University Course

EMA 471
Intermediate Problem Solving for Engineers

University of Wisconsin, Madison

Spring 2016

My Class Notes

Nasser M. Abbasi

Spring 2016

Contents

Chapter 1

Introduction

Took this course in Spring 2016.

Instructor: professor Robert J. Witt

class web site at moodle

1.1 Syllabus

EP 471
Engineering Problem Solving II

Bob Witt
531 ERB; Office Hours:
MW 1 PM – 3 PM or by appointment
263-2760; robert.witt@wisc.edu

Spring 2016
2261 EH
Tu/Th, 8:00 AM, 9:30 AM

Syllabus

Course Description: We will focus on using computers primarily to solve ordinary and partial differential equations that appear in typical engineering problems. Historically this has been done using procedural programming languages (FORTRAN, but also Pascal, C and C++, for instance).We'll use the engineering application Matlab, continuing your experience from CS310/NEEP271. The advantage of Matlab is that many of the capabilities we want are embedded as pre-defined utilities, but this is also a deficiency because the algorithms are hidden from us. We will learn more about both the utilities and the underlying algorithms this semester.

Equations to be solved include systems of ordinary differential equations, both as initial and boundary value problems, "elliptic", "parabolic", and "hyperbolic" partial differential equations, and eigenvalue problems. (We'll discuss formal definitions of "elliptic," "parabolic" and "hyperbolic" partial differential equations later, but for now, "elliptic" equations are those that have to be solved simultaneously everywhere, while "parabolic" and "hyperbolic" equations are propagated or marched from one location to another.)

Outside of differential equations, we'll discuss some additional issues related to numerical integration and also spend some time getting acquainted with "Monte Carlo" methods that can be used to solve engineering problems and model various systems or processes involving parameters described with probability distribution functions.

Course Goals: After completing this course you should be comfortable using Matlab to solve a variety of numerical engineering problems, mostly related to differential equations. You should also have acquired greater confidence with Matlab's syntax and logical structures and improved your ability to write Matlab scripts from scratch.

I would like to offer a couple of suggestions regarding the writing of scripts. **First, ample use of comments is recommended, especially as your scripts become more complicated.** You may work on a script in some capacity, then put it away for several months, then come back to it. Detailed comments within the script can help you remember what you did and why you did it. Also, it is increasingly the case that a large program or script is revised and updated by a team of people. In this case, your detailed comments help someone else to understand what you did and why, and such comments provided by others will help you understand what they did and why when you have to use their work as a basis for continued updates.

Second, use long, descriptive strings for defining variables, scripts and functions. This will again serve as a reminder of what the purpose of the entity is so you don't have

;

to guess later about what you intended. In addition, it prevents you from defining two different entities with the same symbol. If you have a matrix A in your script, and the problem also includes a cross-sectional area that you have labeled A, then you are using the same symbol for a matrix and a scalar. As you can imagine, this will cause your Matlab script to do something you don't intend. Also, keep in mind that Matlab, and many other programming languages, have their own internal functions. Don't define a variable as sin or cos, since Matlab already defines that as an intrinsic function. It is really embarrassing to spend hours stewing about a non-functioning Matlab script, only to find that the problem is a variable or function name that conflicts with one of Matlab's intrinsic functions.

Prior to most class periods, I will post an Exercise that provides some background about a method or Matlab utility. **Please read the exercise notes, then download and examine any accompanying scripts.** There will usually be one or more exercise examples for you to try after reading the notes and examining the scripts. **Please work through the exercise examples before trying the homework.** If you bypass the examples and try to go straight to the homework, you'll usually struggle for a greater length of time than if you had tried the examples first.

Administrative Issues: Course grades will be determined by submitted homework and a project. There will be no quizzes or exams in this class. Seven homework assignments, with due dates spaced more or less equally at two week intervals throughout the semester, will constitute 75% of your grade. The remaining 25% will be determined by a course project. This syllabus, as well as posted homework assignments, sample Matlab scripts, and other materials, will be posted on my Moodle web page.

The physical layout of the classroom obviously requires students to pair up at workstations. The alternative is that you bring your own laptop to class and work by yourself. **If possible, you should find a partner you're comfortable with and work with that person throughout the entire semester. Also, you are permitted to work/submit homework as teams of two. (Teams of three or more are not permitted.) It is not required, however, that you partner with someone. If you're the kind of person who likes to do everything yourself, then you are not obligated to work as a team.**

On each due date, you (or your team) will hand me a paper report *and* electronically submit the appropriate Matlab files so I can evaluate your work and compare your results to what's in your report. The paper report need not be long, but it should summarize your results. A preferable format is to print out any relevant plots and write a description of the problem and results, focusing on the plot(s). You **don't need to print out your scripts** and attach them to your report. Your .m files will be deposited using the Assignment feature in Moodle. For now, I'd like an email from **each of you** stating one of the following:
(1) I'm working with a partner and his/her name is <name here>, or…
(2) I'm working by myself… or
(3) I would like to partner with someone but don't know anyone in class.

2

;

If the last option, it would be helpful to include something about your background (undergrad or grad, major) to assist me in finding a suitable partner.

The schedule we'll follow throughout the semester is shown below. Note that there are numbers in brackets [1] and parentheses (1) next to several days throughout the semester. **A number appearing in brackets [1] indicates the day when HW will be assigned. A number appearing in parenthesis (1) indicates when that HW will be due**. As you can see, the first homework assignment will be distributed Tuesday, 1/28, and is due on Thursday, 2/6. We will maintain a two-week interval throughout the semester for distributing and receiving homework.

The schedule is:

Week 1: Introductions, re-acquaintance with Matlab

1. Tu, 1/19	Introductions, ODEs – IVPs
2. Th, 1/21	ODEs – IVPs

Week 2: First order (and Systems of First Order) ODEs: Initial Value Problems (IVPs)

3. Tu, 1/26 [1]	ODEs – IVPs
4. Th, 1/28	ODEs – IVPs

Week 3: Second order ODEs: Boundary Value Problems (BVPs)

5. Tu, 2/2	ODEs – BVPs
6. Th, 2/4 (1)	ODEs – BVPs

Week 4: Second order ODEs: Boundary Value Problems

7. Tu, 2/9 [2]	ODEs – BVPs
8. Th, 2/11	ODEs – BVPs

Week 5: ODEs: Eigenvalue Problems (EVPs)

9. Tu, 2/16	ODEs – EVPs
10. Th, 2/18 (2)	ODEs – EVPs

Week 6: ODEs: Eigenvalue Problems (EVPs)

11. Tu, 2/23 [3]	ODEs – EVPs
12. Th, 2/25	ODEs – EVPs

;

Week 7: Monte Carlo Methods (MCMs)

13. Tu, 3/1 MCMs
14. Th, 3/3 (3) MCMs

Week 8: Monte Carlo Methods/Numerical Integration (NI)

15. Tu, 3/8 [4] MCMs
16. Th, 3/10 NI

Week 9: Numerical Integration (NI)

17. Tu, 3/15 NI
18. Th, 3/17 (4) NI

No Class: Spring Break (3/19 – 3/27)

Week 10: Elliptic Partial Differential Equations (PDEs)

19. Tu, 3/29 [5] Elliptic PDEs
20. Th, 3/3 Elliptic PDEs

Week 11: Elliptic PDEs

21. Tu, 4/5 Elliptic PDEs
22. Th, 4/7 (5) Elliptic PDEs

Week 12: Parabolic PDEs

23. Tu, 4/12 [6] Parabolic PDEs
24. Th, 4/14 Parabolic PDEs

Week 13: Parabolic PDEs

25. Tu, 4/19 Parabolic PDEs
26. Th, 4/21 (6) Hyperbolic PDEs

Week 14: Hyperbolic PDEs

27. Tu, 4/26 [7] Hyperbolic PDEs
28. Th, 4/28 Hyperbolic PDEs

Week 15: Time to Work on Last HW, Projects

29. Tu, 5/3
30. Th, 5/5 (7)

We have final exam time slots assigned for Monday, May 9th, and Thursday, May 12th, but there will be no final exams. There will be a course project, and I'd like you to submit your project materials (paper report plus uploaded relevant scripts) by the evening of Thursday, May 12th.

5

;

Chapter 2

HWs

Local contents

2.1 HW 1

2.1.1 Problem 1

PROBLEM DESCRIPTION

(1) (12 pts) Consider the first order IVP

$$\frac{dy}{dt} = t^2 - t, \qquad y(0) = 1$$

This is a relatively straightforward ODE that has an analytical solution. Find the analytical solution first to serve as a comparison with your numerical solutions. Using a step size $h = 0.1$, advance this solution to $t = 4$ using the Euler, Modified Euler and 2nd order Runge-Kutta methods we discussed in Exercise 1. Compare your numerical results to the analytical solution.

SOLUTION

$$\begin{aligned} y'(t) &= f(t) \\ &= t^2 - t \end{aligned} \tag{1}$$

This ODE is separable. Integrating both sides gives

$$y = \frac{t^3}{3} - \frac{t^2}{2} + C$$

Where C is the constant of integration which is found from the initial conditions. At $t = 0$, we are given $y(0) = 1$. This results in $C = 1$. The solution becomes

$$\boxed{y = 1 + \frac{t^3}{3} - \frac{t^2}{2}}$$

Matlab program was written to implements Euler, modified Euler and Runge-Kutta using time spacing of 0.1 and was run to 4 seconds.

2.1.1.1 Euler method

$$y_{n+1} = y_n + hy'_n + O\left(h^2\right)$$

In the above, y_0 was taken from given initial conditions and $y'_n = f_n$ is the RHS of (1) evaluated at each time step. h is the time step used (which is 0.1 seconds in this problem). Hence $t_n = nh$. Euler method has local error (per step) $O\left(h^2\right)$ and an overall global error $O(h)$.

2.1.1.2 Modified Euler method

$$y_{n+1} = y_n + h\left(\frac{y'_n + y'_{n+1}}{2}\right) + O\left(h^3\right)$$

Modified Euler method has local error (per step) $O\left(h^3\right)$ and overall global error $O\left(h^2\right)$, therefore it is more accurate than the standard Euler method. In this problem, the RHS $y'_n = f_n(t)$ only depends on t and not on y_n.

2.1.1.3 Second order Runge-Kutta

This method has local error $O\left(h^3\right)$ and global error $O\left(h^2\right)$

$$\begin{aligned} t_n &= nh \\ k_1 &= hf\left(t_n, y_n\right) \\ k_2 &= hf\left(t_n + \alpha h, y_n + \beta k_1\right) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

In this problem y do not appear in the RHS (hence β is not used). The above reduces to

$$\begin{aligned} k_1 &= hf(t_n) \\ k_2 &= hf(t_n + \alpha h) \\ y_{n+1} &= y_n + ak_1 + bk_2 \end{aligned}$$

Using $a = \frac{2}{3}, b = \frac{1}{3}, \alpha = \frac{3}{2}, \beta = \frac{3}{2}$, (as was done in exercise one handout) the above now becomes

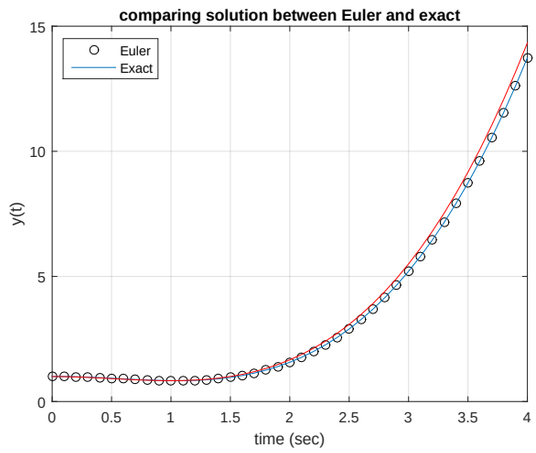
$$\begin{aligned} k_1 &= hf(nh) \\ k_2 &= hf\left(nh + \frac{3}{2}h\right) = hf\left(\left(n + \frac{3}{2}\right)h\right) \\ y_{n+1} &= y_n + \frac{2}{3}hf(nh) + \frac{1}{3}hf\left(\left(n + \frac{3}{2}\right)h\right) \\ &= y_n + \frac{2}{3}h\left[f(nh) + \frac{1}{2}f\left(\left(n + \frac{3}{2}\right)h\right)\right] \end{aligned}$$

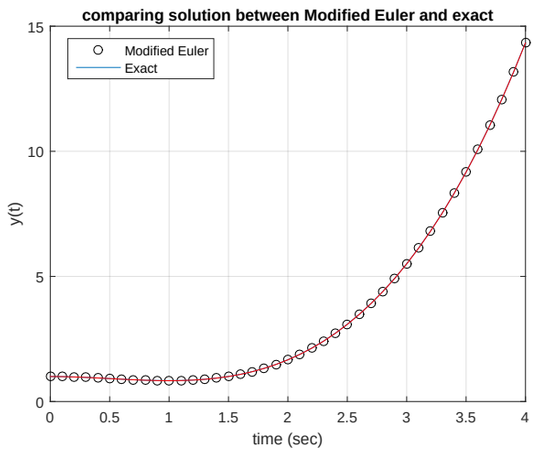
The following is a summary table of the three methods.

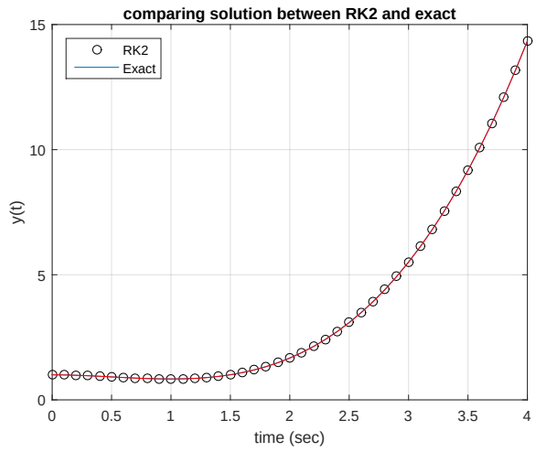
scheme name	main computation	global error
Euler	$y_{n+1} = y_n + hy'_n$	$O(h)$
Modified Euler	$y_{n+1} = y_n + \frac{h}{2}\left(y'_n + y'_{n+1}\right)$	$O(h^2)$
RK2	$y_{n+1} = y_n + \frac{2}{3}h\left(y'_n + \frac{1}{2}y'_{n+\frac{3}{2}}\right)$	$O(h^2)$

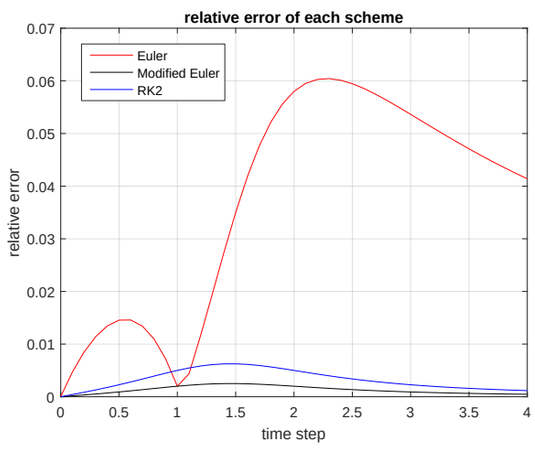
2.1.1.4 Results

Four plots were generated. three plots to show the solution by each scheme next to the exact solution. The fourth plot shows the relative error of each scheme. The relative error was found by calculating $\frac{|\text{exact solution-numerical}|}{|\text{exact}|}$ at each time instance.









2.1.1.5 Discussion of results

Euler method was least accurate as expected since it has global error $O(h)$. There is no large difference between RK2 and modified Euler since both have global error $O(h^2)$.

2.1.1.6 Source code listing

```
function   nma_EMA_471_HW1_part_a
%This function solves HW1, part(a). EMA 471, spring 2016, UW Madison
%Please see report for detailed information about the problem
%by Nasser M. Abbasi

clear; close all;

%First call is to initialize data before each scheme run is made
[h,t,yExact,y] = initializeData();
yEuler         = Euler(h,t,y,yExact);

[h,t,yExact,y] = initializeData();
yModifiedEuler = modifiedEuler(h,t,y,yExact);

[h,t,yExact,y] = initializeData();
yRK2           = RK2(h,t,y,yExact);

%Now we Plot the relative error for each scheme, using returned results
figure();
plot(t,abs(yEuler-yExact)./abs(yExact),'r', ...
     t,abs(yModifiedEuler-yExact)./abs(yExact),'k',...
     t,abs(yRK2-yExact)./abs(yExact),'b');

title('relative error of each scheme');
xlabel('time (sec)'); ylabel('relative error');
legend('Euler','Modified Euler','RK2','Location','NorthWest');
grid;
end

%---------------------------------------
%Implements basic Euler method
function y = Euler(h,t,y,yExact)
%y(1) has been initialized to correct value by caller
%
%  o---------o----------o----.....
%  t=0      t=h         t=2h
%  y(1)     y(2)        y(3)
%
%This function starts computing y(2),y(3),....

currentTime = 0;
for n = 2:length(y)
    f           = rhs(currentTime); %RHS of the ODE
    y(n)        = y(n-1)+h*f;       %Euler scheme
    currentTime = currentTime + h;
end

makePlots(t,y,yExact,'Euler');
end

%---------------------------------------
%Implements Modified Euler solver
function y  = modifiedEuler(h,t,y,yExact)
currentTime = 0;

%y(1) has been initialized to correct value by caller
for n = 2:length(y)
```

```
    fn1         = rhs(currentTime);
    fn2         = rhs(currentTime+h);
    f           = (fn1+fn2)/2;
    y(n)        = y(n-1)+h*f;
    currentTime = currentTime + h;
end

makePlots(t,y,yExact,'Modified Euler');
end

%---------------------------------------
%Implements RK2 solver
function y  = RK2(h,t,y,yExact)
currentTime = 0;
a           = 2/3;
b           = 1/3;
alpha       = 3/2;   %RK2 parameters used
%beta       = 3/2;   %not needed, since RHS f(.) do not depend on y.

%y(1) has been initialized to correct value by caller
for n = 2:length(y)
    k1          = h*rhs(currentTime);
    k2          = h*rhs(currentTime+alpha*h);
    y(n)        = y(n-1)+a*k1+b*k2;
    currentTime = currentTime + h;
end

makePlots(t,y,yExact,'RK2');
end

%-----------------------------------
%Called to initialize data and counters before each solver is called
function [h,t,yExact,y] = initializeData()

h      = 0.1;      %time step we are asked to use
t      = (0:h:4)';
y      = zeros(length(t),1);
y(1)   = 1;        %initial conditions, from problem statement
yExact = 1+t.^3/3-t.^2/2;
end

%-------------------------
%Called to generate plots from each solver
function makePlots(t,y,yExact,schemeName)

figure();
plot(t,y,'ok',t,y); hold on;
plot(t,yExact,'r');
legend(schemeName,'Exact','Location','NorthWest');
title(sprintf('comparing solution between %s and exact',schemeName));
xlabel('time (sec)'); ylabel('y(t)');
grid;

end

%---------------
%RHS function, called by each scheme solver. Notice only t appears in RHS
function v = rhs(t)
  v = t^2-t;
end
```

2.1.2 Problem 2

PROBLEM DESCRIPTION

(2) (12 pts) Consider the second order system of equations:

$$\frac{dx}{dt} = xy + t, \qquad x(0) = 1$$
$$\frac{dy}{dt} = ty + x, \qquad y(0) = -1$$

As this is a non-linear system of equations, it's unlikely that you will be able to find an analytical solution. You can, however, construct a Taylor series expansion in the neighborhood of $t = 0$ that will be a reasonable approximation of the solution as long we include enough terms in the expansion and don't wander too far away from $t = 0$. Construct Taylor series expansions out to terms t^5 for $x(t)$ and $y(t)$ and plot these approximate solutions over the interval $0 \le t \le 0.5$. Then use `ode45` to solve the system over the interval $0 \le t \le 2$. Plot your numerical solutions on the same plot as the Taylor series expansions and discuss your results.

SOLUTION

$$x'(t) = xy + t$$
$$y'(t) = ty + x$$

With $x(0) = 1, y(0) = -1$.

The following calculation shows the evaluation of all the derivatives needed for use with Taylor expansion. The expansion is around $t = 0$. This table gives the result.

derivative
$x'(t) = xy + t$
$y'(t) = ty + x$
$x''(t) = x'y + xy' + 1$
$y''(t) = y + ty' + x'$
$x'''(t) = x''y + x'y' + x'y' + xy''$
$y'''(t) = y' + y' + ty'' + x''$
$x^{(4)}(t) = x'''y + x''y' + x''y' + x'y'' + x''y' + x'y'' + x'y'' + xy'''$
$y^{(4)}(t) = y'' + y'' + y'' + ty''' + x'''$
$x^{(5)}(t) = x^{(4)}y + x'''y' + x'''y' + x''y'' + x'''y' + x''y'' + x''y'' + x'y''' + x'''y' + x''y'' + x''y'' + x'y''' + x''y'' + x'y''' + x'y''' + x'y^{(4)}$
$y^{(5)}(t) = y''' + y''' + y''' + y''' + ty^{(4)} + x^{(4)}$

The numerical value of the above derivatives at $t = 0$ is now calculated and given in another table below

	derivative at $t = 0$	value
$x'(0)$	$(1)(-1) + 0$	-1
$y'(0)$	$(0)(-1) + 1$	1
$x''(0)$	$(-1)(-1) + (1)(1) + 1 = 1 + 1 + 1$	3
$y''(0)$	$(-1) + 0(1) - 1$	-2
$x'''(0)$	$(3)(-1) + (-1)(1) + (-1)(1) + (1)(-2)$	-7
$y'''(0)$	$1 + 1 + 0 + 2$	4
$x^{(4)}(0)$	$(-7)(-1) + 2 + 2 + (-1)(-2) + 2 + (-1)(-2) + (-1)(-2) + 4$	23
$y^{(4)}(0)$	$(-2) + (-2) + (-2) + 0 + (-7)$	-13
$x^{(5)}(0)$	$(-13)(-1) + 4 + 4 + (2)(-2) + (-7) + (2)(-2) + (2)(-2) + (-1)(4) + (-7) + (2)(-2) + (2)(-2) + 4 + (2)(-2) + 4 + 4 - 13$	-22
$y^{(5)}(0)$	$(4) + (4) + (4) + (4) + 0 + 23$	39

The Taylor expansion for $x(t), y(t)$ is

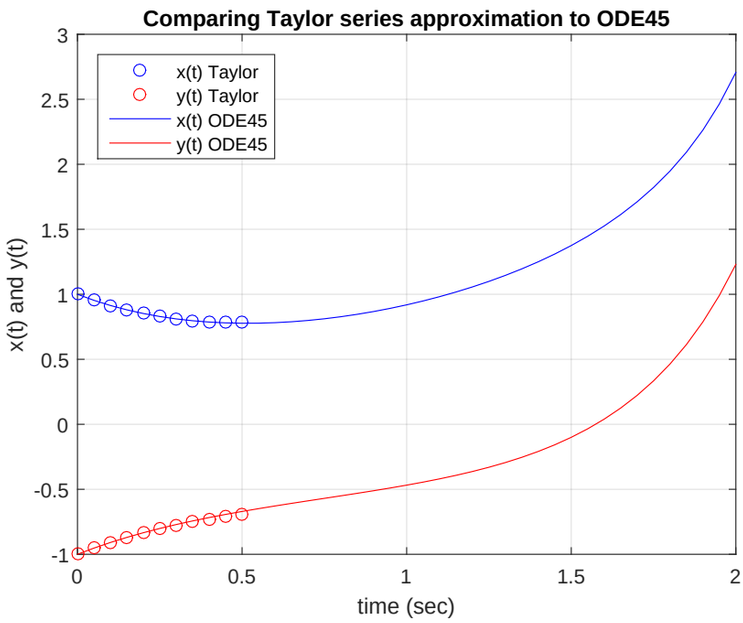
$$x(t) = x(0) + tx'(0) + \frac{t^2}{2}x''(0) + \frac{t^3}{3!}x'''(0) + \frac{t^4}{4!}x^{(4)} + \frac{t^5}{5!}x^{(5)} + O\left(t^6\right)$$
$$y(t) = y(0) + ty'(0) + \frac{t^2}{2}y''(0) + \frac{t^3}{3!}y'''(0) + \frac{t^4}{4!}y^{(4)} + \frac{t^5}{5!}y^{(5)} + O\left(t^6\right)$$

Applying values from the table above gives

$$\begin{aligned} x(t) &= 1 - t + \frac{3}{2}t^2 - 7\frac{t^3}{3!} + 23\frac{t^4}{4!} - 22\frac{t^5}{5!} + O\left(t^6\right) \\ &= 1 - t + \frac{3}{2}t^2 - \frac{7}{6}t^3 + \frac{23}{24}t^4 - \frac{22}{120}t^5 + O\left(t^6\right) \\ y(t) &= -1 + t - t^2 + 4\frac{t^3}{3!} - 13\frac{t^4}{4!} + 39\frac{t^5}{5!} + O\left(t^6\right) \\ &= -1 + t - t^2 + \frac{2}{3}t^3 - \frac{13}{24}t^4 + \frac{39}{120}t^5 + O\left(t^6\right) \end{aligned}$$

2.1.2.1 Results

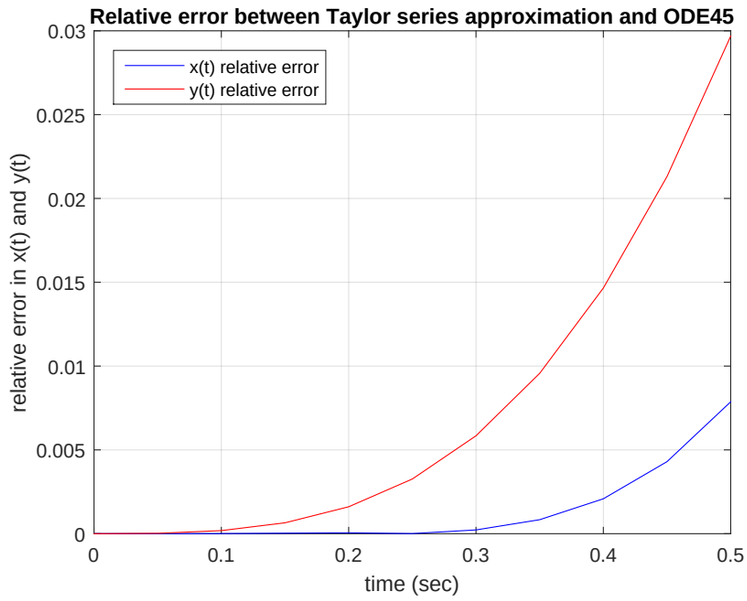
The following diagram shows the solution using Taylor series approximation and using ODE45



We see from the above that Taylor series approximation was good for only small distance from the expansion point, $t = 0$. The $x(t)$ solution using Taylor became worst faster than the $y(t)$ solution did. The relative error was computed and plotted for up to $t = 0.5$ to better compare the results.

