\mathcal{P}_n \boldsymbol{\sigma})_i - \bar{t}_i \right] \eta_i \, \mathrm{d}S = 0 \quad \Rightarrow \quad \int_{S_{t,i}} (\mathcal{P}_n \boldsymbol{\sigma})_i \, \mathrm{d}S = \int_{S_{t,i}} \bar{t}_i \, \mathrm{d}S \tag{12.4}$$

The balance weighted residual equation reads

$$\int_V \boldsymbol{r}_B \cdot \boldsymbol{\eta} \, \mathrm{d}V = \int_V \boldsymbol{\eta} \cdot \boldsymbol{r}_B \, \mathrm{d}V = 0 \, , \tag{12.5}$$

where $\boldsymbol{\eta}$ is a vector test function (with three components). At this point, we only require that the test function be sufficient to 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

$$\int_V [\boldsymbol{r}_B]^T [\boldsymbol{\eta}] \, \mathrm{d}V = \int_V [\boldsymbol{\eta}]^T [\boldsymbol{r}_B] \, \mathrm{d}V = 0 \, .$$

12.1.2 Step 3: Preliminaries

While the first two terms on the right hand side of (12.1) present no difficulties, the stress term needs to be treated similarly to the previous two models to move one differentiation from the stress to the test function. Therefore, in the next few paragraphs we focus on the term

$$\int_V \boldsymbol{\eta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V$$

which will be sought as part of the chain rule formula (analogously to equation (6.4) for the heat conduction). The product of $\boldsymbol{\eta}$ and $\boldsymbol{\sigma}$ may be expressed using the vector-stress vector dot product operator (11.15) in the form $\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}$. Therefore, we need an identity for the chain rule applied to the divergence div $\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)$:

$$\mathrm{div}\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right) = (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} + \boldsymbol{\eta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, . \tag{12.6}$$

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 last piece of (12.6) in terms of the other two, we obtain

$$\int_V \boldsymbol{\eta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_V \mathrm{div}\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right) \, \mathrm{d}V - \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V$$

The divergence theorem (5.10) may be applied to the first term on the right to yield

$$\int_V \boldsymbol{\eta} \cdot \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right) \cdot \boldsymbol{n} \, \mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} \, \mathrm{d}V \, .$$

The traction boundary condition (11.37) references $\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}$. To extricate this form from $\big(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\big)\cdot\boldsymbol{n}$ we notice that the result is a scalar (number), indicating a double dot product: the stress vector with two vectors. Indeed, it is easily verified by multiplying through that

$$\big(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\big)\cdot\boldsymbol{n} = \big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)\cdot\boldsymbol{\eta}\ ,$$

with the result

$$\int_V \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\ \mathrm{d}V = \int_S \boldsymbol{\eta}\cdot\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)\ \mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\ \mathrm{d}V\ .$$

We are ready to write the balance residual equation (12.5) now as

$$\begin{aligned}&\int_V \boldsymbol{\eta}\cdot\rho\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\ \mathrm{d}V - \int_V \boldsymbol{\eta}\cdot\overline{\boldsymbol{b}}\ \mathrm{d}V\\ &- \int_S \boldsymbol{\eta}\cdot\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)\ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\ \mathrm{d}V = 0\end{aligned} \tag{12.7}$$

12.1.3 Step 3: The glorious conclusion

Closely paralleling Section (6.6), the surface will now be split, for each component, into the part where traction is known, and the part where displacement is being prescribed

$$\int_S \boldsymbol{\eta}\cdot\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)\ \mathrm{d}S = \sum_{i=x,y,z}\int_{S_{t,i}} (\boldsymbol{\eta})_i\,\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)_i\ \mathrm{d}S + \int_{S_{u,i}} (\boldsymbol{\eta})_i\,\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)_i\ \mathrm{d}S\ ,$$

where $(\boldsymbol{\eta})_i$ is the i^{th} component of the test function. On the $S_{t,i}$ subset the traction component i is known from (11.37), but on the $S_{u,i}$ subset of the bounding surface the traction, the component i of the traction is not known, it represents the reaction. The same trick as for the other PDE models may be resorted to, namely to put in place a requirement

$$(\boldsymbol{\eta})_i = 0 \quad \text{on} \quad S_{u,i}\ .$$

Therefore, with the constraint on the trial function to satisfy the essential boundary conditions, and a constraint on the test function to vanish on $S_{u,i}$, we have the weighted balance residual equation

$$\int_V \boldsymbol{\eta}\cdot\rho\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\ \mathrm{d}V - \int_V \boldsymbol{\eta}\cdot\overline{\boldsymbol{b}}\ \mathrm{d}V - \sum_{i=x,y,z}\int_{S_{t,i}} (\boldsymbol{\eta})_i\,\big(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\big)_i\ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\ \mathrm{d}V = 0 \tag{12.8}$$

and using the natural boundary condition (12.4) (and subsuming this condition in the balance residual equation), we obtain the ***final form of the weighted residual equation***

$$\begin{gathered}\int_V \boldsymbol{\eta}\cdot\rho\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\ \mathrm{d}V - \int_V \boldsymbol{\eta}\cdot\overline{\boldsymbol{b}}\ \mathrm{d}V - \sum_{i=x,y,z}\int_{S_{t,i}} (\boldsymbol{\eta})_i\,\overline{t}_i\ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\ \mathrm{d}V = 0\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\end{gathered} \tag{12.9}$$

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

$$\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \quad \Rightarrow \quad \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}^2\boldsymbol{v}}{\mathrm{d}t^2} = \ddot{\boldsymbol{u}} \ ,$$

and the constitutive equation (11.34) and the displacement-strain relation (11.32), the form that goes into the discretization process reads

$$\int_V \boldsymbol{\eta} \cdot \rho\ddot{\boldsymbol{u}} \, \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \bar{\boldsymbol{b}} \, \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \bar{t}_i \, \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\mathcal{B}\boldsymbol{u} \, \mathrm{d}V = 0 \quad (12.10)$$

$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

12.2 Method of weighted residuals as the Principle of Virtual Work

An alternative route to equation (12.9) is via the principle of virtual work. The validity of (12.5) is postulated as a principle: the ***principle of virtual work***. The test function is interpreted as ***virtual displacement***. The constraint on the test function is postulated a priori: virtual displacement is ***kinematically admissible***. The various terms in (12.9) are interpreted as the virtual work of the inertial forces, applied body load, applied tractions, and internal forces.

This approach tends to seem completely 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).

12.3 Discretizing

The primary variable, the displacement, is a vector quantity

$$\boldsymbol{u} = u_x\boldsymbol{e}_x + u_y\boldsymbol{e}_y + u_z\boldsymbol{e}_z \ ,$$

where $u_x, \ldots$ are the components, and $\boldsymbol{e}_x, \ldots$ are the basis vectors. All computer manipulations are performed in terms of components, the basis vectors are necessary only when a transition needs to be made from one basis to another. For simplicity, in this work and in the toolbox `SOFEA`, the basis in which the *displacement field* is expressed is the *global Cartesian basis*.

12.3.1 The trial function

Therefore, the *trial* displacement vector function will be expressed in terms of the components in the global Cartesian basis (the basis is implied) as (compare with Sections 2.7 and 6.5)

$$[\boldsymbol{u}(\boldsymbol{x},t)] = \begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \sum_{i=1}^{N} N_i(\boldsymbol{x})[\boldsymbol{u}_i(t)] = \sum_{i=1}^{N} N_i(\boldsymbol{x}) \begin{bmatrix} u_{ix}(t) \\ u_{iy}(t) \\ u_{iz}(t) \end{bmatrix} . \tag{12.11}$$

Here $N_i(\boldsymbol{x})$ is a finite element basis function (at this point we assume it is defined on a three-dimensional mesh), and $u_{ix}(t), \ldots$ are nodal degrees of freedom (displacements at nodes) as functions of time. As in Section 6.5), it will be useful to separate the free degrees of freedom from the prescribed displacements. However, this expression will be complicated by the fact that now we have three components, each of which has different sets of the free degrees of freedom and the prescribed ones. As an illustration consider Figure 12.1: for the x component, the free degrees of freedom are u_{2x}, the prescribed (at zero value) are u_{1x}, u_{3x}; for the y component, the free degrees of freedom are u_{3y}, the prescribed (at zero value) are u_{1y}, u_{2y}.

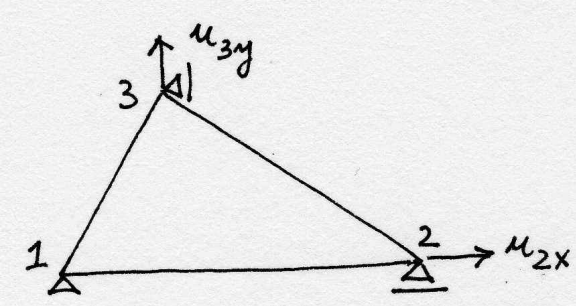

Fig. 12.1. A single element, with free degrees of freedom.

One possibility is to write

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \begin{bmatrix} \sum\limits_{\text{free } i} N_i(\boldsymbol{x})u_{ix}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x})\overline{u}_{ix}(t) \\ \sum\limits_{\text{free } i} N_i(\boldsymbol{x})u_{iy}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x})\overline{u}_{iy}(t) \\ \sum\limits_{\text{free } i} N_i(\boldsymbol{x})u_{iz}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x})\overline{u}_{iz}(t) \end{bmatrix} , \tag{12.12}$$

but a more convenient approach is the following trick:

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \sum_{\text{all } i} N_i(\boldsymbol{x}) \left\{ \begin{bmatrix} u_{ix}(t) \\ u_{iy}(t) \\ u_{iz}(t) \end{bmatrix} + \begin{bmatrix} \overline{u}_{ix}(t) \\ \overline{u}_{iy}(t) \\ \overline{u}_{iz}(t) \end{bmatrix} \right\} , \tag{12.13}$$

where we define

$u_{ix}(t) = 0$ if the x degree of freedom at node i is prescribed;
$u_{iy}(t) = 0$ if the y degree of freedom at node i is prescribed;
$u_{iz}(t) = 0$ if the z degree of freedom at node i is prescribed,

and

$$\overline{u}_{ix}(t) = 0 \text{ if the } x \text{ degree of freedom at node } i \text{ is free;}$$
$$\overline{u}_{iy}(t) = 0 \text{ if the } y \text{ degree of freedom at node } i \text{ is free;}$$
$$\overline{u}_{iz}(t) = 0 \text{ if the } z \text{ degree of freedom at node } i \text{ is free.}$$

Thus, for the example of Figure 12.1 we have the following

$$\begin{bmatrix} u_{1x}(t) \\ u_{1y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{1x}(t) \\ \overline{u}_{1y}(t) \end{bmatrix} = \begin{bmatrix} \text{as given} \\ \text{as given} \end{bmatrix},$$

for node 1,

$$\begin{bmatrix} u_{2x}(t) \\ u_{2y}(t) \end{bmatrix} = \begin{bmatrix} u_{2x}(t) \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{2x}(t) \\ \overline{u}_{2y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ \text{as given} \end{bmatrix},$$

for node 2, and finally,

$$\begin{bmatrix} u_{3x}(t) \\ u_{3y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ u_{3y}(t) \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{3x}(t) \\ \overline{u}_{3y}(t) \end{bmatrix} = \begin{bmatrix} \text{as given} \\ 0 \end{bmatrix},$$

for node 3. In the toolbox code, the free degrees of freedom may be distinguished from the prescribed ones using the attributes of the `field` class. The free degrees of freedom get nonzero equation numbers (attribute `eqnums`), and the magnitudes are stored in the attribute `values`; the prescribed degrees of freedom are marked with the attribute `is_prescribed` and the value to which these degrees of freedom are being set is the `prescribed_value`. The `gather()` method of the `field` class may be used to retrieve all these attributes.

12.3.2 The test function

Using the trick described below equation (12.13), we will write the test function as

$$\begin{bmatrix} \eta_x(\boldsymbol{x}) \\ \eta_y(\boldsymbol{x}) \\ \eta_z(\boldsymbol{x}) \end{bmatrix} = \sum_{\text{all } i} N_i(\boldsymbol{x}) \begin{bmatrix} \eta_{ix} \\ \eta_{iy} \\ \eta_{iz} \end{bmatrix}, \tag{12.14}$$

where we define

$$\eta_{ix} = 0 \text{ if the } x \text{ degree of freedom at node } i \text{ is prescribed;}$$
$$\eta_{iy} = 0 \text{ if the } y \text{ degree of freedom at node } i \text{ is prescribed;}$$
$$\eta_{iz} = 0 \text{ if the } z \text{ degree of freedom at node } i \text{ is prescribed.}$$

Note that $\eta_{ix}, \eta_{iy}, \eta_{iz}$ are arbitrary numbers (except when they are forced to be zero as shown above). We will also use a more succinct version

$$\begin{bmatrix} \eta_x(\boldsymbol{x}) \\ \eta_y(\boldsymbol{x}) \\ \eta_z(\boldsymbol{x}) \end{bmatrix} = [\eta(\boldsymbol{x})] = \sum_{\text{all } i} N_i(\boldsymbol{x}) \begin{bmatrix} \eta_i \end{bmatrix}, \tag{12.15}$$

where $[\eta_i]$ stands for a column matrix holding the components $[\eta_i]_k$ of the vector of the degrees of freedom at node i.

