Peter I. Kattan

MATLAB Guide to Finite Elements

An Interactive Approach

Second Edition
With 108 Figures and 25 Tables

Springer

Peter I. Kattan, PhD
P.O. BOX 1392
Amman 11118
Jordan
pkattan@tedata.net.jo
pkattan@lsu.edu

Library of Congress Control Number: 2007920902

ISBN-13 978-3-540-70697-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable for prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media
springer.com
© Springer-Verlag Berlin Heidelberg 2008

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Integra Software Services Pvt. Ltd., Pondicherry, India
Cover design: Erich Kirchner, Heidelberg

Printed on acid-free paper SPIN: 11301950 42/3100/Integra 5 4 3 2 1 0

Dedicated to My Professor, George Z. Voyiadjis

Preface to the Second Edition

Soon after the first edition of this book was published at the end of 2002, it was realized that a new edition of the book was needed. I received positive feedback from my readers who requested that I provide additional finite elements in other areas like fluid flow and heat transfer. However, I did not want to lengthen the book considerably. Therefore, I decided to add two new chapters thus adding new material while keeping the size of the book reasonable.

The second edition of the book continues with the same successful format that characterized the first edition – which was sold out in less than four years. I continue to emphasize the important features of interactivity of using MATLAB[1] coupled with the simplicity and consistency of presentation of finite elements. One of the most important features also is bypassing the use of numerical integration in favor of exact analytical integration with the use of the MATLAB Symbolic Math Toolbox[2]. The use of this toolbox is emphasized in Chaps. 12, 13, 14, and 16.

In the new edition, two important changes are immediately noted. First, I corrected the handful of typing errors that appeared in the first edition. Second, I added two new chapters. Chap. 16 includes another solid three-dimensional element (the eight-noded brick element) in great detail. The final chapter (Chap. 17) provides a review of the applications of finite elements in other areas like fluid flow, heat transfer, geotechnical engineering, electro-magnetics, structural dynamics, plasticity, etc. In this chapter, I show how the same consistent strategy that was followed in the first sixteen chapters can be used to write MATLAB functions in these areas by providing the MATLAB code for a one-dimensional fluid flow element.

One minor drawback of the first edition as I see it is the absence of a concluding chapter. Therefore, I decided to remedy the situation by adding Chap. 17 as a real concluding chapter to the book. It is clear that this chapter is different from the first sixteen chapters and thus may well provide a well written conclusion to the book.

The second edition still comes with an accompanying CD-ROM that contains the full set of M-files written specifically to be used with this book. These MATLAB functions have been tested with version 7 of MATLAB and should work with any

[1] MATLAB is a registered trademark of The MathWorks, Inc.

[2] The MATLAB Symbolic Math Toolbox is a registered trademark of The MathWorks, Inc.

later versions. In addition, the CD-ROM contains a complete solutions manual that includes detailed solutions to all the problems in the book. If the reader does not wish to consult these solutions, then a brief list of answers is provided in printed form at the end of the book.

I would like to thank my family members for their help and continued support without which this book would not have been possible. I would also like to acknowledge the help of the editior at Springer-Verlag (Dr. Thomas Ditzinger) for his assistance in bringing this book out in its present form. Finally, I would like to thank my brother, Nicola, for preparing most of the line drawings in both editions. In this edition, I am providing two email addresses for my readers to contact me (pkattan@tedata.net.jo and pkattan@lsu.edu). The old email address that appeared in the first edition was cancelled in 2004.

December 2006 Peter I. Kattan

Preface to the First Edition

This is a book for people who love finite elements and MATLAB[3]. We will use the popular computer package MATLAB as a matrix calculator for doing finite element analysis. Problems will be solved mainly using MATLAB to carry out the tedious and lengthy matrix calculations in addition to some manual manipulations especially when applying the boundary conditions. In particular the steps of the finite element method are emphasized in this book. The reader will not find ready-made MATLAB programs for use as black boxes. Instead step-by-step solutions of finite element problems are examined in detail using MATLAB. Problems from linear elastic structural mechanics are used throughout the book. The emphasis is not on mass computation or programming, but rather on learning the finite element method computations and understanding of the underlying concepts. In addition to MATLAB, the MATLAB Symbolic Math Toolbox[4] is used in Chaps. 12, 13, and 14.

