Teaching and Learning Structural Engineering Analysis with MATLAB

Abstract

The paper presents several examples of how the author uses the analysis and visualization software MATLAB in teaching analysis courses in a structural engineering university program. It also addresses how students work with the software to accelerate learning and deepen understanding.

Examples include:

- Animating the vibration behavior of a building frame during earthquake excitation with and without base isolation (only conference presentation, not included in paper).
- Assembling structural stiffness matrices in the direct stiffness and finite element methods.
- Visualizing stresses in a beam with holes.
- Behavior of a shear wall with openings under lateral loading.
- Illustrating the idea behind influence lines using a truss bridge.

It is shown how the many built-in matrix and graphics functions enable students to focus on the logic behind an analysis method rather than time-consuming programming and debugging. The paper includes a remarkably effortless and efficient technique in MATLAB for finite element mesh generation of areas with openings. Shape and number of openings are arbitrary provided we can describe them by mathematical functions.

1.0 Introduction

MATLAB\(^1\) is a software tool with powerful computational and graphics presentation capabilities widely used in education and research. It is valuable for teaching structural analysis, in particular modern matrix procedures like the direct stiffness and finite element methods. The popularity of MATLAB in teaching analysis in structural engineering is due to its ease of use through a variety of built-in functions well suited for structural analysis and its powerful graphic capabilities.

The author has taught structural analysis classes at California Polytechnic State University in San Luis Obispo (Cal Poly) for many years and has used MATLAB in his research and for his lecture notes. His students have worked with the software for their homework assignments and senior projects. The main objective of having the students work with MATLAB is to help them appreciate principles and concepts of structural analysis and the clarity of their formulation and to teach them to think logically. MATLAB is also ideal for understanding structural behavior through parametric studies. In special purpose commercial structural analysis software, changing parameters often requires cumbersome clicking through numerous levels of windows. In MATLAB, on the other hand, users enjoy changing input information in an "old-fashioned" text file, the MATLAB m-file.
2.0 Basics of finite element analysis in MATLAB

In what follows we give a brief overview of key MATLAB statements that are used to carry out important computational steps in a finite element analysis.

2.1 Adding element stiffness matrix to structure stiffness matrix

Let us consider the simple finite element mesh above consisting of 80 four-node elements and 99 nodes. We use the standard four-node element with two degrees-of-freedom per node (eight degrees-of-freedom total), the displacements in the horizontal and vertical directions. The structure thus has 198 degrees-of-freedom. For the sake of argument, we place the 8x8 element stiffness matrix \( k \) of element 24 (the element highlighted above) into the 198x198 structure stiffness matrix \( K \). Since element 24 has nodes 26, 27, 38 and 37, the proper location of the element stiffness matrix in the structure stiffness matrix is defined by the eight-component “location” vector

\[
\text{loc} = \begin{bmatrix}
2 \cdot 26 - 1 & 2 \cdot 26 & 2 \cdot 27 - 1 & 2 \cdot 27 & 2 \cdot 38 - 1 & 2 \cdot 38 & 2 \cdot 37 - 1 & 2 \cdot 37 \\
51 & 52 & 53 & 54 & 75 & 76 & 73 & 74
\end{bmatrix}
\]

The location vector is the fundamental piece of information required in the assembly loop that calculates the structure stiffness matrix \( K \) (see following statements).
$K = \text{sparse}(\text{dim, dim}); \quad \% \text{dim } = 198$

for element = 1:numele \quad \% \text{numele } = 80

\text{%calculate element stiffness matrix}
kele = elestiffness()

\text{%calculate location vector}
loc = ....

$K(\text{loc, loc}) = K(\text{loc, loc}) + \text{kele}$

end

Students usually struggle for some time before they grasp the important statement above, which is one of the key statements in a finite element analysis. Understanding begins, once the student recognizes MATLAB’s submatrix capabilities, i.e. that the location vector $\text{loc}$ temporarily reduces the $198 \times 198$ structure stiffness matrix to a $8 \times 8$ submatrix to which the element stiffness matrix is added.

Figure 2 provides an illustration of the critical statement above, i.e. how we assemble the individual element stiffness matrices to form the structure stiffness matrix. For simplicity, each of the six nodes of the eight-element structure is assumed to have only one degree-of-freedom. Each quartet of x-symbols thus represents the $2 \times 2$ element stiffness matrix. An empty box stands for a zero entry in the structure stiffness matrix. We observe that a matrix element $a_{ij}$ is non-zero only if the corresponding nodes $i$ and $j$ are directly connected by an element. Figure 2 schematically depicts the state of the $6 \times 6$ structure stiffness matrix each time one of the eight elements, represented by their $2 \times 2$ element stiffness matrix, is added to it. Several x-symbols in one box indicate that the corresponding numerical values should be added.