12.3.3 Producing the requisite equations

It may be easily verified that the definition of the test function gives us just enough equations to solve for all the free degrees of freedom. Use (12.14) in equation (12.10) to obtain

$$\begin{aligned}\int_V [\boldsymbol{\eta}]^T \rho[\ddot{\boldsymbol{u}}]\,\mathrm{d}V - \int_V [\boldsymbol{\eta}]^T [\overline{\boldsymbol{b}}]\,\mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i\, \overline{t}_i\,\mathrm{d}S \\ + \int_V (\mathcal{B}[\boldsymbol{\eta}])^T \boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\,\mathrm{d}V = 0 \qquad (12.16)\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\end{aligned}$$

The key is to obtain $[\boldsymbol{\eta}]^T$ in all the terms. The trickiest seems to be $\mathcal{B}[\boldsymbol{\eta}]$. Substituting (12.14), we get

$$\mathcal{B}[\boldsymbol{\eta}] = \mathcal{B}\sum_{\text{all } j} N_j(\boldsymbol{x}) \begin{bmatrix} \eta_{jx} \\ \eta_{jy} \\ \eta_{jz} \end{bmatrix},$$

but since the $\eta_{jx}, \eta_{jy}, \eta_{jz}$'s are just numbers, the symmetric gradient operator works with the basis functions, as if we simply multiplied the matrix of the operator with a scalar

$$\mathcal{B}[\boldsymbol{\eta}] = \sum_{\text{all } j} \mathcal{B}\left(N_j(\boldsymbol{x})\right) \begin{bmatrix} \eta_{jx} \\ \eta_{jy} \\ \eta_{jz} \end{bmatrix}.$$

We are ready to proceed: the test function is substituted as

$$\begin{aligned}\sum_{\text{all } j} [\eta_j]^T \int_V N_k(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]\,\mathrm{d}V - \sum_{\text{all } j} [\eta_j]^T \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]\,\mathrm{d}V \\ - \sum_{i=x,y,z} \sum_{\text{all } j} ([\eta_j])_i \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i\,\mathrm{d}S \\ - + \sum_{\text{all } j} [\eta_j]^T \int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right) \boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\,\mathrm{d}V = 0 \qquad (12.17)\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\end{aligned}$$

The components are independent, hence (12.17) should hold for each component separately

$$\begin{aligned}\sum_{\text{all } j} [\eta_j]_i \int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i\,\mathrm{d}V - \sum_{\text{all } j} [\eta_j]_i \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i\,\mathrm{d}V \\ - \sum_{\text{all } j} [\eta_j]_i \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i\,\mathrm{d}S + \sum_{\text{all } j} [\eta_j]_i \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right) \boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i\,\mathrm{d}V = 0 \qquad (12.18)\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\end{aligned}$$

Collecting all the sums yields finally

$$\sum_{\text{all } j}[\eta_j]_i \left\{ \int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \,\mathrm{d}V - \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \,\mathrm{d}V \right.$$
$$\left. - \int_{S_{t,i}} N_j(\boldsymbol{x})\bar{t}_i \,\mathrm{d}S + \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V \right\} = 0 \tag{12.19}$$
$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

We are ready for the punchline: some of the $[\eta_j]_i$ components are zero. These components do not field any equations, they are simply ignored. The components $[\eta_j]_i$ that are not zero are completely arbitrary. Therefore, the interiors of the braces have to vanish identically, yielding *one equation for each free degree of freedom.*

$$\int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \,\mathrm{d}V - \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \,\mathrm{d}V$$
$$- \int_{S_{t,i}} N_j(\boldsymbol{x})\bar{t}_i \,\mathrm{d}S + \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V = 0 \tag{12.20}$$
$$\text{where} \quad u_i = \overline{u}_i; \quad \text{for all } j, \quad \text{and } i = x, y, z, \qquad \text{such that } [\eta_j]_i \neq 0 \,.$$

Note that $[\eta_j]_i = 0$ whenever node j is on the boundary $S_{u,i}$. The meaning of these equations is ***dynamic force equilibrium***.

12.4 The discrete equations: system of ODE's

Finally, we substitute the trial function from (12.13). We will use the more streamlined version

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = [u(\boldsymbol{x},t)] = \sum_{\text{all } k} N_k(\boldsymbol{x}) \left\{ \left[u_k(t) \right] + \left[\overline{u}_k(t) \right] \right\} \,, \tag{12.21}$$

where $[u_k]$ stands for a column matrix holding the components $[u_k]_m$ of the vector of the free degrees of freedom at node k; analogously for the prescribed degrees of freedom. To satisfy the essential boundary conditions by interpolation, we set for the prescribed m^{th} component at node k

$$[\overline{u}_k(t)]_m = [\overline{u}(\boldsymbol{x}_k, t)]_m$$

where $\boldsymbol{x}_k$ is the location of node k.

The trial function is substituted into (12.20). To keep things orderly and clear, we will substitute term by term.

12.4.1 Inertial term: Mass matrix

Starting with the first:

$$\int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \, \mathrm{d}V =$$
$$\int_V N_j(\boldsymbol{x})\rho \sum_{\text{all } k} N_k(\boldsymbol{x}) \left\{ \left[\ddot{u}_k(t) \right]_i + \left[\overline{\ddot{u}}_k(t) \right]_i \right\} \, \mathrm{d}V =$$
$$\sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \left\{ \left[\ddot{u}_k(t) \right]_i + \left[\overline{\ddot{u}}_k(t) \right]_i \right\} =$$
$$\sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \left[\ddot{u}_k(t) \right]_i + \sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \left[\overline{\ddot{u}}_k(t) \right]_i \quad (12.22)$$

Two terms emerge: firstly, the ***inertial force***

$$F_{a,(j,i)} = \sum_{\text{all } k} M_{(j,i)(k,m)} \left[\ddot{u}_k(t) \right]_m \ , \quad (12.23)$$

produced by the free accelerations which are coupled together by the ***consistent mass matrix***

$$M_{(j,i)(k,m)} = \left[\int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) [\underset{3\times 3}{\mathbf{1}}] \, \mathrm{d}V \right]_{im} \ , \quad (12.24)$$

where (j,i) means *equation number* corresponding to component i at node j (which is a free degree of freedom).

Secondly, there is the ***inertial load*** produced by the acceleration of the supported nodes

$$F_{\overline{a},(j,i)} = \sum_{\text{all } k} \overline{M}_{(j,i)(k,m)} \left[\overline{\ddot{u}}_k(t) \right]_m \ , \quad (12.25)$$

where the matrix elements $\overline{M}_{(j,i)(k,m)}$ are calculated exactly as those of (12.24), but (k,m) corresponds to component m at node k, which is prescribed.

12.4.2 Body loads and traction loads

The next two terms represent external loads. The ***body load vector*** component (j,i) corresponding to free component i at node j is

$$F_{b,(j,i)} = \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \, \mathrm{d}V \ . \quad (12.26)$$

The ***surface traction load vector*** component (j,i) corresponding to free component i at node j is

$$F_{t,(j,i)} = \int_{S_{t,i}} N_j(\boldsymbol{x})\bar{t}_i \, \mathrm{d}S \ . \quad (12.27)$$

12.4.3 Resisting forces: Stiffness matrix

Finally, the trial function (12.21) is substituted into the last term of (12.20).

$$\begin{aligned}
&\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V = \\
&\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\sum_{\text{all } k} N_k(\boldsymbol{x})\left\{\left[u_k(t)\right]+\left[\overline{u}_k(t)\right]\right\}\right]_i \,\mathrm{d}V = \\
&\sum_{\text{all } k}\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\left\{\left[u_k(t)\right]+\left[\overline{u}_k(t)\right]\right\}\right]_i \,\mathrm{d}V = \\
&\sum_{\text{all } k}\sum_{m}\left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,\mathrm{d}V\right]_{im}\left\{\left[u_k(t)\right]_m+\left[\overline{u}_k(t)\right]_m\right\}
\end{aligned} \tag{12.28}$$

Two contributions result: the first is the ***resisting force*** produced by the deformed material.

$$F_{r,(j,i)} = \sum_{\text{all } k} K_{(j,i)(k,m)}\left[u_k(t)\right]_m \ . \tag{12.29}$$

The matrix generating the resisting force is the ***stiffness matrix***

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,\mathrm{d}V\right]_{im} \ . \tag{12.30}$$

where (j,i) $[(k,m)]$ means equation number corresponding to component i at node j (component m at node k); both are free degrees of freedom.

The second is the ***nonzero-displacement load*** produced by the deformation induced by prescribed essential boundary conditions. (Remark: In structural analysis, this kind of load is frequently associated with the loading condition called the *support settlement*.)

$$F_{\overline{r},(j,i)} = \sum_{\text{all } k} \overline{K}_{(j,i)(k,m)}\left[\overline{u}_k(t)\right]_m \ . \tag{12.31}$$

The elements $\overline{K}_{(j,i)(k,m)}$ are computed exactly as in (12.30), but (k,m) corresponds to component m at node k, which is prescribed.

12.4.4 Summary of the elastodynamics ODE's

Summing the forces (12.23), (12.25), (12.26), (12.27), (12.29), and (12.31) yields a system of second-order ordinary differential equations for the free displacements $\left[\ddot{u}_k(t)\right]_m$

$$\sum_{\text{all } k} M_{(j,i)(k,m)} \left[\ddot{u}_k(t)\right]_m + \sum_{\text{all } k} K_{(j,i)(k,m)} \left[u_k(t)\right]_m =$$
$$-\sum_{\text{all } k} \overline{M}_{(j,i)(k,i)} \left[\ddot{\overline{u}}_k(t)\right]_i - \sum_{\text{all } k} \overline{K}_{(j,i)(k,m)} \left[\overline{u}_k(t)\right]_m + F_{b,(j,i)} + F_{t,(j,i)} \ . \quad (12.32)$$

In the convenient matrix notation, we could write

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{L} \ , \quad (12.33)$$

where $\boldsymbol{U}$ collects all the free degrees of freedom. These equations could be directly integrated using a Matlab integrator as indicated in Section 3.4, or even 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.

12.5 Constitutive equations of linearly elastic materials

The strain displacement operator(symmetric gradient operator) (11.32) 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 perpendicularly 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 (6.49) (except that the transformation matrix has three columns).

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: the ***general anisotropic material***. Still, to identify all of these constants represents a major experimental effort, and few engineering materials are characterized as fully anisotropic.

12.5.1 Orthotropic material.

If a material has only three 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

$$\boldsymbol{C} = \begin{bmatrix} {E_1}^{-1}, & -\frac{\nu_{12}}{E_1}, & -\frac{\nu_{13}}{E_1}, & 0, & 0, & 0 \\ -\frac{\nu_{12}}{E_1}, & {E_2}^{-1}, & -\frac{\nu_{23}}{E_2}, & 0, & 0, & 0 \\ -\frac{\nu_{13}}{E_1}, & -\frac{\nu_{23}}{E_2}, & {E_3}^{-1}, & 0, & 0, & 0 \\ 0, & 0, & 0, & {G_{12}}^{-1}, & 0, & 0 \\ 0, & 0, & 0, & 0, & {G_{13}}^{-1}, & 0 \\ 0, & 0, & 0, & 0, & 0, & {G_{23}}^{-1} \end{bmatrix}.$$

All nine coefficients are independent, and needs to be provided as input [**?**]. The material stiffness matrix is a bit of a mess. The nonzero elements are

$$D_{11} = \frac{C_{22}C_{33} - C_{23}C_{23}}{C}, \quad D_{12} = \frac{C_{13}C_{23} - C_{12}C_{33}}{C}, \quad D_{13} = \frac{C_{12}C_{23} - C_{13}C_{22}}{C},$$
$$D_{22} = \frac{C_{33}C_{11} - C_{13}C_{13}}{C}, \quad D_{23} = \frac{C_{12}C_{13} - C_{23}C_{11}}{C}, \quad D_{33} = \frac{C_{11}C_{22} - C_{12}C_{12}}{C},$$
$$D_{44} = G_{12}, \qquad D_{55} = G_{13}, \qquad D_{66} = G_{23}.$$

Here $C = C_{11}C_{22}C_{33} - C_{11}C_{23}C_{23} - C_{22}C_{13}C_{13} - C_{33}C_{12}C_{12} + 2C_{12}C_{23}C_{13}$.

12.5.2 Transversely isotropic material.

If a material has an infinite number of planes of symmetry passing through an axis (in this case, local x-axis), and one plane of symmetry perpendicular to this axis, it is known as an ***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 perpendicular to the fibers. Layered materials are also modeled as transversely isotropic: here the direction perpendicular to the layers is special. Figure 12.2 offers an illustration of these two types.

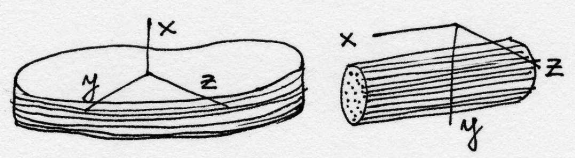

Fig. 12.2. Transversely isotropic model is appropriate for layered or fiber-reinforced materials

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}.