Many types of finite elements are studied in this book including the spring element, the bar element, two-dimensional and three-dimensional truss elements, plane and space beam and frame elements, two-dimensional elasticity elements for plane stress and plane strain problems, and one three-dimensional solid element. Each chapter deals with only one type of element. Also each chapter starts with a summary of the basic equations for the element followed by a number of examples demonstrating the use of the element using the provided MATLAB functions. Special MATLAB functions for finite elements are provided as M-files on the accompanying CD-ROM to be used in the examples. These functions have been tested successfully with MATLAB versions 5.0, 5.3, and 6.1. They should work with other later versions. Each chapter also ends with a number of problems to be used as practice for students.

This book is written primarily for students studying finite element analysis for the first time. It is intended as a supplementary text to be used with a main textbook for an introductory course on the finite element method. Since the computations of finite elements usually involve matrices and matrix manipulations, it is only natural that students use a matrix-based software package like MATLAB to do the calculations.

[3] MATLAB is a registered trademark of The MathWorks, Inc.

[4] The MATLAB Symbolic Math Toolbox is a registered trademark of The MathWorks, Inc.

In fact the word MATLAB stands for MATrix LABoratory. The main features of the book are:

1. The book is divided into fifteen chapters that are well defined ad correlated.
2. The books includes a short tutorial on using MATLAB in Chap. 1.
3. The CD-ROM that accompanies the book includes 75 MATLAB functions (M-files) that are specifically written to be used with this book. These functions comprise what may be called the MATLAB Finite Element Toolbox. It is used mainly for problems in structural mechanics. The provided MATLAB functions are designed to be simple and easy to use.
4. A sequence of six steps is outlined in the first chapter for the finite element method. These six steps are then used systematically in each chapter throughout the book.
5. The book stresses the interactive use of MATLAB. Each example is solved in an interactive session with MATLAB. No ready-made subroutines are provided to be used as black boxes.
6. Answers to the all problems are provided at the end of the book.
7. A solutions manual is also provided on the accompanying CD-ROM. The solutions manual includes detailed solutions to all the problems in the book. It is over 300 pages in length.

The author wishes to thank the editors at Springer-Verlag (especially Dr. Thomas Ditzinger) for their cooperation and assistance during the writing of this book. Special thanks are also given to my family members without whose support and encouragement this book would not have been possible. In particular, I would like to thank Nicola Kattan for preparing most of the figures that appear in the book.

February 2002 Peter I. Kattan

Table of Contents

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.