The relative error for both $x(t)$ and $y(t)$ was computed using

$$\frac{\left|\text{ODE45 solution-Taylor solution}\right|}{\left|\text{ODE45 solution}\right|}$$



The above plot shows that the error compared to ODE45 increased as the distance from the expansion point becomes larger with $y(t)$ solution showing a worst approximation than $y(t)$.

2.1.2.2 Source code listing

```
function   nma_EMA_471_HW1_part_b
%This function solves HW1, problem 2. EMA 471, spring 2016,
%UW Madison see report for detailed information about the problem
%by Nasser M. Abbasi

close all;

h  = 0.05;    %step size
t1 = 0:h:0.5; %time for the Taylor series
t2 = 0:h:2;   %for ODE45 we are asked to go for 2 seconds.

[xApprox,yApprox] = doTaylorApproximation(t1);
[xODE45,yODE45]   = doODE45(t2);

%done, now plot and compare.
compare(t1,xApprox,yApprox,t2,xODE45,yODE45);
end

%---------------------------------
%Do the Taylor series approximation
function [xApprox,yApprox] = doTaylorApproximation(t)
xApproxF = @(t) 1 - t + (3/2)* t.^2 - 7/6 * t.^3  + 23/24 * ...
                                          t.^4 -22/120 * t.^5;
yApproxF = @(t)-1 + t - t.^2 + 2/3 * t.^3  - 13/24 *...
                                          t.^4 +39/120 * t.^5;

xApprox  = xApproxF(t);
yApprox  = yApproxF(t);
end

%-------------------------------------------------
%Do ODE45. This uses Matlab's ode45 to solve the problem.
function [xODE45,yODE45] = doODE45(t)
xInitial = 1;
yInitial = -1;

[~,v]  = ode45( @rhs, t, [xInitial yInitial] );
%extract solutions
```

```
xODE45 = v(:,1);
yODE45 = v(:,2);

%internal function, for RHS
    function v = rhs(t,X)
        x = X(1);
        y = X(2);
        v = [x*y+t;t*y+x];
    end
end

%-------------------------------------------------------
%function to compare Taylor series with ODE 45 output
function compare(t1,xApprox,yApprox,t2,xODE45,yODE45)

figure();
plot(t1,xApprox,'bo',t1,yApprox,'ro'); hold on;
plot(t2,xODE45,'b',t2,yODE45,'r');

legend('x(t) Taylor','y(t) Taylor','x(t) ODE45','y(t) ODE45',...
        'Location','NorthWest');

title('Comparing Taylor series approximation to ODE45');
xlabel('time (sec)'); ylabel('x(t) and y(t)');
grid;

figure();
plot(t1,abs(xODE45(1:length(t1))-xApprox')./...
                                abs(xODE45(1:length(t1))),'b',...
     t1,abs(yODE45(1:length(t1))-yApprox')./...
                                  abs(yODE45(1:length(t1))),'r');
title('Relative error between Taylor series approximation and ODE45');
xlabel('time (sec)');
ylabel('relative error in x(t) and y(t)');
legend('x(t) relative error','y(t) relative error',...
        'Location','NorthWest');
grid;
end
```

2.1.3 Problem 3

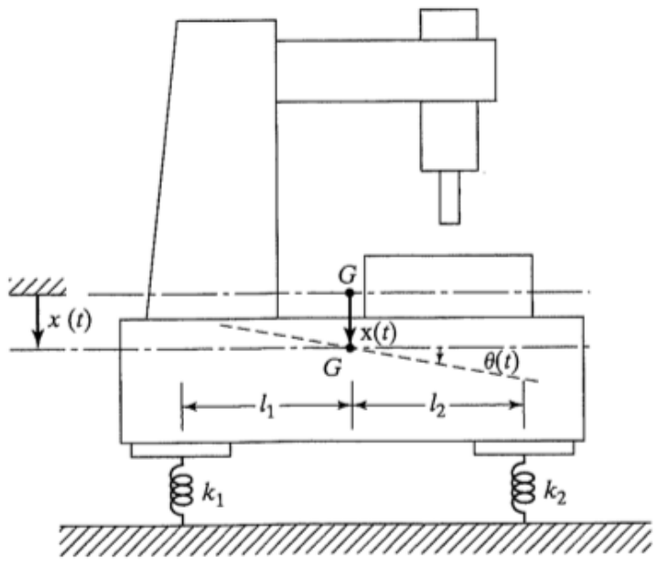
PROBLEM DESCRIPTION

(3) (16 pts) Note: This is a problem you might encounter in EMA 545. A machine tool is mounted on two nonlinear elastic mounts, as shown in the figure at the top of the next page. The equations of motion, in terms of the coordinates $x(t)$ and $\theta(t)$, are given by:

$$m\ddot{x} + k_{11}(x - l_1\theta) + k_{12}(x - l_1\theta)^3 + k_{21}(x + l_2\theta) + k_{22}(x + l_2\theta)^3 = 0$$
$$J_o\ddot{\theta} - k_{11}(x - l_1\theta)l_1 - k_{12}(x - l_1\theta)^3 l_1 + k_{21}(x + l_2\theta)l_2 + k_{22}(x + l_2\theta)^3 l_2 = 0$$

Here m is the mass and J_o is the mass moment of inertia about G of the machine tool. Using `ode45`, find $x(t)$ and $\theta(t)$ over the interval $0 \le t \le 10$ s using the initial conditions:

$$x(0) = 0, \quad \dot{x}(0) = 10 \text{ mm/s}, \quad \theta(0) = 0, \quad \dot{\theta}(0) = 10 \text{ mrad/s}$$



Parameters in the governing equations have values: m = 1000 kg, J_o = 2500 kg-m^2, l_1 = 1 m, l_2 = 1.5 m, k_{11} = 40 kN/m, k_{12} = 10 kN/m^3, k_{21} = 50 kN/m, k_{22} = 5 kN/m^3.

NOTE: The governing equations are equations of motion based on $\Sigma F = ma_G$, $\Sigma M_G = I\alpha$, so ***units are important***. The values for the parameters given are in SI units, so be sure everything you input into your script is consistent with that system of units.

Plot x and θ over the interval with x in [mm] and θ in [mrad]. What are the peak amplitudes of x and θ over the interval?

SOLUTION

$$mx'' + k_{11}(x - l_1\theta) + k_{12}(x - l_1\theta)^3 + k_{21}(x + l_2\theta) + k_{22}(x + l_2\theta)^3 = 0 \tag{1}$$
$$J_0\theta'' - k_{11}(x - l_1\theta)l_1 - k_{12}(x - l_1\theta)^3 l_1 + k_{21}(x + l_2\theta)l_2 + k_{22}(x + l_2\theta)^3 l_2 = 0$$

The first step is to convert the two second order differential equations to a set of four first order differential equations, since ODE numerical solvers work with first order ode's. We need to select the states to use. Using

$$x_1 = x$$
$$x_2 = x'$$
$$x_3 = \theta$$
$$x_4 = \theta'$$

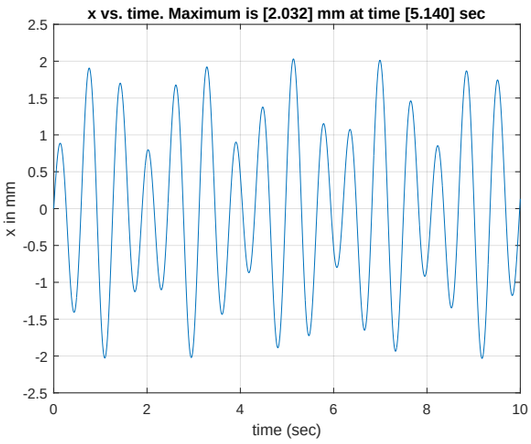
After taking time derivatives, the above becomes

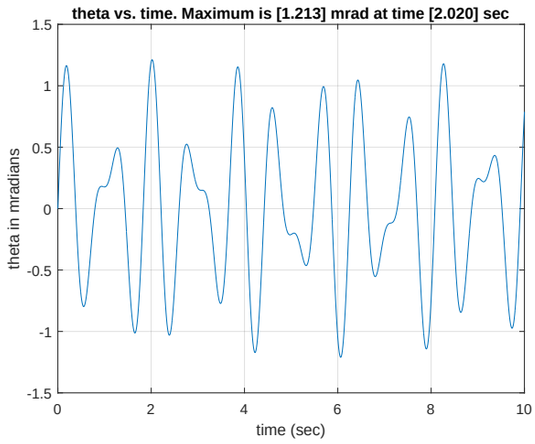
$$\dot{x}_1 = x' = x_2$$
$$\dot{x}_2 = x'' = -\frac{1}{m}\left[k_{11}(x_1 - l_1x_3) + k_{12}(x_1 - l_1x_3)^3 + k_{21}(x_1 + l_2x_3) + k_{22}(x_1 + l_2x_3)^3\right]$$
$$\dot{x}_3 = \theta' = x_4$$
$$\dot{x}_4 = \theta'' = -\frac{1}{J_0}\left[-k_{11}(x_1 - l_1x_3)l_1 - k_{12}(x_1 - l_1x_3)^3 l_1 + k_{21}(x_1 + l_2x_3)l_2 + k_{22}(x_1 + l_2x_3)^3 l_2\right]$$

The above vector $\dot{X}$ in the LHS, is what the Matlab ode45 function will return when called. The following shows the Matlab implementation and the plots generated. It was found that maximum displacement is $\boxed{x_{\max} = 2.032 \text{ mm}}$ and occurred at t = 5.14 seconds. For the angle, the maximum angular displacement was $\boxed{\theta_{\max} = 1.213 \text{ mrad}}$ (or 0.069 degree) and occurred at t = 2.02 seconds.

2.1.3.1 Results

The following are the x and θ solutions plots for t = 10 seconds. The Matlab source code used is given in the next section.





2.1.3.2 Source code listing

```
function   nma_EMA_471_HW1_part_c
%This function solves HW1, part c. EMA 471, spring 2016,
%UW Madison report contains detailed information about the problem
%by Nasser M. Abbasi

close all;

%physical paramters of problem
m    = 1000;    %kg
J0   = 2500;    %kg-m^3
L1   = 1;       %meter
L2   = 1.5;     %meter
k11  = 40*10^3; %N/m
k12  = 10*10^3; %N/m^3
k21  = 50*10^3; %N/m
k22  = 5*10^3;  %N/m

%initialization parameters and initial conditions
h             = 0.01;       %step size. ODE45 did not work
                            %well with large step
t             = 0:h:10;     %time for ODE45
xInitial      = 0;          %meter
xVInitial     = 10*10^(-3); %10 mm/sec
thetaInitial  = 0;          %radians
thetaVInitial = 10*10^(-3); %mrad/sec

%call ODE45 to numerically solve the equations of motions
[t,sol] = ode45( @rhs, t, ...
              [xInitial xVInitial thetaInitial thetaVInitial]);

%Extract the first and third columns. These represet x
%and theta solutions
x          = sol(:,1);
[value,I]  = max(x);

%make x(t) vs. time plot
figure();
plot(t,x*1000); grid;
title(sprintf(...
      'x vs. time. Maximum is [%3.3f] mm at time [%3.3f] sec',...
      value*1000,t(I)));
xlabel('time (sec)'); ylabel('x in mm');

theta     = sol(:,3);
[value,I] = max(theta);

%make theta(t) vs. time plot
```

```
figure();
plot(t,theta*1000); grid;
title(sprintf(...
 'theta vs. time. Maximum is [%3.3f] mrad at time [%3.3f] sec',...
 value*1000,t(I)));
xlabel('time (sec)'); ylabel('theta in mradians');

%internal function, for RHS. Since this is internal, it will
%see all the  physical parameters defined in its parent function,
%hence no need to pass them or make them global

    function v = rhs(~,x)
        f1 = x(2);
        f2 = (-k11*(x(1)-L1*x(3))- k12*(x(1)-L1*x(3))^3 - ...
                    k21*(x(1)+L2*x(3)) - k22*(x(1)+L2*x(3))^3)/m;
        f3 = x(4);
        f4 =(k11*(x(1)-L1*x(3))*L1 + k12*(x(1)-L1*x(3))^3*L1 - ...
             k21*(x(1)+L2*x(3))*L2 - k22*(x(1)+L2*x(3))^3*L2)/J0;

        v = [f1;f2;f3;f4];
    end
end
```

2.2 HW 2

2.2.1 Problem 1

PROBLEM DESCRIPTION

(1) (10 pts) Solve the non-linear boundary value problem:

$$\ddot{y} + 10\ddot{y} - 5y^3 + ty = t^2$$

over the interval, $0 \le t \le 2$, subject to: $y(0) = 0$, $y(2) = -1$, $\ddot{y}(2) = 0$. Use the `bvp4c` utility.

SOLUTION

The first step is to convert the system to state space.

$$y''' + 10y'' - 5y^3 + ty = t^2$$

Let $x_1 = y, x_2 = y', x_3 = y''$, therefore

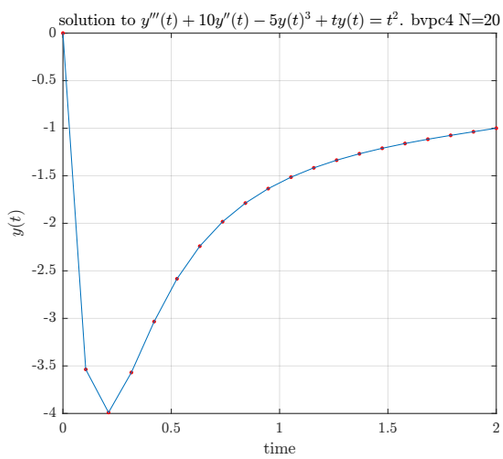
$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= x_3 \\
\dot{x}_3 &= -10y'' + 5y^3 - ty + t^2 \\
&= -10x_3 + 5x_1^3 - tx_1 + t^2
\end{aligned}$$

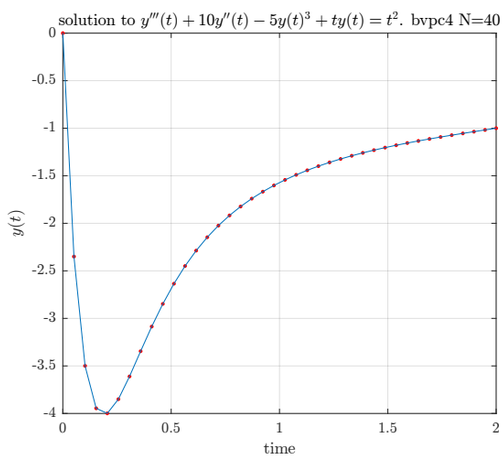
The above $\dot{x}$ vector is what returned back in the RHS call used by bvp4c. The following shows the solution obtained and the code used. One difficulty with this problem was to guess the correct initial solution to use. If the wrong guess was used, then Matlab gives an error

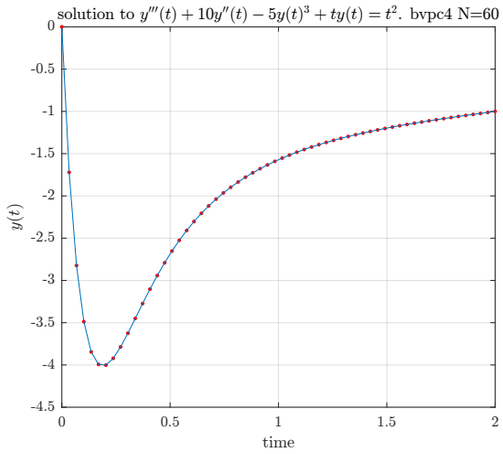
```
Unable to solve the collocation equations -- a singular Jacobian encountered.
```

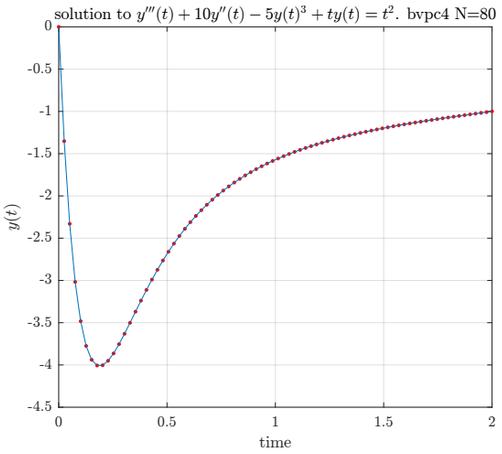
2.2.1.1 Output

Four plots were generated, with different number of grid points, using $N = 20, 40, 60, 80$ grid point, to see how the solution improves with more grid points added. At $N = 80$, the solution was smooth. Here are the results









2.2.1.2 Source code

```
function nma_EMA471_HW2_prob_1
% Solves y'''+10 y''-5 y^3 + t y= t^2
% over 0<=t<=2, with y(0)=0,y(2)=-1,y''(2)=0 using bvp4c
%
% see HW2, EMA 471
% by Nasser M. Abbasi
%
clc; close all;

reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

%solve on different grids  20,40,60,80 points
for i=20:20:80
    make_test(i);
end

end

function make_test(N)
%N is the number of grid points.
%solves the problem using bvp4c

%Important, must use the following guess initial solution [-1 0 0]
%else this error
% Unable to solve the collocation equations -- a singular
%Jacobian encountered. will be generated (Matlab 2015a)

x_bvp4c  = linspace(0,2,N);
solinit1 = bvpinit(x_bvp4c,[-1 0 0]);
sol      = bvp4c(@rhs,@bc,solinit1);

 %evaluate at our grid point, to compare with FDM
y_bvp4c = deval(sol,x_bvp4c);
y_bvp4c = y_bvp4c(1,:)';

figure();
plot(x_bvp4c,y_bvp4c,'r.',x_bvp4c,y_bvp4c);
xlabel('time'); ylabel('$y(t)$');
title(sprintf( ...
 'solution to $y''''''(t)+10 y''''(t)-5 y(t)^3 + t y(t)= t^2$. bvpc4 N=%d',N));

grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
```

```
47
48 end
49 %---------------------------------------------------------------------------%
50 function f = rhs(t,x)
51 %This function sets up the RHS of the state space setup
52 %for this problem. similar to ode45 RHS
53
54 x1 = x(2);
55 x2 = x(3);
56 x3 = -10*x(3)+5*x(1)^3-t*x(1)+t^2;
57 f = [ x1
58       x2
59       x3 ];
60 end
61 %---------------------------------------------------------------------------%
62 function res = bc(ya,yb)
63 %This sets up the boundary conditions vector
64 res = [ ya(1)
65         yb(1)+1
66         yb(3)
67       ];
68 end
```

2.2.2 Problem 2

PROBLEM DESCRIPTION

(2) (15 pts) Consider the linear equation:

$$y'''' - y' = e^x$$

Solve this over the interval, $0 \le x \le 1$, subject to: $y(0) = 0, \quad y'(0) = -1, \quad y(1) = 1, \quad y'(1) = 0$. As this is a linear equation, you can employ a finite-difference approximation to compare to the analytical solution as well as the solution generated by `bvp4c`.

SOLUTION

The first step is to convert the system to state space. (I used t below as the independent variable, instead of x as given in the problem statement, to reduce confusion with the x_i used for state space setup).

$$y^{(4)} - y' = e^t$$

Let $x_1 = y, x_2 = y', x_3 = y'', x_4 = y'''$, therefore

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= x_3 \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= y' + e^t \\
&= x_1 + e^t
\end{aligned}$$

The above $\dot{x}$ vector is what returned back in the RHS call used by bvp4c. Now we will solve it analytically in order to compare the solutions. The homogenous ODE is $y'''' - y' = 0$, hence the characteristic equation is $\lambda^4 - \lambda = 0$, which has solutions

$$\begin{aligned}
\lambda &= 0 \\
\lambda &= 1^{\frac{1}{3}} \\
&= \left\{1, -\frac{1}{2} - \frac{\sqrt{3}}{2}i, -\frac{1}{2} + \frac{\sqrt{3}}{2}i\right\}
\end{aligned}$$

Therefore the homogenous solution is

$$\begin{aligned} y_h &= Ae^{\lambda_1} + Be^{\lambda_2} + Ce^{\lambda_3} + De^{\lambda_4} \\ &= A + Be^t + Ce^{\left(-\frac{1}{2}-\frac{\sqrt{3}}{2}i\right)t} + De^{\left(-\frac{1}{2}+\frac{\sqrt{3}}{2}i\right)t} \end{aligned}$$

Since the homogenous solution contains e^t solution, and the forcing function is also e^t, we can't guess e^t as particular solution. We try $y_p = cte^t$. Hence

$$\begin{aligned} y_p' &= \left(cte^t + ce^t\right) \\ y_p'' &= \left(cte^t + ce^t\right) + ce^t \\ y_p''' &= \left(\left(cte^t + ce^t\right) + ce^t\right) + ce^t \\ y_p'''' &= \left(\left(\left(cte^t + ce^t\right) + ce^t\right) + ce^t\right) + ce^t \end{aligned}$$

Substituting this in the original ODE we obtain

$$\begin{aligned} \left(\left(\left(cte^t + ce^t\right) + ce^t\right) + ce^t\right) + ce^t - \left(cte^t + ce^t\right) &= e^t \\ 3ce^t &= e^t \end{aligned}$$

Hence $3c = 1$ or $c = \frac{1}{3}$, therefore $y_p = \frac{1}{3}te^t$ and the full solution is

$$\begin{aligned} y &= y_h + y_p \\ &= A + Be^t + Ce^{\left(-\frac{1}{2}-\frac{3}{2}i\right)t} + De^{\left(-\frac{1}{2}+\frac{3}{2}i\right)t} + \frac{1}{3}te^t \\ &= A + Be^t + Ce^{\frac{-1}{2}t}e^{\frac{-3}{2}it} + De^{\frac{-1}{2}t}e^{\frac{3}{2}it} + \frac{1}{3}te^t \\ &= A + Be^t + e^{\frac{-1}{2}t}\left(C\cos\left(\frac{\sqrt{3}}{2}t\right) + D\sin\left(\frac{\sqrt{3}}{2}t\right)\right) + \frac{1}{3}te^t \end{aligned} \tag{A}$$

Now we find the constants from initial and boundary conditions. At $y(0) = 0$ hence

$$0 = A + B + C \tag{1}$$

From $y(1) = 1$ we obtain

$$1 = A + Be + e^{\frac{-1}{2}}\left(C\cos\left(\frac{\sqrt{3}}{2}\right) + D\sin\left(\frac{\sqrt{3}}{2}\right)\right) + \frac{1}{3}e \tag{2}$$

To apply the other conditions, we need $y'(t)$. Taking derivative of (A) gives

$$y' = Be^t + e^{\frac{-1}{2}t}\left(C\cos\left(\frac{\sqrt{3}}{2}t\right) + D\sin\left(\frac{\sqrt{3}}{2}t\right)\right) + e^{\frac{-1}{2}t}\left(-C\frac{\sqrt{3}}{2}\sin\left(\frac{\sqrt{3}}{2}t\right) + D\frac{\sqrt{3}}{2}\cos\left(\frac{\sqrt{3}}{2}t\right)\right) + \frac{1}{3}e^t + \frac{1}{3}te^t$$

Using $y'(0) = 1$ gives

$$0 = B + C + D\frac{\sqrt{3}}{2} + \frac{1}{3} \tag{3}$$

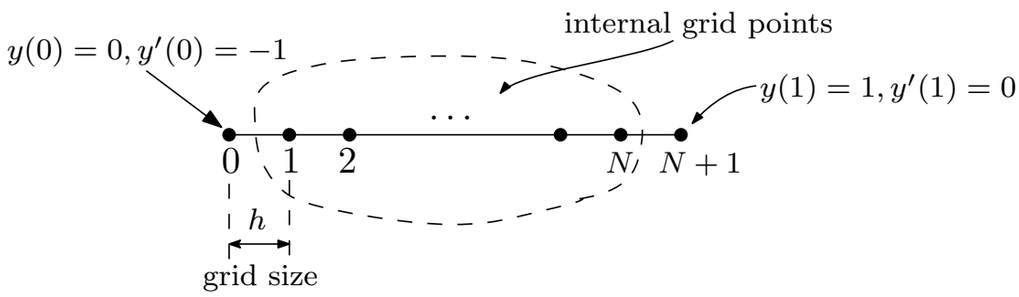
And from $y'(1) = 0$ we find

$$0 = Be + e^{\frac{-1}{2}}\left(C\cos\left(\frac{\sqrt{3}}{2}\right) + D\sin\left(\frac{\sqrt{3}}{2}\right)\right) + e^{\frac{-1}{2}}\left(-C\frac{\sqrt{3}}{2}\sin\left(\frac{\sqrt{3}}{2}\right) + D\frac{\sqrt{3}}{2}\cos\left(\frac{\sqrt{3}}{2}\right)\right) + \frac{2}{3}e \tag{4}$$

Now we solve Eq (1,2,3,4) for A, B, C, D. With the help of the computer, the above were now solved and the coefficients substituted back into (A) to give the analytical solution below

$$y(t) = \frac{1}{3}te^t - 3.956e^t - 16.1397e^{\frac{-1}{2}t}\cos\left(\frac{\sqrt{3}}{2}t\right) - 6.2898e^{\frac{-1}{2}t}\sin\left(\frac{\sqrt{3}}{2}t\right) + 20.096$$

Now that we have the analytical solution, we now need to find a solution using finite differences as well as per problem statement. The first step is to set up the grid. The following diagram shows the grid used



N grid points. $N-2$ internal grid points

Total of N grid points is used. Since the solution is known at $i = 0$ and $i = N-1$, the solution at the remaining only $N-2$ points needs to be determined using finite difference scheme. We will now derive the FD equations for grid points $i = 1, 2, 3$. From grid point $i = 3$ to $i = N$, the same pattern repeats, and the matrix $A_{N\times N}$ will be filled using an iteration process as shown below.