12.5.3 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

$$\boldsymbol{C} = \begin{bmatrix} E^{-1}, & -\frac{\nu}{E}, & -\frac{\nu}{E}, & 0, & 0, & 0 \\ -\frac{\nu}{E}, & E^{-1}, & -\frac{\nu}{E}, & 0, & 0, & 0 \\ -\frac{\nu}{E}, & -\frac{\nu}{E}, & E^{-1}, & 0, & 0, & 0 \\ 0, & 0, & 0, & G^{-1}, & 0, & 0 \\ 0, & 0, & 0, & 0, & G^{-1}, & 0 \\ 0, & 0, & 0, & 0, & 0, & G^{-1} \end{bmatrix}.$$

The shear modulus G is not independent, but is expressed as $G = E/2/(1+\nu)$. 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 $\lambda = E\nu/(1+\nu)/(1-2\nu)$ for convenience.

12.6 Imposed (thermal) strains

Often the material from which a structure is built up reacts to the environment by deformation. The material around it experiences the environment in possibly different ways or different measures, and stresses are produced. To simplify matters, 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. To fix ideas, 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

$$[\boldsymbol{\epsilon}_\Theta] = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \Delta T \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \\ 0 \\ 0 \end{bmatrix} . \tag{12.34}$$

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.

Let us now consider a more general effect of environmentally-imposed strains on the small sample. 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 a 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

$$\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_{\text{total}} = \boldsymbol{\epsilon}_\Theta + \boldsymbol{C}\boldsymbol{\sigma} . \tag{12.35}$$

Therefore, we have a modification of the constitutive equation

$$\boldsymbol{\sigma} = \boldsymbol{D}\left(\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\Theta\right) . \tag{12.36}$$

The total strain is related to the displacement via the strain-displacement relation (11.32), and thus we may write

$$\boldsymbol{\sigma} = \boldsymbol{D}\left(\mathcal{B}\boldsymbol{u} - \boldsymbol{\epsilon}_\Theta\right) .$$

This is sometimes written as

$$\boldsymbol{\sigma} = \boldsymbol{D}\mathcal{B}\boldsymbol{u} + \boldsymbol{\sigma}_\Theta ,$$

where we introduce the so-called ***thermal stress*** $\boldsymbol{\sigma}_\Theta$, but purely as a convenience; the primary quantity is the measurable thermal strain.

As expected, the residual equation form that enters the discretization process needs to be augmented with respect to (12.10) to become

$$\int_V \boldsymbol{\eta} \cdot \rho\ddot{\boldsymbol{u}} \, \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \, \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \overline{t}_i \, \mathrm{d}S \tag{12.37}$$

$$+ \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\left(\mathcal{B}\boldsymbol{u} - \boldsymbol{\epsilon}_\Theta\right) \, \mathrm{d}V = 0 \tag{12.38}$$

$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

Clearly, there will be one more term to discretize

$$- \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\boldsymbol{\epsilon}_\Theta \, \mathrm{d}V , \tag{12.39}$$

yielding the ***thermal strain load***

$$F_{\Theta,(j,i)} = \left[\int_V \mathcal{B}^T \left(N_j(\boldsymbol{x}) \right) \boldsymbol{D} \boldsymbol{\epsilon}_\Theta \, \mathrm{d}V \right]_i \, . \tag{12.40}$$

Therefore, with the inclusion of the thermal strains, the system of ordinary differential equations (12.32) that describe the discrete problem becomes

$$\begin{aligned} \sum_{\text{all } k} M_{(j,i)(k,m)} \left[\ddot{u}_k(t) \right]_m + \sum_{\text{all } k} K_{(j,i)(k,m)} \left[u_k(t) \right]_m = \\ - \sum_{\text{all } k} \overline{M}_{(j,i)(k,i)} \left[\ddot{\overline{u}}_k(t) \right]_i - \sum_{\text{all } k} \overline{K}_{(j,i)(k,m)} \left[\overline{u}_k(t) \right]_m \\ + F_{b,(j,i)} + F_{t,(j,i)} + F_{\Theta,(j,i)} \, . \end{aligned} \tag{12.41}$$

12.7 Strain-displacement matrix

The elements of the stiffness matrix (12.30) are evaluated using numerical quadrature element-by-element (explained in detail in Section 6.8 for the conductivity matrix). This operation produces element-level stiffness matrices that couple together the finite element nodes of each element. To simplify the notation, we will tentatively define the strain-displacement matrix (one further refinement follows)

$$\boldsymbol{B}_k^e = \mathcal{B} \left(N_k(\boldsymbol{x}) \right) \quad \text{for } \boldsymbol{x} \in \text{element } e \, , \tag{12.42}$$

where k indicates the node number, and e identifies the element (it bears emphasis that the strain-displacement matrix of node k is different in each element that shares this node).

Consider now a structure that consists of a single element, say a tetrahedron with four nodes, A, B, C and D (Figure 12.3). Let us also assume that all the degrees of freedom are free (hypothetically: if they really were, the stiffness matrix would be singular). The stiffness matrix of such a structure (i. e. of this single element) has 12 rows and columns.

$$K_{(j,i)(k,m)} = \left[\int_{V_e} \boldsymbol{B}_j^{eT} \boldsymbol{D} \boldsymbol{B}_k^e \, \mathrm{d}V \right]_{im} \, , \quad j, i = A, B, C, D \, . \tag{12.43}$$

For j, k fixed, say $j = B$ and $k = D$, the matrix

$$\int_{V_e} \boldsymbol{B}_B^{e\,T} \boldsymbol{D} \boldsymbol{B}_D^e \, \mathrm{d}V \, , \tag{12.44}$$

is a 3×3 submatrix of the element stiffness matrix: compare with the illustration in Figure 12.3 (the off-diagonal block). Similarly, for $j = B$ and $k = B$ we would obtain the diagonal block. Therefore, the whole element stiffness matrix may be computed in one shot as

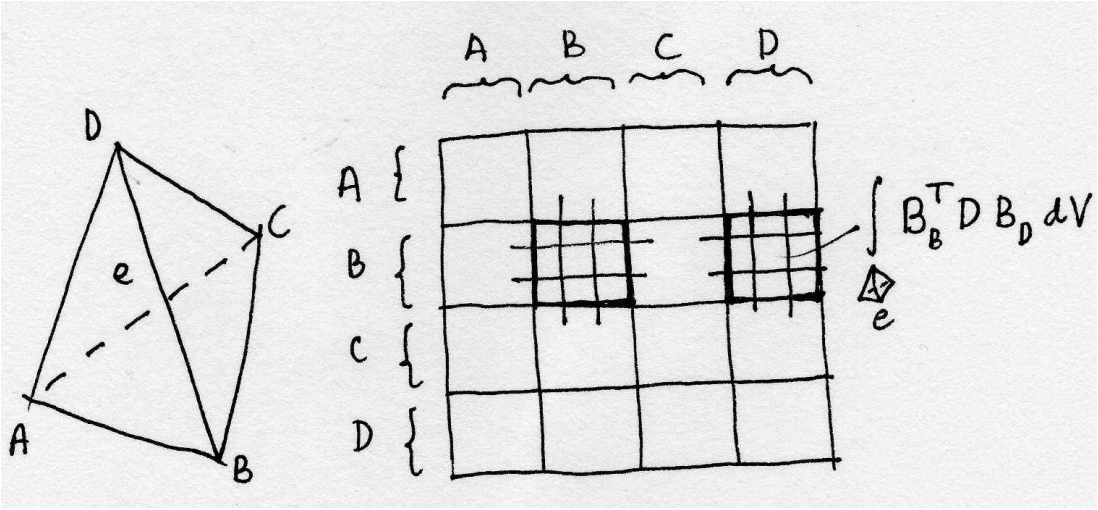

Fig. 12.3. Assembly of the element stiffness matrix

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \, \mathrm{d}V \ , \tag{12.45}$$

using the blocked matrix (the elementwise strain-displacement matrix)

$$\boldsymbol{B}_e = [\boldsymbol{B}_A^e, \boldsymbol{B}_B^e, \boldsymbol{B}_C^e, \boldsymbol{B}_D^e] \ . \tag{12.46}$$

Correspondingly, the displacements at the notes will be ordered into a column vector of displacement components in the global Cartesian basis (with the Matlab syntax: semicolon means new line)

$$\boldsymbol{U}_e = [[u_A]; [u_B]; [u_C]; [u_D]] \ . \tag{12.47}$$

12.7.1 Transformation of basis

It remains to discuss the issue of the choice of coordinate systems when evaluating the stiffness matrix integrals. As discussed in Section 12.5, the components of the material stiffness matrix are expressed on the local Cartesian basis of material orientation directions. Referring to the definition of the stiffness matrix (12.30), we see that there is a need to mediate between the local material directions and the global Cartesian basis. Drawing on the example of the element stiffness matrix (12.45), the element's restoring force may be expressed as

$$\boldsymbol{F}_e = \boldsymbol{K}_e \boldsymbol{U}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \, \mathrm{d}V \, \boldsymbol{U}_e \ , \tag{12.48}$$

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 another.

There are several ways in which this transformation could be carried out. In the SOFEA toolbox the following approach is used: the strain-displacement matrix is defined to produce strains in the local material basis, while taking displacement components in the global basis. This can be accomplished writing

$$[\boldsymbol{\epsilon}]^{(\overline{x})} = \mathcal{B}^{\overline{x}}[\boldsymbol{u}]^{(\overline{x})} = \mathcal{B}^{(\overline{x})}\left([T][\boldsymbol{u}]^{(x)}\right) = \left(\mathcal{B}^{(\overline{x})}[T]\right)[\boldsymbol{u}]^{(x)} = \mathcal{B}^{(\overline{x},x)}[\boldsymbol{u}]^{(x)} \tag{12.49}$$

where we use the superscript $(\overline{x})$ or (x) to indicate in which coordinate system the components are expressed. The strain-displacement operator $\mathcal{B}^{(\overline{x},x)} = \mathcal{B}^{(\overline{x})}[T]$ incorporates the geometric transformation $[T]$ of the displacement vector components from the global basis into the local material directions basis. This transformation may be derived from the equality

$$\left[\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z\right] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \left[\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}\right] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} . \tag{12.50}$$

Pre-multiplying with

$$\begin{bmatrix} \boldsymbol{e}_{\overline{x}} \\ \boldsymbol{e}_{\overline{y}} \\ \boldsymbol{e}_{\overline{z}} \end{bmatrix}$$

leads to the transformation of vector components

$$[\boldsymbol{R}_m]^T \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = [T] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} . \tag{12.51}$$

The transformation $[T]$ is thus identified with an orthogonal (rotation) matrix

$$[\boldsymbol{R}_m] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} , \tag{12.52}$$

which is just a three-dimensional analog of the two-dimensional transformation (6.49).

The global-to-local ***strain-displacement matrix*** is therefore defined as

$$\boldsymbol{B}_k^e = \mathcal{B}^{(\overline{x},x)}\left(N_k(\boldsymbol{x})\right) = \mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)[\boldsymbol{R}_m]^T \quad \text{for } \boldsymbol{x} \in \text{element } e\ . \tag{12.53}$$

The $\boldsymbol{B}_k^e$ matrix is used to compose the element-wise strain-displacement matrix (12.46) In the SOFEA toolbox, the strain-displacement matrix is computed for a three-dimensional solid geometric cell by the method Blmat:

```
0023 function B = Blmat[1](self, pc, x, Rm)
```

[1] Folder: SOFEA/classes/gcell/@gcell_3_manifold

```
0024      Nder = Ndermat_param (self, pc);
0025      Ndersp=Ndermat_spatial(self,Nder,x*Rm);
0026      nfens = size(Ndersp, 1);
0027      dim=3;
0028      B = zeros(6,nfens*dim);
0029      for i= 1:nfens
0030          B(:,dim*(i-1)+1:dim*i)...
0031              = [ Ndersp(i,1) 0           0  ; ...
0032              0           Ndersp(i,2) 0  ; ...
0033              0           0           Ndersp(i,3) ; ...
0034              Ndersp(i,2) Ndersp(i,1) 0  ; ...
0035              Ndersp(i,3) 0           Ndersp(i,1) ; ...
0036              0           Ndersp(i,3) Ndersp(i,2) ]*Rm';
0037      end
0038      return;
0039  end
```

The input arguments are the parametric coordinates `pc`, the array of nodal coordinates `x` (the 3-D version of (6.39)), and the transformation matrix $\boldsymbol{R}_m$. Comparing with the definitions (12.53) and (12.46), the Matlab code is an almost literal translation of the formulas. Note that the global location vectors for each node `x` are transformed into the local Cartesian basis of material directions, `x*Rm`, as the matrix $\mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)$ in equation (12.53) requires derivatives with respect to the material basis.

It is of interest to note that the method `Blmat` is for solid elements defined for the `gcell_3_manifold` class. Analogously, one-dimensional and two-dimensional elements have their versions defined by the `gcell_1_manifold` and `gcell_2_manifold` classes; in other words, it is not specific to any particular element type or shape.

12.8 Stiffness matrix

In Chapter 3 and further and Section 6.8, it was pointed out and discussed that all the problem dependent code is concentrated in a descendent of the `feblock` class. The elastodynamics model of this chapter is implemented in the `feblock_defor_ss` finite element block class.