When the structure stiffness matrix assembly process is animated in class according to Fig. 2, students generally grasp the underlying concept quickly. As mentioned before, it requires much more effort to convert this visual understanding into appropriate programming statements as the ones above.
Figure 2: Illustration of structure stiffness matrix assembly
2.2 Displacement boundary conditions

To efficiently incorporate the displacement boundary conditions into MATLAB (we consider only homogeneous boundary conditions here), we use the *find* command to locate the fixed degrees-of-freedom (see Fig. 1).

```matlab
%boundary condition at pin connection
fixnode = find(x==0 & y==0)
fixdof  = [2*fixnode-1 2*fixnode];

%boundary condition at roller connection
fixnode = find(x==Lx & y==0)
fixdof  = [fixdof 2*fixnode];
```

In order to eliminate the fixed degrees-of-freedom from the stiffness matrix it is practical to specify the complementary set, the free degrees-of-freedom, which are obtained from the fixed degrees-of-freedom by the operations

```matlab
dof        = zeros(1,dim)  %dim= number of dofs(fixed and free combined)
dof(fixdof)=1;
free       = find(dof==0);
```

We reduce the structure force vector $F$ and the structure stiffness matrix $K$ to form the corresponding quantities $F_{\text{free}}$ and $K_{\text{Free}}$, solve the set of linear equations for the vector $q_{\text{free}}$ of structure displacements and finally add the prescribed zero displacements to the solution vector using the following statements.

```matlab
%initialize displacement vector
q       = zeros(dim,1);

%reduce stiffness matrix (eliminate rows and columns representing fixed dofs)
Kfree   = K(free,free);

%reduce force vector
Ffree   = F(free);

%solve equations
qfree   = Kfree \ Ffree;

%include fixed degrees of freedom in displacement vector
q(free) = qfree;
```
Similarly to the statement

\[ K(\text{loc},\text{loc}) = K(\text{loc},\text{loc}) + k_{\text{ele}} \]

discussed before, which adds the element stiffness matrix to the structure stiffness matrix, students need time to understand the sequence of commands above involving sub-matrix and sub-vector operations.

### 2.3 Applying forces to finite element model

We apply the load to the model (see Fig. 1) by using the `find` command again.

```matlab
%distributed load in vertical direction (2nd degree of freedom at loaded %nodes)
loadnode = find(y==Ly); %find all nodes on top of beam
F(2*loadnode) = F;

%load at corner points
loadnode = find(y==Ly & (x==0 | x==Lx));
F(2*loadnode) = F/2;
```
3.0 Example applications

3.1 General remarks

In what follows we present two examples of student assignments in a first course on finite element analysis for structural engineers in the Architectural Engineering Master Program at Cal Poly San Luis Obispo. The instructor provides students with a core version of a MATLAB program containing functions that calculate the $x$- and $y$-coordinates of a rectangular grid, the connectivity information, i.e. the four nodes defining each element, the element stiffness matrix, the displacement-to-strain transformation matrix and the elasticity matrix. Students then perform parametric studies varying the finite element mesh density and other structural parameters.

This section also contains an illustration generated in MATLAB to visualize the concept of influence lines that the author uses in a classical third-year structural analysis class.

3.2 Beam with holes

Figure 3: (a) Beam with holes. (b) Corresponding finite element model

In the first example students investigate how the number of holes and their diameter relative to the depth of the cross section affect the stiffness and the maximum stresses in the beam. The plane stress four-node element with incompatible modes is used. The interested reader finds a MATLAB function calculating the element stiffness matrix in the Appendix. Stiffness and maximum stress of the perforated beam are compared to the corresponding values for a beam without holes obtained from conventional beam theory. Students experiment with a brute force method of meshing around the holes by starting with a finite element mesh of the solid beam and then calculating the distance from the center of each circle to the midpoint of all elements. Only those elements are retained in the analysis whose midpoint lies outside of all circles. If the finite element mesh is fine, this extraordinary simple and efficient procedure, which constitutes a particular highlight of this project, approximates the circles well (see Fig. 3). A summary of the corresponding MATLAB statements is provided below.
%CALCULATE COORDINATES OF CENTER OF EACH CIRCLE
delcir = Lx/numcir; %distance between centers of circles
for cir = 1:numcir;
    XR(cir) = delcir/2 + (cir-1)*delcir;
    YR(cir) = Ly/2;
end

%CALCULATE DISTANCE FROM CENTER OF EACH CIRCLE TO MIDPOINT OF ALL
%ELEMENTS AND COMPARE TO RADIUS OF CIRCLE
%RETAIN ELEMENTS THAT LIE OUTSIDES OF ALL CIRCLES
eleretain = [1:nele]; %initialize
for i = 1:numcir
    r1 = sqrt((xm-XR(i)).^2 + (ym-YR(i)).^2);
    a1 = find( r1 > rad); %Find elements whose midpoints are outside circle i
    e1eretain = intersect(e1eretain,a1);
end

Students use MATLAB’s patch and surf commands to plot the response of the beam (see Figs. 4 and 5).