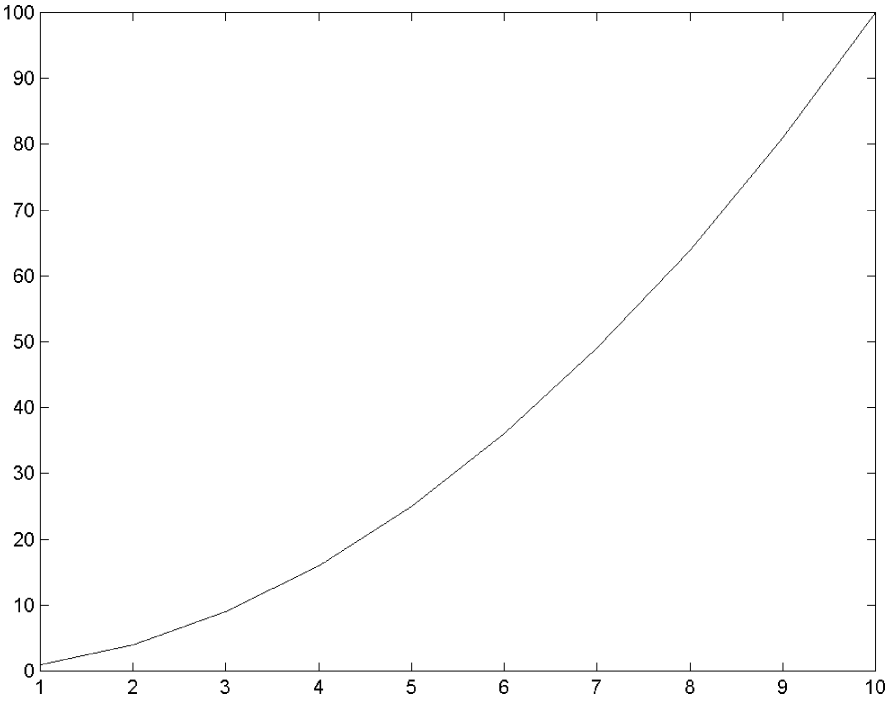

Fig. 1.1. Using the MATLAB `Plot` Command

2 The Spring Element

2.1 Basic Equations

The spring element is a one-dimensional finite element where the local and global coordinates coincide. It should be noted that the spring element is the simplest finite element available. Each spring element has two nodes as shown in Fig. 2.1. Let the stiffness of the spring be denoted by k. In this case the element stiffness matrix is given by (see [1], [8], and [18]).

$$k = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{2.1}$$

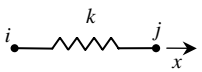

Fig. 2.1. The Spring Element

Obviously the element stiffness matrix for the spring element is a 2×2 matrix since the spring element has only two degrees of freedom – one at each node. Consequently for a system of spring elements with n nodes, the size of the global stiffness matrix K will be of size $n \times n$ (since we have one degree of freedom at each node). The global stiffness matrix K is obtained by assembling the element stiffness matrices $k_i (i = 1, 2, 3, \ldots .., n)$ using the direct stiffness approach. For example the element stiffness matrix k for a spring connecting nodes 4 and 5 in a system will be assembled into the global stiffness matrix K by adding its rows and columns to rows 4 and 5 and columns 4 and 5 of K. A special MATLAB function called *SpringAssemble* is written specifically for this purpose. This process will be illustrated in detail in the examples.

Once the global stiffness matrix K is obtained we have the following system equation:

$$[K]\{U\} = \{F\} \tag{2.2}$$

where U is the global nodal displacement vector and F is the global nodal force vector. At this step the boundary conditions are applied manually to the vectors U and F. Then the matrix (2.2) is solved by partitioning and Gaussian elimination. Finally once the unknown displacements and reactions are found, the element forces are obtained for each element as follows:

$$\{f\} = [k]\{u\} \tag{2.3}$$

where f is the 2×1 element force vector and u is the 2×1 element displacement vector.

2.2 MATLAB Functions Used

The three MATLAB functions used for the spring element are:

"*SpringElementStiffness*(k) – This function calculates the element stiffness matrix for each spring with stiffness k. It returns the 2×2 element stiffness matrix k."

SpringAssemble(K, k, i, j) – This functions assembles the element stiffness matrix k of the spring joining nodes i (at the left end) and j (at the right end) into the global stiffness matrix K. It returns the $n \times n$ global stiffness matrix K every time an element is assembled.

SpringElementForces(k, u) – This function calculates the element force vector using the element stiffness matrix k and the element displacement vector u. It returns the 2×1 element force vector f.

The following is a listing of the MATLAB source code for each function:

```
function y = SpringElementStiffness(k)
%SpringElementStiffness    This function returns the element stiffness
%                          matrix for a spring with stiffness k.
%                          The size of the element stiffness matrix
%                          is 2 x 2.
y = [k -k; -k k];
```

```
function y = SpringAssemble(K,k,i,j)
%SpringAssemble    This function assembles the element stiffness
%                  matrix k of the spring with nodes i and j into the
%                  global stiffness matrix K.
%                  This function returns the global stiffness matrix K
%                  after the element stiffness matrix k is assembled.
K(i,i) = K(i,i) + k(1,1);
K(i,j) = K(i,j) + k(1,2);
```

```
K(j,i) = K(j,i) + k(2,1);
K(j,j) = K(j,j) + k(2,2);
y = K;
```

```
function y = SpringElementForces(k,u)
%SpringElementForces   This function returns the element nodal force
%                      vector given the element stiffness matrix k
%                      and the element nodal displacement vector
u.
y = k * u;
```