The stiffness matrix (12.30) is constructed from element-wise stiffness matrices (12.45). These are calculated by the `stiffness` method, and returned in the `ems` array. The `stiffness` method takes as arguments the geometry of the mesh (field `geom`), in order to have access to the locations of the nodes, and the displacement field `u` to provide the global equation numbers of the unknowns.

```
0008 function ems = stiffness[2] (self, geom, u)
0009      gcells = get(self.feblock,'gcells');
0010      nfens = get(gcells(1),'nfens');
```

[2] Folder: `SOFEA/classes/feblock/@feblock_defor_ss`

```
0011      ems(1:length(gcells)) = deal(elemat);% Pre-allocate
0012      % Integration rule
0013      integration_rule = get(self.feblock, 'integration_rule');
0014      pc = get(integration_rule, 'param_coords');
0015      w  = get(integration_rule, 'weights');
0016      npts_per_gcell = get(integration_rule, 'npts');
0017      % Material
0018      mat = get(self.feblock, 'mater');
```

The material stiffness matrix is pre-allocated, and the loop over all the geometric cells within the block begins. The connectivity of the geometric cell is retrieved, and, based on the connectivity, the locations of the nodes `x` are gathered from the geometry field `geom`. Finally, the matrix `Ke` is zeroed out in preparation for the numerical integration.

```
0019      Ke = zeros(get(geom,'dim')*nfens); % preallocate
0020      % Now loop over all gcells in the block
0021      for i=1:length(gcells)
0022          conn = get(gcells(i), 'conn'); % connectivity
0023          x = gather(geom, conn, 'values', 'noreshape'); % coord
0024          Ke = 0*Ke; % zero out element stiffness
```

The loop over the quadrature points begins by calculating the basis functions in order to be able to evaluate the location of the current integration point.

```
0025          % Loop over all integration points
0026          for j=1:npts_per_gcell
0027              N = Nmat(gcells(i),pc(j,:));
```

The matrix of the local material directions (material orientation matrix) is computed. It may depend on the location of the quadrature point, `xyz`, and/or on the tangents to the parametric curves through the quadrature point, `x'*Nder`. If indeed required, these quantities would be calculated by the method `material_directions` from the supplied arguments.

```
0028              Rm = material_directions(self,gcells(i),pc(j,:),x);
```

Compute the Jacobian associated with the integration point, and since the integration is to be performed over the 3-D volume, the method `Jacobian_volume` means to be used; for the solid elements it is identical to the `Jacobian` method.

```
0029              detJ = Jacobian_volume(gcells(i),pc(j,:),x);
```

The strain-displacement matrix for the element is calculated from the spatial gradients of the basis functions, and the material directions matrix.

```
0030              B = Blmat(gcells(i), pc(j,:), x, Rm);
```

The material stiffness matrix is calculated by the material object `mat`, and since it may change from point to point, the location of the current integration point, `xyz`, is passed to the method.

```
0031              D = tangent_moduli(mat,struct('xyz',N'*x));
```

The product of the strain-displacement matrix and the tangent moduli (the material stiffness matrix) is accumulated.

```
0032              Ke = Ke + B'*D*B * detJ * w(j);
0033          end
```

Finally, the computed element stiffness matrix is stored in the `ems(i)` object of the `elemat` class. Note that the equation numbers are gathered from the displacement field.

```
0034          ems(i) = set(ems(i), 'mat', Ke);
0035          ems(i) = set(ems(i), 'eqnums', gather(u,conn,'eqnums'));
0036      end
0037      return;
0038  end
```

13

Examples and further developments

With the formulation sketched out, it only remains to pick an element formulation to perform an analysis. We begin with the tetrahedron T4. All the necessary formulas have been put into place in Section 9.5, and we may immediately proceed to the first example.

The equation of motion (12.33) stands for the so-called free vibration when there is no forcing, $\boldsymbol{L} = \boldsymbol{0}$. (Compare also with (3.5).) The built-in Matlab eigenvalue solver will be used to obtain the numerical solution, and the task for the `SOFEA` is relatively simple: compute the stiffness matrix and the mass matrix.

13.1 Modal analysis with the tetrahedron T4: the drum

As our first example, we consider the vibration of a moderately thick circular plate as shown in Figure 13.2. The cylindrical surface is fully clamped (all displacements zero). The material is isotropic. The analytical solution has been worked out in dependence on the number of "nodes" (locations of approximately zero displacement) radially and circumferentially [Blevins]. Therefore, this is a very good example with which to test the finite element formulation, and the finite element T4 in particular.

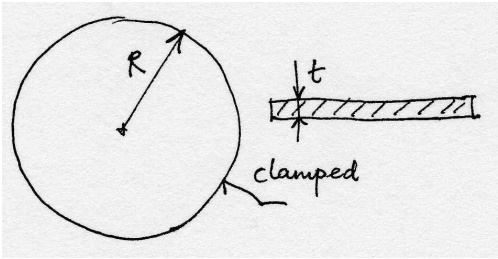

Fig. 13.1. Clamped circular plate drum.

The Matlab script `drum_t4`[1] solves the vibration problem for the four lowest eigenvalues (shown in Figure 13.2). First, a few variables are defined and a mesh is produced

[1] Folder: `SOFEA/examples/stress`

using the mesh function `t4cylinderdel`. The mesh is relatively coarse, with only two element edges through the thickness.

```
0001 E=0.1e6;% Pa
0002 nu=0.3;
0003 rho=1000;% kg
0004 R= 25.0e-3;% m
0005 t= 2.0e-3;% m
0006 rand('state',0);% try to comment out this line and compare
0007 %                   results for several subsequent runs
0008
0009 % Mesh
0010 [fens,gcells] = t4cylinderdel(t,R, 2,7);
```

The material is small-strain (`ss`), linearly elastic (`linel`), isotropic (`iso`), and triaxial (`triax`). The property class `property_linel_iso` supplies methods to compute the material stiffness matrix for this type of material. Each type of material has its own property class, and the material class `mater_defor_ss_linel_triax` undertakes to insulate the methods of the finite element block from the details of the properties. Note that the mass density needs also be supplied (dynamics!).

```
0012 prop = property_linel_iso (struct('E',E,'nu',nu,'rho',rho));
0013 mater = mater_defor_ss_linel_triax (struct('property',prop));
```

The finite element block is of class `feblock_defor_ss`. The integration rule is a one-point quadrature, provided by the class `tet_rule`.

```
0013 % Finite element block
0014 feb = feblock_defor_ss (struct ('mater',mater, 'gcells',gcells,...
0015     'integration_rule',tet_rule (1)));
```

The geometry field and the displacement field are set up as usual. The essential boundary condition is applied to all finite element nodes on the cylindrical surface (line 0024).

```
0018 geom = field(struct ('name',['geom'], 'dim', 3, 'fens',fens));
0019 % Define the displacement field
0020 u   = 0*geom; % zero out
0021 % Apply EBC's
0022 for i=1:length(fens)
0023     xyz = get(fens(i),'xyz');
0024     if abs(R-norm(xyz(2:3))) <0.001*R
0025         u   = set_ebc(u, i, [1], [], 0.0);
0026     end
0027 end
0028 u   = apply_ebc (u);
0029 % Number equations
0030 u   = numbereqns (u);
```

The two methods, `stiffness` and `mass`, discussed previously are invoked. Notice that this will lead to the free vibration problem with a consistent mass matrix.

```
0032 K = start (sparse_sysmat, get(u, 'neqns'));
0033 K = assemble (K, stiffness(feb, geom, u));
0034 M = start (sparse_sysmat, get(u, 'neqns'));
0035 M = assemble (M, mass(feb, geom, u));
```

Finally, the built-in Matlab solver `eigs` is called. Four lowest eigenvalues are requested (the flag `'SM'`). The returned eigenvalues needs to be sorted from the smallest to the largest (line `0039-0040`).

```
0037 neigvs = 4;
0038 [W,Omega]=eigs(get(K,'mat'),get(M,'mat'),neigvs,'SM');
0039 [Omegas,ix]=sort(diag(Omega));
0040 Omega= diag(Omegas);
```

The plotting section of the script is omitted, but the resulting shapes and the calculated frequencies are summarized in Figure 13.2: taking the analytical solution as reference, it is clear that the errors are huge. Evidently, a much more refined mesh would be required to obtain reasonable answers (recall: smaller elements, smaller error). A valuable observation: all the calculated values are above the reference frequencies. This is called "convergence from above", and it is an indication that the discrete model (in other words, the T4 finite element) is *too stiff*.

13.2 Modal analysis with the tetrahedron T4: the composite rod

As the second example we will again consider a free vibration problem, but this time the material of the structure is modeled as transversely isotropic. The structure is a straight rod of circular cross-section, manufactured from carbon fiber infused with polymer resin (Figure 13.3). The reinforcing fibers are twisted, and for lack of other information, we assume that the twist angle decreases from the outside surface (15°) to zero along its axis. The rod is clamped at both ends. Of interest is the lowest natural frequency of the structure.

The Matlab solution of the problem is in the form of a function, `twist_t4`[2], in order to allow for a convergence analysis to be performed on a series of meshes. Therefore, an internal function will be run repeatedly to solve for the first natural frequency for different resolutions radially (number of element edges `nR`) and longitudinally (number of element edges`nt`).

```
0001 function twist_t4
0002     nR = [2, 3, 4];
0003     nt = [30, 40, 50, 60, 70, 80];
0004     fs =zeros(length(nR),length(nt));
0005     for i=1:length(nR)
```

[2]Folder: `SOFEA/examples/stress`

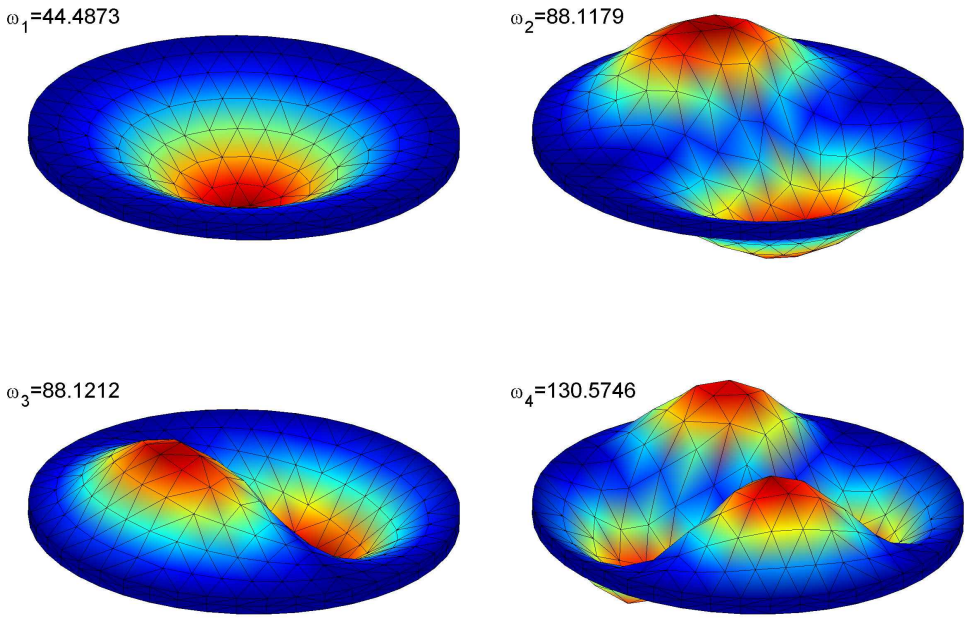

Fig. 13.2. Mode shapes of thick clamped plate. The analytical solution for the natural frequencies yields for these modes $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz

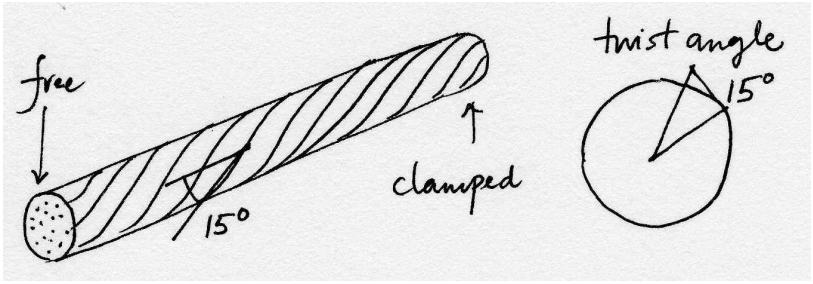

Fig. 13.3. The cantilever Rod.

```
0006          for j=1:length(nt)
0007              fs(i,j) =do_twist_t4(nR(i),nt(j))
0008          end
0009          save fs
0010      end
0011  end
0012
0013
0014  function frequency1 =do_twist_t4(nR,nt)
0015   ...
```

The transversely isotropic material from Section 12.5.2 requires the definition of the local material orientation matrix (12.52) at each integration point. The axis $\overline{x}$ needs to be oriented along the fibers; the orientation of the remaining two basis vectors in the isotropy plane is arbitrary. In the function `twist`, the orientation matrix is derived by first turning a basis triad around the first vector (line 0022-0024), and then twisting the intermediate triad by an angle ramped up from 0° to 15° proportionally to the distance from the axis of the rod (line 0025).

```
0031    function Rm = twist (XYZ, ts)
0032        r= norm(XYZ( 2:3));
0033        if r>0
0034            y=XYZ(2);z=XYZ(3);
0035            e2 = [0, y/r, z/r]';
0036            e3 = skewmat([1, 0, 0])*e2;
0037            Rm= [[1, 0, 0]',e2,e3];
0038            Rm=rotmat(r/R*twist_angle*skewmat(e2))*Rm;
0039        else
0040            Rm= eye(3);
0041        end
0042    end
```

The property class `property_linel_transv_iso` defines the material properties for the transversely isotropic material model. Note that five independent material constants need to be supplied.

```
0046    prop = property_linel_transv_iso (...
0047        struct('E1',E1,
                'E2',E2,'G12',G12,'nu12',nu12,'nu23',nu23,'rho',rho));
0048    mater = mater_defor_ss_linel_triax (struct('property',prop));
```

It is noteworthy that the function `twist` that defines the directions of the local material basis is supplied to the finite element block as a function handle. Call for Section 12.8 for details on the use of the material orientation matrix.

```
0050    feb = feblock_defor_ss (struct ('mater',mater,
                'gcells',gcells,...
0051        'integration_rule',tet_rule (integration_order),
                'Rm',@twist));
```

The rest of the function `twist_t4` is omitted. The shape of the composite rod for a relatively fine mesh is shown in Figure 13.4: notice that the rod twists as well as bends due to the orientation of the reinforcing fibers in the form of a helix.