The differential equation

$$y''''(x) - y'(x) = e^t$$

In finite differences form is

$$\frac{y_{i-2} - 4y_{i-1} + 6y_i - 4y_{i+1} + y_{i+2}}{h^4} - \frac{y_{i+1} - y_{i-1}}{2h} = e^{x_i}$$

Where we used centered difference with $O\left(h^2\right)$ local truncation error for the approximation of $y'(x)$ and 5 points centered difference for the approximation of $y''''(x)$

At $i = 1$

$$\frac{y_{-1} - 4y_0 + 6y_1 - 4y_2 + y_3}{h^4} - \frac{y_2 - y_0}{2h} = e^{x_1}$$

$$y_{-1} - 4y_0 + 6y_1 - 4y_2 + y_3 - \frac{1}{2}h^3\left(y_2 - y_0\right) = h^4 e^{x_1}$$

To find y_{-1}, since $y'(0)$ is known, using $y'(0) = y'_0 = \frac{y_1 - y_{-1}}{2h}$ given $y_{-1} = y_1 - 2hy'_0$ and the above becomes

$$\left(y_1 - 2hy'_0\right) - 4y_0 + 6y_1 - 4y_2 + y_3 - \frac{1}{2}h^3\left(y_2 - y_0\right) = h^4 e^{x_1}$$

$$y_0\left(-4 + \frac{1}{2}h^3\right) + 7y_1 + y_2\left(-4 - \frac{1}{2}h^3\right) + y_3 = h^4 e^{x_1} + 2hy'_0$$

$$7y_1 + y_2\left(-4 - \frac{1}{2}h^3\right) + y_3 = h^4 e^{x_1} + 2hy'_0 + y_0\left(4 - \frac{1}{2}h^3\right)$$

At $i = 2$

$$\frac{y_0 - 4y_1 + 6y_2 - 4y_3 + y_4}{h^4} - \frac{y_3 - y_1}{2h} = e^{x_i}$$

$$y_0 - 4y_1 + 6y_2 - 4y_3 + y_4 - \frac{1}{2}h^3\left(y_3 - y_1\right) = h^4 e^{x_2}$$

$$y_1\left(-4 + \frac{1}{2}h^3\right) + 6y_2 + y_3\left(-4 - \frac{1}{2}h^3\right) + y_4 = h^4 e^{x_2} - y_0$$

At $i = 3$

$$\frac{y_1 - 4y_2 + 6y_3 - 4y_4 + y_5}{h^4} - \frac{y_4 - y_2}{2h} = e^{x_3}$$

$$y_1 - 4y_2 + 6y_3 - 4y_4 + y_5 - \frac{1}{2}h^3\left(y_4 - y_2\right) = h^4 e^{x_3}$$

$$y_1 + y_2\left(-4 + \frac{1}{2}h^3\right) + 6y_3 + y_4\left(-4 - \frac{1}{2}h^3\right) + y_5 = h^4 e^{x_3}$$

The rest will now be repeated with i being increased by one for each new row, and shifted

to the right by one for each row. For example for $i = 4$

$$\frac{y_2 - 4y_3 + 6y_4 - 4y_5 + y_6}{h^4} - \frac{y_5 - y_3}{2h} = e^{x_4}$$

$$y_2 - 4y_3 + 6y_4 - 4y_5 + y_6 - \frac{1}{2}h^3\left(y_5 - y_3\right) = h^4 e^{x_4}$$

$$y_2 + y_3\left(-4 + \frac{1}{2}h^3\right) + 6y_4 + y_5\left(-4 + \frac{1}{2}h^3\right) + y_6 = h^4 e^{x_4}$$

Therefore, the $Ax = b$ system to solve is

$$\begin{bmatrix} 7 & \left(-4 - \frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 & 0 & 0 \\ \left(-4 + \frac{1}{2}h^3\right) & 6 & \left(-4 - \frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 & 0 \\ 1 & \left(-4 + \frac{1}{2}h^3\right) & 6 & \left(-4 - \frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 \\ 0 & 1 & \left(-4 + \frac{1}{2}h^3\right) & 6 & \left(-4 + \frac{1}{2}h^3\right) & 1 & 0 & \cdots \\ 0 & 0 & \left(-4 + \frac{1}{2}h^3\right) & 6 & \left(-4 + \frac{1}{2}h^3\right) & 1 & 0 & \cdots \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} h^4 e^h - 2hy_0' + y_0\left(4 - \frac{1}{2}h^3\right) \\ h^4 e^{2h} - y_0 \\ h^4 e^{3h} \\ h^4 e^{4h} \\ \vdots \\ \\ \\ \end{bmatrix}$$

Now to fill in the last 3 rows. At $i = N$

$$\frac{y_{N-2} - 4y_{N-1} + 6y_N - 4y_{N+1} + y_{N+2}}{h^4} - \frac{y_{N+1} - y_{N-1}}{2h} = e^{x_i}$$

But we do not know y_{N+2} but since we know $y'(1) = 0$, then using

$$y'(1) = y'_{N+1} = \frac{y_{N+2} - y_N}{2h}$$

Hence $y_{N+2} = 2hy'(1) + y_N$ and the above becomes (by also replacing $y_{N+1} = y(1)$, which is known).

$$\frac{y_{N-2} - 4y_{N-1} + 6y_N - 4y_{N+1} + 2hy'(1) + y_N}{h^4} - \frac{y_{N+1} - y_{N-1}}{2h} = e^{x_N}$$

$$y_{N-2} - 4y_{N-1} + 6y_N - 4y(1) + 2hy'(1) + y_N - \frac{1}{2}h^3\left(y(1) - y_{N-1}\right) = h^4 e^{x_N}$$

$$y_{N-2} + y_{N-1}\left(-4 + \frac{1}{2}h^3\right) + 7y_N = h^4 e^{x_N} - 2hy'(1) + \left(\frac{1}{2}h^3 + 4\right)y(1)$$

At $i = N - 1$

$$\frac{y_{N-3} - 4y_{N-2} + 6y_{N-1} - 4y_N + y_{N+1}}{h^4} - \frac{y_N - y_{N-2}}{2h} = e^{x_{N-1}}$$

$$y_{N-3} - 4y_{N-2} + 6y_{N-1} - 4y_N + y(1) - \frac{1}{2}h^3\left(y_N - y_{N-2}\right) = h^4 e^{x_{N-1}}$$

$$y_{N-3} + y_{N-2}\left(-4 + \frac{1}{2}h^3\right) + 6y_{N-1} + y_N\left(-4 - \frac{1}{2}h^3\right) = h^4 e^{x_{N-1}} - y(1)$$

At $i = N - 2$

$$\frac{y_{N-4} - 4y_{N-3} + 6y_{N-2} - 4y_{N-1} + y_N}{h^4} - \frac{y_{N-1} - y_{N-3}}{2h} = e^{x_{N-2}}$$

$$y_{N-4} - 4y_{N-3} + 6y_{N-2} - 4y_{N-1} + y_N - \frac{1}{2}h^3\left(y_{N-1} - y_{N-3}\right) = h^4 e^{x_{N-2}}$$

$$y_{N-4} + y_{N-3}\left(-4 + \frac{1}{2}h^3\right) + 6y_{N-2} + y_{N-1}\left(-4 - \frac{1}{2}h^3\right) + y_N = h^4 e^{x_{N-2}}$$

The $Ax = b$ system becomes

$$\begin{bmatrix} 7 & \left(-4-\frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 & 0 & 0 \\ \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4-\frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 & 0 \\ 1 & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4-\frac{1}{2}h^3\right) & 1 & 0 & \cdots & 0 \\ 0 & 1 & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4+\frac{1}{2}h^3\right) & 1 & 0 & \cdots \\ 0 & 0 & 1 & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4+\frac{1}{2}h^3\right) & 1 & 0 \\ \cdots & \cdots & \ddots & \cdots & \ddots & \cdots & \cdots & \cdots \\ 0 & \cdots & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4+\frac{1}{2}h^3\right) & 1 & 0 & \cdots \\ & & & 1 & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4-\frac{1}{2}h^3\right) & 1 \\ 0 & 0 & \cdots & 0 & 1 & \left(-4+\frac{1}{2}h^3\right) & 6 & \left(-4-\frac{1}{2}h^3\right) \\ 0 & 0 & 0 & \cdots & 0 & 1 & \left(-4+\frac{1}{2}h^3\right) & 7 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ \vdots \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} =$$

$$\begin{bmatrix} h^4 e^h - 2hy'_0 + y_0\left(4-\frac{1}{2}h^3\right) \\ h^4 e^{2h} - y(0) \\ h^4 e^{3h} \\ h^4 e^{4h} \\ \vdots \\ \vdots \\ \vdots \\ h^4 e^{x_{N-2}} \\ h^4 e^{x_{N-1}} - y(1) \\ h^4 e^{x_N} - 2hy'(1) + \left(\frac{1}{2}h^3+4\right)y(1) \end{bmatrix}$$

The system is now solved for x, which is the y_i solution and plotted. The Matlab code is given below.

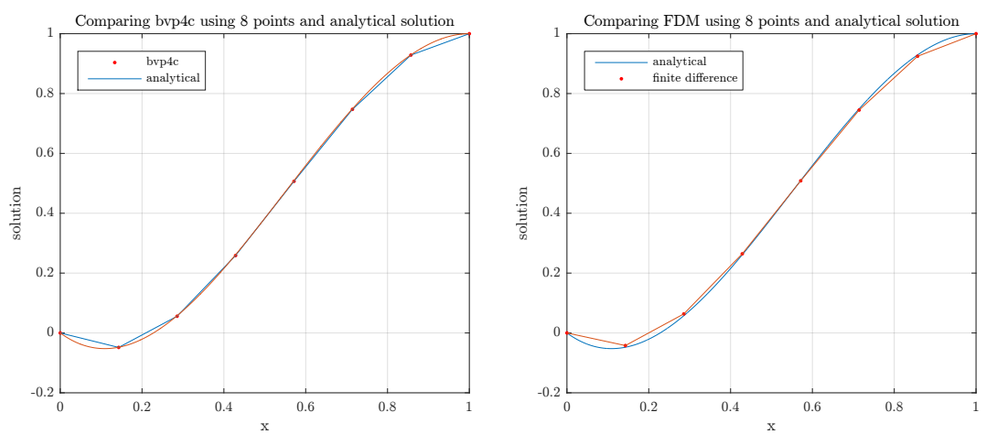
2.2.2.1 Results

The program `nma_EMA_471_HW2_prob_2.m` generates 4 result for different grid sizes. It uses $8, 15, 30, 100$ grid points each time, to compare the result. bvp4c and the finite difference method, both used the same grid size each time. Each time, the result is compare with the analytical solution (which used a much smaller grid than both).

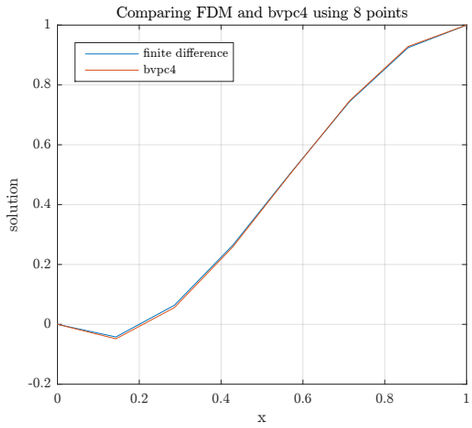
```
....
x_for_analytic=0:0.001:1;
analytical_solution = get_analytical_solution(x_for_analytic);
make_on_test(8,x_for_analytic,analytical_solution);
make_on_test(15,x_for_analytic,analytical_solution);
make_on_test(30,x_for_analytic,analytical_solution);
make_on_test(100,x_for_analytic,analytical_solution);
....
```

.

At small number of grid points (large h), bvp4c seems to be more accurate at the boundary. Here is side by side showing the result for grid size of 8 points over the whole range.



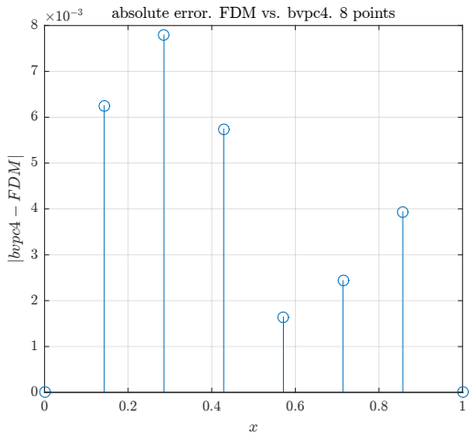
The finite difference was also plotted against the bvp4c solution. The differences between them show up near where the solution changes most rapidly, around $x = 0.2$ and near the right boundary also. Here is the plot when using 8 grid points



To see more clearly the difference, the absolute difference between bvp4c and finite difference solution was plotted, by plotting $|y_{\text{FDM}} - y_{\text{bvp4c}}|$ at each grid point

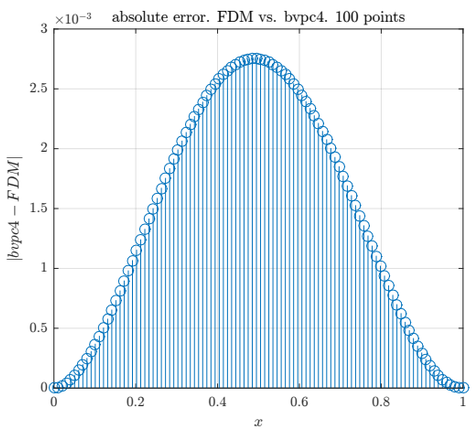
```
....
figure();
stem(x_bvp4c,abs(y_bvp4c-y_FDM));
...
```

.



As more grid points added, the difference between bvp4c and the analytical solution became smaller. The same for finite difference solution. At 100 grid points, the largest absolute difference between bvp4c and FDM was 0.00275, near the middle of the range. This is compared to the difference being 0.008 when using 8 grid points.

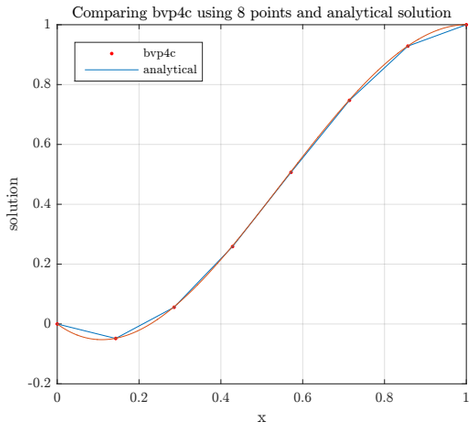
This plot shows the difference at 100 grid points

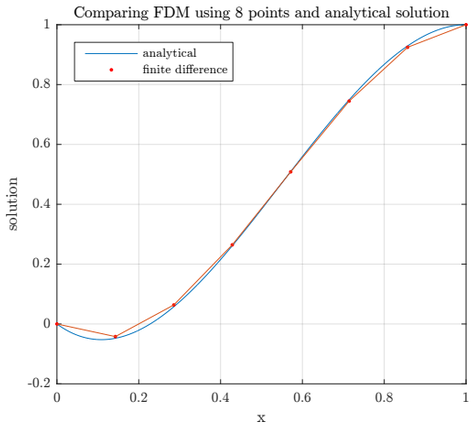


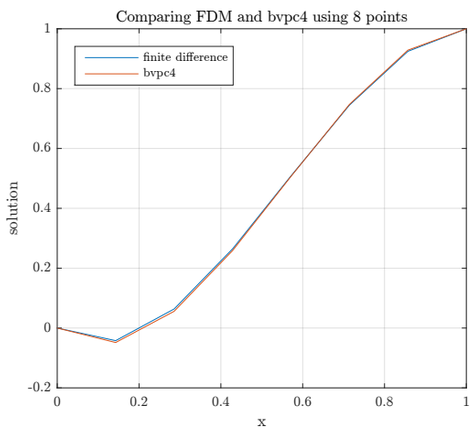
A total of 16 plots generated (4 for each test case) as described above. Below all 16 plots are given, with description title of each plot. Then the source code used to generate these plots is listed.

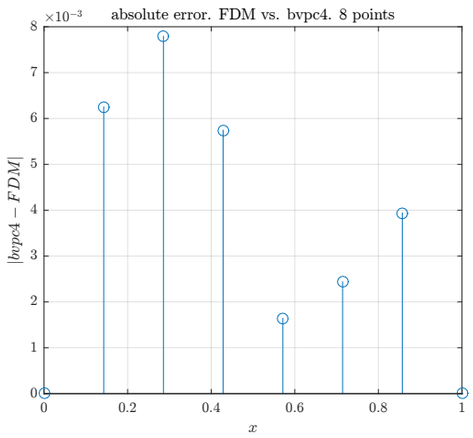
In conclusion: As more grid points added, both bvp4c and FDM approached the analytical solution. There remains difference between bvp4c and FDM in the middle range. Both converged to the same result at the boundaries. This is expected, since the solution there is given from the problem boundary conditions.