Example 2.1:

Consider the two-element spring system shown in Fig. 2.2. Given $k_1 = 100\,\mathrm{kN/m}$, $k_2 = 200\,\mathrm{kN/m}$, and $P = 15\,\mathrm{kN}$, determine:

1. the global stiffness matrix for the system.
2. the displacements at nodes 2 and 3.
3. the reaction at node 1.
4. the force in each spring.

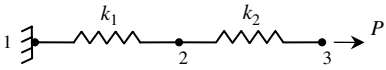

Fig. 2.2. Two-Element Spring System for Example 2.1

Solution:

Use the six steps outlined in Chap. 1 to solve this problem using the spring element.

Step 1 – Discretizing the Domain:

This problem is already discretized. The domain is subdivided into two elements and three nodes. Table 2.1 shows the element connectivity for this example.

Table 2.1. Element Connectivity for Example 2.1

Element Number	Node i	Node j
1	1	2
2	2	3

Step 2 – Writing the Element Stiffness Matrices:

The two element stiffness matrices k_1 and k_2 are obtained by making calls to the MATLAB function *SpringElementStiffness*. Each matrix has size 2×2.

```
» k1=SpringElementStiffness(100)

k1 =

   100  -100
  -100   100


» k2=SpringElementStiffness(200)

k2 =

   200  -200
  -200   200
```

Step 3 – Assembling the Global Stiffness Matrix:

Since the spring system has three nodes, the size of the global stiffness matrix is 3×3. Therefore to obtain K we first set up a zero matrix of size 3×3 then make two calls to the MATLAB function *SpringAssemble* since we have two spring elements in the system. Each call to the function will assemble one element. The following are the MATLAB commands:

```
» K=zeros(3,3)

K =

  0    0    0
  0    0    0
  0    0    0


» K=SpringAssemble(K,k1,1,2)

K =

   100   -100   0
  -100    100   0
     0      0   0
```

```
» K=SpringAssemble(K,k2,2,3)

K =

   100  -100     0
  -100   300  -200
     0  -200   200
```

Step 4 – Applying the Boundary Conditions:

The matrix (2.2) for this system is obtained as follows using the global stiffness matrix obtained in the previous step:

$$\begin{bmatrix} 100 & -100 & 0 \\ -100 & 300 & -200 \\ 0 & -200 & 200 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \tag{2.4}$$

The boundary conditions for this problem are given as:

$$U_1 = 0, F_2 = 0, F_3 = 15\,kN \tag{2.5}$$

Inserting the above conditions into (2.4) we obtain:

$$\begin{bmatrix} 100 & -100 & 0 \\ -100 & 300 & -200 \\ 0 & -200 & 200 \end{bmatrix} \begin{Bmatrix} 0 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ 0 \\ 15 \end{Bmatrix} \tag{2.6}$$

Step 5 – Solving the Equations:

Solving the system of equations in (2.6) will be performed by partitioning (manually) and Gaussian elimination (with MATLAB). First we partition (2.6) by extracting the submatrix in rows 2 and 3 and columns 2 and 3. Therefore we obtain:

$$\begin{bmatrix} 300 & -200 \\ -200 & 200 \end{bmatrix} \begin{Bmatrix} U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 15 \end{Bmatrix} \tag{2.7}$$

The solution of the above system is obtained using MATLAB as follows. Note that the backslash operator "\" is used for Gaussian elimination.

```
» k=K(2:3,2:3)

k =

     300    -200
    -200     200


» f=[0 ; 15]

f =

     0
    15

» u=k\f

u =

    0.1500
    0.2250
```

It is now clear that the displacements at nodes 2 and 3 are 0.15 m and 0.225 m, respectively.

