*A.I. 33

Instructional Use of MATLAB Software Components for Computational Structural Engineering Applications

Petr Krysl and Abhishek Trivedi
Jacobs school of engineering,
University of California San Diego
9500 Gilman Dr., Mail code 0085
La Jolla CA 92093-0085
pkrysl@ucsd.edu, <http://hogwarts.ucsd.edu/~pkrysl>

Abstract

Object oriented MATLAB software components that have been developed for and used in instruction of computational structural engineering are described. It is shown that the object orientation leads to elegant and concise implementations, which helps promote learning and understanding of the subject at the level of the algorithms.

Introduction

Using open software tools for instruction of computational methods to engineering students, not only graduate, but also undergraduate students, seems to be making a lot of sense. To become truly expert, the students need hands-on practice. Black-box type software may provide some needed experience, but the tired old joke about a monkey that had been taught which buttons to press to get a banana should ring in our ears. Knowledge of how to operate an analysis tool GUI must not be considered a replacement for the understanding of the guts of the tool. Tools that the students may inspect in depth, preferably at the source code level, are ideally positioned as instructional tools at all levels of university instruction.

We describe some experiences with instructional software that has been written for and used in some courses taught in the department of Structural engineering at the UCSD. These courses span from the Introduction to numerical and graphical tools (SE102), Numerical methods in engineering (SE121), and Finite element analysis (SE131) at the undergraduate level, to Advanced nonlinear finite element analysis at the graduate level. Importantly, we are also able to report experiences obtained with these tools in the setting of undergraduate research in the scope of the Faculty Mentor Program at UCSD. Four students have developed MATLAB tools and used them in research in the past three years.

```
% L = length [m]
% n = number of nodes in the interior
% T0 = prestressing force [N]
% mu = mass density [kg/m]
function [M,K] = cablemats(L, n, T0, mu)
h=L/(n+1);
M = mu * h * eye(n,n);
aux = -ones(n,n) + 3*eye(n,n);
K = (T0/h) * triu(tril(aux,1),-1);
```

Figure 1. Compute the mass and stiffness matrices for the prestressed cable.

Some of these tools are self-contained library routines (the likes of LU and QR factorizations), some are ad hoc programs (optimization of the period of oscillation of a nonlinear pendulum or dynamics of prestressed cables), and some are comprehensive object oriented toolkits for engineering models of continua (mechanics of solids, heat conduction, rigid body dynamics). All of these tools are pure MATLAB, with particular attention to the intended use in instruction. We go to great lengths to ensure that sound software engineering practices are never abandoned.

MATLAB as exploratory tool

MATLAB is quite well suited to the classroom. The interactive nature lends itself well to exploration, and its expressiveness, and completeness of the overall computational environment is a distinct benefit compared to languages like Fortran or C++. Thus, for instance we may express the lumped-mass discretization of a prestressed cable in a few lines (Figure 1), integrate the resulting ODE's with the centered difference (Newmark) time-stepper (Figure 2), and plot the results in a sophisticated way (Figure 4: surface of transversal displacement for an off-center plucked prestressed string). All of this in just a few lines, even while observing good software engineering practices. Non-negligible is also the ability to inspect the workings of the code by running in the debugger, since it any point all the MATLAB sophistication is available, be it running numerical algorithms or visualization.

```
% M = mass matrix
% K = stiffness matrix
% C = damping matrix
% V0 = initial deflection (column vector) [m]
% V0v = initial velocity (column vector) [m/s]
% dt = time step [s]
% nsteps = number of steps to take [ND]
function [ts,ys] = dampode_expl_newm(M, K, C, V0, V0v, dt, nsteps)
n=length(M);
ts=zeros(nsteps,1); ys=zeros(2*n,nsteps); % init output
Vj  = V0; % initial displacement
Vjv = V0v; % initial velocity
Vja = M \ (-K * Vj - C * Vjv); % initial acceleration
t=0;
for j=1:nsteps
   ts(j) = t; ys(:,j)=[Vj; Vjv]; % output
   Vj1  = Vj + dt * Vjv + ((dt^2)/2) * Vja; % update displacement
   Vj1a = M \ (-K * Vj1 - C * (Vj1 – Vj)/dt); % compute acceleration
   Vj1v = Vjv + (dt/2) * (Vja + Vj1a); % update velocity
   Vj = Vj1;   Vjv = Vj1v;   Vja = Vj1a; % swap
   t = t + dt;
end
ts(nsteps+1) = t; ys(:,nsteps+1)=[Vj; Vjv]; % output
```

Figure 2. Explicit Newmark integrator for linear second-order systems (in our case cable vibration).