Let us now address the issue of convergence: The lowest frequency has been computed for a number of meshes with varying number of elements radially, and along the length of the rod. The lowest natural frequency is displayed for these different meshes in Figure 13.5. The frequency varies from approximately 2300Hz to slightly less than 1900Hz. It should be noted that the first natural frequency decreases in

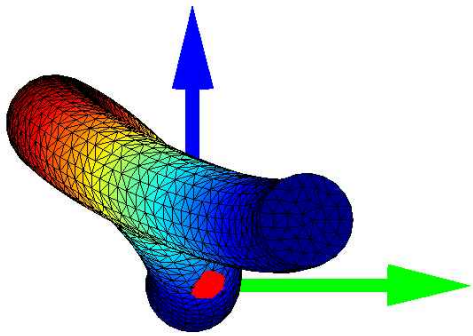

Fig. 13.4. The shape of the first eigenmode.

magnitude with refinement (convergence from above), but the resulting surface is not smooth. (The bad shapes of some elements in the mesh have a profound effect on the accuracy.) The numerical answers are still changing significantly with the refinement, and it is therefore not clear how far from the "exact" solution we might be.

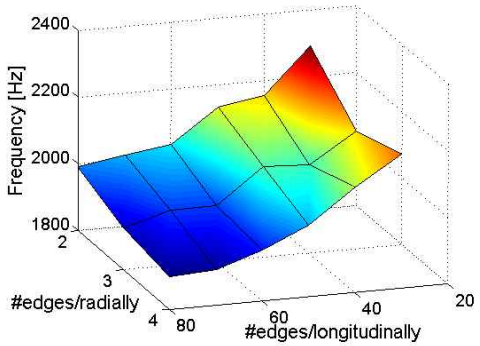

Fig. 13.5. Convergence of the lowest natural frequency with the T4 element.

13.3 Tetrahedron T10

The quadratic tetrahedron is a simple extension of the quadratic triangle to three dimensions. 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

$$\begin{aligned} N_1 &= (1-\xi-\eta-\zeta)(2(1-\xi-\eta-\zeta)-1)\,, \quad N_2 = \xi(2\xi-1)\,, \\ N_3 &= \eta(2\eta-1)\,, \quad N_4 = \zeta(2\zeta-1)\,, \quad N_5 = 4(1-\xi-\eta-\zeta)\xi\,, \\ N_6 &= 4\xi\eta\,, \quad N_7 = 4\eta(1-\xi-\eta-\zeta)\,, \quad N_8 = 4(1-\xi-\eta-\zeta)\zeta\,, \\ & \qquad\qquad N_9 = 4\xi\zeta\,, \quad N_{10} = 4\eta\zeta\,. \end{aligned} \tag{13.1}$$

The basis functions are quadratic in the parametric coordinates. Therefore, their gradients with respect to the parametric coordinates will be linear functions of ξ, η, ζ. Provided the Jacobian matrix in equation (6.37) is constant (independent of ξ, η, ζ), the gradients of the basis functions with respect to x, y, z as computed from (6.32) are going to be linear in those coordinates. Hence, computing the elements of the stiffness matrix can be done exactly with the four-point rule from Table 9.2. On the other hand

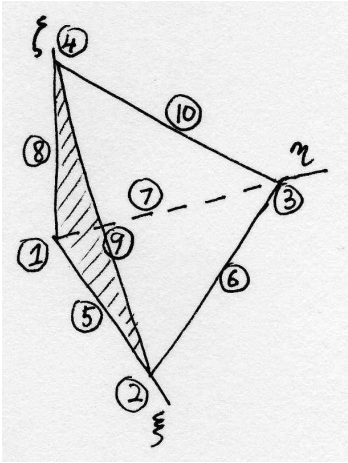

Fig. 13.6. The standard quadratic tetrahedron for the T10 element.

13.3.1 Example: the drum revisited

The Matlab script `drum_t10`[3] is a variation on `drum_t4` which applies the quadratic element instead of T4. The natural frequencies obtained for different meshes with the two elements, T4, and T10, are illustrated in Figure 13.7. While increasing the number of unknowns by making the elements (uniformly) smaller leads generally to more accurate answers, the element T4 is clearly not performing very well. Even for a fairly fine mesh, the results are probably not of much use. On the other hand, the element T10 produces estimates of the frequencies which are of good engineering accuracy. The first frequency is estimated quite well even with the coarsest mesh (only a single element through the thickness, and two elements radially).

The graph of Figure 13.7 may serve as a crude guide to the relative accuracy of the two tetrahedral elements. Sometimes the linear tetrahedron will fare better than in this example, oftentimes much worse.

[3] Folder: `SOFEA/examples/stress/3-D`

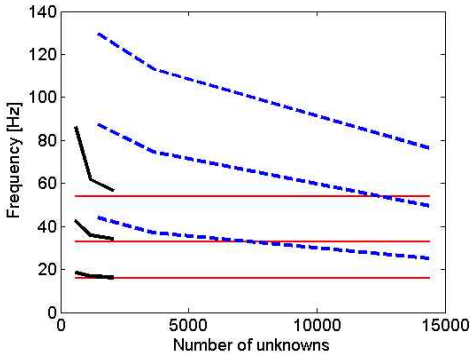

Fig. 13.7. Comparison of the first four natural frequencies of the drum computed with the two tetrahedral elements, the T4 (dashed line) and the T10 (solid line). The analytical solution for the natural frequencies, $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz, is indicated with horizontal lines.

13.4 The composite rod with the tetrahedron T10

The problem from Section 13.2 is addressed here with the quadratic tetrahedron T10. The Matlab function `twist_t10`[4] remarkably differs from `twist_t4` in only two lines. Firstly, the mesh generated for the rod is converted from the type T4 to the type T10 by inserting additional nodes at the midpoints of the edges with the utility `T4_to_T10`.

```
0042      [fens,gcells] = t4cylinderdel(t,R, nt,nR);
0043      [fens,gcells] = T4_to_T10(fens,gcells);
```

Secondly, the integration rule is boosted to a four-point formula.

```
0049      feb = feblock_defor_ss (struct ('mater',mater,
                'gcells',gcells,...
0050          'integration_rule',tet_rule (4),
                'Rm',@twist));
```

Otherwise, the two functions are identical. On the other hand, the results are very different. The higher order tetrahedron converges very quickly and produces monotonically converging answer for the lowest frequency (approximately 1766.6Hz). The convergence behavior with respect to the number of elements radially and longitudinally is compared with the earlier analysis with T4 in Figure 13.8.

13.5 Static analysis with hexahedron H8

13.6 Analyzing the effects of thermal strains

[4] Folder: `SOFEA/examples/stress`

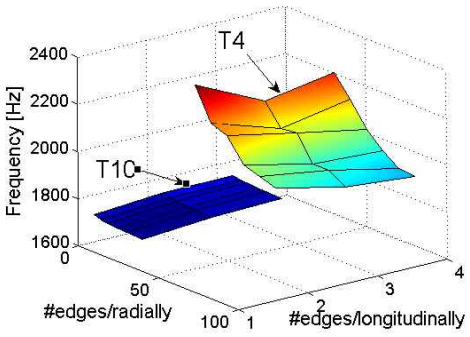

Fig. 13.8. Convergence of the lowest natural frequency. Comparison of the two tetrahedral elements.

14

Reduced-dimension models: plane strain, plane stress, axisymmetric

In this chapter we will reduce the three-dimensional elastic dynamic model to just two dimensions. This is possible by introducing assumptions for strains or stresses (and some restrictions on the form of the constitutive equation and loads).

14.1 Plane strain model reduction

The starting point is the observation that for solids of uniform cross-section (right angle cylinders), such as the one shown in Figure 14.1, 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

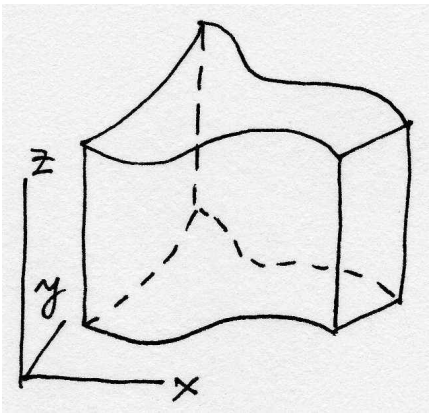

Fig. 14.1. Right-angle cylinder

reduction to two dimensions will be possible if the third balance equation is satisfied by design

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z = \rho \ddot{u}_z \ .$$

Due to the assumptions on the displacements, we have for the shear strains

$$\gamma_{zx} = 0, \quad \gamma_{zy} = 0 \ .$$

Note that the identically vanishing u_z also implies $\epsilon_z = 0$. 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 Figure 14.2: coefficients which could be nonzero are hatched, 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 of Section 12.5.1 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.

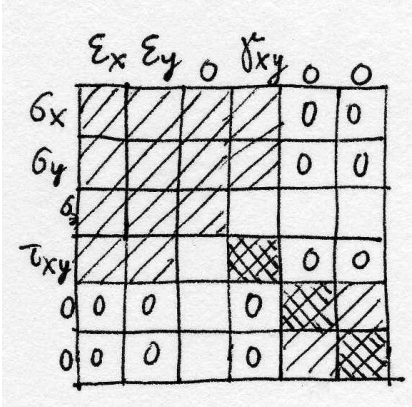

Fig. 14.2. Entries of the material stiffness matrix for plane strain

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

$$\frac{\partial \sigma_z}{\partial z} + \overline{b}_z = 0 \ .$$

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 also functions of x, y but not of z. Then, the equilibrium equation simply becomes a condition on the admissible loads **(v)**

$$\overline{b}_z = 0 \, .$$

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(v)** are

- top and bottom plane: $\overline{u}_z = 0$, $\overline{t}_x = 0$, $\overline{t}_y = 0$;
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the prescribed 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

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:2),(1:2)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \, . \tag{14.1}$$

The stress-divergence operator for plane strain with just two equations and three stresses assumes the form

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & \partial/\partial y \\ 0 & \partial/\partial y & \partial/\partial x \end{bmatrix} \, . \tag{14.2}$$

14.1.1 Discretization

The model for the fully three-dimensional elasticity resulted in the ODE system (12.32). In the present case, at this point we are still dealing with the same three-dimensional problem. However, when we simply take all the assumptions and constraints, and plug them into the equations (12.23), (12.25), (12.26), (12.27), (12.29), and (12.31), 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 (12.30) may be written as

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T \left(N_j(\boldsymbol{x}) \right) \boldsymbol{D} \mathcal{B} \left(N_k(\boldsymbol{x}) \right) \, \mathrm{d}V \right]_{im} = \tag{14.3}$$

$$\left[\int_S \mathcal{B}^T \left(N_j(\boldsymbol{x}) \right) \boldsymbol{D} \mathcal{B} \left(N_k(\boldsymbol{x}) \right) \, \Delta z \, \mathrm{d}S \right]_{im} \, . \tag{14.4}$$

Here Δz is the thickness of the slab in the z direction, S is the cross-sectional area of the cylinder. As only σ_x, σ_y, τ_{xy} (and correspondingly ϵ_x, ϵ_y, γ_{xy}) matter, the strain-displacement matrix uses the transpose of (14.2).

14.2 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 Figure 14.3) 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.

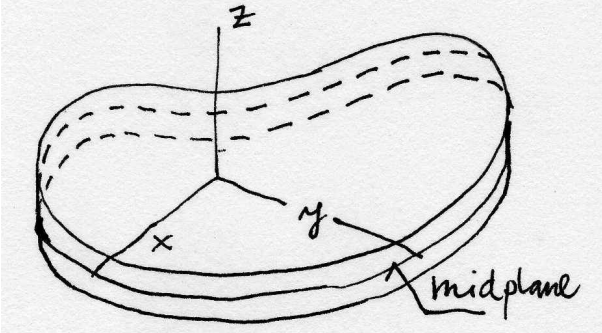

Fig. 14.3. Right-angle cylinder/thin slab/membrane

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 a 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.