2.2.2.1.1 8 grid points plots

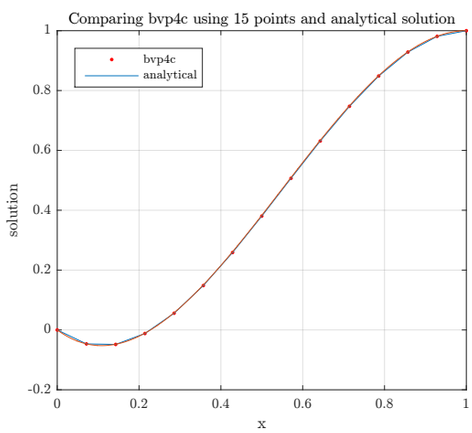


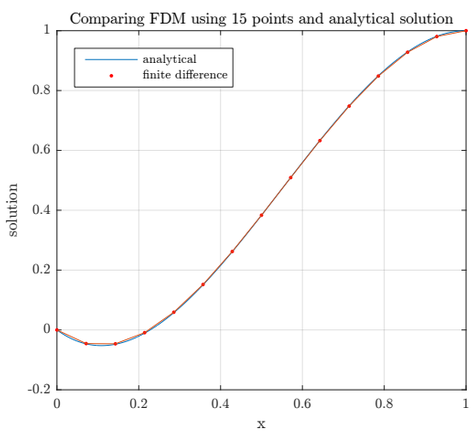


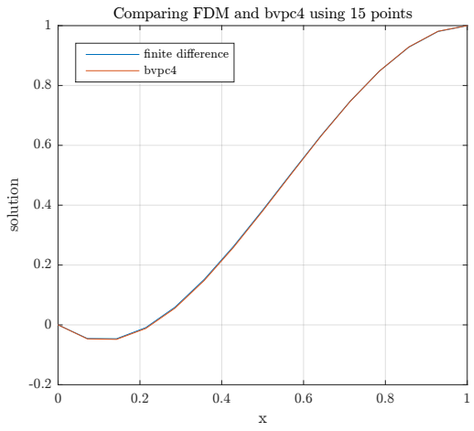


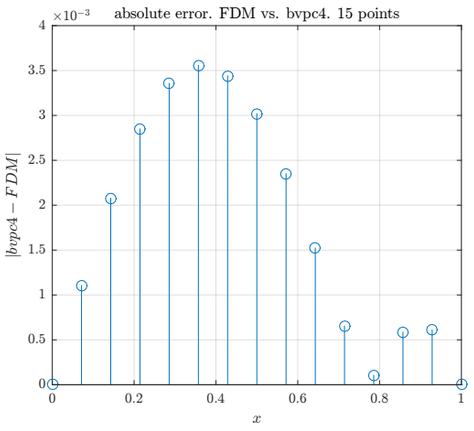


2.2.2.1.2 15 grid points plots

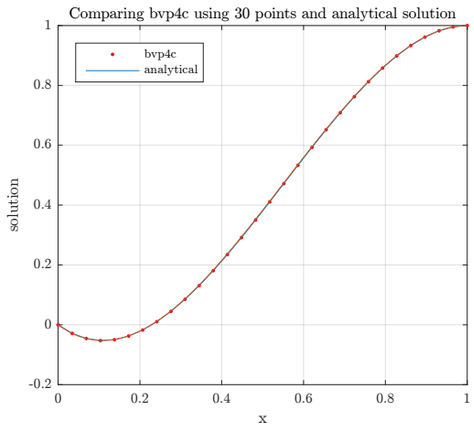


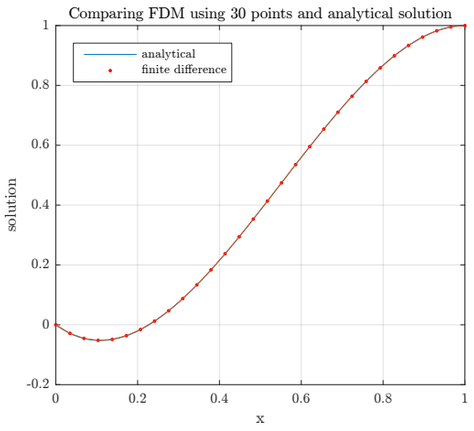


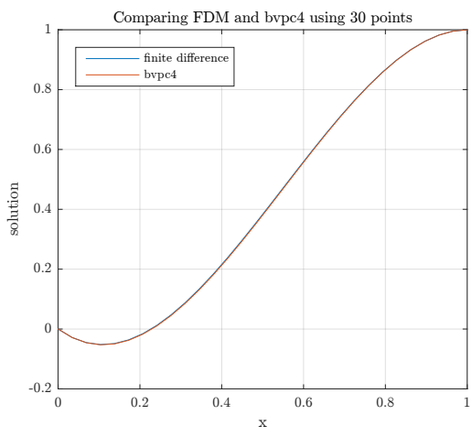


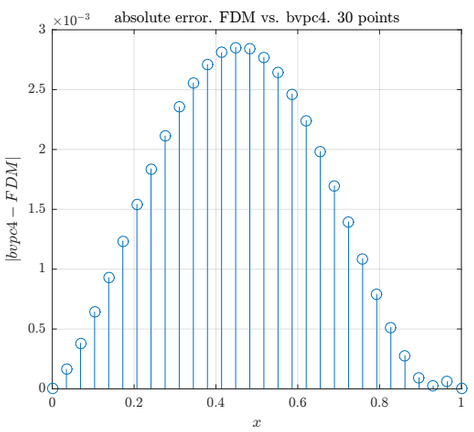


2.2.2.1.3 30 grid points plots

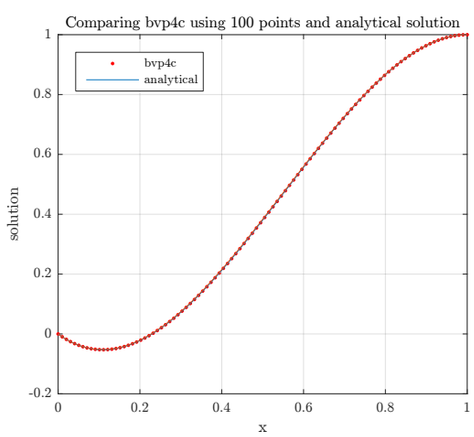


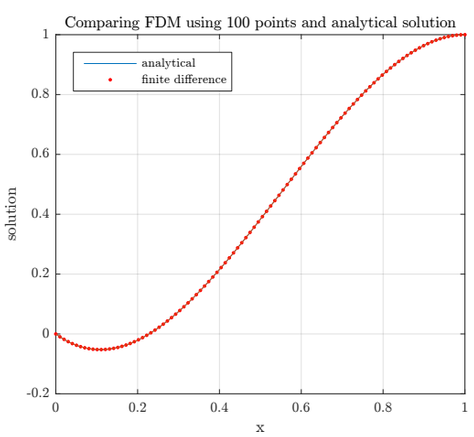


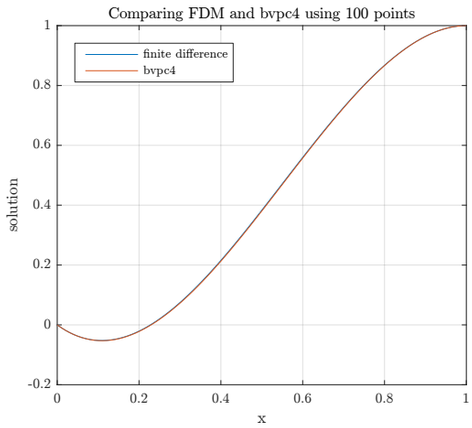


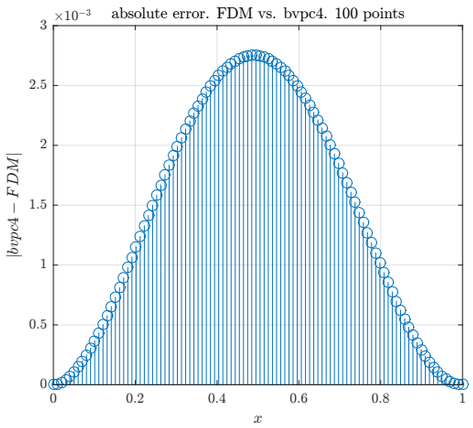


2.2.2.1.4 100 grid points plots









2.2.2.2 Source code

```
function nma_EMA471_HW2_prob_2
%Nasser M. Abbasi

clc; close all;

%generate the analytical solution first, to use to compare
%the numerical results against.
x_for_analytic=0:0.001:1;
analytical_solution = get_analytical_solution(x_for_analytic);

%generate results for different grid sizes
make_one_test(8,x_for_analytic,analytical_solution);
make_one_test(15,x_for_analytic,analytical_solution);
make_one_test(30,x_for_analytic,analytical_solution);
make_one_test(100,x_for_analytic,analytical_solution);
end

%-----------------------------------------------
%This function generates all the plots for bvp4c and FDM
%using specific number of grid points
%
function make_one_test(N,x_anaytic,y_analytical)

%reset for plotting only
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

x_bvp4c  = linspace(0,1,N);
solinit1 = bvpinit(x_bvp4c,[1 0 0 0]); %use specificed N grid points

%options = bvpset('RelTol',1e-6,'AbsTol',1e-6);  Was not
%needed at home pc! The above is only needed at school Matlab,
%which is 2014a. But not 2015a for some reason. One only need
%to pick the correct initial guess

sol = bvp4c(@rhs,@bc,solinit1);

%extract the x and y solution for plotting. These
%variables are used later in the plots

%evaluate at our grid point, to compare with FDM
y_bvp4c = deval(sol,x_bvp4c);
y_bvp4c = y_bvp4c(1,:)';

%plot bvp4c vs. analytical
```

```
figure();
plot(x_bvp4c,y_bvp4c,'r.',x_bvp4c,y_bvp4c);
hold on;
plot(x_anaytic,y_analytical);
xlabel('x'); ylabel('solution');
title(sprintf( ...
   'Comparing bvp4c using %d points and analytical solution',N));
legend('bvp4c','analytical','location','northwest');
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%Obtain FDM solution. Use the same grid spacing as
%bvp4c (ie. same x)
y0    = 0; yL = 1; yd0 = -1; ydL = 0;
y_FDM = finite_difference_solution(x_bvp4c,y0,yL,yd0,ydL);

%plot finite difference vs. analytical
figure();
plot(x_anaytic,y_analytical);
hold on;
plot(x_bvp4c,y_FDM,'r.',x_bvp4c,y_FDM);
legend('analytical','finite difference','location','northwest');
xlabel('x'); ylabel('solution');
title(sprintf(...
    'Comparing FDM using %d points and analytical solution',N));
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);


%plot finite difference vs. bvp4c
figure();
plot(x_bvp4c,y_FDM);
hold on;
plot(x_bvp4c,y_bvp4c);
legend('finite difference','bvpc4','location','northwest');
xlabel('x'); ylabel('solution');
title(sprintf('Comparing FDM and bvpc4 using %d points',N));
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);


%plot error between bvp4c and FDM, to see more
%clearly the difference at each grid point.
figure();
stem(x_bvp4c,abs(y_bvp4c-y_FDM));
xlabel('$x$'); ylabel('$\left|bvpc4-FDM\right|$');
title(sprintf('absolute error. FDM vs. bvpc4. %d points',N));
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

end

%-----------------------------------------
%This function generates the Finite difference solution
%The HW report contains the derivation used
%
function y = finite_difference_solution(x,y0,yL,yd0,ydL)

%Allocate A matrix and b vector
h = x(2)-x(1);   %h spacing is the same between each grid point
N = length(x)-2; %number of internal points.
A = zeros(N,N);
b = zeros(N,1);  %allocate RHS, for Ax=b use
```

```
111
112 %Start filling the A matrix
113
114 %fill in the first 2 rows by hand
115 A(1,1) = 7;
116 A(1,2) = -4-1/2*h^3;
117 A(1,3) = 1;
118
119 A(2,1) = -4+1/2*h^3;
120 A(2,2) = 6;
121 A(2,3) = -4-1/2*h^3;
122 A(2,4) = 1;
123
124 k=0; %column index, used for the loop below, to fill the rest of A
125 for i = 3:N-2
126      k = k+1;
127      A(i,k:k+4) = [1,(-4+1/2*h^3),6,(-4-1/2*h^3),1];
128 end
129
130 %fill in the last 2 rows by hand
131 k = k+1;
132 A(N-1,k:k+3) = [1,(-4+1/2*h^3),6,(-4-1/2*h^3)];
133 k = k+1;
134 A(N,k:k+2)   = [1,(-4+1/2*h^3),7];
135
136 %fill in the b matrix. The first 2 rows by hand
137 b(1) = h^4*exp(h)+2*h*yd0+y0*(4-1/2*h^3);
138 b(2) = h^4*exp(2*h)-y0;
139
140 %fill the rest of b using loop
141 for i = 3:N-2
142     b(i) = h^4*exp(i*h);
143 end
144
145 %fill the last 2 rows of b
146 b(N-1) = h^4*exp((N-1)*h)-yL;
147 b(N)   = h^4*exp(N*h)-2*h*ydL+yL*(4+1/2*h^3);
148
149 %now we solve the system.
150 y=A\b;
151
152 %pad in the left and right boundary values. Known values.
153 y=[y0;y;yL];
154
155 end
156
157 %-----------------------------------------
158 %This function returns the analytical solution. Solved in the HW report
159 function y=get_analytical_solution(t)
160 y=1/3*t.*exp(t)-3.956115643*exp(t)-16.13974994*exp(-.5*t).*...
161                                                 cos(.866025404*t)...
162     -6.289760819*exp(-.5*t).*sin(.866025404*t)+20.09586558;
163 end
164
165 %-------------------------------------------------------------------------%
166 %This function used by bvp4c, the RHS. Same as ODE45 RHS function
167
168 function f = rhs(t,x)
169 x1 = x(2);
170 x2 = x(3);
171 x3 = x(4);
172 x4 = x(1)+exp(t);
173 f = [ x1
```

```
    x2
    x3
    x4 ];
end

%-------------------------------------------------------------------------%
%This is the boundary conditions function for bvp4c
function res = bc(ya,yb)
res = [ ya(1)
    ya(2)+1
    yb(1)-1
    yb(2)
    ];
end
```

2.2.3 Problem 3

PROBLEM DESCRIPTION

(3) (15 pts) One of the topics often discussed in Advanced Mechanics of Materials is the thick rotating cylinder. It is possible to show that the governing equation can be obtained in terms of the radial stress:

$$r\frac{d^2\sigma_r}{dr^2}+3\frac{d\sigma_r}{dr}=-(3+\nu)\rho\omega^2 r$$

with the hoop stress obtained after the fact from:

$$\sigma_\theta = r\frac{d\sigma_r}{dr}+\sigma_r+\rho\omega^2 r^2$$

The second order equation in the radial stress is solved subjected to the constraint that the radial stress equal the negative of the pressure applied at the inner and outer radii. If there is no internal or external pressure, the boundary conditions become:

$$\sigma_r(r_i)=0 \qquad , \qquad \sigma_r(r_o)=0$$

Suppose we have a thick cylinder with inner radius 1 cm, outer radius 10 cm with mass density 1000 kg/m^3. It is subjected to a angular velocity of 700 rad/s. Find the peak radial and hoop stresses for this cylinder. There is an analytical solution to this linear equation, and it also lends itself to a finite difference solution. Compare both of these to the solution generated using `bvp4c`.

SOLUTION

The first step is to determine the analytical solution, in order to compare with the numerical solutions. The ODE to solve is (below, σ is used instead of σ_r to simplify the notation)

$$r\frac{d^2\sigma}{dr^2}+3\frac{d\sigma}{dr}=-(3+\upsilon)\rho\omega^2 r \tag{1}$$

Since υ, ρ, ω are all constants, the above can be written as

$$r\frac{d^2\sigma}{dr^2}+3\frac{d\sigma}{dr}=kr$$

Where $k=-(3+\upsilon)\rho\omega^2$. To solve the above, we introduce $f=\frac{d\sigma}{dr}$ and it becomes a first order ODE

$$\begin{aligned} r\frac{df}{dr}+3f &= kr \\ \frac{df}{dr}+\frac{3}{r}f &= k \end{aligned} \tag{2}$$

This is separable now, and can be solved for f. Looking at the homogenous ODE first,

$$\begin{aligned}\frac{df}{dr} + \frac{3}{r}f &= 0\\ \frac{df}{f} &= -\frac{3}{r}dr\end{aligned}$$

Integrating both sides

$$\begin{aligned}\ln f &= -3\ln r + c_1\\ f &= e^{-3\ln r + c_1}\\ &= c_2 e^{-3\ln r}\end{aligned}$$

Hence the homogenous solution is

$$f_h = \frac{c_2}{r^3}$$

Where c_2 is constant of integration. To find the particular solution f_p, and since the RHS of (2) is constant k, then we guess $f_p = crk$, where c is constant. Hence (2) becomes

$$\begin{aligned}ck + \frac{3}{r}rck &= k\\ 4c &= 1\\ c &= \frac{1}{4}\end{aligned}$$

Hence

$$f_p = \frac{r}{4}k$$

And the complete solution is

$$\begin{aligned}f &= f_h + f_p\\ &= \frac{c_2}{r^3} + \frac{r}{4}k\end{aligned}$$

Now that we found $f(r)$, and since $f = \frac{d\sigma}{dr}$ then we have

$$\frac{d\sigma}{dr} = \frac{c_2}{r^3} + \frac{r}{4}k$$

Which is separable. Hence

$$\begin{aligned}d\sigma &= \left(\frac{c_2}{r^3} + \frac{r}{4}k\right)dr\\ \sigma &= \int\left(\frac{c_2}{r^3} + \frac{r}{4}k\right)dr\\ &= \frac{-c_2}{2}\frac{1}{r^2} + \frac{r^2}{8}k + c_3\end{aligned}$$

Hence the analytical solution is

$$\sigma_r(r) = \frac{-C_1}{2}\frac{1}{r^2} + \frac{r^2}{8}k + C_2 \tag{3}$$

Where C_1, C_2 are constants of integration, which we will now find from boundary conditions. At $r = r_i, \sigma_r = 0$ and at $r = r_o, \sigma_r = 0$, hence we obtain two equations

$$\begin{aligned}0 &= \frac{-C_1}{2}\frac{1}{r_i^2} + \frac{r_i^2}{8}k + C_2\\ 0 &= \frac{-C_1}{2}\frac{1}{r_o^2} + \frac{r_o^2}{8}k + C_2\end{aligned}$$

Solving these gives

$$\begin{aligned}C_1 &= \frac{-k}{4}r_i^2 r_o^2\\ C_2 &= \frac{-k}{8}\left(r_i^2 + r_o^2\right)\end{aligned}$$

Therefore (3) becomes

$$\sigma_r(r) = \frac{k}{8}\left(r_i^2 r_o^2\frac{1}{r^2} + r^2 - \left(r_i^2 + r_o^2\right)\right)$$

But $k = -(3+\upsilon)\rho\omega^2$, hence the above becomes

$$\boxed{\sigma_r(r) = \frac{(3+\upsilon)\rho\omega^2}{8}\left(\left(r_i^2 + r_o^2\right) - r_i^2 r_o^2 \frac{1}{r^2} - r^2\right)} \tag{4}$$

Now we will find the analytical solution for the hoop stress

$$\sigma_\theta = r\frac{d\sigma}{dr} + \sigma_r + \rho\omega^2 r^2$$

From (4), we see that

$$\frac{d\sigma}{dr} = \frac{(3+\upsilon)\rho\omega^2}{8}\left(\frac{2}{r^3} r_i^2 r_o^2 - 2r\right)$$

Hence

$$\boxed{\sigma_\theta = \frac{(3+\upsilon)\rho\omega^2}{8}\left(\frac{2}{r^2} r_i^2 r_o^2 - 2r^2\right) + \sigma_r + \rho\omega^2 r^2} \tag{5}$$

Now that we have found the analytical solutions, we can implement the numerical solution and compare. We start with bvp4c to verify the analytical solution with, then implement the finite difference method. We need first to convert the ODE to state space.

$$\frac{d^2\sigma}{dr^2} + \frac{3}{r}\frac{d\sigma}{dr} = -(3+\upsilon)\rho\omega^2$$

Let $x_1 = \sigma_r$, and let $x_2 = \frac{d\sigma}{dr}$, hence

$$\dot{x}_1 = x_2$$

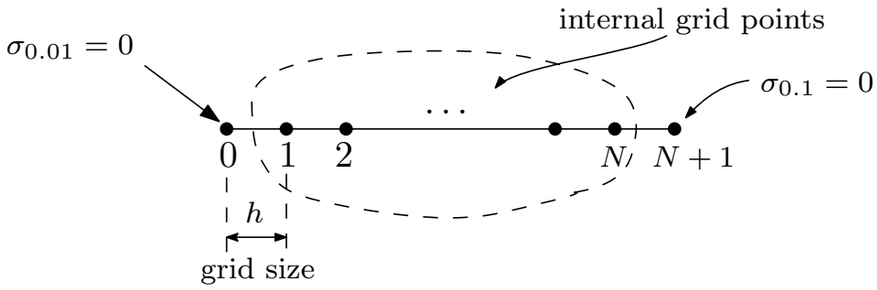
$$\dot{x}_2 = -(3+\upsilon)\rho\omega^2 - \frac{3}{r}x_2$$

The boundary conditions are (using bvp4c notations) $y_a(1) = 0, y_b(1) = 0$.

2.2.3.1 Finite difference method

$$\frac{d^2\sigma}{dr^2} + 3\frac{1}{r}\frac{d\sigma}{dr} = -(3+\upsilon)\rho\omega^2$$

The following grid is used



N grid points. $N-2$ internal grid points

The grid is only over $r = 0.01 \cdots 0.1$ meters since this is the distance between the internal radius and the outer radius. For $\frac{d^2\sigma}{dr^2}$ we use 3 points centered difference approximation, for $i = 1 \cdots N$ which has $O\left(h^2\right)$ local truncation error

$$\frac{d^2\sigma}{dr^2} = \frac{\sigma_{i+1} - 2\sigma_i + \sigma_{i-1}}{h^2}$$

Where σ_0, σ_{N+1} are given as the boundary conditions (both are zero). For $\frac{d\sigma}{dr}$ we use two points centered difference, which also has $O\left(h^2\right)$ local truncation error

$$\frac{d\sigma}{dr} = \frac{\sigma_{i+1} - \sigma_{i-1}}{2h}$$

Therefore, the ODE becomes

$$r_i \frac{\sigma_{i+1} - 2\sigma_i + \sigma_{i-1}}{h^2} + 3\frac{\sigma_{i+1} - \sigma_{i-1}}{2h} = -(3+\upsilon)\rho\omega^2 r_i$$

$$\sigma_{i+1} - 2\sigma_i + \sigma_{i-1} + \frac{3}{2r_i}h(\sigma_{i+1} - \sigma_{i-1}) = -h^2(3+\upsilon)\rho\omega^2$$

$$\sigma_{i-1}\left(1 - \frac{3}{2r_i}h\right) - 2\sigma_i + \sigma_{i+1}\left(1 + \frac{3}{2r_i}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

Where $r_i = r_i + ih = 0.01 + ih$, since are starting from 0.01, the above becomes

$$\sigma_{i-1}\left(1 - \frac{3}{2(r_i+ih)}h\right) - 2\sigma_i + \sigma_{i+1}\left(1 + \frac{3}{2(r_i+ih)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

For $i = 1$,

$$\sigma_0\left(1 - \frac{3}{2(r_i+h)}h\right) - 2\sigma_1 + \sigma_2\left(1 + \frac{3}{2(r_i+h)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

But σ_0 is the boundary conditions, which we move to the RHS, hence

$$-2\sigma_1 + \sigma_2\left(1 + \frac{3}{2(r_i+h)}h\right) = -h^2(3+\upsilon)\rho\omega^2 - \sigma_0\left(1 - \frac{3}{2(r_i+h)}h\right)$$

For $i = 2$

$$\sigma_1\left(1 - \frac{3}{2(r_i+2h)}h\right) - 2\sigma_2 + \sigma_3\left(1 + \frac{3}{2(r_i+2h)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

For $i = 3$

$$\sigma_2\left(1 - \frac{3}{2(r_i+3h)}h\right) - 2\sigma_3 + \sigma_4\left(1 + \frac{3}{2(r_i+3h)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

The same pattern repeats. For $i = N$

$$\sigma_{N-1}\left(1 - \frac{3}{2(r_i+Nh)}h\right) - 2\sigma_N + \sigma_{N+1}\left(1 + \frac{3}{2(r_i+Nh)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

But σ_{N+1} is given (the right side boundary conditions, hence

$$\sigma_{N-1}\left(1 - \frac{3}{2(r_i+Nh)}h\right) - 2\sigma_N = -h^2(3+\upsilon)\rho\omega^2 - \sigma_{N+1}\left(1 + \frac{3}{2(r_i+Nh)}h\right)$$

And for $i = N-1$

$$\sigma_{N-2}\left(1 - \frac{3}{2(r_i+(N-1)h)}h\right) - 2\sigma_{N-1} + \sigma_N\left(1 + \frac{3}{2(r_i+(N-1)h)}h\right) = -h^2(3+\upsilon)\rho\omega^2$$

Therefore the $Ax = b$ system is below. The A matrix is

$$\begin{bmatrix}
-2 & 1+\frac{3h}{2(r_i+h)} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots \\
1-\frac{3h}{2(r_i+2h)} & -2 & 1+\frac{3h}{2(r_i+2h)} & 0 & \cdots & \cdots & \cdots & \cdots \\
0 & 1-\frac{3h}{2(r_i+3h)} & -2 & 1+\frac{3h}{2(r_i+3h)} & 0 & \cdots & \cdots & \cdots \\
\vdots & 0 & 0 & 0 & 0 & \ddots & \cdots & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \vdots & \vdots & 0 & 1-\frac{3h}{2(r_i+(N-1)h)} & -2 & 1+\frac{3h}{2(r_i+(N-1)h)} \\
0 & 0 & 0 & 0 & 0 & 0 & 1-\frac{3h}{2(r_i+Nh)} & -2
\end{bmatrix}$$

And the x vector is $\begin{bmatrix}\sigma_1 & \sigma_1 & \cdots & \sigma_{N-2} & \sigma_{N-1} & \sigma_N\end{bmatrix}^T$ and the b vector is

$$\begin{bmatrix} -h^2(3+\upsilon)\rho\omega^2 - \sigma_0\left(1-\frac{3h}{2(r_i+h)}\right) \\ -h^2(3+\upsilon)\rho\omega^2 \\ -h^2(3+\upsilon)\rho\omega^2 \\ \vdots \\ \vdots \\ -h^2(3+\upsilon)\rho\omega^2 \\ -h^2(3+\upsilon)\rho\omega^2 \\ -h^2(3+\upsilon)\rho\omega^2 - \sigma_{N+1}\left(1+\frac{3h}{2(r_i+Nh)}\right) \end{bmatrix}$$

2.2.3.1.1 Finding numerical solution for hoop stress To compare the analytical solution for σ_θ with the numerical one, we need to evaluate $\sigma_\theta = r\frac{d\sigma}{dr} + \sigma_r + \rho\omega^2 r^2$ numerically from the numerical solution we found about using finite difference method. Using finite difference, this expression becomes

$$\sigma_{\theta_i} = (r_{int} + ih)\frac{\sigma_{i+1} - \sigma_{i-1}}{2h} + \sigma_i + \rho\omega^2(r_{int} + ih)^2$$

Where $r_{int} = 0.01$ meter. Since we do not know σ_{-1} and can not obtain derivative at the edge to approximate using phantom grid point, we will start from $i = 1 \cdots N$, so that $\sigma_{i-1} = \sigma_0$ which is known.

Hence for $i = 1$ we have

$$\sigma_{\theta_1} = (r_{int} + h)\frac{\sigma_2 - \sigma_0}{2h} + \sigma_1 + \rho\omega^2(r_{int} + h)^2$$

And for $i = N$

$$\sigma_{\theta_N} = (r_{int} + Nh)\frac{\sigma_{N+1} - \sigma_{N-1}}{2h} + \sigma_N + \rho\omega^2(r_{int} + Nh)^2 \tag{6}$$

In the above, σ_{N+1} and σ_0 are the boundary conditions we are given. All other σ_i values are obtained from the numerical solution we did in the last section (the finite difference solution). This was implemented in Matlab.

2.2.3.2 Results

The program was run for $10, 15, 20, 25, 30, 100$ grid points. Each time, 4 plots were generated:

1. Plot comparing the analytical and FDM result for σ_r Title contains location and value of maximum σ_r reported by FDM based method.
2. Plot comparing the analytical and bvp4c result for σ_r Title contains location and value of maximum σ_r reported by bvp4c based method
3. Plot comparing bvp4c and FDM result for σ_r
4. Plot comparing analytical result and FDM for σ_θ Title contains location and value of maximum σ_θ reported by FDM based method.

The result shows that bvp4c was more accurate than FDM for small number of grid points (large h) since the bvp4c curve was closer to the analytical solution than FDM curve. To get more accurate numerical hoop stress, the number of grid points needed to be over 50. At $N = 100$ grid points, the result of σ_θ matched the analytical solution extremely well as can be seen from the plots below. The following table gives the maximum radial stress found and the location as reported by bvp4c and FDM based methods. Units for stress is N/m^2 and units for r is meters. This shows the maximum radial stress occurs at left edge $r = 0.01$ meter as expected. This is where the inner hole edge starts. The maximum radial stress is located at $r = 0.032$ meter. About 2 cm after the inner hole edge.

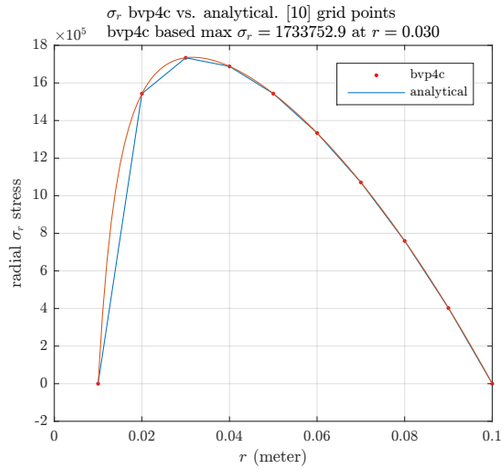
N (grid points)	Max σ_r FDM based, and r location	Max σ_r bvp4c based, and location
10	σ_r =1,783,600 at $r = 0.03$	σ_r =1,733,752 at $r = 0.03$
15	σ_r =1,731,188 at $r = 0.29$	σ_r =1,752,483 at $r = 0.029$
20	σ_r =1,732,767 at $r = 0.034$	σ_r =1,741,523 at $r = 0.034$
25	σ_r =1,735,518 at $r = 0.033$	σ_r =1,741,592 at $r = 0.033$
30	σ_r =1,735,984 at $r = 0.032$	σ_r =1,740,593 at $r = 0.032$
100	σ_r =1,736,401 at $r = 0.032$	σ_r =1,736,759 at $r = 0.032$

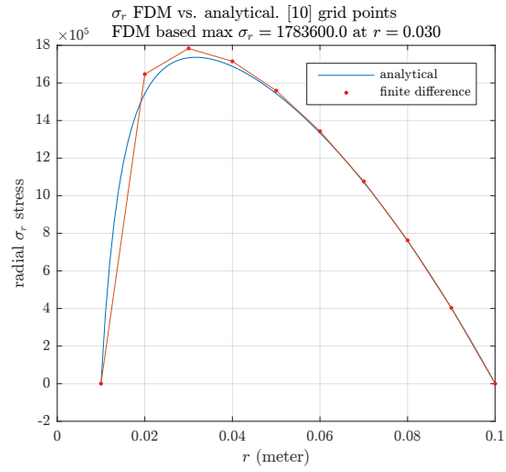
The following table gives the maximum hoop stress σ_θ found and the location as reported FDM based method. Units for stress is N/m^2 and units for r is meters. Each plot below also display this information in the title.

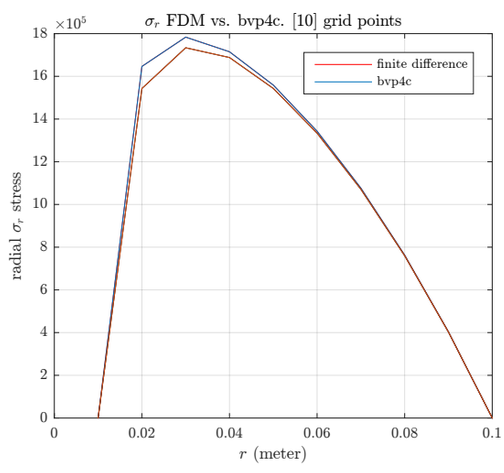
N (grid points)	Max σ_θ value and location
10	σ_θ =3,626,000 at $r = 0.01$
15	σ_θ =3,639,120 at $r = 0.01$
20	σ_θ =3,665,659 at $r = 0.01$
25	σ_θ =3,697,797 at $r = 0.01$
30	σ_θ =3,730,811 at $r = 0.01$
100	σ_θ =4,004,798 at $r = 0.01$

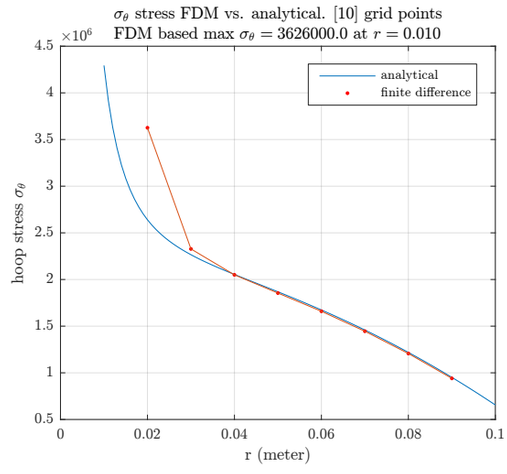
There are total of 24 plots below. Four plots for each N. This helps show that more grid points resulted in more accurate numerical solution.