Simple computational tools may be designed in the form of a collection of small, self-contained units that together constitute a library. However, this often lead to the use of the array as the only data structure. Browsing through the legions of textbooks that deal with numerical algorithms implemented in MATLAB one might think that was the "MATLAB way". From a software engineering point of view that is not very appealing, and not surprisingly there is anecdotal evidence that students eventually find out why they dislike rummaging through poorly written software, no matter how sophisticated and elegant the algorithms it implements.

```
function cablehistsurf(ts,ys)
nsteps=length(ts);     n=round(size(ys,1)/2);    vs=[0*ys(1,:); ys(1:n,:); 0*ys(1,:)];
surf(ts,(1:n+2),vs, 'FaceColor', 'interp','EdgeColor', 'none');
hold on;   camlight headlight ;
contour3(ts,(1:n+2),vs, 10, 'k-');
```

Figure 3. Plot the motion of the cable as a surface in 3-D space-time.

Object-oriented computational tools in MATLAB?

Many engineering students nowadays learn C++ or Java as freshmen, which exposes them to object-based ideas in the form of classes with data abstraction and data encapsulation, polymorphism, and inheritance. It is hardly necessary to belabor the advantages that such an approach to programming contributes to engineering education (an object-oriented approach seems in many cases natural and often appeals more to human cognition than other methodologies), even though it bears emphasis that the object-oriented approach is just one of several alternatives.

MATLAB has incorporated object-oriented concepts as an addition to its core strengths. For instance the graphical user interface and visualization capabilities are accessible from the Java programming language, and the instances of figures and other components may be treated as objects. MATLAB's object-oriented features have been recently applied to very sophisticated mathematical modeling [9], and to challenging simulations on a fairly large-scale with multiple coupled partial differential equation models [8].

This paper does not intend to provide an introduction to object-oriented programming with MATLAB. For reference and additional information please refer to MATLAB online technical documentation [6]; for a gentle introduction see Reference 7. However, several technical issues have a bearing on the subject of instructional use. Perhaps the trickiest problems is related to the way MATLAB passes arguments: In MATLAB, there is no passing of variables by reference. When writing methods that update an object, you must pass back the updated object and use an assignment statement. For instance, this call to the **set** method updates the **name** field of the object **A** and returns the updated object.

```
A = set(A,'name','John Doe');
```

Unfortunately, that is rather counterintuitive, especially to students that are used to C++ or Java, where the actions on objects are explicitly allowed to change the state of the object. As an alternative, one could use the mechanism of assigning objects in the name space of the caller. (This is not recommended as a rule, since the practice of modifying the caller's data is dangerous.) The **assignin** function may be used to emulate the change of the state as a side effect. The methods called on objects needs to invoke the **assignin** function as

```
assignin('caller',self_name(self),self);
```

to achieve the desired change in the object state to propagate to the environment of the caller of the method. This is the approach used in the MATLAB object-oriented tool Hexcomp [1]. For historical reasons, the finite element toolkit FEALAB described below does not use this mechanism, and the first technique detailed above of assigning the object returned by a method to propagate the change of state is used. However, the second approach using **assignin** is under consideration as an alternative to which the whole toolkit maybe adapted at some point as it is closer to what the students expect, given their grounding in C++ and Java.

Another troublesome issue may be encountered when designing abstract data structures. MATLAB does not allow for storage of actual pointers, and that would make design of complex data structures more challenging [7]. However, in our applications we have not actually encountered a situation which could not be resolved with the available technical facilities of the language.

```
clf;
gm1=graphics_model(...
        make_graphics_reps(feb, geom, u, 'gcells'));
draw(gm1, 'dataidx', 3, 'scale', 100000, ...
        'displace_by', [1 2 3], 'facecolor', 'none')
gm2=graphics_model(...
        make_graphics_reps(feb, geom, u, 'integration_points'));
draw(gm2, 'dataidx', 5, ...
        'scale', [100000 100000 100000 0.2], 'displace_by', [1 2 3])
```

Figure 4. Code to plot the stress tensor ellipsoids at the quadrature points.