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \bar{b}_z = \rho \ddot{u}_z \ .$$

However, neither of the stresses in it will be exactly zero, since the corresponding strains are in general nonzero. Nevertheless, inferring from the uniform, and small, thickness of the membrane, we may 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 perpendicular 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 perpendicular to the midplane. The material stiffness matrix will therefore have the appearance shown in Figure 14.4.

As the last step, the strain ϵ_z is eliminated from the constitutive equation. Assuming $\sigma_z \approx 0$, we have

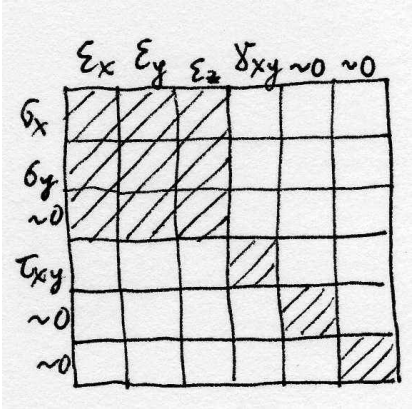

Fig. 14.4. Entries of the material stiffness matrix for plane stress

$$\sigma_z = [\boldsymbol{D}]_{3,(1:3)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} \approx 0 \ ,$$

which gives

$$\epsilon_z = -D_{33}^{-1} [\boldsymbol{D}]_{3,(1:2)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \ .$$

Therefore, we obtain

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \left[[\boldsymbol{D}]_{(1:2),(1:2)}, [\boldsymbol{D}]_{(1:2),3} \right] \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} =$$

$$\left([\boldsymbol{D}]_{(1:2),(1:2)}, -D_{33}^{-1} [\boldsymbol{D}]_{(1:2),3} [\boldsymbol{D}]_{3,(1:2)} \right) \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \ , \tag{14.5}$$

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 $\overline{b}_x = \overline{b}_x(x, y)$, $\overline{b}_y = \overline{b}_y(x, y)$.

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(vi)** are

- top and bottom plane: $\overline{t}_z = 0$, $\overline{t}_x = 0$, $\overline{t}_y = 0$;
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the prescribed 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 z.

14.3 Model reduction for axial symmetry

The last model reduction approach to be discussed in this chapter, is the case of *torsionless* axial symmetry (Figure 14.5). The main assumption is that **(i)** all planes

passing through the axis of symmetry are symmetry planes. 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 (zero circumferential displacement). Furthermore, points on circles in planes perpendicular to the axis of symmetry, with centers on the axis of symmetry, experience the same radial and axial displacements.

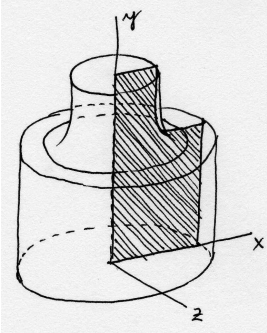

Fig. 14.5. Axially symmetric geometry

Let us consider one particular cross-section, for instance as indicated in Figure 14.5. 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 Figure 14.5). 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 perpendicular to the plane of the cross-section (z) is satisfied implicitly. 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, we must have $\bar{b}_z = 0$.

Now for the stresses: As indicated in Figure 14.6, by assumption the circles in planes perpendicular to the axis of symmetry are transformed by the deformation again into circles in planes perpendicular 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 $\gamma_{zx} = 0$, and $\gamma_{zy} = 0$. If the material is **(ii)** orthotropic, with one axis of orthotropy perpendicular to the cross-section, 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.

It remains to express the circumferential strain in terms of the displacements in the plane of the cross-section. By inspection of Figure 14.6, displacement of the circle in the y direction does not change its circumference, while displacement radially means

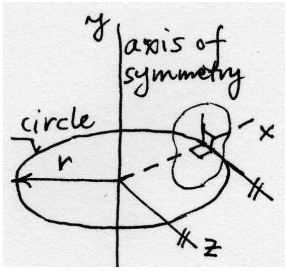

Fig. 14.6. Axially symmetric geometry: geometry of circles in planes perpendicular to the axis of symmetry

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 be therefore written as

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ 1/x & 0 \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \, . \tag{14.6}$$

The stress divergence operator (the transpose of (14.6)) gives the reduced form for the equations of equilibrium (11.25)

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\sigma_z}{x} + \bar{b}_x &= \rho \ddot{u}_x \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \bar{b}_y &= \rho \ddot{u}_y \end{aligned}$$

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 in the radial direction.

The constitutive equation is simply obtained from the full three-dimensional relationship by extracting appropriate rows and columns

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:3),(1:3)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \end{bmatrix} \, . \tag{14.7}$$

14.4 Material stiffness for two-dimensional models

The material stiffness matrices are (14.1) for plain strain, equation (14.5) for plain stress, and (14.7) for axial symmetry. While all are for models reduced to two displacement components, the stiffness matrices are clearly different. The material class

`mater_defor_ss_linel_biax` provides the tangent moduli by translating the material stiffness of a three-dimensional material for a particular reduced two-dimensional model. The class method `tangent_moduli` first retrieves the three-dimensional material stiffness from the property object (line 0014), and then the switch statement implements the computation (or subsampling) expressed in the three formulas.

```
0013 function D = tangent_moduli[1](self, context)
0014     D= get(self.property,'D', context);
0015     switch self.reduction
0016         case 'axisymm'
0017             D =D(1:4, 1:4);
0018         case 'strain'
0019             D =D([1, 2, 4],[1, 2, 4]);
0020         case 'stress'
0021             Dt =D(1:2, 1:2)-D(1:2,3)*D(3,1:2)/D(3,3);
0022             D =D([1, 2, 4],[1, 2, 4]);
0023             D(1:2, 1:2)= Dt;
0024         otherwise
0025             error([' Reduction ' self.reduction ' not recognized']);
0026     end
0027 end
```

Note well that there is no error checking concerning the assumptions in the three models. In particular, all three assume there is no coupling between normal stresses and the in-plane shear stress. If the three-dimensional material stiffness was retrieved from a property class which did not comply with these requirements, unpredictable results may be expected.

14.5 Strain-displacement matrices for two-dimensional models

The three models need different strain displacement matrices. The plane strain/planes stress defines the stress-displacement matrix as the transpose of (14.2)

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} . \tag{14.8}$$

while the axially symmetric model relies on (14.6). Consider the discretization of a particular structure with triangles T3. It is not possible in general to tell whether the model is axisymmetric, plain strain, or plain stress just by looking at the mesh. The finite elements (geometric cells) need to be told which they are. For instance, an axisymmetric triangle is created by the constructor being supplied with a flag:

```
gcell_T3(struct('conn', [2, 4, 7],'axisymm', true));
```

[1] Folder: `SOFEA/classes/mater/@mater_defor_ss_linel_biax`

Based on this flag, the two-manifold class `gcell_2_manifold` decides which type of the string displacement matrix should be produced by the method `Blmat`.

```
0024 function B = Blmat[2](self, pc, x, Rm)
0025     Nder = Ndermat_param (self, pc);
0026     Ndersp=Ndermat_spatial(self,Nder,x*Rm);
0027     nfens = size(Ndersp, 1);
0028     dim=length (Rm(:,1));
0029     if self.axisymm
0030         N = Nmat(self, pc);
0031         xyz =N'*x;
0032         r=xyz(1);
0033         B = zeros(4,nfens*dim);
0034         for i= 1:nfens
0035             B(:,dim*(i-1)+1:dim*i)=...
0036                 [Ndersp(i,1) 0; ...
0037                 0           Ndersp(i,2); ...
0038                 N(i)/r 0; ...
0039                 Ndersp(i,2) Ndersp(i,1) ]*Rm(:,1:2)';
0040         end
0041     else
0042         B = zeros(3,nfens*dim);
0043         for i= 1:nfens
0044             B(:,dim*(i-1)+1:dim*i)=...
0045                 [Ndersp(i,1) 0; ...
0046                 0           Ndersp(i,2); ...
0047                 Ndersp(i,2) Ndersp(i,1) ]*Rm(:,1:2)';
0048         end
0049     end
0050     return;
0051 end
0052
```

14.6 Integration for two-dimensional models

As discussed in Section 14.1.1, 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 thickness reduces multiplication.

The volume integrals for the axial-symmetry model reduction approach differ from plane stress or plane strain in that the integration "for free" is performed along the circumference of the rotationally symmetric body, not through the thickness. For instance, the stiffness matrix integral (12.30) may be written as

[2] Folder: `SOFEA/classes/gcell/@gcell_2_manifold`

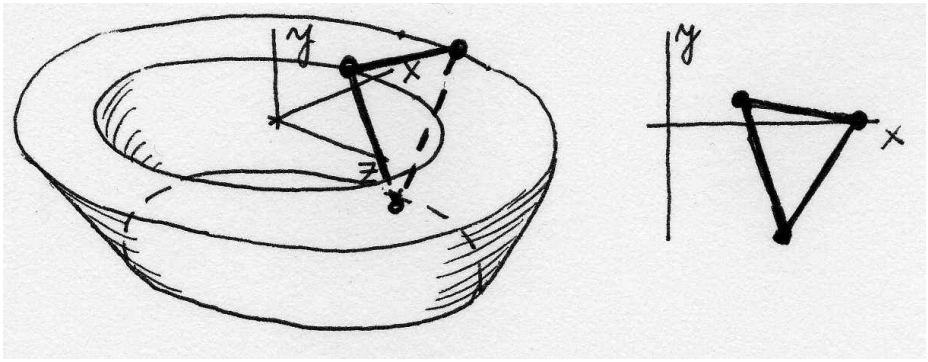

Fig. 14.7. Volume integration for axisymmetric analysis

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right) \boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\, \mathrm{d}V\right]_{im} = \left[\int_S \mathcal{B}^T\left(N_j(\boldsymbol{x})\right) \boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\, 2\pi x\, \mathrm{d}S\right]_{im} , \tag{14.9}$$

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 Figure 14.7. The volume integration may be performed by numerical quadrature over the area S but using a volume Jacobian

$$\int_V f\, \mathrm{d}V = \int_S f 2\pi x\, \mathrm{d}S \approx \sum_{k=1}^{M} f(\xi_k, \eta_k) J_{\mathrm{vol}}(\xi_k, \eta_k) W_k ,$$

where the volume Jacobian is defined as

$$J_{\mathrm{vol}}(\xi_k, \eta_k) = 2\pi x(\xi_k, \eta_k) \det\left[J(\xi_k, \eta_k)\right] . \tag{14.10}$$

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). For the two-dimensional models of this chapter, the volume Jacobian is computed by the method `Jacobian_volume` defined for the class `gcell_2_manifold`. The Matlab code closely mirrors the formulas; the method `other_dimension` simply computes the thickness at the given integration point `pc` (which in the plane-stress case could be variable).

```
0015 function detJ = Jacobian_volume[3](self, pc, x)
0016     if self.axisymm
0017         N = Nmat(self,pc);
0018         xyz =N'*x;
0019         detJ = Jacobian_surface(self, pc, x)*2*pi*xyz(1);
0020     else
```

[3] Folder: `SOFEA/classes/gcell/@gcell_2_manifold`

```
0021          detJ=Jacobian_surface(self,pc,x)*other_dimension(self,pc,x);
0022      end
0023 end
```

Traction loads, such as prescribed pressure on part of the bounding surface, needs to be evaluated with surface integrals. Quite analogously to the volume integration, the surface integrals are numerically evaluated along curves, but the Jacobians are surface Jacobians. Compare with Figures 14.3 and 14.8: for instance, for axial symmetry the surface Jacobian is

$$J_{\text{surf}}(\xi_k) = 2\pi x(\xi_k) \det\left[J(\xi_k)\right] \ . \tag{14.11}$$

For the two-dimensional models, the surface Jacobian is computed by the method `Jacobian_surface` defined for the class `gcell_1_manifold`.

```
0015 function detJ = Jacobian_surface[4](self, pc, x)
0016     if self.axisymm
0017         N = Nmat(self,pc);
0018         xyz =N'*x;
0019         detJ = Jacobian_curve(self, pc, x)*2*pi*xyz(1);
0020     else
0021         detJ = Jacobian_curve(self,pc,x)*other_dimension(self,pc,x);
0022     end
0023 end
```

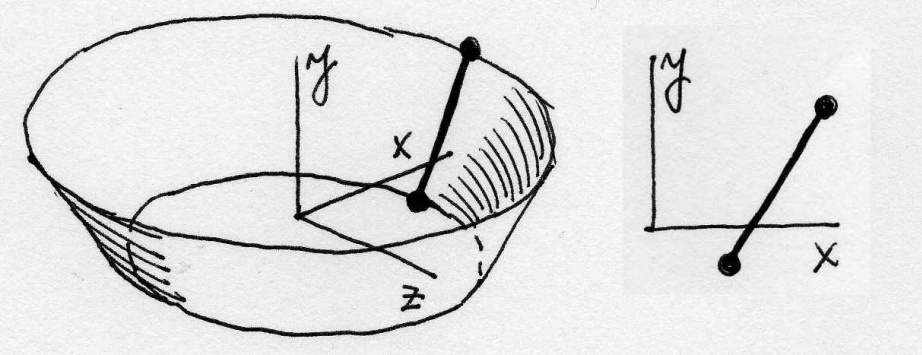

Fig. 14.8. Surface integration for axisymmetric analysis

14.7 Thermal strains in two-dimensional models

The thermal strains must be of the form (12.34) as the two-dimensional models are require properly aligned orthotropic constitutive relations. In order to properly defined the thermal strain loads (12.40), three ingredients are needed: the strain-displacement

[4] Folder: `SOFEA/classes/gcell/@gcell_1_manifold`

matrix, the material stiffness, and the thermal strains. The former two have been discussed already, it remains to outline the calculation of the thermal strains when the model is reduced.