2.2.3.2.1 10 grid points

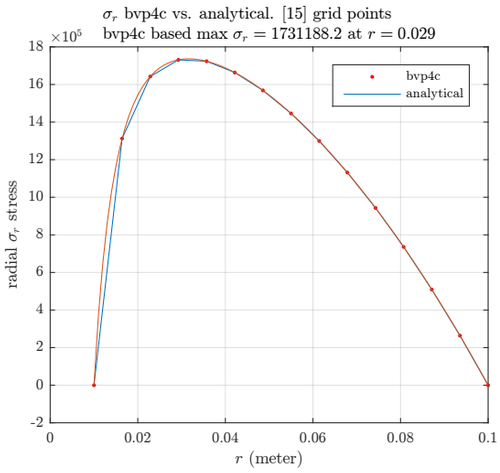


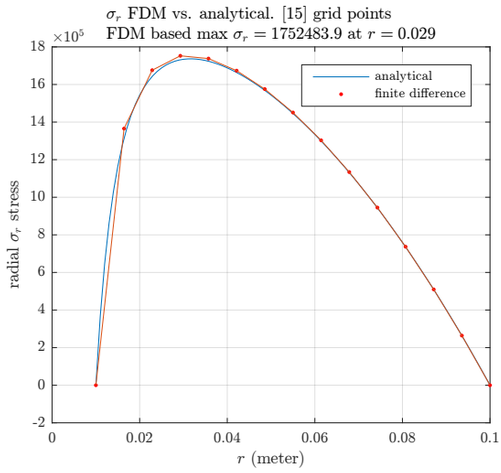


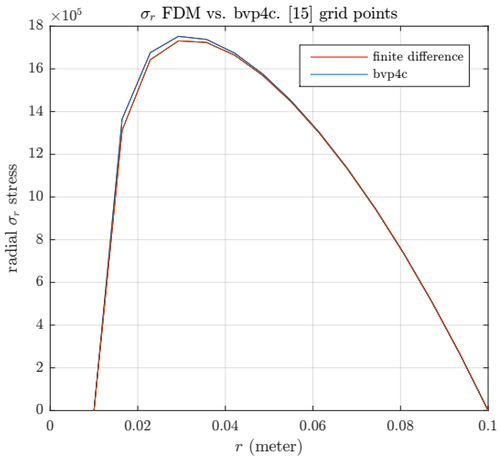


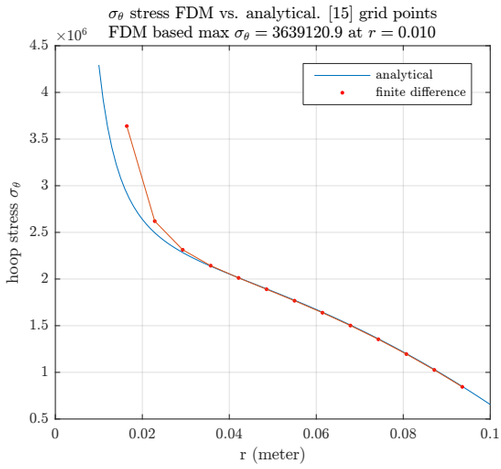


2.2.3.2.2 15 grid points

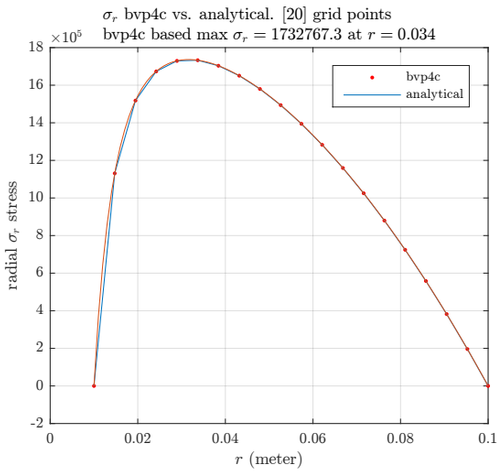


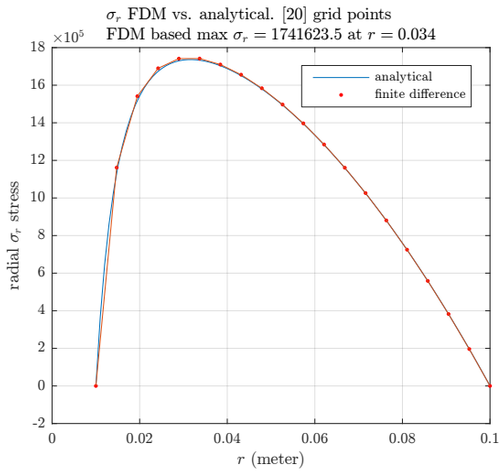


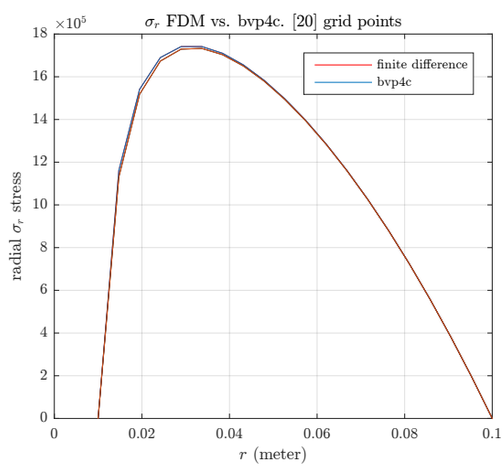


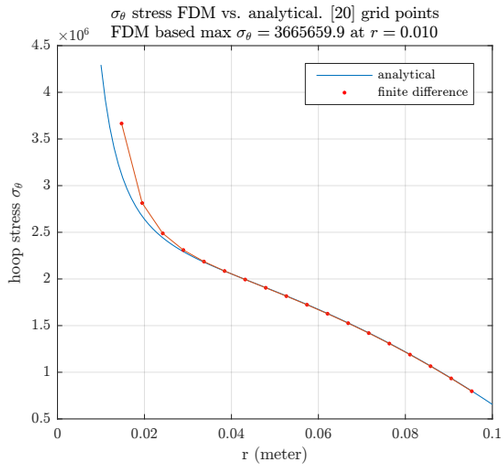


2.2.3.2.3 20 grid points

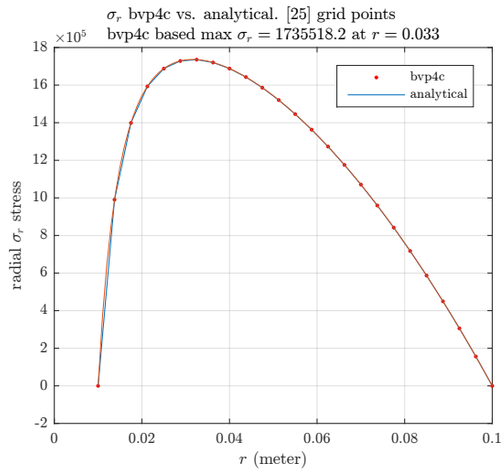


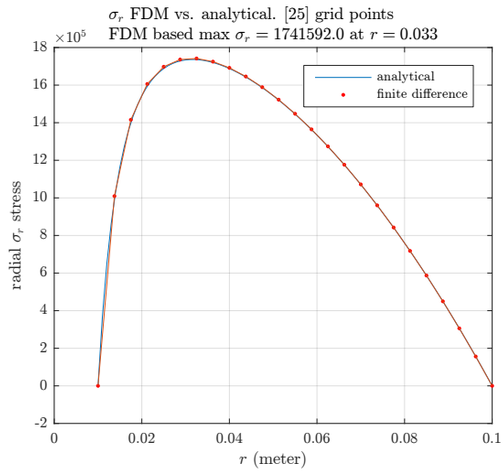


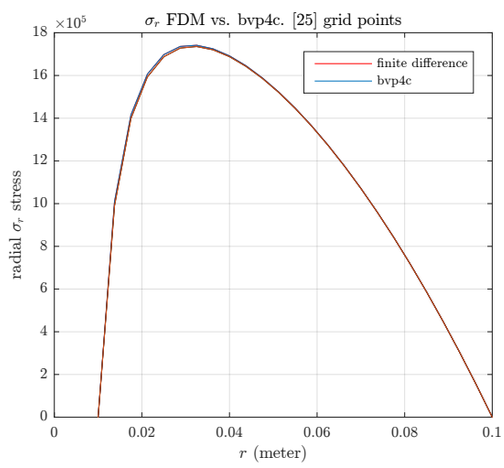


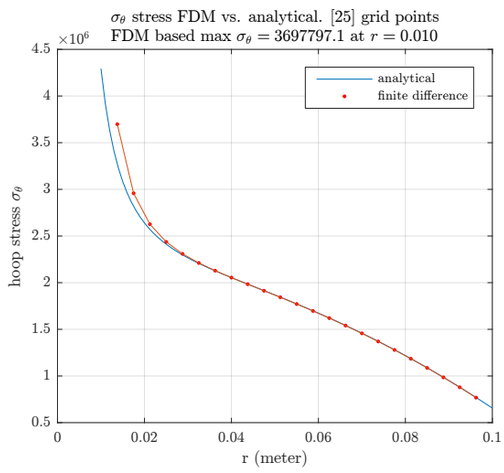


2.2.3.2.4 25 grid points

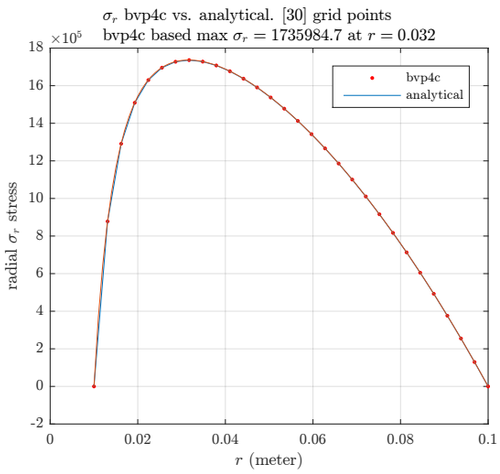


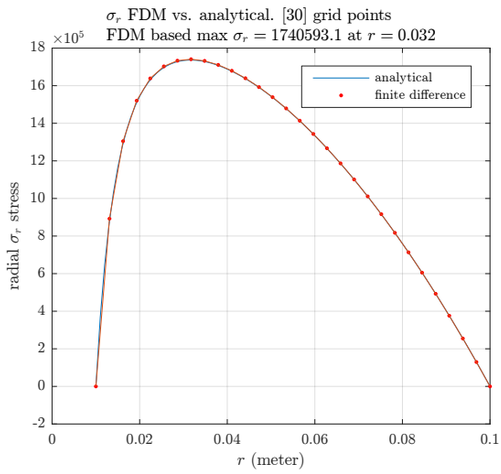


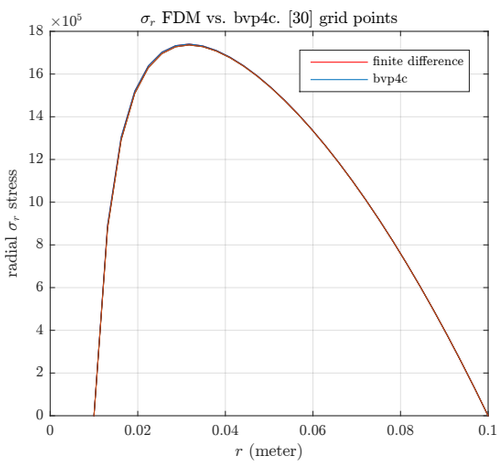


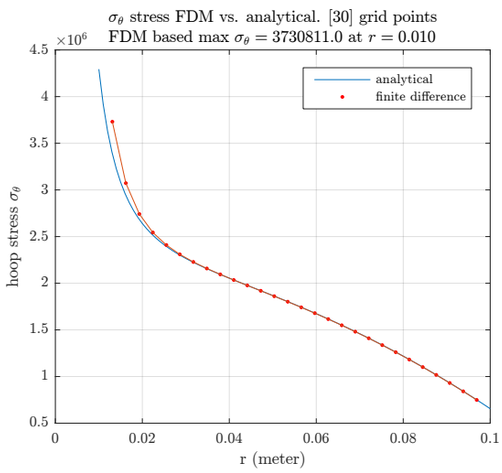


2.2.3.2.5 30 grid points

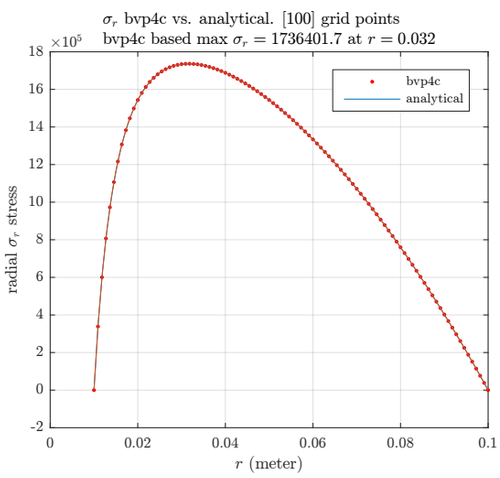


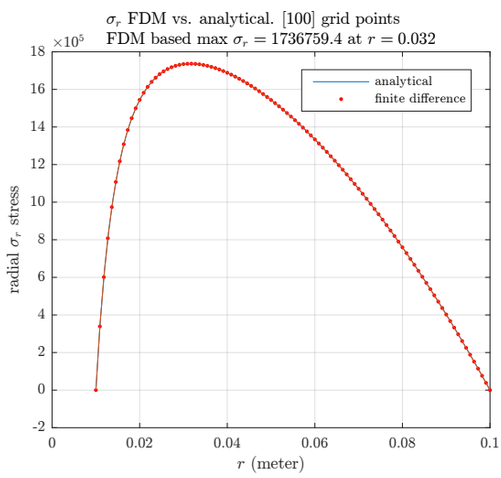


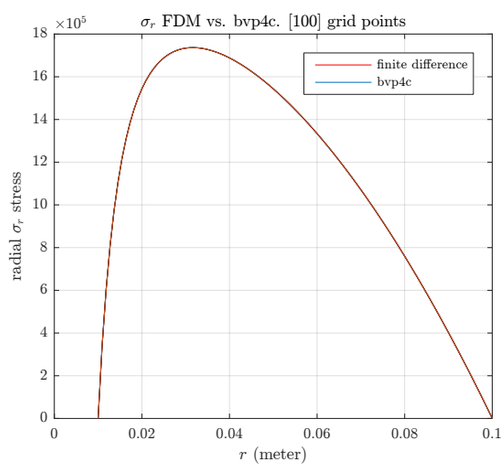


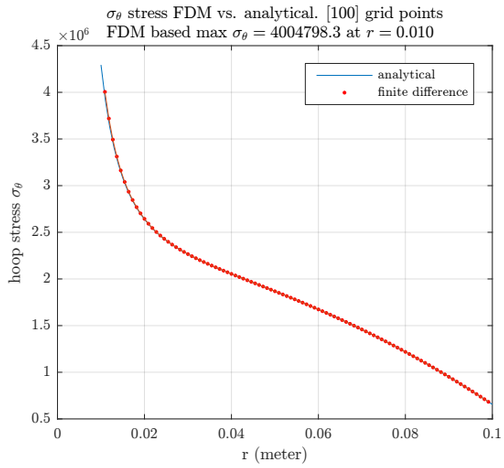


2.2.3.2.6 100 grid points









2.2.3.3 source code

```
function nma_EMA471_HW2_prob_3
%Solves problem 3, HW2. EMA 471, spring 2016
%Nasser M. Abbasi

clc; close all;

%generate the analytical solution first, to use to compare the numerical
%results against.

ri          = 0.01; %meter
ro          = 0.1;  %meter
x_analytic  = ri:0.001:ro;
density     = 1000; %kg/m^3
omega       = 700;  %radians per second
nu          = 0.5;  %Poisson's ratio

y_analytic = get_analytical_solution(x_analytic,ri,ro,density,omega,nu);

%generate results for different grid sizes
N             = 100; %select the number of grid points
[x_FDM,y_FDM] = make_one_test(N,x_analytic,y_analytic,ri,ro,density,omega,nu);

%now find the analytical solution for hoop stress, and compare with
%the numerical one
y_analytic_hoop = get_analytical_solution_hoop(x_analytic, ...
                                        y_analytic,ri,ro,density,omega,nu);

%compare with numerical result for hoop stress
make_one_test_hoop(N,x_FDM,y_FDM,x_analytic,y_analytic_hoop,ri,density,omega);

end

%-----------------------------------------------
%This function generates all the plots for bvp4c and FDM
%using specific number of grid points
%
function [x_bvp4c,y_FDM] = make_one_test(N,x_anaytic,...
                                    y_analytical,ri,ro,density,omega,nu)

%reset for plotting only
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

x_bvp4c  = linspace(ri,ro,N);
```

```
solinit1 = bvpinit(x_bvp4c,[1 1]); %use specificed N grid points
%options = bvpset('RelTol',1e-6,'AbsTol',1e-6);  %Was not needed for this

sol = bvp4c(@rhs,@bc,solinit1);

%extract the x and y solution for plotting. These variables are used
%later in the plots
y_bvp4c = deval(sol,x_bvp4c); %evaluate at our grid point, to compare with FDM
y_bvp4c = y_bvp4c(1,:)';

%plot bvp4c vs. analytical
figure();
plot(x_bvp4c,y_bvp4c,'r.',x_bvp4c,y_bvp4c);
[max_radial_stress,I]=max(y_bvp4c);
hold on;
plot(x_anaytic,y_analytical);
xlabel('$r$ (meter)'); ylabel('radial $\sigma_r$ stress');
title({sprintf('$\\sigma_r$ bvp4c vs. analytical. [%d] grid points',N),...
      sprintf('bvp4c based max $\\sigma_r = %3.1f$ at $r= %3.3f$',max_radial_stress,x_bvp4c(I))
legend('bvp4c','analytical','location','northeast');
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%Obtain FDM solution. Use the same grid spacing as bvp4c (ie. same x spacing)
y0    = 0;
yL    = 0;
y_FDM = finite_difference_solution(x_bvp4c,y0,yL,ri,density,omega,nu);
[max_radial_stress,I] = max(y_FDM);

%plot finite difference vs. analytical
figure();
plot(x_anaytic,y_analytical);

hold on;
plot(x_bvp4c,y_FDM,'r.',x_bvp4c,y_FDM);
legend('analytical','finite difference','location','northeast');
xlabel('$r$ (meter)'); ylabel('radial $\sigma_r$ stress');
title({sprintf('$\\sigma_r$ FDM vs. analytical. [%d] grid points',N),...
      sprintf('FDM based max $\\sigma_r = %3.1f$ at $r= %3.3f$',max_radial_stress,x_bvp4c(I))}
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

%plot finite difference vs. bvp4c
figure();
plot(x_bvp4c,y_FDM,'r',x_bvp4c,y_FDM);

hold on;
plot(x_bvp4c,y_bvp4c,'k',x_bvp4c,y_bvp4c);
legend('finite difference','bvp4c','location','northeast');
xlabel('$r$ (meter)'); ylabel('radial $\sigma_r$ stress');
title(sprintf('$\\sigma_r$ FDM vs. bvp4c. [%d] grid points',N));
grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);


    %------------------------------------------------------------------------%
    %This internal function used by bvp4c, the RHS. Same as ODE45 RHS function

    function f = rhs(t,x)
        x1 = x(2);
        x2 = -(3+nu)*density*omega^2 - 3/t * x(2);
        f = [ x1
              x2];
```

```
    end
    %-------------------------------------------------------------------------%
    %This internal is the boundary conditions function for bvp4c
    function res = bc(ya,yb)
        res = [ ya(1)
                yb(1)];
    end
end
%-----------------------------------------
%This function generate numerical solution for hoop stress, using the
%solution found earlier for the stress from finite difference and
%compare the result to the analytical solution of hoop stress.
function make_one_test_hoop(N,x_FDM,sigma_FDM,x_analytic,y_analytic_hoop,...
                                                    ri,density,omega)

%Obtain FDM based hoop stress, uses stress allready solved using FDM
h=x_FDM(2)-x_FDM(1);
i=(2:length(x_FDM)-1)';

%tried one point difference for finding hoop stress at initial and
%end points, but since O(h), it gives bad result. So keep the calculation
%for the internal points only. Block commented out
%
%hoop_stress_FDM=zeros(length(x_FDM),1);
%hoop_stress_FDM(1) = ri * (sigma_FDM(2)-sigma_FDM(1))/h + ...
%                      sigma_FDM(1) + density*omega^2*ri^2;
%hoop_stress_FDM(end) = ro * (sigma_FDM(end)-sigma_FDM(end-1))/h + ...
%                      sigma_FDM(end)+density*omega^2*(ro)^2;
%hoop_stress_FDM(2:end-1) = (ri+(i-1)*h) .* (sigma_FDM(i+1)-sigma_FDM(i-1))/(2*h) + ...
%                      sigma_FDM(i)+density*omega^2*(ri+(i-1)*h).^2;

%This below implements eq(6) in the HW report.
hoop_stress_FDM = (ri+(i-1)*h) .* (sigma_FDM(i+1)-sigma_FDM(i-1))/(2*h) + ...
                     sigma_FDM(i)+density*omega^2*(ri+(i-1)*h).^2;
[max_hoop_stress,I] = max(hoop_stress_FDM);
%plot finite difference vs. analytical hoop stress
figure();
plot(x_analytic,y_analytic_hoop);
hold on;
plot(x_FDM(2:end-1),hoop_stress_FDM,'r.',x_FDM(2:end-1),hoop_stress_FDM);
legend('analytical','finite difference','location','northeast');
xlabel('r (meter)');  ylabel('hoop stress $\sigma_\theta$');
title({sprintf('$\\sigma_\\theta$ stress FDM vs. analytical. [%d] grid points',N),...
  sprintf('FDM based max $\\sigma_\\theta = %3.1f$ at $r= %3.3f$',max_hoop_stress,x_FDM(I))}

grid;
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

end
%-----------------------------------------
%This function returns the analytical solution for hoop stress.
%Solved in the HW report
function y = get_analytical_solution_hoop(r,sigma,ri,ro,density,omega,nu)

k = (3+nu)*density*omega^2/8;
y = k*( 2./(r.^2) * ri^2*ro^2 -2 * r.^2) + sigma + density*omega^2*r .^2;

end
%-----------------------------------------
%This function returns the analytical solution. Solved in the HW report
function y=get_analytical_solution(x,ri,ro,density,omega,nu)

k = (3+nu)*density*omega^2/8;
```

```
y = k*( (ri^2+ro^2)- ri^2*ro^2 ./ (x.^2) - x.^2);
end
%-------------------------------------------
%This function generates the Finite difference solution
%The HW report contains the derivation used for Ax=b setup
%
function y = finite_difference_solution(x,y0,yL,ri,density,omega,nu)
%Allocate A matrix and b vector
h = x(2)-x(1);   %h spacing is the same between each grid point
N = length(x)-2; %number of internal points.
A = zeros(N,N);
b = zeros(N,1);  %allocate RHS, for Ax=b use

%Start filling the A matrix

%fill in the first row by hand
A(1,1) = -2;
A(1,2) = (3*h)/(2*(ri+h))+1;

k=0; %column index, used for the loop below, to fill the rest of A
for i = 2:N-1
     k = k+1;
     A(i,k:k+2) = [1-(3*h)/(2*(ri+i*h)),-2,(3*h)/(2*(ri+i*h))+1];
end

%fill in the last  row by hand
k = k+1;
A(N,k:k+1)   = [1-(3*h)/(2*(ri+N*h)),-2];

%fill in the b matrix. The first  row1 by hand
b(1) = -h^2*(3+nu)*density*omega^2-y0*(1-(3*h)/(2*(ri+h)));

%fill the rest of b using loop
for i = 2:N-1
    b(i) = -h^2*(3+nu)*density*omega^2;
end

%fill the last row of b
b(N)   = -h^2*(3+nu)*density*omega^2-yL*((3*h)/(2*(ri+N*h))+1);

%now we solve the system.
y=A\b;
y=[y0;y;yL]; %pad in the left and right boundary values. Known values.
end
```

2.3 HW 3

2.3.1 Problem 1

(1) (10 pts) Consider the eigenvalue problem:

$$y'' + 2y' + \lambda^2 y = 0,$$

$$y(0) = y(1) = 0,$$

valid over the interval $0 \le x \le 1$. Find the first two eigenvalues and mode shapes for this problem using the `bvp4c` and `eig` utilites. This problem does have an analytical solution, and the results are that the eigenfunctions and associated eigenvalues are:

$$y_n(x) = A_n \exp(-x)\sin(\sqrt{\lambda_n^2 - 1}\, x) \qquad \sqrt{\lambda_n^2 - 1} = n\pi$$

Figure 2.1: problem 1 description

The ODE is

$$y'' + 2y' + \lambda^2 y = 0$$

With $y(0) = y(1) = 0$. The first step is to find the state space representation. Let $x_1 = y, x_2 = y'$. Taking derivatives gives

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -2x_2 - \lambda^2 x_1$$

The above is used with bvp4c as shown in the source code. To use eig, the problem is converted to the form $Ay = \alpha By$ and then Matlab $eig(A, B)$ is used to find the eigenvalues. Using second order centered difference gives

$$\left.\frac{dy}{dx}\right|_i = \frac{y_{i+1} - y_{i-1}}{2h}$$
$$\left.\frac{d^2y}{dx^2}\right|_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

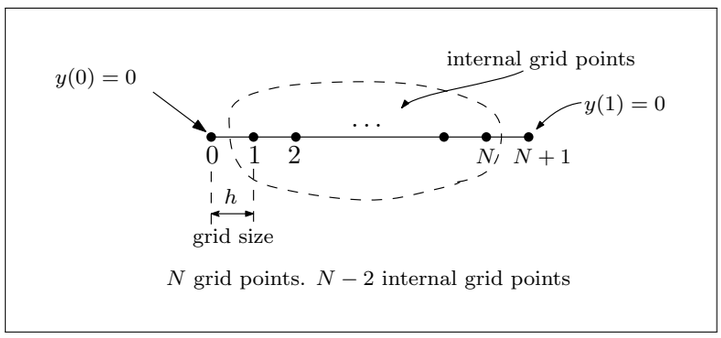


Figure 2.2: grid numbering used in problem 1

Therefore, the approximation to the differential equation at grid i (on the internal nodes as shown in the above diagram) is

$$\left.y'' + 2y' + \lambda^2 y\right|_i \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + 2\frac{y_{i+1} - y_{i-1}}{2h} + \lambda^2 y_i$$

Hence

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + 2\frac{y_{i+1} - y_{i-1}}{2h} + \lambda^2 y_i = 0$$
$$y_{i+1} - 2y_i + y_{i-1} + h\left(y_{i+1} - y_{i-1}\right) + h^2\lambda^2 y_i = 0$$
$$y_{i-1}(1 - h) + y_i\left(h^2\lambda^2 - 2\right) + y_{i+1}(1 + h) = 0$$

At node $i = 1$,

$$y_0(1-h) + y_1\left(h^2\lambda^2 - 2\right) + y_2(1+h) = 0$$

Moving the known quantities and any quantity with λ to the right side

$$-2y_1 + y_2(1+h) = y_0(h-1) - y_1\left(h^2\lambda^2\right)$$

At node $i = 2$

$$y_1(1-h) + y_2\left(h^2\lambda^2 - 2\right) + y_3(1+h) = 0$$
$$y_1(1-h) - 2y_2 + y_3(1+h) = -y_2\left(h^2\lambda^2\right)$$

And so on. At the last node, $i = N$

$$y_{N-1}(1-h) + y_N\left(h^2\lambda^2 - 2\right) + y_{N+1}(1+h) = 0$$
$$y_{N-1}(1-h) - 2y_N = -y_N\left(h^2\lambda^2\right) - y_{N+1}(1+h)$$

At $i = N-1$

$$y_{N-2}(1-h) + y_{N-1}\left(h^2\lambda^2 - 2\right) + y_N(1+h) = 0$$
$$y_{N-2}(1-h) - 2y_{N-1} + y_N(1+h) = -y_{N-1}\left(h^2\lambda^2\right)$$

Hence the structure is

$$\begin{bmatrix} -2 & 1+h & 0 & 0 & 0 & \cdots & 0 \\ 1-h & -2 & 1+h & 0 & 0 & \cdots & \vdots \\ 0 & 1-h & -2 & 1+h & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & 1-h & -2 & 1+h \\ 0 & \cdots & \cdots & \cdots & 0 & 1-h & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} y_0(h-1) \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -y_{N+1}(1+h) \end{bmatrix} - h^2\lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

Since $y_0 = y_{N+1} = 0$ the above reduces to

$$\begin{bmatrix} -2 & 1+h & 0 & 0 & 0 & \cdots & 0 \\ 1-h & -2 & 1+h & 0 & 0 & \cdots & \vdots \\ 0 & 1-h & -2 & 1+h & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & 1-h & -2 & 1+h \\ 0 & \cdots & \cdots & \cdots & 0 & 1-h & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} = -h^2\lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$Ay = \alpha By$$

Where $\alpha = \lambda^2$ and $B = -h^2\boldsymbol{I}$. The above is now implemented in Matlab and eig is used to find α.