Finite element analysis toolkit: FEALAB

Some ideas from our research made it successfully into our instructional software (Figure 5). Thus, for instance the concept of a *field* is used in our adaptive finite element research tools [2]. *Field* is an abstraction used in virtually all finite element continuum mechanics models, where it represents the unknown quantities such as temperatures, dislocation densities, displacements or velocities, etc. In FEALAB it became an actual class, with a number of methods that take care of operations such as numbering of equations, application of essential boundary conditions, scatter and gather of values, and so on (Figure 8). Even the geometry itself is represented as a field, which makes formulation of geometrically nonlinear algorithms quite straightforward. Use of such an abstract construct would seem hopelessly advanced for undergraduate classes, but surprisingly it is possible to achieve a high rate of acceptance when this concept is represented graphically in the form of a table that allows only such modifications that preserve consistency of the entries.

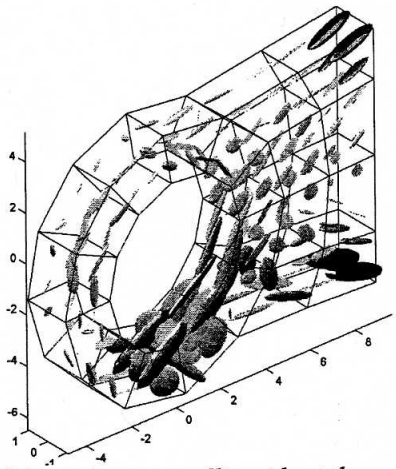

Figure 5. Plot of the stress tensor ellipsoids at the quadrature points.

Inheritance is used extensively for the finite elements as geometry cells (gcell) that support essentially only geometrical calculations: calculation of the basis functions and their derivatives in the form of parametric derivatives, or spatial derivatives with respect to some geometry field. There is an abstract base class (in the parlance of C++) whose purpose is essentially to define a protocol that the finite elements need to support, and the pure virtual functions of the protocol are then implemented in the derived classes.

```
w=clone(u,'w'); % make a copy of u
gm1=graphics_model(make_graphics_reps(feb, geom, w, 'gcells'));
w = scatter_sysvec(w, W(:,2)); % mode number 2
v = get(w,'values');
wmag = field('wmag', 1, get(w,'nfens'));
wmag = scatter(wmag,...
                1:get(wmag,'nfens'),...
                sqrt(v(:,1).^2+v(:,2).^2+v(:,3).^2));
wwmag = combine(w,wmag); % combine eigenvector with its magnitude
gm2=graphics_model(make_graphics_reps(feb, geom, wwmag, 'gcells'));
draw(gm1, 'dataidx', 3, 'scale', 0, 'displace_by', [1 2 3], 'facecolor', 'none')
draw(gm2, 'dataidx', 4, 'scale', 1, 'displace_by', [1 2 3], 'facecolor', 'interp')
```

Figure 6. Combine the eigenvector with this magnitude into another field, ...

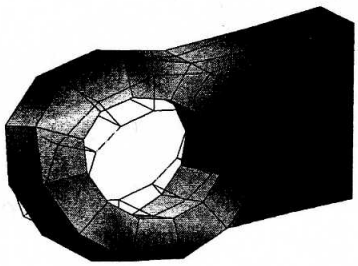

Figure 8. Plot the free-vibration mode.

Examples

We shall give some examples of the use of the MATLAB software framework in structural analysis classroom applications.

```
conn = get(gcells(i), 'conn'); % connectivity
x = gather(geom, conn, 'values', 'noreshape'); % coordinates of nodes
Ke = zeros(dim*nfens); % element stiffness matrix
for j=1:npts_per_gcell % Loop over all integration points
    Nder = Ndermat_param (gcells(i), pc(j,:)); % parametric derivatives
    [Nspatialder,J] = Ndermat_spatial (gcells(i), Nder, x); % xyz derivatives
    B = Blmat (gcells(i), Nspatialder); % strain-displacement matrix
    D = tangent_moduli (mat, matstates{i,j}, eye(3,3), eye(3,3), 'Lagrangean');
    Ke = Ke + w(j)*B'*D*B *J;
end
```

Figure 7. Calculation of the element stiffness matrix.