The plane stress model and the axially symmetric model may refer directly to the definition (12.34) as the stress in the former is independent of the strain in the third direction, while it incorporates all strains in the latter. On the other hand, in the plane strain model the stress depends on the strain in the z direction (it must be zero), and nonzero thermal strain in the z direction will therefore have an effect which needs to be incorporated into the reduced constitutive equation.

The relationship between the normal stresses and the normal strains may be written for the plane strain model as a subset of the full three-dimensional relation

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix} = [\boldsymbol{D}]_{(1:3),(1:3)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z = 0 \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \\ \epsilon_{\Theta z} \end{bmatrix} \right) . \tag{14.12}$$

Since we are not interested directly in σ_z and since $\epsilon_z = 0$, we may write

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = [\boldsymbol{D}]_{(1:2),(1:2)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} \right) - [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} . \tag{14.13}$$

Grouping the strains with the same meaning leads to

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = [\boldsymbol{D}]_{(1:2),(1:2)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} - [\boldsymbol{D}]^{-1}_{(1:2),(1:2)} [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} \right) , \tag{14.14}$$

where

$$- \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} - [\boldsymbol{D}]^{-1}_{(1:2),(1:2)} [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} ,$$

represents an effective in-plane thermal strain.

14.8 Examples

14.8.1 Thermal strains in a bimetallic assembly

Consider an assembly of two thin metal slabs, the inset is of aluminum, while the outer plate is of steel: see Figure 14.9. 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 discretized.

The Matlab script `alusteel`[5] solves the problem with triangular adaptive meshes. The initial phases are omitted for brevity, we just mention that two sets of properties,

[5] Folder: `SOFEA/examples/thermo_mechanical`

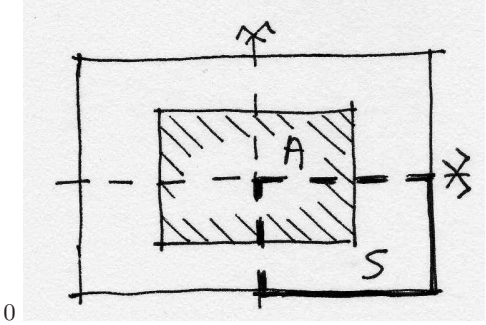

Fig. 14.9. Bimetallic assembly with thermal strain load

materials, and finite element blocks need to be created because the assembly consists of two distinct materials.

The calculation proceeds by assuming that the thermal strains are generated by temperature distribution described by a field. Therefore, such field is created, and all its degrees of freedom are set to the prescribed temperature.

```
0063 dT = field(struct ('name',['dT'], 'dim', 1, ...
0064     'data',zeros(length(fens),1)+70));
```

Both finite element blocks contribute to the stiffness matrix, and the assembly is performed on the concatenated arrays of element stiffness matrices produced for these two blocks. The thermal strains loads are again computed for each block separately, and assembled into the global load vector.

```
0066 K = start (sparse_sysmat, get(u, 'neqns'));
0067 K = assemble (K, cat(2,stiffness(febs, geom, u),stiffness(feba, geom, u)))
0068 % Load
0069 F = start (sysvec, get(u, 'neqns'));
0070 F = assemble (F, cat(2,thermal_strain_loads(feba, geom, u, dT),...
0071     thermal_strain_loads(febs, geom, u, dT)));
```

It might be of interest to point out how to visualize a particular stress component. The first step is to create the graphic viewer, class `graphic_viewer`. The viewport is reset with the method `reset`.

```
0077 gv=graphic_viewer;
0078 gv=reset (gv,struct ('limits', [0, 100, -8, 40]));
0079 set(gca,'FontSize', 12)
0080 cmap=jet;
0081 cmpn=3;
```

Two fields are created by interpolating the stresses from the integration points to produce continuous representations by estimating stress at the finite element nodes. This is inherently fraught with dangers, since we know the stress is in general discontinuous at the nodes. The method `field_from_integration_points` of the class `feblock_defor_ss` uses inverse-distance interpolation. Two distinct fields are used,

because the elastic properties of the two materials are different along their common interface.

```
0082 flda = field_from_integration_points(feba, geom, u, dT,
                'Cauchy', cmpn);
0083 flds = field_from_integration_points(febs, geom, u, dT,
                'Cauchy', cmpn);
0084 nvalsa=get(flda,'values');
0085 nvalss=get(flds,'values');
0086 nvalmin =min(min(nvalsa),min(nvalss));
0087 nvalmax =max(max(nvalsa),max(nvalss));
```

An object to map values of the stress to colors, `data_colormap`, is created and initialized with the range of the stress. The stress is then mapped to another field, `colorfield`, whose nodal parameters are colors. The computed color field is used to draw the geometric cells of the block `feba`, using the magnified (scaled) displacement field `u` so that the deformations are readily distinguishable. The same process is then repeated for the block `febs`.

```
0088 dcm=data_colormap(struct ('range',[nvalmin,nvalmax],
        'colormap',cmap));
0089 colorfield=field(struct ('name', ['colorfield'],
        'data',map_data(dcm, nvalsa)));
0090 draw(feba, gv, struct ('x',geom,'u', scale*u,
        'colorfield',colorfield, 'shrink',1));
0091 % draw(feba, gv, struct ('x',geom,'u', 0*u, 'facecolor',' none'));
0092 colorfield=field(struct ('name', ['colorfield'],
        'data',map_data(dcm, nvalss)));
0093 draw(febs, gv, struct ('x',geom,'u', scale*u,
        'colorfield',colorfield, 'shrink',1));
```

The problem is solved for four different meshes, each time scaling the element size down by the factor 2. The shear stress is shown in Figure 14.10. As can be seen, the stress highs are in the vicinity of the corner in the assembly where the two materials meet. We should probably realize that the reentrant corner in the steel piece is going to generate a stress singularity. To compute the largest stress then becomes futile: it tends to infinity, and any subsequent refinement would only tend to cost more without any real improvements. However, in any real design the corner would not really be sharp, there would be a radius to take care exactly of such stress concentrations.

The Matlab script `alusteelround` [6] solves for deformation and stress in a modified geometry, with the reentrant corner rounded to remove the stress singularity. The solution displayed in Figure 14.11 shows that the shear stress is now convergent.

[6] Folder: `SOFEA/examples/thermo_mechanical`

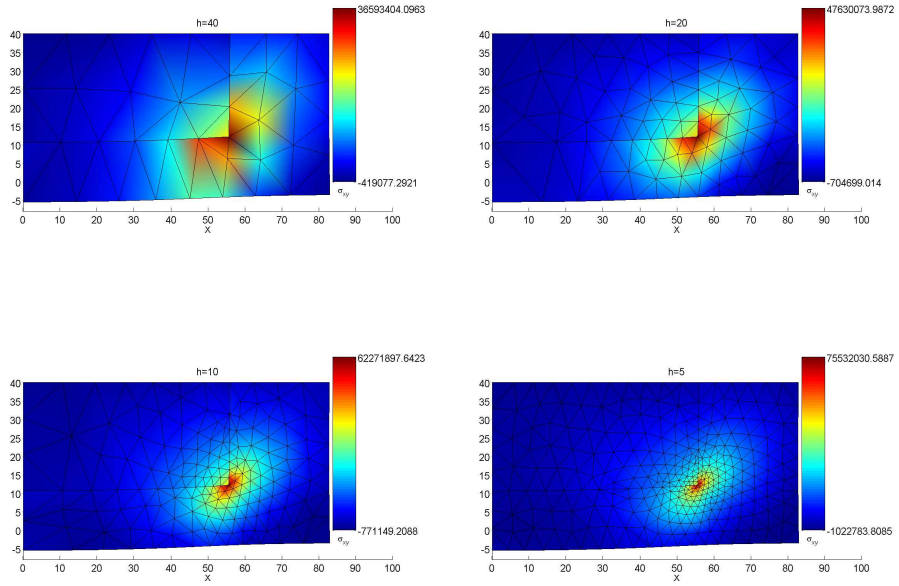

Fig. 14.10. Shear stress in the bimetallic assembly

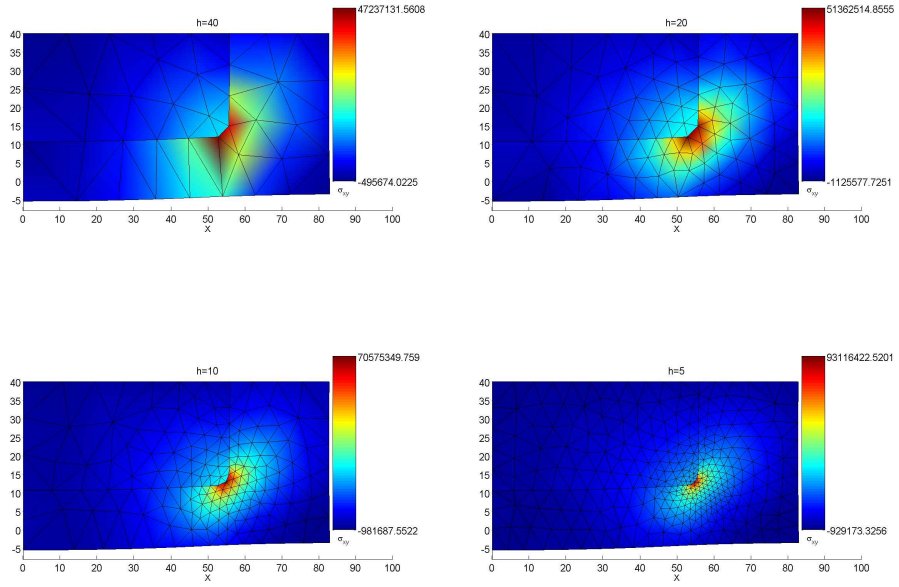

Fig. 14.11. Shear stress in the bimetallic assembly with rounded corner

14.8.2 Orthotropic balloon

In this example we consider an elastic balloon, with orthotropic properties induced by reinforcing fibers along circles generated by rotation of particles around the axis of symmetry: see Figure 14.12. The balloon is loaded by internal pressure. The model is based on the axial symmetry of the expected deformation. The modeled section is hatched, as we use not only the axial symmetry, but also symmetry with respect to the plane perpendicular to the axis of symmetry.

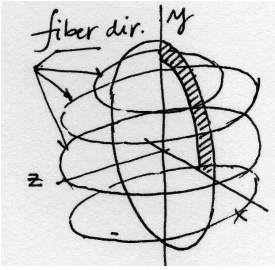

Fig. 14.12. Orthotropic balloon

```
0021 [fens,gcells]=block2d(rex-rin,pi/2,5,20, 0);
0022 for i = 1:length(gcells)
0023     gcells(i) = gcell_Q4(struct('conn', get(gcells(i),'conn'),...
0024         'axisymm', true));
0025 end

0026 bdry_gcells = mesh_bdry(gcells, struct('axisymm', true,
            'area', 0.0001));
0027 icl = gcell_select(fens, bdry_gcells, struct('box', [0,0,0,pi/2],
            'inflate',rin/100));

0028 for i=1:length (fens)
0029     xy=get (fens(i),'xyz');
0030     r=rin+xy(1); a=xy(2);
0031     xy=[r*cos(a) r*sin(a)];
0032     fens(i)=set(fens(i),'xyz', xy);
0033 end
```

The material model is based on the orthotropic property class, `property_linel_ortho`, but we choose to material constants so that in the x, y plane the material is isotropic, and the material is therefore effectively transversely isotropic.

```
0035 prop = property_linel_ortho(struct('E1',E1,'E2',E2,'E3',E3,...
0036     'nu12',nu12,'nu13',nu13,'nu23',nu23,...
0037     'G12',G12,'G13',G13,'G23',G23));
0038 %     prop = property_linel_iso (struct('E',E1,'nu',nu12));
```

```
0039 mater = mater_defor_ss_linel_biax (struct('property',prop, ...
0040     'reduction','axisymm'));
```

Two finite element blocks are created: `feb` collects the finite elements that represent a cross-section (quadrilaterals), and `efeb` for the finite elements along the pressure-loaded edge (the inside of the balloon).

```
0042 feb = febf (struct ('mater',mater, 'gcells',gcells,...
0043     'integration_rule',gauss_rule (2, integration_order)));
0044 efeb = febf (struct ('mater',mater, 'gcells',bdry_gcells(icl),...
0045     'integration_rule',gauss_rule (1, 2)));
```

The boundary conditions are applied in the form off rollers on the axis of symmetry, and on the transverse plane of symmetry.

```
0052 % First the plane of symmetry
0053 ebc_fenids=fenode_select (fens,struct('box',[0 rex 0 0],
            'inflate',rex/10000));
0054 ebc_prescribed=ones(1,length (ebc_fenids));
0055 ebc_comp=ebc_prescribed*0+2;
0056 ebc_val=ebc_fenids*0;
0057 u   = set_ebc(u, ebc_fenids, ebc_prescribed, ebc_comp, ebc_val);
0058 % The axis of symmetry
0059 ebc_fenids=fenode_select (fens,struct('box',[0 0 0 rex],
            'inflate',rex/10000));
0060 ebc_prescribed=ones(1,length (ebc_fenids));
0061 ebc_comp=ebc_prescribed*0+1;
0062 ebc_val=ebc_fenids*0;
0063 u   = set_ebc(u, ebc_fenids, ebc_prescribed, ebc_comp, ebc_val);
```

The force intensity object of class `force_intensity` is given a function handle that corresponds to the internal pressure applied radially. The method `surface_loads` integrates the force intensity over the block `efeb` of the edges on the internal surface, producing an array of element-load vectors.

```
0071 fi=force_intensity(struct('magn',@(x) (p*x'/norm(x))));
0072 F = start (sysvec, get(u, 'neqns'));
0073 F = assemble (F, surface_loads(efeb, geom, u, fi));
```

Figure 14.13 displays the deformed shape, which due to the reinforcing fibers running in planes perpendicular to the axis of symmetry assumes the general resemblance of a football. The circumferential stress is displayed on the deformed shape, while the unformed shape is shown as shrunk outlines of the cross-section elements.