The analytical value of the eigenvalue is given from

$$\sqrt{\lambda_n^2 - 1} = n\pi$$
$$\lambda_n = \sqrt{(n\pi)^2 + 1}$$

Hence the first three eigenvalues are

$$\lambda_1 = \sqrt{\pi^2 + 1} = 3.2969$$
$$\lambda_2 = \sqrt{(2\pi)^2 + 1} = 6.3623$$
$$\lambda_3 = \sqrt{(3\pi)^2 + 1} = 9.4777$$

And the corresponding analytical mode shapes, using $A_n = 1$ when normalized is

$$y_1(x) = e^{-x}\sin(\pi x)$$
$$y_2(x) = e^{-x}\sin(2\pi x)$$

These are used to compare the numerical solutions from bvp4c and from eig against. The following plots show the result for the first three eigenvalues and eigenfunctions found. The main difficulty with using bvp4c for solving the eigenvalue problem is on deciding which guess λ to use for each mode shape to solve for. The first three mode shapes are solved for, and also a plot of the initial mode shape guess passed to bvp4c is plotted. Using large grid size, the solution by eig and bvp4c matched very well as can be seen from the plots below. The eigenvalue produced by bvp4c was little closer to the analytical one than

the eigenvalue produced by eig() command.

2.3.1.1 Results

Each mode shape plot is given, showing the eigenvalue produced by each solver and the initial mode shape guess used. There are 3 plots, one for each mode shape. The first, second and third. (the problem asked for only the first two mode shapes, but the third one was added for verification).

1. First mode shape

Table 2.1: First eigenvalue

Solver	eigenvalue found
analytical	$\lambda_1 = \sqrt{\pi^2 + 1} = 3.296\,9$
bvp4c	3.2969
eig	3.2960997

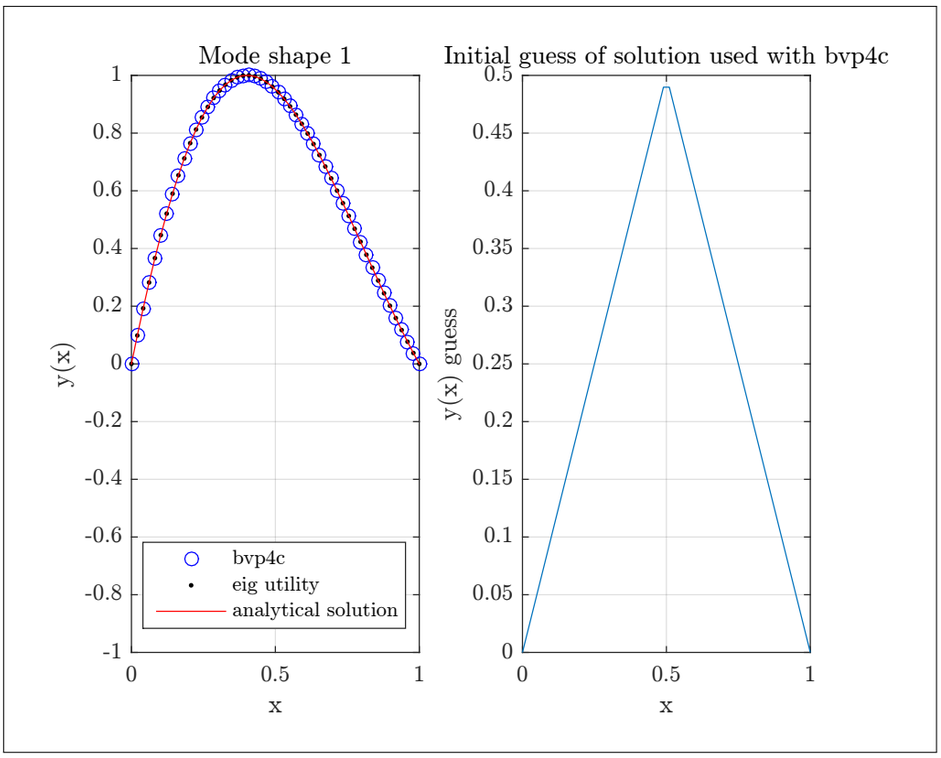


Figure 2.3: First mode shape

2. Second mode shape

Table 2.2: second eigenvalue

Solver	eigenvalue found
analytical	$\lambda_1 = \sqrt{(2\pi)^2 + 1} = 6.3623$
bvp4c	6.3622
eig	6.35738

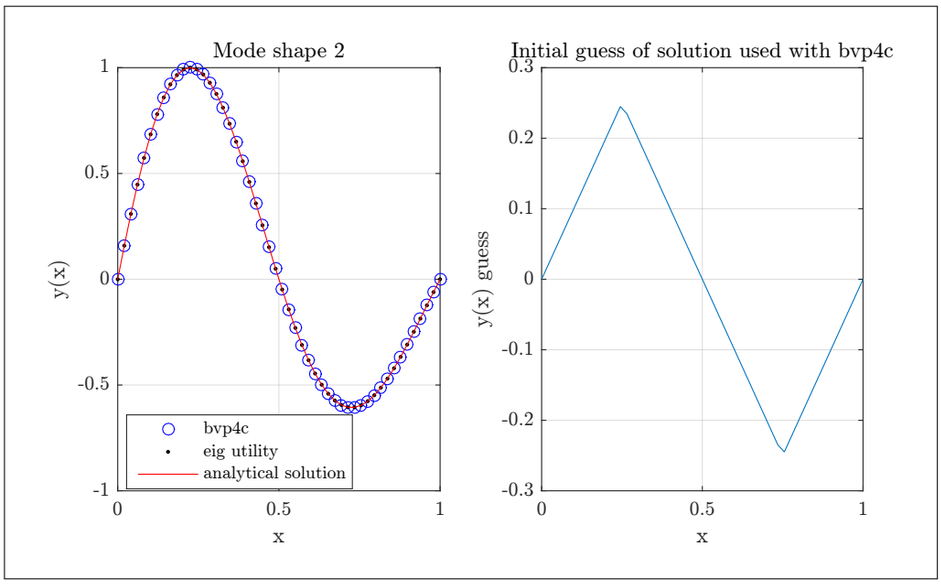


Figure 2.4: Second mode shape

3. Third mode shape

Table 2.3: Third eigenvalue

Solver	eigenvalue found
analytical	$\lambda_1 = \sqrt{(3\pi)^2 + 1} = 9.4777$
bvp4c	9.4777
eig	9.4623

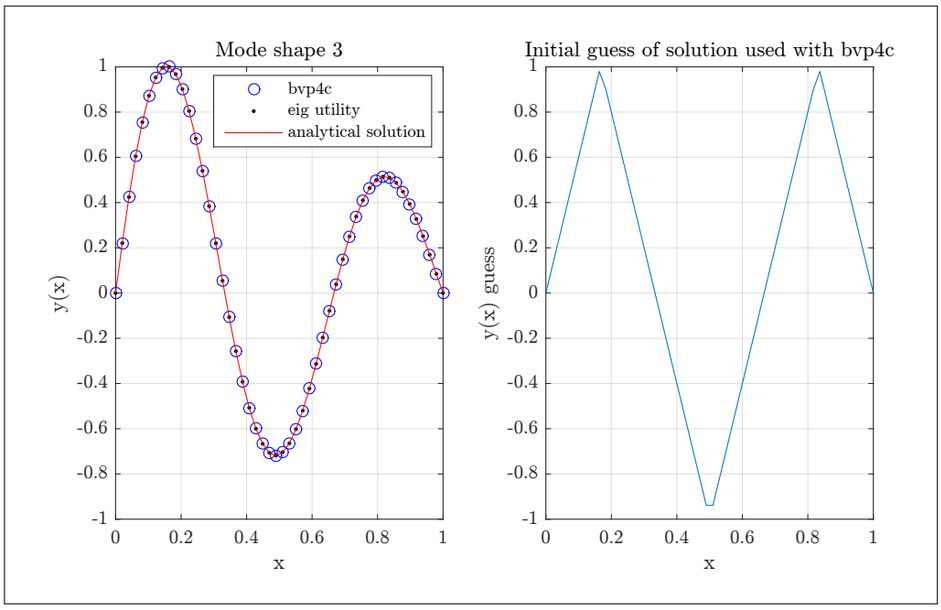


Figure 2.5: Third mode shape

Printout of Matlab console running the program

```
>>nma_HW3_EMA_471_problem_1
**********************
running mode 1. Eigenvalue, obtained with bvp4c, is 3.2968962.
eigenvalue from eig is 3.2960997
**********************
running mode 2. Eigenvalue, obtained with bvp4c, is 6.3622025.
eigenvalue from eig is 6.3573774
**********************
running mode 3. Eigenvalue, obtained with bvp4c, is 9.4777434.
eigenvalue from eig is 9.4622719
```

2.3.1.2 Source code

```
function  nma_HW3_EMA_471_problem_1()
% Solves y''+2 y' + lam^2 y = 0
%
% see HW3, EMA 471
% by Nasser M. Abbasi
%
clc; close all;
initialize();   %GUI
N  = 50;  %number of grid points. Smaller will also work.
x  = linspace(0,1,N); %all grid used is based on this one same grid.

guess_lambda_for_bvp4c = [3,6,9];%guess eigenvalue for bvp4c only

%look at first 3 mode shapes (one more than asked for, to verify)
for mode_shape = 1:3
    make_test(mode_shape, x,guess_lambda_for_bvp4c(mode_shape), N);
end

end
%=============================================%
function make_test(mode_shape_number, x, guess_lambda, N)

y_bvp4c    = get_y_bvp4c(x, guess_lambda, mode_shape_number);
y_eig      = get_eigenvector_matlab_eig(x, N, mode_shape_number);
y_analytic = get_y_analytic(x, mode_shape_number);

%done. Plot all mode shapes
plot_result(x, y_bvp4c, y_eig, y_analytic, mode_shape_number);
end
%=========================================
%This is the bvp4c solver only
function y_bvp4c = get_y_bvp4c(x, guess_lambda, mode_shape_number)

initial_solution = bvpinit(x,@set_initial_mode_shape,guess_lambda);
y_bvp4c          = bvp4c(@rhs, @bc, initial_solution);
value            = y_bvp4c.parameters;

fprintf('\n*********************\n');
fprintf(['running mode %d. Eigenvalue, obtained', ...
         'with bvp4c, is %9.7f.\n'],...
          mode_shape_number, value)

y_bvp4c = deval(y_bvp4c,x);    %interpolate on our own grid
y_bvp4c = y_bvp4c(1,:);
y_bvp4c = y_bvp4c/max(y_bvp4c); %normalize

    %-----------------------------------
    %  internal function
    %  This defines the initial guess for the eigenvector
    %  the fundamental mode shape is a sawtooth
    function solinit = set_initial_mode_shape(x)
        switch mode_shape_number
            case 1
                if x <= 0.5
                    f = x;
                    fp = 1;
                else
                    f = 1 - x;
                    fp = -1;
                end
            case 2
                if x <= 0.25
```

```
                    f = x;
                    fp = 1;
                elseif x > 0.25 && x <= 0.75
                    f = 0.5 - x;
                    fp = -1;
                else
                    f = x - 1;
                    fp = 1;
                end
            case 3
                h = 1/6;
                if x<=h
                    f=1/h*x;
                    fp=1/h;
                elseif x>h&&x<=3*h
                    f=2-x/h;
                    fp=-1/h;
                elseif x>3*h&&x<(5*h)
                    f=(-4+1/h*x);
                    fp=1/h;
                elseif x>5*h
                    f=(6-x/h);
                    fp=-1/h;
                end
        end
        solinit = [ f ; fp ];
    end
    %---------------------------------------
    % internal function. sets up the RHS of the
    %state space for bvp4c
    % similar to ode45 RHS
    function f = rhs(~,x,lam)

        x1 = x(2);
        x2 = -2*x(2)-lam^2*x(1);
        f = [ x1
            x2];
    end
    %-------------------------------------
    % Internal function. sets up the boundary
    % conditions vector. Must have ~
    % above in third arg
    function res = bc(ya,yb,~)
        res = [ ya(1)
            yb(1)
            ya(2)-1
            ];
    end
end
%=====================================================%
%This is the solver using Matlab eig
function y_eig = get_eigenvector_matlab_eig(x, N, mode_shape)

h   = x(2)-x(1);   % find grid spacing to set up A for eig() use
A   = setup_A_matrix(h,N-2);
B   = -eye(N-2)*h^2;

[eig_vec,eig_values] = eig(A,B);   %eigenvalue/vector from matlab
eig_values           = diag(eig_values);
sorted_eig_values    = sort(eig_values);     %sort them

%now match the original positiion of the eigenvalue with its
%correspoding eigenvectr. Hence find the index of
```

```
%correct eigevalue so use to index to eigenvector
found_eig_vector = eig_vec(:,eig_values == ...
                           sorted_eig_values(mode_shape));

%Set the sign correctly
if found_eig_vector(1) > 0
    y_eig = [0 ;  found_eig_vector ; 0];
else
    y_eig = [0 ; -found_eig_vector ; 0];
end

y_eig = y_eig/max(y_eig);  %normalize

fprintf('eigenvalue from eig is %9.7f\n',...
        sqrt(sorted_eig_values(mode_shape)));
   %---------------------------------------
   %Internal function, sets up the A matrix for use
   %for the eig() method
    function A = setup_A_matrix(h,N)
        A      = zeros(N);
        A(1,1) = -2;
        A(1,2) = 1+h;
        for i = 2:N-1
            A(i,i-1:i+1) = [1-h,-2,1+h];
        end
        A(N,N)   = -2;
        A(N,N-1) = 1-h;
    end
end
%=======================================
function y_analytic = get_y_analytic(x, mode_shape)
% This is the known analytical solution. From problem statement

y_analytic  = exp(-x) .* sin(mode_shape*pi*x);
y_analytic  = y_analytic / max(y_analytic);   %normalize
end

%=====================================================
%This function just plots the eigenshapes found from all solvers
function plot_result(x, y_bvp4c, y_eig, y_analytic,mode_shape)
figure();
subplot(1,2,1);
plot(x,y_bvp4c,'bo',...
    x,y_eig,'k.',...
    x,y_analytic,'r')
axis([0 1 -1 1]);
title(sprintf('Mode shape %d',mode_shape));
xlabel('x')
ylabel('y(x)')
legend('bvp4c','eig utility','analytical solution',...
        'Location','southwest')
grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(1,2,2);
initial_mode_shape = set_initial_mode_shape_plot(x, mode_shape);
plot(x,initial_mode_shape);
grid;
title('Initial guess of solution used with bvp4c');
xlabel('x'); ylabel('y(x) guess');
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

    %-----------------------------------
```

```
    %Internal function. To display guess mode shape for plotting
    function f = set_initial_mode_shape_plot(x, mode_shape)
        %  Internal function.
        %  plots the initial mode shape guess used.
        %
        switch mode_shape
            case 1
                f  = x.*(x<=0.5)+(1-x).*(x>0.5);
            case 2
                f  = x.*(x<=0.25)+(0.5-x).*(x>0.25&x<=0.75)+...
                     (x-1).*(x>0.75);
            case 3
                h = 1/6;
                f  = 1/h*x.*(x<=h)+(2-x/h).*(x>h&x<=3*h)+...
                    (-4+1/h*x).*(x>3*h&x<(5*h))+(6-x/h).*(x>5*h);
        end
    end
end
%=========================================================%
function initialize()
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');
end
```

2.3.2 Problem 2

(2) (15 pts) An axial load P is applied to a column of circular cross-section with linear taper, so that

$$I(x) = I_o\left(\frac{x}{b}\right)^4$$

where x is measured from the point at which the column would taper to a point if it were extended and I_o is the value of I at the end $x = b$. If the column is hinged at ends $x = a$ and $x = b$, the governing equation can be put in the form:

$$x^4\frac{d^2y}{dx^2} + \mu^2 y = 0, \qquad \mu^2 \equiv \frac{Pb^4}{EI_o}, \qquad y(a) = y(b) = 0$$

If we write the governing equation in terms of the dimensionless variable $z = x/l$, where $l = b - a$ is the length of the column, the result is:

$$z^4\frac{d^2y}{dz^2} + \lambda^2 y = 0, \qquad \lambda^2 = \frac{\mu^2}{l^2} = \frac{Pb^4}{El^2I_o}, \qquad y(a/l) = y(b/l) = 0$$

This latter form is preferable in that the independent variable z and the eigenvalue λ are both dimensionless. For the specific case $a = 3$ m, $b = 6$ m, $E = 1$ GPa, and circular cross-section radii $r_a = 10$ cm and $r_b = 20$ cm, find the critical buckling load and buckled shape for this column. Use all three methods we discussed (`bvp4c`, `eig`, power iteration) to verify your results. Compare your results to analytical solutions to the critical load and buckled shape, expressed as:

$$P_n = n^2\pi^2\left(\frac{a}{b}\right)^2\frac{EI_o}{l^2} \qquad y_n(z) = A_n z\sin\left[n\pi\frac{b}{l}\left(1-\frac{a}{lz}\right)\right]$$

Figure 2.6: problem 2 description

The geometry of the problem is as follows

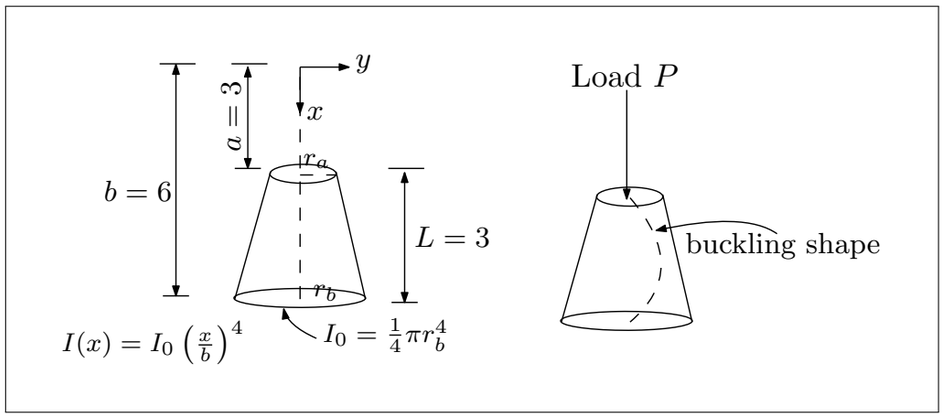


Figure 2.7: problem 2 geometry

Using the normalized ODE

$$z^4 y'' + \lambda^2 y = 0$$

With BC

$$y\left(\frac{a}{L}\right) = y\left(\frac{3}{3}\right) = y(1) = 0$$
$$y\left(\frac{b}{L}\right) = y\left(\frac{6}{3}\right) = y(2) = 0$$

And

$$\lambda^2 = \frac{Pb^4}{EL^2 I_0}$$

For domain $1 \le z \le 2$. The analytical solution is $P_n = n^2\pi^2 \left(\frac{a}{b}\right)^2 \frac{EI_0}{L^2}$ and $y_n(z) = A_n z \sin\left(n\pi \frac{b}{L}\left(1 - \frac{a}{Lz}\right)\right)$. The first step is to convert the ODE into state space for use with bvp4c. Let $x_1 = y, x_2 = y'$. Taking derivatives gives

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\frac{\lambda^2}{z^4} x_1$$

For using eig, the problem needs to discretized first. The following shows the grid used

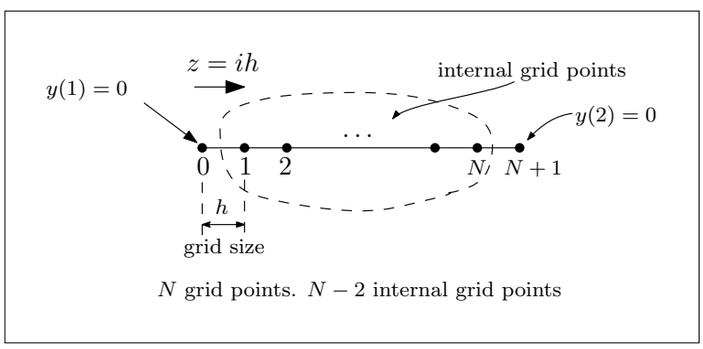


Figure 2.8: Grid used for problem 2

The grid starts at $z = 1$ and ends at $z = 2$ since this is the domain of the differential equation being solved. Therefore $i = 0$ corresponds to the left boundary conditions which is $y(z = 1) = 0$ and $i = (N + 2)h$ corresponds to the right boundary conditions which is $y(z = 2) = 0$.

Using second order centered difference gives

$$\left.\frac{d^2y}{dx^2}\right|_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

Therefore, the approximation to the differential equation at grid i (on the internal nodes as shown in the above diagram) is as follows. Notice we needed to add 1 to the grid spacing since the left boundary starts at $z=1$ in this case and not at $z=0$ as normally the case in other problems.

$$z^4 y'' + \lambda^2 y\Big|_i \approx (1+ih)^4 \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \lambda^2 y_i$$

Hence

$$(1+ih)^4 \left(y_{i+1} - 2y_i + y_{i-1} \right) + h^2 \lambda^2 y_i = 0$$

$$(1+ih)^4 y_{i+1} - 2(1+ih)^4 y_i + (1+ih)^4 y_{i-1} = -h^2 \lambda^2 y_i$$

At node $i=1$,

$$(1+h)^4 y_2 - 2(1+h)^4 y_1 + (1+h)^4 y_0 = -h^2 \lambda^2 y_1$$

Moving the known quantities to the right side

$$(1+h)^4 y_2 - 2(1+h)^4 y_1 = -(1+h)^4 y_0 - h^2 \lambda^2 y_1$$

At node $i=2$

$$(1+2h)^4 y_3 - 2(1+2h)^4 y_2 + (1+2h)^4 y_1 = -h^2 \lambda^2 y_2$$

And so on. At the last node, $i=N$

$$(1+Nh)^4 y_{N+1} - 2(1+Nh)^4 y_N + (1+Nh)^4 y_{N-1} = -h^2 \lambda^2 y_N$$

$$-2(Nh)^4 y_N + (Nh)^4 y_{N-1} = -(1+Nh)^4 y_{N+1} - h^2 \lambda^2 y_N$$

At $i=N-1$

$$(1+(N-1)h)^4 y_N - 2(1+(N-1)h)^4 y_{N-1} + (1+(N-1)h)^4 y_{N-2} = -h^2 \lambda^2 y_{(N-1)}$$

Hence the structure is

$$\begin{bmatrix} -2(1+h)^4 & (1+h)^4 & 0 & 0 & 0 & \cdots & 0 \\ (1+2h)^4 & -2(1+2h)^4 & (1+2h)^4 & 0 & 0 & \cdots & \vdots \\ 0 & (1+3h)^4 & -2(1+3h)^4 & (1+3h)^4 & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & (1+(N-1)h)^4 & -2(1+(N-1)h)^4 & (1+(N-1)h)^4 \\ 0 & \cdots & \cdots & \cdots & 0 & (1+Nh)^4 & -2(1+Nh)^4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} =$$

$$= \begin{bmatrix} -(1+h)^4 y_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -(1+Nh)^4 y_{N+1} \end{bmatrix} - h^2 \lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

Since $y_0 = y_{N+1} = 0$ the above reduces to

$$\begin{bmatrix} -2(1+h)^4 & (1+h)^4 & 0 & 0 & 0 & \cdots & 0 \\ (1+2h)^4 & -2(1+2h)^4 & (1+2h)^4 & 0 & 0 & \cdots & \vdots \\ 0 & (1+3h)^4 & -2(1+3h)^4 & (1+3h)^4 & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & (1+(N-1)h)^4 & -2(1+(N-1)h)^4 & (1+(N-1)h)^4 \\ 0 & \cdots & \cdots & \cdots & 0 & (1+Nh)^4 & -2(1+Nh)^4 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$= -h^2\lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$Ay = \alpha By$$

Where $\alpha = \lambda^2$ and $B = -h^2\boldsymbol{I}$. The above is implemented in Matlab and eig is used to find α.