Calculation of element stiffness matrices

One of the most attractive attributes of MATLAB code is its succinctness. For instance, the stiffness matrix for linear elastic stress analysis is expressed as

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^T \boldsymbol{D}\, \boldsymbol{B}\, dV \tag{1}$$

where $\boldsymbol{B}$ is the strain displacement matrix, $\boldsymbol{D}$ is the property matrix (material stiffness), and V_e is the element volume. Equation (1) may be approximated with numerical quadrature as

$$\boldsymbol{K}_e \approx \sum_{i=1}^{n} w_i \boldsymbol{B}^T(\xi_i)\, \boldsymbol{D}(\xi_i)\, \boldsymbol{B}(\xi_i)\, J(\xi_i) \tag{2}$$

where ξ_i is the location of the quadrature point in the parametric coordinates, J is the Jacobian of the map from the parametric coordinates to the physical space, and w_i is the quadrature point weight. This may be expressed in very compact form in MATLAB as shown in Figure 9. In the first two lines the connectivity of the element and the coordinates of the nodes are collected into local variables. **Ndermat_param** calculates the derivatives of the basis functions with respect to the parametric coordinates, and **Ndermat_spatial** converts these into spatial derivatives using the isoparametric interpolation concept. In

particular, note that the spatial coordinates are retrieved from the geometry field, **geom,** which makes it straightforward to compute the derivatives in various configurations. The concept of using a geometry field is surprisingly easy to explain to the students, and leads naturally to better understanding of the concepts of total and updated Lagrangean and Eulerian formulations of nonlinear kinematics.

Note that **Ndermat_param, Ndermat_spatial,** and **Blmat** are methods defined on the various types of finite elements (hexahedra, tetrahedra, linear, quadratic,...) and are dispatched dynamically which leads to natural dynamic polymorphism. Thus, the loop in Figure 9 is valid for any isoparametric element type.

The call to **tangent_moduli** is another example of dynamic dispatch, this time on the material object. Note that the material state is passed along for uniformity even though the material is linearly elastic as advertised above.

Development of material models

Formulation and development of complex material models is at the heart of computational solid and structural mechanics. The MATLAB software framework FEALAB supports these activities by providing a set of classes that represent material state. In nonlinear calculations, the material update is a crucial operation. Figure 10 shows the code for the material state update for a hyper elastic neo-Hookean material type. The method is actually defined for a material, and the material state **ms** is passed in as an argument. The update is deformation driven (the deformation gradients at the beginning of the timestep and at the end of the timestep are passed as arguments). The updated material state and the calculated stress are returned as output. This abstraction allows us to formulate simple MATLAB drivers that test material models by supplying manufactured deformation gradients and compare calculated stress to expected values of stress. Students find this way of programming the material state update easier to understand than conventionally coded material routines because the parcel of information that needs to be understood in one lump is very compact.

```
function [stress, newms] = update (self, ms, F1, F, stress_type)
E = self.E; nu = self.nu;
lambda = E * nu / (1 + nu) / (1 - 2*(nu));        mu = E / (2 * (1 + nu));

b=F1*F1'; % Finger deformation tensor
J=det(F1);
sigma = mu/J * (b - eye(3,3)) + lambda *log(J)/J * eye(3,3);
switch stress_type
   case '2ndPK'
      invF1=inv(F1);
      stress = stress_tensor_to_6_vector(self.mater,J * invF1*sigma*invF1');

   otherwise
      stress = stress_tensor_to_6_vector(self.mater,sigma);
end
C=F1'*F1; % Green deformation tensor
ms.strain_energy = mu/2*(trace(C)-3) - mu*log(J) + lambda/2*(log(J))^2;
newms = ms;
return;
```

Figure 8. Method for updating the state of the neo-Hookean hyperelastic material.

Exploration of mass lumping

```
M = sparse_sysmat;
M = start (M, get(u, 'neqns'));
ems = mass(feb, geom, u);
M = assemble (M, ems);
M = finish (M);
```

Figure 9. Calculation of element mass matrices and assembly of the system mass matrix.