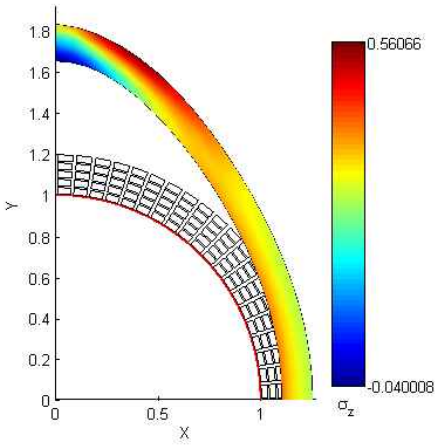

Fig. 14.13. Orthotropic balloon

A

Appendix: GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The **"Document"**, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as **"you"**. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A **"Modified Version"** of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A **"Secondary Section"** is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The **"Invariant Sections"** are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document

may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The **"Cover Texts"** are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A **"Transparent"** copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called **"Opaque"**.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The **"Title Page"** means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section **"Entitled XYZ"** means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as **"Acknowledgements"**, **"Dedications"**, **"Endorsements"**, or **"History"**.) To **"Preserve the Title"** of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque

copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of

Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright ©YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

References

Blevins. Blevins, R. D., Formulas for Natural Frequency and Mode Shape, Krieger publishing company, reprint edition 2001.

BS89. Bogacki, P. and L. F. Shampine, A 3(2) pair of Runge-Kutta formulas, Appl. Math. Letters, Vol. 2, 1989, pp 1-9.

CCS. A.D. CAMERON, J.A. CASEY, G.B. SIMPSON: Benchmark Tests for Thermal Analyses, NAFEMS Documentation.

CC2005. Chapra, SC and RP Canale: Numerical Methods for Engineers. McGraw-Hill (2005)

Gra91. Graff, KF: Wave motion in elastic solids. Dover publications, New York (1991)

Hug00. Hughes, TJR: The finite element method. Linear static and dynamic finite element analysis. Dover publications, New York (2000).
This book is a must-read for anyone serious about learning finite element analysis properly. Meticulously written, and brimming with insights. Quite mathematical.

IL05. `http://www.infolytica.com/en/coolstuff/ex0047/`

LL05. John H. Lienhard IV, and John H. Lienhard V, A Heat Transfer Textbook, 3rd edition, `http://web.mit.edu/lienhard/www/ahtt.html`

Sok. Sokolnikoff, I.S.: Mathematical Theory of Elasticity. Krieger publishing company. Reprint edition 1983.

Index

`OBgui`, 21

adaptive mesh control, 99
anisotropic material, 137
anti-symmetry, 121, 123
argument
  passed by value, 21
assembly, 61
average acceleration method, 27
axial symmetry, 161, 168
axisymmetric, 161, 168

balance equation, 3
  global, 38
  local, 38
balance of angular momentum, 108
balance of linear momentum, 105
balance residual, 7
barycenter, 99
basis function
  derivative, 57
  triangle T3, 50
  triangle T6, 89
basis functions, 12
body load, 135
boundary condition
  displacement, 119
  essential, 5, 40, 119
  natural, 5, 41, 119
  overspecified, 32
  sufficient, 41
  traction, 119
boundary conditions, 4, 5
Boussinesq, 120
Bubnov-Galerkin method, 7

capacity matrix, 54
Cauchy stress tensor, 107
chain rule, 17
change of coordinates in integrals, 59
circular frequency, 23
class
  `body_load`, 22
  `data_colormap`, 170
  `dense_sysmat`, 21, 61
  `elemat`, 61, 63
  `feblock_defor_ss`, 144, 148, 169
  `feblock_diffusion`, 60, 70
  `feblock`, 20, 60, 90, 144
  `field`, 20, 83, 132
  `force_intensity`, 173
  `gcell_1_manifold`, 66, 167
  `gcell_2_manifold`, 165, 166
  `gcell_3_manifold`, 144
  `gcell_T3`, 58
  `gcell`, 57
  `graphic_viewer`, 169
  `mater_defor_ss_linel_biax`, 164
  `mater_defor_ss_linel_triax`, 148
  `property_linel_iso`, 148
  `property_linel_ortho`, 172
  `property_linel_transv_iso`, 151
  `sparse_sysmat`, 61
  `sysvec`, 22
  `tet_rule`, 96, 148

tri_rule, 70, 96
fenode, 20
class method
Blmat, 143, 165
Jacobian_curve, 66, 67
Jacobian_surface, 60, 67, 92, 167
Jacobian_volume, 62, 145, 166, 167
Jacobian, 92, 145
Nder_param, 58
Ndermat_spatial, 57
Nmat, 58
apply_ebc, 21
assemble, 61
conductivity, 61
field_from_integration_points, 169
gather, 22
get, 21
material_directions, 63, 145
measure, 91
numbereqns, 21
other_dimension, 166
property_diffusion, 70
scatter_sysvec, 22
set_ebc, 21
stiffness, 144
surface_loads, 173
surface_transfer_loads, 68
surface_transfer, 67
tangent_moduli, 164
Clough, 48
coefficient of thermal expansion, 140
compliance matrix, 137
concentrated force, 119
conductivity matrix, 54
connectivity, 20
consistent mass matrix, 23
constitutive equation, 116, 140
constructor, 20
contact, 124
control volume, 37
convergence, 23
convex hull, 96
Courant, 48
Crank-Nicholson method, 81
cross product, 58, 92
curved boundary
approximation, 52

degree of freedom, 15
diagonal mass matrix, 17
Dirac delta function, 10
displacement, 113
displacement boundary condition, 119
divergence theorem, 39, 109, 112, 128
dynamic equilibrium, 112
dynamic force equilibrium, 134
dynamic method dispatch, 67, 90

eigenmode, 23
eigenvalue, 107
elastic coefficients, 116
energy of deformation, 116
equation number, 135
error
heat flux, 98
essential boundary condition, 5, 29, 40

fiber-reinforced composite, 137
finite difference
backward Euler, 81
forward Euler, 81
finite element, 13
edge, 48
H8, 96
isoparametric, 55
L2, 13
L3, 89
node, 13, 48
P1, 90
Q4, 96
T10, 152
T3, 58
T4, 94, 147
T6, 87
finite element mesh, 13
flux boundary condition, 41
flux variable, 117
free vibration, 23, 147

Galerkin method, 7
generalized eigenvalue problem, 23
generalized trapezoidal method, 79, 80
geometric cells, 20

hat function, 49
heat
  conduction, 37
  diffusion, 37
heat energy, 37
heat flux, 38

inadmissible boundary condition, 119
inertial force, 112, 135
inertial load, 135
infinite half space, 120
initial boundary value problem, 6
initial conditions, 6
integration by parts, 11
integration rule
  tetrahedron, 94
  triangle, 60, 89
interpolation, 12
isoparametric element, 55
isotropic material, 40, 139

Jacobian, 57, 59, 92
  surface, 67, 167
  volume, 63, 166
Jacobian determinant, 16
Jacobian matrix, 56, 58

kinematically admissible displacement, 130
Kronecker delta, 14, 88

Lagrange interpolation polynomial, 13
Lamé constant λ, 139
linear elasticity, 116
linear momentum, 104
lumped mass matrix, 23

manifold dimension, 90
map
  of areas, 59
  of points, 58
  of vectors, 58
mass density, 109
mass matrix, 13
  consistent, 13, 23, 135
  lumped, 23
material curve, 113
material orientation matrix, 63, 137, 143, 145
material point, 113
material stiffness, 116
Matlab script
  `alusteelround`, 170
  `alusteel`, 168
  `drum _t10`, 153
  `drum_t4`, 147
  `helixcooled`, 95
  `lshape1`, 69
  `shrinkfit`, 84
  `squareinsquare`, 72
  `t3nafems`, 82
  `t4nafems`, 75
  `test_measure`, 93
  `transcool`, 90
  `twist_t10`, 154
  `twist_t4`, 149
  `w1`, 19
  `woverspec`, 33
  `wtransient1`, 25
  `wtransient2`, 27
  `wvib`, 23
membrane, 160
mesh generator `targe2_mesher`, 69
method
  backward Euler, 81
  Crank-Nicholson, 81
  forward Euler, 81
modal equations, 137
modeling error, 117
motion, 113

natural boundary condition, 5, 29, 41
natural frequency, 23
Neumann problem, 31, 42
Newmark algorithm, 27

Newmark average-acceleration integrator, 27, 137
Newmark explicit algorithm, 18
Newmark integrator, 27
Newton's equation of motion, 103
Newton's law, 3
node, 13
nonzero-displacement load, 136
normal strain, 113
normal stresses, 107
numerical quadrature, 15
  point, 90
  triangles, 59

ordinary differential equations, 13
orthogonal matrix, 143
orthotropic material, 40, 137
outer normal, 38

parametric coordinates, 17
particle, 103
piecewise linear, 9
piecewise linear approximation, 13
plane strain, 118, 157
plane stress, 160
point support, 119
Poisson's ratio ν, 139
polymorphism, 90
pressure, 118
primary variable, 117
principal direction, 107
principal stress, 107
principle of virtual work, 130
pure-traction problem, 31, 42, 123

quadratic form, 116, 124
  positive semi-definite, 124
quadrature point, 15

rate of heat generation, 38
reaction, 31, 129
reactions, 118
reduction
  dimension, 46
reflection, 121
residual, 7, 10, 45
  balance, 45
resisting force, 136
Richardson extrapolation, 98
rigid body
  rotation, 124
  translation, 124
rotation matrix, 63, 137, 143
Runge-Kutta ODE integrator, 25

Saint-Vénant's principle, 125
shear modulus, 139
shear strain, 113
shear stresses, 107
shear tractions, 124
simplex element, 96
Simpson's 1/3 rule, 16
singular stiffness matrix, 32
skew-symmetric matrix, 92
smoothness, 10
sparse matrix, 13
specific heat, 38
standard interval, 16, 65
standard tetrahedron, 94
standard triangle, 50, 87
static equilibrium, 19
stiffness
  singular, 124
stiffness matrix, 12, 136, 144
strain displacement operator, 137
strain tensor, 115
strain-displacement matrix, 141, 143
strain-displacement operator, 115
strains, 105
stress, 106
  normal, 107
  shear, 107
stress divergence, 112
stress interpolation, 169
stress singularity, 170
stress-divergence operator, 112, 159
stretch, 113
surface heat transfer matrix, 55
surface traction load, 135

symmetric gradient operator, 112, 115, 137
symmetric matrix, 13
symmetry, 121

tangent to material curve, 113
tangent vector, 58, 65, 92, 145
taut string, 3
temperature gradient, 40
tensor, 107
tent, 49
test function, 8, 128
tetrahedron
  linear, 94
  quadratic, 152
  standard, 94, 152
thermal conductivity, 40
thermal expansion, 139
thermal strain, 139
thermal strain load, 141, 167, 169
thermal strains, 167
thermal stress, 140
traction, 103
traction boundary condition, 119
traction-free, 118
transformation matrix, 63, 137
transformation of vector components, 143
transversely isotropic material, 138
Trapezoidal ODE integrator, 27
trapezoidal rule, 26
trial function, 10, 12
trial-and-test approximate method, 9
triangle
  linear, 55, 57
  quadratic, 87
  standard, 50, 87
triangulation, 48

utility
  `T4_to_T10`, 154
  `block1d`, 82
  `drawmesh`, 95
  `fenode_select`, 71
  `mesh_bdry`, 95
  `transform_apply`, 95

vector-stress vector dot product operator, 108, 128
virtual displacement, 130
virtual work, 130

weighted residual equation
  stress analysis, 129
weighted residual method, 8
work, 116
work-conjugate, 118

Young's modulus E, 139