The analytical value of the eigenvalue is given from

$$\lambda_n = \sqrt{\frac{P_n b^4}{EL^2 I_0}}$$

Where

$$P_n = n^2\pi^2 \left(\frac{a}{b}\right)^2 \frac{EI_0}{L^2}$$

Hence

$$\lambda_n = \sqrt{\frac{n^2\pi^2 \left(\frac{a}{b}\right)^2 \frac{EI_0}{L^2} b^4}{EL^2 I_0}} = \sqrt{\frac{n^2\pi^2 a^2 b^2}{L^4}}$$

Using $a = 3, b = 6, L = 3$,the first three eigenvalues are

$$\lambda_1 = \sqrt{\frac{\pi^2\left(3^2\right)\left(6^2\right)}{\left(3^4\right)}} = 6.2832$$

$$\lambda_2 = \sqrt{\frac{2^2\pi^2\left(3^2\right)\left(6^2\right)}{\left(3^4\right)}} = 12.566$$

$$\lambda_3 = \sqrt{\frac{3^2\pi^2\left(3^2\right)\left(6^2\right)}{\left(3^4\right)}} = 18.850$$

And the corresponding analytical mode shapes, using $A_n = 1$ when normalized is

$$y_1(z) = z\sin\left(\pi\frac{b}{L}\left(1 - \frac{a}{Lz}\right)\right) = z\sin\left(2\pi\left(1 - \frac{1}{z}\right)\right)$$

$$y_2(x) = z\sin\left(2\pi\frac{b}{L}\left(1 - \frac{a}{Lz}\right)\right) = z\sin\left(4\pi\left(1 - \frac{1}{z}\right)\right)$$

$$y_3(x) = z\sin\left(3\pi\frac{b}{L}\left(1 - \frac{a}{Lz}\right)\right) = z\sin\left(6\pi\left(1 - \frac{1}{z}\right)\right)$$

And the corresponding buckling loads at each mode shape are, using $E = 10^9$Pa, $I_0 = \frac{1}{4}\pi\left(r_b\right)^4$ where $r_b = 0.2$ meter are

$$P_n = n^2\pi^2\left(\frac{a}{b}\right)^2\frac{EI_0}{L^2} = n^2\pi^2\left(\frac{3}{6}\right)^2\frac{\left(10^9\right)\frac{1}{4}\pi\left(0.2\right)^4}{\left(3^2\right)} = n^2\left(3.4451\times10^5\right)\ \text{N}$$

Hence

$$P_1 = 3.4451\times10^5\ \text{N}$$

$$P_2 = 4\left(3.4451\times10^5\right) = 1.3781\times10^6\ \text{N}$$

$$P_3 = 9\left(3.4451\times10^5\right) = 3.1006\times10^6\ \text{N}$$

These are used to compare the numerical solutions from bvp4c, eig and power method against. (for power method, only the lowest eigenvalue is obtained). For the numerical computation of P_n, after finding the numerical eigenvalue λ_n, then P_n is found from

$$P_n = \frac{\lambda_n^2 EL^2 I_0}{b^4}$$

And the values obtained are compared to the analytical P_n. The following plots show the result for the first three eigenvalues and eigenfunctions found.

2.3.2.1 Power method

For the power method, the A matrix is setup a little different than with the above eig method. Starting from

$$z^4 y'' + \lambda^2 y\Big|_i \approx (1+ih)^4 \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \lambda^2 y_i$$

Hence

$$(1+ih)^4 \left(y_{i+1} - 2y_i + y_{i-1}\right) + h^2\lambda^2 y_i = 0$$

$$\frac{-(1+ih)^4 y_{i+1} + 2(1+ih)^4 y_i - (1+ih)^4 y_{i-1}}{h^2} = \lambda^2 y_i$$

At node $i = 1$,

$$\frac{-(1+h)^4 y_2 + 2(1+h)^4 y_1 - (1+h)^4 y_0}{h^2} = \lambda^2 y_1$$

Since $y_0 = 0$ then

$$\frac{-(1+h)^4 y_2 + 2(1+h)^4 y_1}{h^2} = \lambda^2 y_1$$

At node $i = 2$

$$\frac{-(1+2h)^4 y_3 + 2(1+2h)^4 y_2 - (1+2h)^4 y_1}{h^2} = \lambda^2 y_2$$

And so on. At the last node, $i = N$

$$\frac{-(1+Nh)^4 y_{N+1} + 2(1+Nh)^4 y_N - (1+Nh)^4 y_{N-1}}{h^2} = \lambda^2 y_N$$

Since $y_{N+1} = 0$ then

$$\frac{2(1+Nh)^4 y_N - (1+Nh)^4 y_{N-1}}{h^2} = \lambda^2 y_N$$

At $i = N-1$

$$\frac{-(1+(N-1)h)^4 y_N + 2(1+(N-1)h)^4 y_{(N-1)} - (1+(N-1)h)^4 y_{N-2}}{h^2} = \lambda^2 y_{(N-1)}$$

Hence the structure is

$$\begin{bmatrix} \frac{2(1+h)^4}{h^2} & \frac{-(1+h)^4}{h^2} & 0 & 0 & 0 & \cdots & 0 \\ \frac{-(1+2h)^4}{h^2} & \frac{2(1+2h)^4}{h^2} & \frac{-(1+2h)^4}{h^2} & 0 & 0 & \cdots & \vdots \\ 0 & \frac{-(1+3h)^4}{h^2} & \frac{2(1+3h)^4}{h^2} & \frac{-(1+3h)^4}{h^2} & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \frac{-(1+(N-1)h)^4}{h^2} & \frac{2(1+(N-1)h)^4}{h^2} & \frac{-(1+(N-1)h)^4}{h^2} \\ 0 & \cdots & \cdots & \cdots & 0 & \frac{-(1+Nh)^4}{h^2} & \frac{2(1+Nh)^4}{h^2} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix} = \lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{N-2} \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$Ay = \lambda^2 y$$

The above structure is now used to solve for lowest eigenvalue and corresponding eigenvector.

2.3.2.2 Results

Each mode shape plot is given, showing the eigenvalue produced by each solver and the initial mode shape guess used. There are 3 plots, one for each mode shape. The first, second and third. (the problem asked for only the first mode shape, but the second and

third were added for verification). For power method, only the lowest eigenvalue and corresponding eigenvector are found.

1. First mode shape

Table 2.4: First (smallest) eigenvalue λ_1

Solver	eigenvalue found λ_n	Corresponding Critical load P_n (N)
analytical	6.2817063	344352.012
bvp4c	6.2821629	344402.076
Matlab eig	6.2817063	344352.012
Power method	6.2817055	344351.929

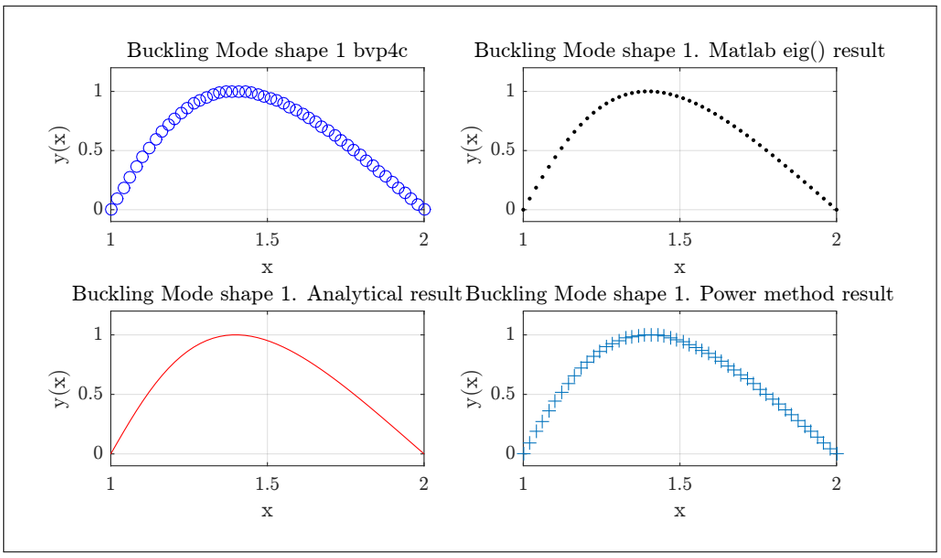


Figure 2.9: First mode shape, each solver on separate plot

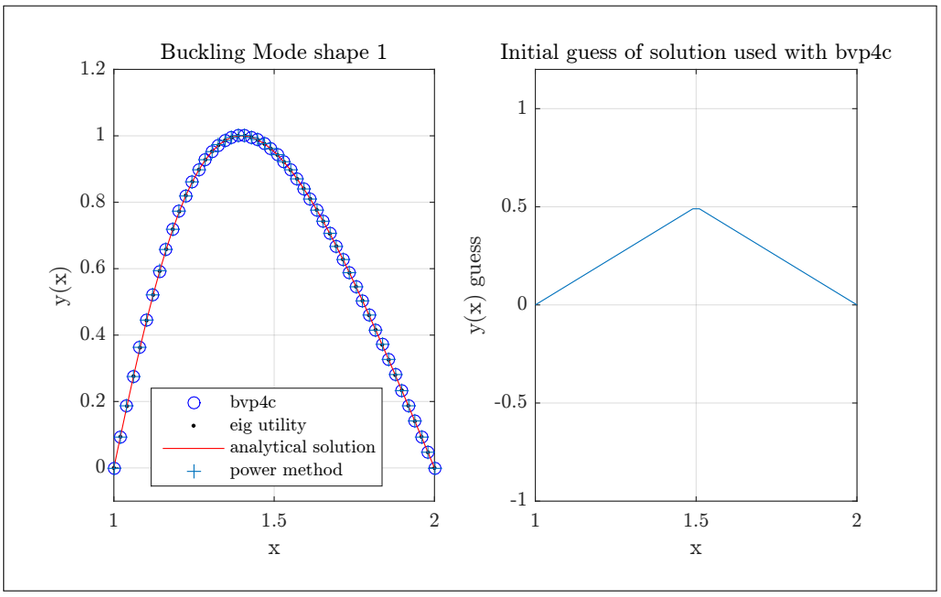


Figure 2.10: First mode shape, combined plot

2. Second mode shape

Table 2.5: second eigenvalue

Solver	eigenvalue found λ_n	Corresponding Critical load P_n (N)
analytical	12.5663706	1378056.741
bvp4c	12.5663983	1378062.820
eig	12.5534143	1375216.578

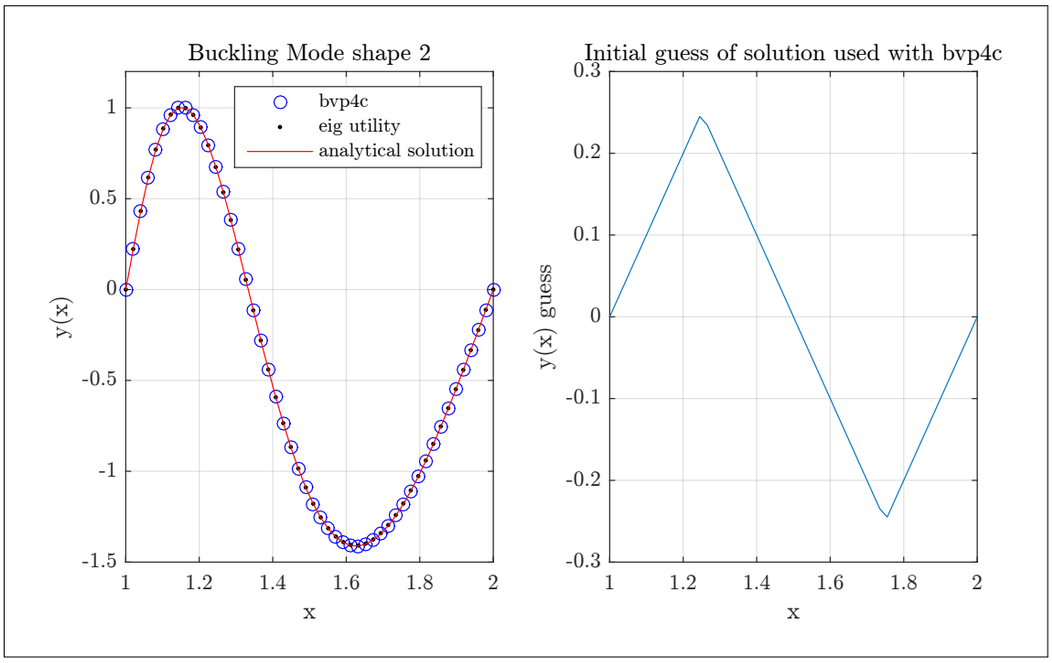


Figure 2.11: Second mode shape

3. Third mode shape

Table 2.6: Third eigenvalue

Solver	eigenvalue found λ_n	Corresponding Critical load P_n (N)
analytical	18.8495559	3100627.668
bvp4c	18.8499237	3100748.676
eig	18.80506	3086006.365

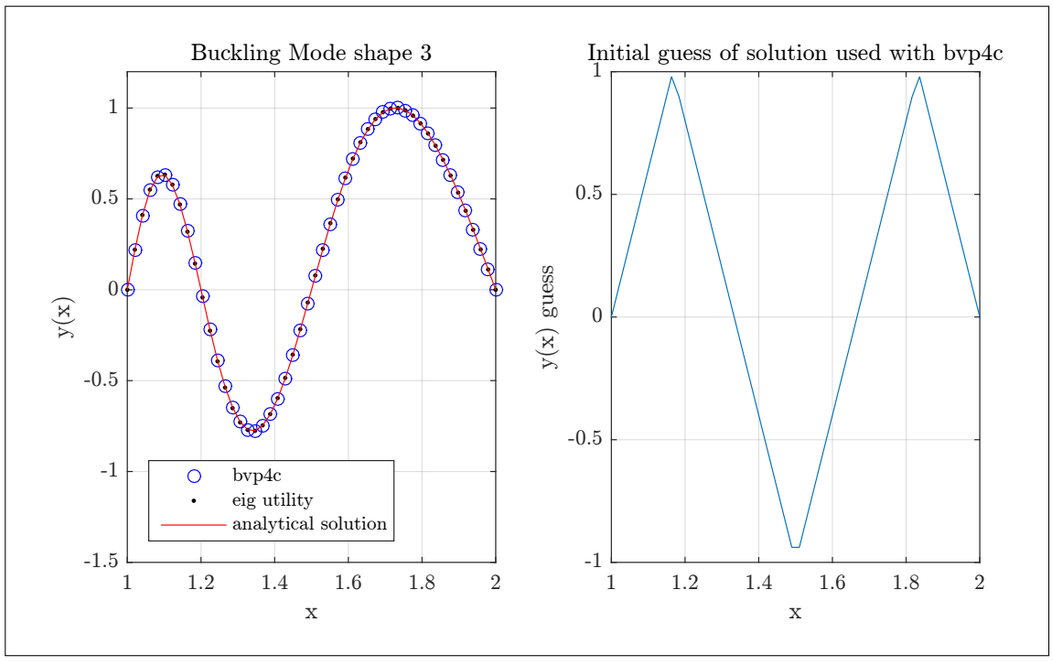


Figure 2.12: Third mode shape

Printout of Matlab console running the program

```
>>nma_HW3_EMA_471_problem_2
**********************
running mode 1
Eigenvalue obtained with bvp4c, is 6.2821629
Critical load is 344402.076.
eigenvalue from eig is 6.2817063
Critical load is 344352.012.
eigenvalue from analytical is 6.2831853
critical load from analytical is 344514.185
eigenvalue obtained with the power iteration method 6.2817055
Critical load is 344351.929.
**********************
running mode 2
Eigenvalue obtained with bvp4c, is 12.5663983
Critical load is 1378062.820.
eigenvalue from eig is 12.5534143
Critical load is 1375216.578.
eigenvalue from analytical is 12.5663706
critical load from analytical is 1378056.741
**********************
running mode 3
Eigenvalue obtained with bvp4c, is 18.8499237
Critical load is 3100748.676.
eigenvalue from eig is 18.8050600
Critical load is 3086006.365.
eigenvalue from analytical is 18.8495559
critical load from analytical is 3100627.668
```

2.3.2.3 Source code

```
function  nma_HW3_EMA_471_problem_2()
% Solves z^4 y''+lam^2 y = 0
%
% see HW3, EMA 471, Spring 2016
% by Nasser M. Abbasi
%
clc; close all; initialize();

%look at first 3 mode shapes (one more than asked for,
%in order to verify)
N            = 50;  %number of grid points.

```

```
%domain of problem, in normalized z-space.
x            = linspace(1,2,N);

%these are guess values for lambda for bvp4c only
guess_lambda = [6,12,18];

% try three mode shapes
for k = 1:3
    process(k, x, guess_lambda(k), N);
end

end
%=============================================================
%Main process function. Calls all solvers and call
%the main plot function
function process(mode_shape_number, x, guess_lambda, N)

  y_bvp4c    = get_y_bvp4c(x, guess_lambda, mode_shape_number);
  y_eig      = get_eigenvector_matlab_eig(x,N-2,mode_shape_number);
  y_analytic = get_y_analytic(x, mode_shape_number);

 %power method only for lowest eigenvalue
  if mode_shape_number==1
     y_power    = get_y_power(x,N-2);
     plot_result_1(x, y_bvp4c, y_eig, y_analytic, ...
                   y_power, mode_shape_number);
  else
     plot_result_2(x, y_bvp4c, y_eig, ...
                   y_analytic, mode_shape_number);
  end

end
%=====================================================================%
%This function finds the eigenvalue and eigenvector
%using Matlab eig()
function y_eig = get_eigenvector_matlab_eig(x,N,mode_shape_number)

h                       = x(2)-x(1);  % find grid spacing
A                       = setup_A_matrix(h,N);
B                       = -eye(N)*h^2;
[eig_vector,eig_values] = eig(A,B);
eig_values              = diag(eig_values); %they are on diagonal
sorted_eig_values       = sort(eig_values); %sort, small->large

%now need to match the original position of the
%eigenvalue with its correspoding eigenvectr. Hence find the
%index of correct eigevalue to use as index to eigenvector
found_eig_vector = eig_vector(:,...
            eig_values == sorted_eig_values(mode_shape_number));

%Set is sign correctly
if found_eig_vector(1) > 0
    y_eig = [0 ;  found_eig_vector ; 0];
else
    y_eig = [0 ; -found_eig_vector ; 0];
end

y_eig     = y_eig/max(y_eig); %normalize

%normalize eigevalues
sorted_eig_values = sqrt(sorted_eig_values)/pi;

fprintf('eigenvalue from eig is %9.7f\n',...
```

```
        sorted_eig_values(mode_shape_number)*pi);

calculate_critial_load(sorted_eig_values(mode_shape_number)*pi);

    %------------------------------%
    function A = setup_A_matrix(h,N)
        A      = zeros(N);
        A(1,1) = -2*(1+h)^4;
        A(1,2) = (1+h)^4;
        for i = 2:N-1
            A(i,i-1:i+1) = [(1+i*h)^4,-2*(1+i*h)^4,(1+i*h)^4];
        end
        A(N,N)   = -2*(1+N*h)^4;
        A(N,N-1) = (1+N*h)^4;
    end
end
%=================================================================%
function y = get_y_power(x,N)

h                       = x(2)-x(1);  % find grid spacing
A                       = setup_A_matrix_for_power(h,N);
A_inv                   = setup_A_inv_matrix_for_power(N);

% Starting guess for the eigenvector. Use unit vector
y = ones(N,1);

% This below from EX 11, applied it here:
% set tolerance; "while" loop will run until there is
%no difference between old and new estimates for eigenvalues
%to within the tolerance

tol              = 1e-6;
eigenvalue_1_old = 0;
eigenvalue_1_new = 1;

while abs(eigenvalue_1_new - eigenvalue_1_old)/abs(eigenvalue_1_new) > tol
    y_new = A_inv*y;    % generate updated value for eigenvector
    eigenvalue_1_old = eigenvalue_1_new; % update old eigenvalue
    eigenvalue_1_new = max(y_new);       % update new eigenvalue
    y = y_new/eigenvalue_1_new; % renormalize eigenvector estimate
end

y = [0;y;0];
y = y/max(y);   %normalize

% Taken Per EX 11:
%   add boundary conditions to complete eigenvector; also
%   note that we have found the largest value of the inverse
%   of what we're looking for, so...

% the lambda we're seeking is actually the
% inverse of the square root of what we've found
lam = 1/sqrt(eigenvalue_1_new);

fprintf('eigenvalue obtained with the power iteration method %9.7f\n',...
         lam);
calculate_critial_load(lam);

    %------------------------------%
    function A = setup_A_matrix_for_power(h,N)
        A      = zeros(N);
        A(1,1) = 2/h^2*(1+h)^4;
        A(1,2) = -1/h^2*(1+h)^4;
```

```
        for i = 2:N-1
            A(i,i-1:i+1) = [-1/h^2*(1+i*h)^4,2/h^2*(1+i*h)^4,...
                            -1/h^2*(1+i*h)^4];
        end
        A(N,N)   = 2/h^2*(1+N*h)^4;
        A(N,N-1) = -1/h^2*(1+N*h)^4;
    end
    %--------------------------------%
    function A_inv = setup_A_inv_matrix_for_power(N)
        %We are looking for smallest eigenvalue. Use inverse.
        A_inv = zeros(N);
        for i = 1:N
            b_rhs = zeros(N,1);
            b_rhs(i,1) = 1;
            A_inv(:,i) = A\b_rhs;
        end
    end
end

%=====================================================================%
function y_analytic = get_y_analytic(z,n)
b = 6; %meter
a = 3; %meter
L = b-a; %meter length of column

%from question statement
y_analytic     = z.*sin(n*pi*(b/L).*(1-a./(L*z)));

y_analytic     = y_analytic/max(y_analytic);  %normalize

E  = 10^9;
rb = 0.2; %meter, radius of lower section
I0 = (1/4)*pi*(rb)^4;

critical_load  = n^2*pi^2*(a/b)^2* E*I0/L^2;
lam            = sqrt(critical_load*b^4 / (E*L^2*I0) );

fprintf('eigenvalue from analytical is %9.7f\n',lam);
fprintf('critical load from analytical is %9.3f\n\n',...
         critical_load);

end

%=====================================================================%
function plot_result_1(x, y_bvp4c_normalized, y_eig, ...
                           y_analytic, ...
                           y_power, mode_shape_number)
figure();
subplot(1,2,1);
plot(x,y_bvp4c_normalized(1,:),'bo', ...
    x,y_eig,'k.', ...
    x,y_analytic,'r',...
    x,y_power,'+');
axis([1 2 -.1 1.2])
title(sprintf('Buckling Mode shape %d',mode_shape_number));
xlabel('x')
ylabel('y(x)')
legend('bvp4c','eig utility','analytical solution',....
       'power method','Location','southwest')
grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(1,2,2);
```

```
initial_mode_shape = set_initial_mode_shape_plot(x-1,...
                                                 mode_shape_number);
plot(x,initial_mode_shape); axis([1 2 -1 1.2]);
grid;
title('Initial guess of solution used with bvp4c');
xlabel('x'); ylabel('y(x) guess');
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

figure();
subplot(2,2,1);
plot(x,y_bvp4c_normalized(1,:),'bo');
title(sprintf('Buckling Mode shape %d bvp4c',mode_shape_number));
xlabel('x'); axis([1 2 -.1 1.2]);
ylabel('y(x)'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,2);
plot(x,y_eig,'k.');
title(sprintf('Buckling Mode shape %d. Matlab eig() result',...
                                                 mode_shape_number));
xlabel('x'); axis([1 2 -.1 1.2]);
ylabel('y(x)'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,3);
plot(x,y_analytic,'r');
title(sprintf('Buckling Mode shape %d. Analytical result',...
                                                 mode_shape_number));
xlabel('x'); axis([1 2 -.1 1.2]);
ylabel('y(x)'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,4);
plot(x,y_power,'+');
title(sprintf('Buckling Mode shape %d. Power method result',...
                                                 mode_shape_number));
xlabel('x'); axis([1 2 -.1 1.2]);
ylabel('y(x)'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

end

%=============================================================%
function plot_result_2(x, y_bvp4c_normalized, y_eig, ...
                       y_analytic, mode_shape_number)

figure();
subplot(1,2,1);
plot(x,y_bvp4c_normalized(1,:),'bo',...
     x,y_eig,'k.',...
     x,y_analytic,'r')

axis([1 2 -1.5 1.2]);
title(sprintf('Buckling Mode shape %d',mode_shape_number));
xlabel('x')
ylabel('y(x)')
legend('bvp4c','eig utility','analytical solution',...
        'Location','southwest')
grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(1,2,2);
initial_mode_shape = set_initial_mode_shape_plot(x-1,...
```

```
                                                mode_shape_number);
plot(x,initial_mode_shape);
grid;
title('Initial guess of solution used with bvp4c');
xlabel('x'); ylabel('y(x) guess');
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

end

%=====================================================================
function f = set_initial_mode_shape_plot(x,mode_shape_number)
        %  Internal function.
        %  plots the initial mode shape guess used.
        %
        switch mode_shape_number
            case 1
                f  = x.*(x<=0.5)+(1-x).*(x>0.5);
            case 2
                f  = x.*(x<=0.25)+(0.5-x).*(x>0.25&x<=0.75)+...
                                                (x-1).*(x>0.75);
            case 3
                h = 1/6;
                f  = 1/h*x.*(x<=h)+(2-x/h).*(x>h&x<=3*h)+...
                    (-4+1/h*x).*(x>3*h&x<(5*h))+(6-x/h).*(x>5*h);
        end
end
%=====================================================================%
function  y_bvp4c_normalized = ...
                  get_y_bvp4c(x,guess_lambda,mode_shape_number)

initial_solution = bvpinit(x,@set_initial_mode_shape,guess_lambda);
y_bvp4c          = bvp4c(@rhs,@bc,initial_solution);
value            = y_bvp4c.parameters;
fprintf('\n**********************\n');
fprintf('running mode %d\nEigenvalue obtained with bvp4c, is %9.7f\n',...
    mode_shape_number,value);
calculate_critial_load(value);

y_bvp4c            = deval(y_bvp4c,x);    %interpolate
y_bvp4c_normalized = y_bvp4c/max(y_bvp4c(1,:)); %normalize

    %--------------------------------%
    function solinit = set_initial_mode_shape(x)
        %  internal function
        %  This defines the initial guess for the eigenvector;
        %  the first guess of
        %  the fundamental mode shape is a sawtooth
        %
        switch mode_shape_number
            case 1
                if x <= 0.5
                    f = x;
                    fp = 1;
                else
                    f = 1 - x;
                    fp = -1;
                end
            case 2
                if x <= 0.25
                    f = x;
                    fp = 1;
                elseif x > 0.25 && x <= 0.75
                    f = 0.5 - x;
```

```
328                     fp = -1;
329                 else
330                     f = x - 1;
331                     fp = 1;
332                 end
333             case 3
334                 h = 1/6;
335                 if x<=h
336                     f=1/h*x;
337                     fp=1/h;
338                 elseif x>h&&x<=3*h
339                     f=2-x/h;
340                     fp=-1/h;
341                 elseif x>3*h&&x<(5*h)
342                     f=(-4+1/h*x);
343                     fp=1/h;
344                 elseif x>5*h
345                     f=(6-x/h);
346                     fp=-1/h;
347                 end
348         end
349         solinit = [ f ; fp ];
350     end
351     %-----------------------------%
352     function f = rhs(t,x,lam)
353         %This function sets up the RHS of the state space
354         %setup for this problem.
355         %similar to ode45 RHS
356 
357         x1 = x(2);
358         x2 = -lam^2*x(1)/t^4;
359         f = [ x1
360             x2];
361     end
362     %-----------------------------%
363     function res = bc(ya,yb,~)
364         %This sets up the boundary conditions vector.
365         %Must have ~ above in third agrs!
366         res = [ ya(1)
367             yb(1)
368             ya(2)-1
369             ];
370     end
371 end
372 %=========================================================%
373 function calculate_critial_load(lam)
374 
375 E  = 10^9;
376 b  = 6; %meter
377 a  = 3; %meter
378 L  = b-a; %meter length of column
379 rb = 0.2; %meter, radius of lower section
380 I0 = (1/4)*pi*(rb)^4;
381 
382 P = lam^2 * E * L^2 * I0/ b^4;
383 fprintf('Critical load is %9.3f.\n\n',P);
384 end
385 %=================================================================%
386 function initialize()
387 reset(0);
388 set(groot,'defaulttextinterpreter','Latex');
389 set(groot, 'defaultAxesTickLabelInterpreter','Latex');
390 set(groot, 'defaultLegendInterpreter','Latex');
```

```
391 end
```