Figure 4: Deflected shape of beam with five holes ($D/H = 0.6$)

Figure 5: Qualitative axial stresses (a, b) and shear stresses (c, d) for beam with and without holes ($D/H = 0.6$)
3.3 Shear wall with opening

Most of the Architectural Engineering students at Cal Poly enter the structural engineering profession in California, which is heavily focused on seismic design. Assessing the lateral stiffness of shear walls, an important parameter in predicting the response of shear wall buildings during earthquakes, is thus a typical problem facing our graduates in professional practice.

The behavior of a solid prismatic wall is well understood and a sufficiently accurate structural model is that of a deep beam. The structural behavior of shear walls containing openings, however, is significantly more complex because the openings cause disturbed regions in which “loads turn corners” resulting in irregular stress flow such that standard deep beam theory does not apply. The structural problem of shear walls with openings is that of general plane stress for which analytical solutions generally do not exist (see Fig. 6).

![Figure 6: (a) Qualitative deflected shape and (b) qualitative shear stresses for wall with and without opening under lateral loading](image)

This example considers a shear wall of variable aspect ratio $H / L$ with a horizontally and vertically centered rectangular opening of dimensions $L/3 \times H/3$ as shown in Fig. 7. We choose $H = 3.60\, \text{m}$ for the height of the wall and vary its length $L$ corresponding to a selected aspect ratio $H / L$. We determine the lateral stiffness of the wall using both a popular method for hand calculation (see reference 2) and finite element analysis. Selected values for the aspect ratio are $H / L = 0.4, 0.6, 0.8, \ldots, 3$. 
Figure 7: Shear wall with variable aspect ratio $H/L$

Figure 8(a) shows the stiffness of the wall with opening relative to that of the solid wall for both hand calculation and finite element analysis. In the hand method, the opening reduces the stiffness fairly uniformly by about 10-15% for all aspect ratios. The reduction in stiffness due to the opening is thus not sensitive to the aspect ratio of the wall. Results of finite element analysis contrast sharply. For the shear-dominated wall with small aspect ratio the opening reduces the stiffness by almost 50% whereas for the flexure-dominated wall with large aspect ratio the stiffness is only about 20% smaller than that of the solid wall. This suggests that the stiffness reduction strongly depends on the aspect ratio, a remarkable finding considering that the area of the opening is $(1/3) \cdot (1/3) = 11\%$ of the area of the wall regardless of the aspect ratio.

Figure 8(b) summarizes the discrepancy between hand and finite element analyses. For walls governed by shear deformation, the hand procedure overestimates the lateral stiffness by as much as 60%. Hand calculation results improve with increasing aspect ratio. For tall walls dominated by flexural deformation the error reduces to about 15%.
3.4 Illustration of influence lines

Influence lines provide important information in structural engineering regarding the controlling load position of moving loads and are particularly valuable in the analysis and design of bridges. It is important for students to distinguish between the concept of an internal force diagram and that of an influence line. An internal force diagram shows the magnitude of an internal force, say the axial force, in all members of the structure due to a stationary loading. An influence line, on the other hand, is a diagram showing the variation of an internal force in a single member of the structure as a load moves across the structure. The author uses his structural analysis program in MATLAB to visualize this concept. Each of the four diagrams in Fig. 9 constitutes an internal force diagram. An influence line by contrast is the variation (the change in color in Fig. 9) of a force in one single member, say the vertical bar in the leftmost triangular panel of the truss bridge, as the load changes its position.
Figure 9: Illustration of influence line and internal force diagram

Figure 9: Illustration of influence line and internal force diagram

Compression

Tension
4.0 Summary

The author considers MATLAB an invaluable tool for teaching and learning the analytical aspects of structural engineering. Its superb graphic capabilities and vast array of built-in functions are particularly suited for courses on matrix structural analysis like the direct stiffness and finite element methods. Students are relieved from often time consuming and frustrating programming details and often report that implementing a structural analysis methodology in MATLAB has been enjoyable and has helped them deepen their understanding. This perception is particularly prevalent among Master students.

