1 Introduction

This short introductory chapter is divided into two parts. In the first part there is a summary of the steps of the finite element method. The second part includes a short tutorial on MATLAB.

1.1 Steps of the Finite Element Method

There are many excellent textbooks available on finite element analysis like those in [1–18]. Therefore this book will not present any theoretical formulations or derivations of finite element equations. Only the main equations are summarized for each chapter followed by examples. In addition only problems from linear elastic structural mechanics are used throughout the book.

The finite element method is a numerical procedure for solving engineering problems. Linear elastic behavior is assumed throughout this book. The problems in this book are taken from structural engineering but the method can be applied to other fields of engineering as well. In this book six steps are used to solve each problem using finite elements. The six steps of finite element analysis are summarized as follows:

1. Discretizing the domain – this step involves subdividing the domain into elements and nodes. For discrete systems like trusses and frames the system is already discretized and this step is unnecessary. In this case the answers obtained are exact. However, for continuous systems like plates and shells this step becomes very important and the answers obtained are only approximate. In this case, the accuracy of the solution depends on the discretization used. In this book this step will be performed manually (for continuous systems).
2. Writing the element stiffness matrices – the element stiffness equations need to be written for each element in the domain. In this book this step will be performed using MATLAB.
3. Assembling the global stiffness matrix – this will be done using the direct stiffness approach. In this book this step will be performed using MATLAB.

4. Applying the boundary conditions – like supports and applied loads and displacements. In this book this step will be performed manually.
5. Solving the equations – this will be done by partitioning the global stiffness matrix and then solving the resulting equations using Gaussian elimination. In this book the partitioning process will be performed manually while the solution part will be performed using MATLAB with Gaussian elimination.
6. Post-processing – to obtain additional information like the reactions and element forces and stresses. In this book this step will be performed using MATLAB.

It is seen from the above steps that the solution process involves using a combination of MATLAB and some limited manual operations. The manual operations employed are very simple dealing only with discretization (step 1), applying boundary conditions (step 4) and partitioning the global stiffness matrix (part of step 5). It can be seen that all the tedious, lengthy and repetitive calculations will be performed using MATLAB.

1.2 MATLAB Functions for Finite Element Analysis

The CD-ROM accompanying this book includes 84 MATLAB functions (M-files) specifically written by the author to be used for finite element analysis with this book. They comprise what may be called the MATLAB Finite Element Toolbox. These functions have been tested with version 7 of MATLAB and should work with any later versions. The following is a listing of all the functions available on the CD-ROM. The reader can refer to each chapter for specific usage details.

SpringElementStiffness(k)
SpringAssemble(K, k, i, j)
SpringElementForces(k, u)

LinearBarElementStiffness(E, A, L)
LinearBarAssemble(K, k, i, j)
LinearBarElementForces(k, u)
LinearBarElementStresses(k, u, A)

QuadraticBarElementStiffness(E, A, L)
QuadraticBarAssemble(K, k, i, j, m)
QuadraticBarElementForces(k, u)
QuadraticBarElementStresses(k, u, A)

PlaneTrussElementLength(x_1, y_1, x_2, y_2)
PlaneTrussElementStiffness(E, A, L, *theta*)
PlaneTrussAssemble(K, k, i, j)
PlaneTrussElementForce(E, A, L, *theta*, u)

PlaneTrussElementStress(E, L, *theta*, u)
PlaneTrussInclinedSupport(T, i, *alpha*)

SpaceTrussElementLength(x_1, y_1, z_1, x_2, y_2, z_2)
SpaceTrussElementStiffness(E, A, L, *thetax*, *thetay*, *thetaz*)
SpaceTrussAssemble(K, k, i, j)
SpaceTrussElementForce(E, A, L, *thetax*, *thetay*, *thetaz*, u)
SpaceTrussElementStress(E, L, *thetax*, *thetay*, *thetaz*, u)

BeamElementStiffness(E, I, L)
BeamAssemble(K, k, i, j)
BeamElementForces(k, u)
BeamElementShearDiagram(f, L)
BeamElementMomentDiagram(f, L)

PlaneFrameElementLength(x_1, y_1, x_2, y_2)
PlaneFrameElementStiffness(E, A, I, L, *theta*)
PlaneFrameAssemble(K, k, i, j)
PlaneFrameElementForces(E, A, I, L, *theta*, u)
PlaneFrameElementAxialDiagram(f, L)
PlaneFrameElementShearDiagram(f, L)
PlaneFrameElementMomentDiagram(f, L)
PlaneFrameInclinedSupport(T, i, *alpha*)

GridElementLength(x_1, y_1, x_2, y_2)
GridElementStiffness(E, G, I, J, L, *theta*)
GridAssemble(K, k, i, j)
GridElementForces(E, G, I, J, L, *theta*, u)