Mass lumping techniques have a way of affecting results for vibration and wave propagation problems in solid and structural dynamics. FEALAB provides a mechanism for calculation of element and system matrices that is amenable to easy to understand explorations of the effects of mass lumping. Figure 11 shows code to assemble the consistent mass matrix. The first two lines initialize the system mass matrix as an empty sparse matrix, and the last line essentially gives the mass matrix the chance to finalize its state before it gets used. The third and fourth lines carry out the calculation of the element mass matrices, and the assembly of these elements matrices into the system matrix. Therefore, the FEALAB Toolkit does not need to provide a specialized method for the calculation of lumped mass matrices, since all the element mass matrices are available for modification. Figure 12 demonstrates how the lumping of the mass matrices may be effected by the row-sum technique.

```
M = sparse_sysmat;
M = start (M, get(u, 'neqns'));
ems = mass(feb, geom, u);
for i=1:length(ems)
   Me=get(ems(i),'mat');  % get as consistent element mass matrix
   ems(i)=set(ems(i),'mat',diag(sum(Me))); % store as lumped element mass matrix
end
M = assemble (M, ems);
M = finish (M);
```

Figure 10. Calculation of consistent element mass matrices, conversion to lumped mass matrices, and assembly of the system mass matrix.

Conclusions

We have described some of our efforts aimed at providing undergraduate and graduate students with tools for numerical experimentation, and in particular for hands-on exploration of simulation tools for finite element solid and structural analysis. The object-oriented approach has proven to be successful at doling out the information that needs to be digested in small, easily understandable packages. Student feedback shows this quite clearly.

MATLAB proved to be a good tool for the intended use, especially due to its interactivity and completeness of the computational environment. The brevity with which matrix operations may be expressed is particularly helpful.

When designing object-oriented software whose primary goal is to facilitate understanding in MATLAB, we have to deal with one or two troublesome aspects. In particular, the MATLAB way of passing arguments forces the designer to use non-obvious tricks to propagate the change of state of objects affected by method invocations. That makes for trouble, as we have experienced with our students who have often had to struggle, especially on the undergraduate level where students typically come equipped with fresh knowledge of C++ or Java.

We never worry about computational inefficiency here. The problems that may be solved with the described MATLAB finite element toolkit are really small (typically several hundred equations), but large-scale computation was never the goal. We have always aimed at creating software for experimentation and interactive exploration.

References

1. P. Krysl, W. T. Ramroth, L. K. Stewart, and R. J. Asaro (2004): **Finite Element Modelling of Fiber Reinforced Polymer Sandwich Panels Exposed to Heat**, International Journal for Numerical Methods in Engineering, to appear.
2. Petr Krysl, Abhishek Trivedi, Baozhi Zhu (2004): **Object-Oriented Hierarchical Mesh Refinement with CHARMS**, International Journal for Numerical Methods in Engineering, 60 (8): 1401-1424.
3. Endres, L. and P. Krysl (2004): **Octasection-based Refinement of Finite Element Approximations on Tetrahedral Meshes that Guarantees Shape Quality,** *International Journal for Numerical Methods in Engineering*, 59 (1): 69-82.
4. Hammon, P. and P. Krysl (2003): **Implementation of a General Mesh Refinement Technique,** *CTU Reports, Czech Technical University in Prague, 1, Vol. 7, 57—70.*
5. Krysl, P., E. Grinspun, and P. Schröder (2003): **Natural Hierarchical Refinement for Finite Element Methods**, *International Journal for Numerical Methods in Engineering*, v.56(8), 1109-1124.
6. MATLAB documentation on the Web: http://www.mathworks.com/access/helpdesk/help/helpdesk.html
7. Peter Webb and Gregory V. Wilson (1999): **MATLAB as a Scripting Language. A simple way to do powerful things,** *Dr. Dobb's Journal* January 1999.
8. **FEMLAB, multiphysics modeling.** www.comsol.com (2003).
9. **Diffman, an object oriented MATLAB toolbox for solving differential equations on manifolds.** www.ii.uib.no/diffman/ (2003).

Biographical information

PETR KRYSL

Petr Krysl is an assistant professor at the University of California, San Diego. He holds a Ph.D. in Theoretical and Applied Mechanics 1993, and MSc. in Statics and dynamics of structures 1987, both from the Czech technical University in Prague. Petr Krysl teaches mechanics, numerical mathematics, and structural analysis courses, and does research in various areas of computational mechanics.

ABHISHEK TRIVEDI

Abhishek Trivedi is a Ph.D. student at the University of California, San Diego, and contributed to this work as teaching assistant for the undergraduate course SE131 "Finite element analysis" which uses the described MATLAB tools.