5.0 Acknowledgement

The author would like to express his sincere gratitude to his colleagues Kevin Dong and Allen Estes, Associate Professor and Department Head, respectively, in the Architectural Engineering Department at Cal Poly. Their tireless efforts in developing an ARCE Master Program and obtaining its approval have given the author the opportunity to teach his much-loved finite elements.

Bibliography

1. MATLAB, The MathWorks, Inc., Natick, MA
%CALCULATES THE DISPLACEMENT-TO-STRAIN MATRIX B
%FOR A 4-node RECTANGULAR PLANE STRESS ELEMENT

%INPUT:
%xi, eta: element coordinates
%a, b: element length and width
%incom=0: conventional formulation
%incom=1: formulation with incompatible modes

%OUTPUT:
%B: 3x8 matrix

function [B] = BMATRIXMEMBRANE(xi,eta,a,b,incom);

if nargin==4; incom=0; end

if incom==0;
    B = zeros(3,8);
else
    B = zeros(3,12);
end

c1 = (1 - eta)/ a;
c2 = (1 + eta)/ a;
c3 = (1 - xi) / b;
c4 = (1 + xi) / b;
c5 = - 4*2*xi  / a^2;
c6 = - 4*2*eta / b^2;

if incom==0;
    B(1,:)  = [-c1   0    c1   0   c2  0  -c2    0];
    B(2,:)  = [   0  -c3 0  -c4  0  c4  0  c3 ];
    B(3,:)  = [-c3 -c1 -c4  c1  c4 c2 c3  -c2 ];
else
    B(1,:)  = [-c1   0    c1   0   c2 0  -c2    0  c5  0  0  0];
    B(2,:)  = [ 0  -c3 0  -c4 0  c4 0  c3 0  0  0  c6 ];
    B(3,:)  = [-c3 -c1 -c4 c1 c4 c2 c3 -c2 0  c6 c5 0];
end

B = B / 4;

%****************************
%CALCULATES THE 3x3 ELASTICITY MATRIX
%FOR PLANE STRESS
%****************************

function C = PlaneStressElastMatrix(E,nu);

%E: Young's modulus
%nu: Poisson's ratio

%C: 3x3 elasticity matrix

C = zeros(3,3);
C(1,1) = 1;
C(2,2) = 1;
C(1,2) = nu;
C(2,1) = nu;
C(3,3) = (1 - nu) ./ 2;
C = C .* E ./ (1 - nu.^2);
function KK = EleStiffMembrane4Node(a,b,E,nu,thick,incom);
%INPUT
%a:       half of element length in x-direction
%b:       half of element length in y-direction
%E:       modulus of elasticity
%nu:      POISSON's ratio
%thick:   thickness of element (out-of-plane dimension)
%incom=0: conventional formulation
%incom=1: formulation with incompatible modes

%OUTPUT
%KK = 8x8 ESM
if nargin==5; incom=0; end

%2 GAUSS points
g(1) = -1 / sqrt(3);
w(1) = 1;
g(2) = -g(1);
w(2) = 1;

%Elasticity matrix
C = PlaneStressElastMatrix(E,nu);
if incom==0;
    KK = zeros(8,8);
else
    KK = zeros(12,12);
end

%GAUSS integration
ngauss = 2;
for i = 1 : ngauss
    xi = g(i);
    for j = 1 : ngauss
        eta = g(j);
        B = BMatrixMembrane(xi,eta,a,b,incom);
        KK = KK + B' * C * B * w(i) .* w(j);
    end
end
KK = KK * thick * a * b;

%Static condensation to 8x8 matrix for incompatible modes
if incom==1
    dof1 = [1:8]; dof2 = [9:12];
    K11 = KK(dof1,dof1);
    K12 = KK(dof1,dof2);
    K21 = KK(dof2,dof1);
    K22 = KK(dof2,dof2);
    KK = K11 - K12*inv(K22)*K21;
end
%*******************************************************
%Main program to analyze beam with holes in Section 3.2
%*******************************************************

%length of beam
Lx = 12;
%depth of beam
Ly = 1.5;

x = [0 Lx Lx 0]';
y = [0 0 Ly Ly]';

rad = 0.3*Ly;
umcir = 5;

%icom=1 = include extra displacement shapes in element formulation
%icom=0 = exclude extra displacement shapes in element formulation
incom=1;

%number of elements in x-direction
nx = 120;
%number of elements in y-direction
ny = 30;

numele = nx * ny;

delx = Lx/nx;
dely = Ly/ny;
a = delx/2;
b = dely/2;

generate topology of structure (x and y coordinates and connectivity)
[x,y,nodes] = gen_grid_membrane(Lx,Ly,nx,ny);