SpaceFrameElementLength(x_1, y_1, z_1, x_2, y_2, z_2)
SpaceFrameElementStiffness(E, G, A, I_y, I_z, J, x_1, y_1, z_1, x_2, y_2, z_2)
SpaceFrameAssemble(K, k, i, j)
SpaceFrameElementForces(E, G, A, I_y, I_z, J, x_1, y_1, z_1, x_2, y_2, z_2, u)
SpaceFrameElementAxialDiagram(f, L)
SpaceFrameElementShearZDiagram(f, L)
SpaceFrameElementShearYDiagram(f, L)
SpaceFrameElementTorsionDiagram(f, L)
SpaceFrameElementMomentZDiagram(f, L)
SpaceFrameElementMomentYDiagram(f, L)

LinearTriangleElementArea(x_i, y_i, x_j, y_j, x_m, y_m)
LinearTriangleElementStiffness(E, *NU*, t, x_i, y_i, x_j, y_j, x_m, y_m, p)
LinearTriangleAssemble(K, k, i, j, m)
LinearTriangleElementStresses(E, *NU*, t, x_i, y_i, x_j, y_j, x_m, y_m, p, u)
LinearTriangleElementPStresses(*sigma*)

QuadTriangleElementArea(x_1, y_1, x_2, y_2, x_3, y_3)
QuadTriangleElementStiffness(E, *NU*, t, x_1, y_1, x_2, y_2, x_3, y_3, p)
QuadTriangleAssemble(K, k, i, j, m, p, q, r)
QuadTriangleElementStresses(E, *NU*, t, x_1, y_1, x_2, y_2, x_3, y_3, p, u)
QuadTriangleElementPStresses(*sigma*)

BilinearQuadElementArea(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)
BilinearQuadElementStiffness(E, *NU*, t, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, p)
BilinearQuadElementStiffness2(E, *NU*, t, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, p)
BilinearQuadAssemble(K, k, i, j, m, n)
BilinearQuadElementStresses(E, *NU*, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, p, u)
BilinearQuadElementPStresses(*sigma*)

QuadraticQuadElementArea(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)
QuadraticQuadElementStiffness(E, *NU*, t, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, p)
QuadraticQuadAssemble(K, k, i, j, m, p, q, r, s, t)
QuadraticQuadElementStresses(E, *NU*, x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4, p, u)
QuadraticQuadElementPStresses(*sigma*)

TetrahedronElementVolume(x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4)
TetrahedronElementStiffness(E, *NU*, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4)
TetrahedronAssemble(K, k, i, j, m, n)
TetrahedronElementStresses(E, *NU*, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, u)
TetrahedronElementPStresses(*sigma*)

LinearBrickElementVolume(x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7, x_8, y_8, z_8)
LinearBrickElementStiffness(E, *NU*, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7, x_8, y_8, z_8)
LinearBrickAssemble(K, k, i, j, m, n, p, q, r, s)
LinearBrickElementStresses(E, *NU*, x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3, x_4, y_4, z_4, x_5, y_5, z_5, x_6, y_6, z_6, x_7, y_7, z_7, x_8, y_8, z_8, u)
LinearBrickElementPStresses(*sigma*)

FluidFlow1DElementStiffness(K_{xx}, A, L)
FluidFlow1DAssemble(K, k, i, j)
FluidFlow1DElementVelocities(K_{xx}, L, p)
FluidFlow1DElementVFR(K_{xx}, L, p, A)

1.3 MATLAB Tutorial

In this section a very short MATLAB tutorial is provided. For more details consult the excellent books listed in [19–27] or the numerous freely available tutorials on the internet – see [28–35]. This tutorial is not comprehensive but describes the basic MATLAB commands that are used in this book.

In this tutorial it is assumed that you have started MATLAB on your system successfully and you are ready to type the commands at the MATLAB prompt (which is denoted by double arrows ">>"). Entering scalars and simple operations is easy as is shown in the examples below:

```
» 3*4+5

ans =

    17

» cos (30*pi/180)

ans =

    0.8660

» x=4

x =

    4

» 2/sqrt(3+x)

ans =

    0.7559
```

To suppress the output in MATLAB use a semicolon to end the command line as in the following examples. If the semicolon is not used then the output will be shown by MATLAB:

```
» y=32;
» z=5;
» x=2*y-z;
» w=3*y+4*z

w =

   116
```

MATLAB is case-sensitive, i.e. variables with lowercase letters are different than variables with uppercase letters. Consider the following examples using the variables x and X.

```
» x=1

x =

     1

» X=2

X =

     2

» x

x =

     1
```

Use the help command to obtain help on any particular MATLAB command. The following example demonstrates the use of `help` to obtain help on the `inv` command.

```
» help inv

INV    Matrix inverse.
   INV(X) is the inverse of the square matrix X.
   A warning message is printed if X is badly scaled or
   nearly singular.

   See also SLASH, PINV, COND, CONDEST, NNLS, LSCOV.

Overloaded methods
   help sym/inv.m
   help zpk/inv.m
   help tf/inv.m
   help ss/inv.m
   help lti/inv.m
   help frd/inv.m
```

The following examples show how to enter matrices and perform some simple matrix operations:

```
» x=[1 2 3 ; 4 5 6 ; 7 8 9]

x =

     1   2   3
     4   5   6
     7   8   9
```

```
» y=[2 ; 0 ; -3]

y =
     2
     0
    -3


» w=x*y

w =
    -7
   -10
   -13
```