Step 6 – Post-processing:

In this step, we obtain the reaction at node 1 and the force in each spring using MATLAB as follows. First we set up the global nodal displacement vector U, then we calculate the global nodal force vector F.

```
» U=[0 ; u]

U =

         0
    0.1500
    0.2250

» F=K*U

F =

    -15
      0
     15
```

Thus the reaction at node 1 is a force of 15 kN (directed to the left). Finally we set up the element nodal displacement vectors u_1 and u_2, then we calculate the element force vectors f_1 and f_2 by making calls to the MATLAB function S*pringElementForces*.

```
» u1=[0 ; U(2)]

u1 =

           0
      0.1500

» f1=SpringElementForces(k1,u1)

f1 =

      -15
       15

» u2=[U(2) ; U(3)]

u2 =

      0.1500
      0.2250

» f2=SpringElementForces(k2,u2)

f2 =

      -15
       15
```

Thus it is clear that the force in element 1 is 15 kN (tensile) and the force in element 2 is also 15 kN (tensile).

Example 2.2:

Consider the spring system composed of six springs as shown in Fig. 2.3. Given $k = 120\,\mathrm{kN/m}$ and $P = 20\,\mathrm{kN}$, determine:

1. the global stiffness matrix for the system.
2. the displacements at nodes 3, 4, and 5.
3. the reactions at nodes 1 and 2.
4. the force in each spring.

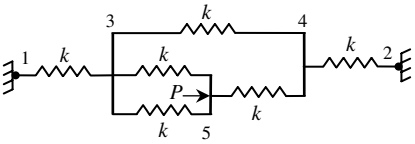

Fig. 2.3. Six-Element Spring System for Example 2.2

Solution:

Use the six steps outlined in Chap. 1 to solve this problem using the spring element.

Step 1 – Discretizing the Domain:

This problem is already discretized. The domain is subdivided into six elements and five nodes. Table 2.2 shows the element connectivity for this example.

Table 2.2. Element Connectivity for Example 2.2

Element Number	Node i	Node j
1	1	3
2	3	4
3	3	5
4	3	5
5	5	4
6	4	2

Step 2 – Writing the Element Stiffness Matrices:

The six element stiffness matrices k_1, k_2, k_3, k_4, k_5, and k_6 are obtained by making calls to the MATLAB function *SpringElementStiffness*. Each matrix has size 2×2.

```
» k1=SpringElementStiffness(120)

k1 =

     120 -120
    -120  120
```

```
» k2=SpringElementStiffness(120)

k2 =


     120  -120
    -120   120

» k3=SpringElementStiffness(120)

k3 =

     120  -120
    -120   120


» k4=SpringElementStiffness(120)

k4 =

     120  -120
    -120   120


» k5=SpringElementStiffness(120)

k5 =

     120  -120
    -120   120


» k6=SpringElementStiffness(120)

k6 =

     120  -120
    -120   120
```

Step 3 – Assembling the Global Stiffness Matrix:

Since the spring system has five nodes, the size of the global stiffness matrix is 5×5. Therefore to obtain K we first set up a zero matrix of size 5×5 then make six calls to the MATLAB function *SpringAssemble* since we have six spring elements in the system. Each call to the function will assemble one element. The following are the MATLAB commands:

```
» K=zeros(5,5)

K =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0


» K=SpringAssemble(K,k1,1,3)

K =

     120     0    -120     0     0
       0     0       0     0     0
    -120     0     120     0     0
       0     0       0     0     0
       0     0       0     0     0


» K=SpringAssemble(K,k2,3,4)

K =

     120     0    -120       0     0
       0     0       0       0     0
    -120     0     240    -120     0
       0     0    -120     120     0
       0     0       0       0     0


» K=SpringAssemble(K,k3,3,5)

K =


     120     0    -120       0       0
       0     0       0       0       0
    -120     0     360    -120    -120
       0     0    -120     120       0
       0     0    -120       0     120
```