%calculate midpoint of elements for circle cut-out
for i = 1:numele
  xm(i) = sum(x(nodes(i,:))) / 4;
  ym(i) = sum(y(nodes(i,:))) / 4;
end

%CALCULATE COORDINATES OF CENTER OF EACH CIRCLE
%distance between centers of circles
delcir = Lx/numcir;
for cir = 1:numcir
  XR(cir) = delcir/2 + (cir-1)*delcir;
  YR(cir) = Ly/2;
end

%CALCULATE DISTANCE FROM CENTER OF EACH CIRCLE TO MIDPOINT OF ALL ELEMENTS AND COMPARE TO RADIUS OF CIRCLE
%RETAIN ELEMENTS THAT LIE OUTSIDES OF ALL CIRCLES
elein = [1:numele];
for i=1:numcir
  r1 = sqrt( (xm-XR(i)).^2 + (ym-YR(i)).^2 );
  a1 = find( r1 > rad);
  elein = intersect(elein,a1);
end

nodes = nodes(elein,:);
[numele aux] = size(nodes);
umno = (nx+1) * (ny+1);
dim = 2*numno;
E = 1600;
nu = 0.2;
thick = 1;
K = sparse(dim,dim);
for i=1:dim;K(i,i) = eps; end
F = zeros(dim,1);
q = zeros(dim,1);

%distributed load
load = find( abs(y-Ly) < 0.00001);
F(2*load) = -delx;

%distributed load corner points
load = find( (abs(x) < 0.00001 & abs(y-Ly) < 0.00001 ) | (abs(x-Lx) < 0.00001 & abs(y-Ly) < 0.00001 ));
F(2*load) = -delx/2;

%boundary condition in x-direction
fix = find( abs(x-Lx)< 0.0001 & abs(y < 0.0001 ) );
fix = 2*fix;

%boundary condition in y-direction
fix1 = find( (x == 0   & y==0) );
fix1 = [2*fix1-1 2*fix1];
fix = [fix fix1];

dof = zeros(1,dim);dof(fix) =1;
free = find(dof==0);

%calculate element stiffness matrix (same for all elements)
k = EleStiffMembrane4Node(a,b,E,nu,thick,incom);

%calculate structure stiffness matrix
for i=1:numele
  no = nodes(i,1:4)';
  loc = zeros(8,1);loc(1:2:8) = 2*no-1;loc(2:2:8) = 2*no;
  K(loc,loc) = K(loc,loc) + k;
end

%reduce structure stiffness matrix to include boundary conditions
Kfree = K(free,free);
Ffree = F(free);

%solve equations
qfree = Kfree \ Ffree;
q(free) = qfree;

%draw deflected shape
figure;hold
mag = 0.10 * Lx / max(abs(q));
xmag = q(1:2:2*numno) * mag + x;
ymag = q(2:2:2*numno) * mag + y;
draw PATCH(xmag(nodes'),ymag(nodes'))
set(gca,'fontsize',16) 
axis('equal') 
colorbar 
h = colorbar; 
set(h,'fontsize',20) 
shading interp

%function stress = calculateStressesPlane2D(a,b,E,nu,thick,q,nodes,incom)
stress = calculateStressesPlane2D(a,b,E,nu,thick,q,nodes,incom)

%CONTOUR PLOT OF STRESSES
figure;hold
for i=1:numele
  no = nodes(i,1:4)';
  sxx = stress(i,:,1);
sxx1 = [sxx(1) sxx(2);sxx(4) sxx(3)];
  xx = x(no(1:2));
  yy = y(no(2:3));
surf(xx,yy,sxx1)
end
set(gca,'fontsize',16) 
axis('equal') 
colorbar 
h = colorbar; 
set(h,'fontsize',20) 
shading interp
title('STRESSES $\sigma_{x\,x}$ [KSF]', 'fontsize', 16)

% CONTOUR PLOT OF SHEAR STRESSES
figure; hold
for i = 1:nele
    no   = nodes(i, 1:4);
    sxx  = stress(i, :, 3);
    sxx1 = [sxx(1) sxx(2); sxx(4) sxx(3)];
    xx   = x(no(1:2));
    yy   = y(no(2:3));
surf(xx, yy, sxx1)
end
set(gca, 'fontsize', 16)
axis('equal')
shading interp
colorbar
h = colorbar;
set(h, 'fontsize', 20)
title('SHEAR STRESSES $\sigma_{x\,y}$ [KSF]', 'fontsize', 16)