Let us now solve the following system of simultaneous algebraic equations:

$$\begin{bmatrix} 2 & -1 & 3 & 0 \\ 1 & 5 & -2 & 4 \\ 2 & 0 & 3 & -2 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{Bmatrix} 3 \\ 1 \\ -2 \\ 2 \end{Bmatrix} \tag{1.1}$$

We will use Gaussian elimination to solve the above system of equations. This is performed in MATLAB by using the backslash operator "\" as follows:

```
» A=[2 -1 3 0 ; 1 5 -2 4 ; 2 0 3 -2 ; 1 2 3 4]

A =

     2   -1    3    0
     1    5   -2    4
     2    0    3   -2
     1    2    3    4


» b=[3 ; 1 ; -2 ; 2]

b =

     3
     1
    -2
     2
```

```
» x= A\b

x =

    1.9259
   -1.8148
   -0.8889
    1.5926
```

It is clear that the solution is $x_1 = 1.9259$, $x_2 = -1.8148$, $x_3 = -0.8889$, and $x_4 = 1.5926$. Alternatively, one can use the inverse matrix of A to obtain the same solution directly as follows:

```
» x=inv (A)*b

x =

     1.9259
    -1.8148
    -0.8889
     1.5926
```

It should be noted that using the inverse method usually takes longer that using Gaussian elimination especially for large systems. In this book we will use Gaussian elimination (i.e. the backslash operator "\").

Consider now the following 5×5 matrix D:

```
» D=[1 2 3 4 5 ; 2 4 6 8 9 ; 2 4 6 2 4 ; 1 1 2 3 -2 ; 9 0 2 3 1]

D =

    1   2   3   4    5
    2   4   6   8    9
    2   4   6   2    4
    1   1   2   3   -2
    9   0   2   3    1
```

We can extract the submatrix in rows 2 to 4 and columns 3 to 5 as follows:

```
» E=D (2:4, 3:5)

E =

    6   8    9
    6   2    4
    2   3   -2
```

We can extract the third column of D as follows:

```
» F=D(1:5,3)

F =

     3
     6
     6
     2
     2
```

We can also extract the second row of D as follows:

```
» G=D(2,1:5)

G =

     2     4     6     8     9
```

We can extract the element in row 4 and column 3 as follows:

```
» H=D(4,3)

H =

     2
```

Finally in order to plot a graph of the function $y = f(x)$, we use the MATLAB command `plot(x,y)` after we have adequately defined both vectors x and y. The following is a simple example.

```
» x=[1 2 3 4 5 6 7 8 9 10]

x =

     1     2     3     4     5     6     7     8     9    10

» y=x.^2

y =

     1     4     9    16    25    36    49    64    81   100

» plot(x,y)
```

Figure 1.1 shows the plot obtained by MATLAB. It is usually shown in a separate graphics window. In this figure no titles are given to the x and y-axes. These titles may be easily added to the figure using the `x-label` and `y-label` commands.

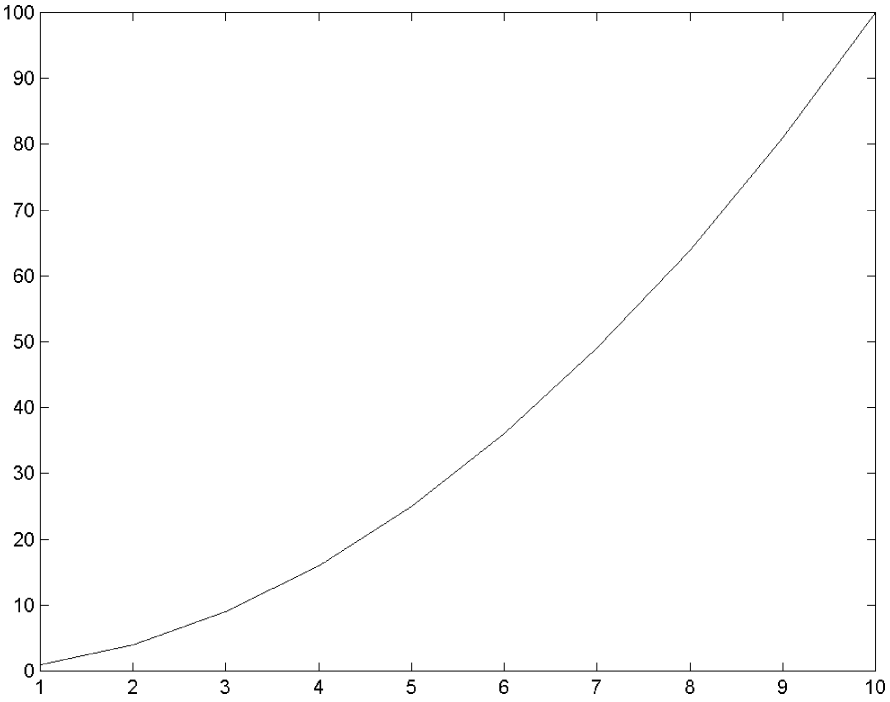

Fig. 1.1. Using the MATLAB `Plot` Command