2.3.3 Problem 3

(3) (15 pts) In the case of a column of uniform cross-section for which *EI* is a constant, the buckling of the column due to its own weight, given one end free and the other built-in, can be written in terms of rotation θ as:

$$\frac{d^2\theta}{dz^2} + \lambda^2 z\theta = 0, \qquad \lambda^2 \equiv \frac{\rho g A l^3}{EI}, \qquad \theta'(0) = \theta(1) = 0$$

Here again the problem has been written in terms of the dimensionless length $z = x/l$ and the eigenvalue λ is dimensionless. Given a uniform cross-section of 1 cm diameter bar, mass density 7500 kg/m^3 and modulus E = 100 GPa, what is the limiting height that causes the bar to buckle under its own weight? As with problem 2, use all three methods to verify your result. The buckled shape can be compared to its analytical form:

$$\theta_n(z) = A_n \sqrt{z} J_{-1/3}\left(\frac{2}{3}\lambda_n z^{3/2}\right)$$

Figure 2.13: problem 3 description

$$\frac{d^2\theta}{dz^2} + \lambda^2 z\theta = 0$$
$$\theta'(1) = 0$$
$$\theta(0) = 0$$

For domain $0 \le z \le 1$. By numerically solving for the lowest eigenvalue λ_1, the limiting height L can next be found from solving for L in $\lambda^2 = \frac{\rho g A L^3}{EI}$. Three methods are used to find λ_1: Power method, bpv4c and Matlab eig. The buckled shape (eigen shapes) found from the numerical method is compared to the analytical shape given

$$\theta_1(z) = A_1 \sqrt{z} J_{\left(-\frac{1}{3}\right)}\left(\frac{2}{3}\lambda_1 z^{\frac{3}{2}}\right)$$

A_1 is taken as 1 due to the normalization used and J is the Bessel function of first kind.

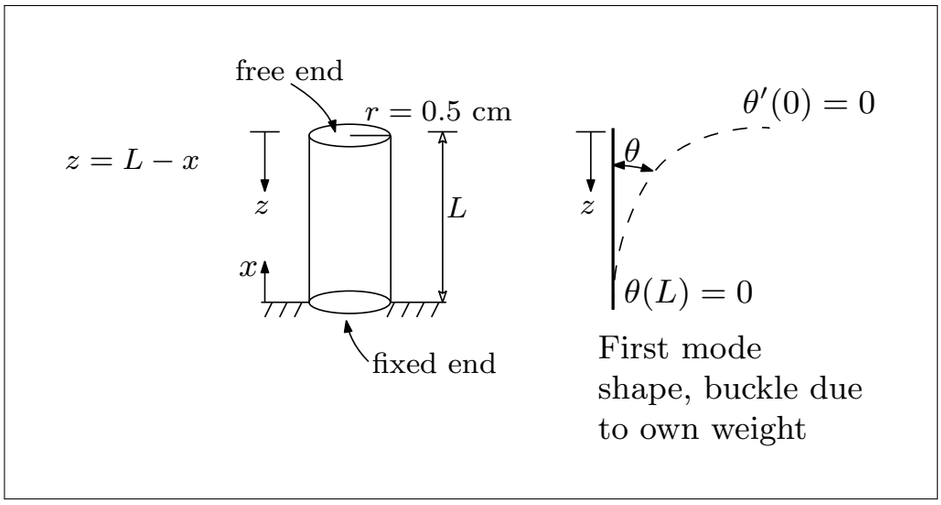


Figure 2.14: problem 3 geometry

The first step is to convert the ODE into state space for use with bvp4c. Let $x_1 = \theta, x_2 = \theta'$.

Taking derivatives gives

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\lambda^2 z x_1$$

For using eig, the problem needs to discretized first. The following shows the grid used

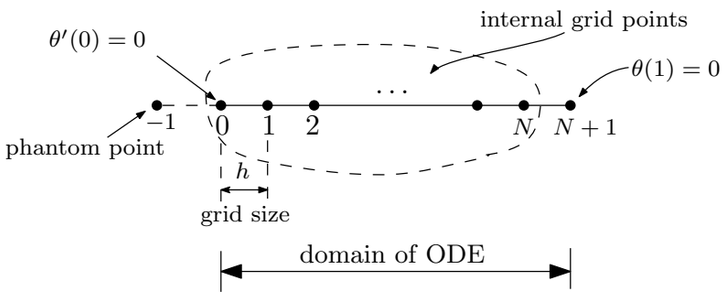


Figure 2.15: Grid used for problem 3

The grid starts at $i = 0$ which corresponds to $z = 0$ and ends at $i = N+1$ which corresponds to $z = 1$. Since θ is not known at $z = 0$, then in this problem $i = 0$ is included in the internal grid points, hence the A matrix will have size $(N+1)\times(N+1)$. Using second order centered difference gives

$$\left.\frac{d^2\theta}{dz^2}\right|_i = \frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{h^2}$$

Therefore, the approximation to the differential equation at grid i (on the internal nodes as shown in the above diagram) is as follows.

$$\left.\frac{1}{z}\frac{d^2\theta}{dz^2} + \lambda^2\theta = 0\right|_i \approx \frac{1}{ih}\frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{h^2} + \lambda^2\theta_i$$

Hence

$$\frac{1}{ih}\frac{\theta_{i+1} - 2\theta_i + \theta_{i-1}}{h^2} + \lambda^2\theta_i = 0$$
$$\frac{1}{ih}(\theta_{i+1} - 2\theta_i + \theta_{i-1}) = -h^2\lambda^2\theta_i$$

At node $i = 0$

$$\frac{\theta_1 - 2\theta_0 + \theta_{-1}}{ih + \varepsilon} = -h^2\lambda^2\theta_0$$
$$\frac{\theta_1 - 2\theta_0 + \theta_{-1}}{\varepsilon} = -h^2\lambda^2\theta_0$$

Where ε is small value 10^{-6} in order to handle the condition at $z = 0$.

To find $\theta_{i=-1}$, the condition $\theta'(0) = 0$ is used. Since $\theta'(0) = \frac{\theta_1 - \theta_{-1}}{2h} = 0$ then $\theta_{-1} = \theta_1$ and the above becomes

$$\boxed{\frac{2\theta_1 - 2\theta_0}{\varepsilon} = -h^2\lambda^2\theta_0}$$

At $i = 1$

$$\boxed{\frac{\theta_2 - 2\theta_1 + \theta_0}{h} = -h^2\lambda^2\theta_1}$$

At node $i = 2$

$$\boxed{\frac{\theta_3 - 2\theta_2 + \theta_1}{2h} = -h^2\lambda^2\theta_2}$$

And so on. At the last internal node, $i = N$

$$\frac{\theta_{N+1} - 2\theta_N + \theta_{N-1}}{Nh} = -h^2\lambda^2\theta_N$$

But $\theta_{N+1} = 0$ from boundary conditions, hence

$$\boxed{\frac{-2\theta_N+\theta_{N-1}}{Nh} = -h^2\lambda^2\theta_N}$$

At $i = N-1$

$$\boxed{\frac{\theta_N-2\theta_{N-1}+\theta_{N-2}}{(N-1)h} = -h^2\lambda^2\theta_N - 1}$$

Hence the structure is

$$\begin{bmatrix} -\frac{2}{\varepsilon} & \frac{2}{\varepsilon} & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{h} & -\frac{2}{h} & \frac{1}{h} & 0 & 0 & \cdots & \vdots \\ 0 & \frac{1}{2h} & -\frac{2}{2h} & \frac{1}{2h} & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \frac{1}{(N-1)h} & -\frac{2}{(N-1)h} & \frac{1}{(N-1)h} \\ 0 & \cdots & \cdots & \cdots & 0 & \frac{1}{Nh} & -\frac{2}{Nh} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N-2} \\ \theta_{N-1} \\ \theta_N \end{bmatrix} = -h^2\lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N-2} \\ \theta_{N-1} \\ \theta_N \end{bmatrix}$$

$$A\theta = \alpha B\theta$$

Where $\alpha = \lambda^2$ and $B = -h^2\boldsymbol{I}$. The above is implemented in Matlab and eig is used to find α.

2.3.3.1 Power method

For the power method, the A matrix is setup a little different than with the above eig method which results in

$$\frac{-1}{h^2}\begin{bmatrix} -\frac{2}{\varepsilon} & \frac{2}{\varepsilon} & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{h} & -\frac{2}{h} & \frac{1}{h} & 0 & 0 & \cdots & \vdots \\ 0 & \frac{1}{2h} & -\frac{2}{2h} & \frac{1}{2h} & 0 & \cdots & \vdots \\ 0 & 0 & \cdots & \ddots & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \frac{1}{(N-1)h} & -\frac{2}{(N-1)h} & \frac{1}{(N-1)h} \\ 0 & \cdots & \cdots & \cdots & 0 & \frac{1}{Nh} & -\frac{2}{Nh} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N-2} \\ \theta_{N-1} \\ \theta_N \end{bmatrix} = \lambda^2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 1 & 0 & 0 & \cdots & \vdots \\ 0 & 0 & 0 & 1 & 0 & \cdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{N-2} \\ \theta_{N-1} \\ \theta_N \end{bmatrix}$$

$$A\theta = \lambda^2\theta$$

The above structure is now used to solve for lowest eigenvalue and corresponding eigenvector.

One the system is solved for the lowest eigenvalue, the critical length of the column is found by solving for L from $\lambda^2 = \frac{\rho g A L^3}{EI}$.

2.3.3.2 Results

The following table shows the lowest eigenvalue found by each method, and the corresponding L found.

method	λ	$L_{critical}$ meter
bvp4c	2.7995616	18.81235953
eig	2.71870438	18.44836607
power	2.71870430	18.44836567

The following are the plots of the mode shape by each method. There is little difference that can be seen between the eig and the power methods since they are both based on the same finite difference scheme. The bvp4c is the most similar to the analytical solution. In order to evaluate and plot the analytical solution given in the problem, the eigenvalue found from bvp4c was used.

The following plot shows the result on one plot for all the methods. As can be seen, they are very similar to each others.

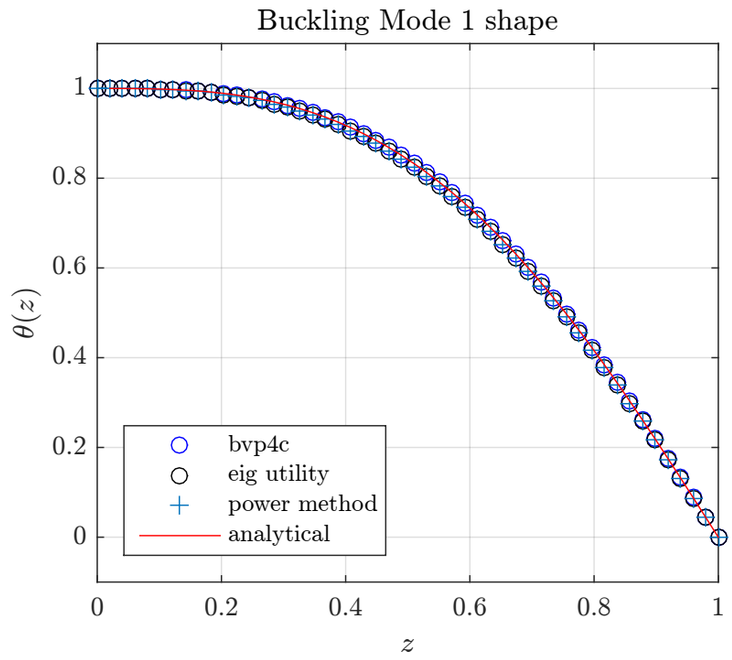


Figure 2.16: mode shape result from the three numerical method on one plot

Below is a zoomed version, showing the bvp4c is in very good agreement with the analytical plot. The power method and the eig methods are almost exactly the same. All methods become very close to each others at the boundaries and they are most different in the middle of the range.

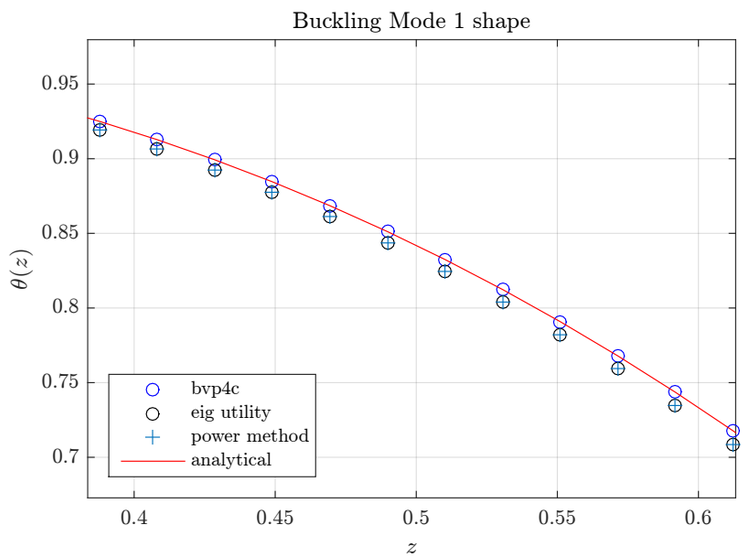


Figure 2.17: zoom in showing the result of the three methods

The following shows the result in separate plots

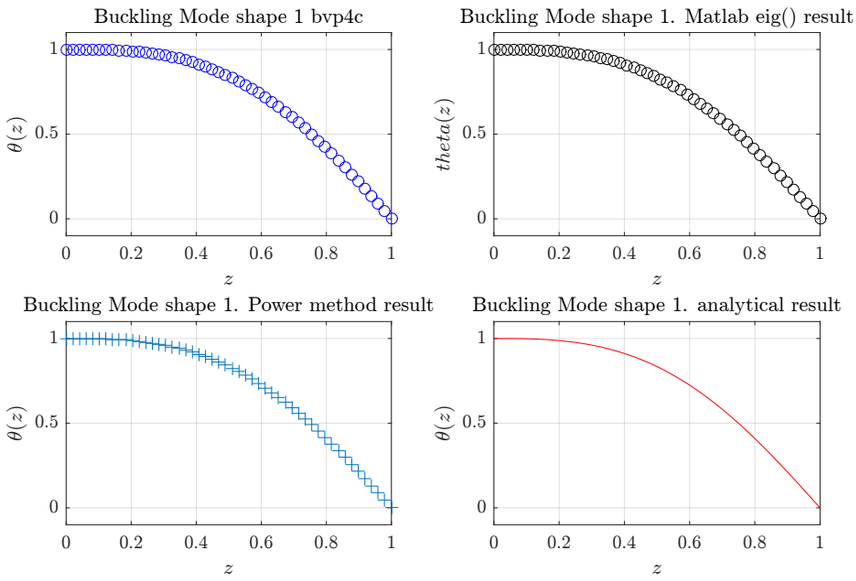


Figure 2.18: mode shape result from the three numerical method

The following is printout of Matlab console running the program

```
>>nma_HW3_EMA_471_problem_3
**********************
Eigenvalue obtained with bvp4c is
          2.79956162718772

Critical length is
          18.8123595369211
**********************
eigenvalue from eig is
          2.71870438941484

Critical length is
          18.4483660724537
**********************
eigenvalue obtained with the power iteration method
           2.7187043018523

Critical length is
          18.4483656763372
```

2.3.3.3 Source code

```
function  nma_HW3_EMA_471_problem_3()
% Solves z^4 y''+lam^2 y = 0
%
% see HW3, EMA 471, Spring 2016
% by Nasser M. Abbasi
%
clc; close all; initialize();

N            = 50;               %number of grid points.

%domain of problem, in normalized z-space.
x            = linspace(0,1,N);
guess_lambda = 2.8;

[y_bvp4c,eig_bvp4c]    = get_y_bvp4c(x, guess_lambda);
y_eig                  = get_eigenvector_matlab_eig(x, N-1);
y_power                = get_y_power(x,N-1);

%use bvp4c found eigenvalue to find analytical solution
%by using expression given in problem statement
```

```
y_analytic             = get_y_analytic(x,eig_bvp4c);

plot_result(x, y_bvp4c, y_eig, y_power, y_analytic);
end
%=====================================================================%
%This function find the eigenvalue and eigenvector
%using Matlab eig()
function y_eig = get_eigenvector_matlab_eig(x,N)

h                      = x(2)-x(1);  % find grid spacing
A                      = setup_A_matrix(N,h);
B                      = setup_B_matrix(N,h);
[eig_vector,eig_values] = eig(A,B);
eig_values             = diag(eig_values); %they are on diagonal
sorted_eig_values      = sort(eig_values); %sort , small to large

%now need to match the original positiion of the eigenvalue
%with its correspoding eigenvectr. Hence find the index of
%correct eigevalue so  use to index to eigenvector
found_eig_vector = eig_vector(:,eig_values == sorted_eig_values(1));

%Set is sign correctly
if found_eig_vector(1) > 0
    y_eig = [found_eig_vector ; 0];
else
    y_eig = [-found_eig_vector ; 0];
end

y_eig = y_eig/max(y_eig);  %normalize

%normalize eigevalues
sorted_eig_values = sqrt(sorted_eig_values)/pi;
fprintf('\n**********************\n');
fprintf('eigenvalue from eig is\n');
disp(sorted_eig_values(1)*pi);

calculate_critial_length(sorted_eig_values(1)*pi);

    %-------------------------------%
    function A = setup_A_matrix(N,h)
        A      = zeros(N);
        eps    = 1e-6;
        A(1,1) = -2/eps;
        A(1,2) = 2/eps;
        for i = 2:N-1
            A(i,i-1:i+1) = [1,-2,1]/((i-1)*h);
        end
        A(N,N)   = -2/(N*h);
        A(N,N-1) = 1/(N*h);
    end
    %-------------------------------%
    function B = setup_B_matrix(N,h)
        B = -h^2 * eye(N);
    end

end
%=====================================================================%
function y = get_y_power(x,N)

h     = x(2)-x(1);  % find grid spacing
A     = setup_A_matrix_for_power(h,N);
A_inv = setup_A_inv_matrix_for_power(N);

```

```
% Starting guess for the eigenvector. Use unit vector
y = ones(N,1);

% This below from EX 11, apply it here:
% set tolerance; "while" loop will run until there is no
%difference between  old and new estimates for eigenvalues to
%within the tolerance

tol = 1e-6;
eigenvalue_1_old = 0;
eigenvalue_1_new = 1;

while abs(eigenvalue_1_new - eigenvalue_1_old)/abs(eigenvalue_1_new) > tol

    % generate updated value for eigenvector
    y_new = A_inv*y;


    eigenvalue_1_old = eigenvalue_1_new; % update old eigenvalue
    eigenvalue_1_new = max(y_new);       % update new eigenvalue
    y = y_new/eigenvalue_1_new; %renormalize eigenvector estimate
end

y = [y;0];
y = y/max(y);   %normalize

% Taken Per EX 11:
% add boundary conditions to complete eigenvector; also
%note that we have  found the largest value of the inverse of
%what we're looking for, so...

% the lambda we're seeking is actually the
% inverse of the square root of what we've found
lam = 1/sqrt(eigenvalue_1_new);

fprintf('\n**********************\n');
fprintf('eigenvalue obtained with the power iteration method\n');
disp(lam);

calculate_critial_length(lam);

    %--------------------------------%
    function A = setup_A_matrix_for_power(h,N)
        A      = zeros(N);
        eps    = 1e-6;
        A(1,1) = -2/eps;
        A(1,2) = 2/eps;
        for i = 2:N-1
            A(i,i-1:i+1) = [1,-2,1]/((i-1)*h);
        end
        A(N,N)   = -2/(N*h);
        A(N,N-1) = 1/(N*h);
        A        = -A/h^2;
    end
    %--------------------------------%
    function A_inv = setup_A_inv_matrix_for_power(N)
        %We are looking for smallest eigenvalue. Use inverse.
        A_inv = zeros(N);
        for i = 1:N
            b_rhs = zeros(N,1);
            b_rhs(i,1) = 1;
            A_inv(:,i) = A\b_rhs;
        end
```

```
147
148     end
149 end
150
151 %=====================================================================%
152 function  [y_bvp4c_normalized, eigen_value] = ...
153                                     get_y_bvp4c(x,guess_lambda)
154
155 initial_solution  = bvpinit(x,@set_initial_mode_shape,...
156                             guess_lambda);
157 y_bvp4c           = bvp4c(@rhs,@bc,initial_solution);
158 eigen_value       = y_bvp4c.parameters;
159 fprintf('\n**********************\n');
160 fprintf('Eigenvalue obtained with bvp4c is\n');
161 disp(eigen_value);
162
163 calculate_critial_length(eigen_value);
164
165 y_bvp4c            = deval(y_bvp4c,x);    %interpolate
166 y_bvp4c_normalized = y_bvp4c/max(y_bvp4c(1,:)); %normalize
167
168 %--------------------------------%
169     function solinit = set_initial_mode_shape(x)
170         %  internal function
171         %  This defines the initial guess for the eigenvector;
172         %   the first guess of
173         %  the fundamental mode shape is a sawtooth
174         %
175         f  = 1-x;
176         fp = -1;
177         solinit = [ f ; fp ];
178     end
179     %-----------------------------%
180     function f = rhs(t,x,lam)
181         %This function sets up the RHS of the state space
182         %setup for this problem.
183         %similar to ode45 RHS
184         x1 = x(2);
185         x2 = -t*lam^2*x(1);
186         f = [ x1
187               x2];
188     end
189     %-----------------------------%
190     function res = bc(ya,yb,~)
191         %This sets up the boundary conditions vector.
192         %Must have ~ above in third agrs!
193         res = [ ya(2)
194                 yb(1)
195                 yb(2)+1
196               ];
197     end
198 end
199
200 %=====================================================%
201 function y_analytic = get_y_analytic(z,eigen_value)
202
203   y_analytic = sqrt(z) .* besselj(-1/3,(2/3)*eigen_value*z.^(3/2));
204   y_analytic = y_analytic/max(y_analytic);  %normalize
205
206
207 end
208 %=============================================================-===%
209 function plot_result(x, y_bvp4c_normalized, y_eig,...
```

```
                                        y_power, y_analytic)

figure();
plot(x,y_bvp4c_normalized(1,:),'bo',...
     x,y_eig,'ko',...
     x,y_power,'+',...
     x,y_analytic,'r');

axis([0 1 -0.1 1.1])
title('Buckling Mode 1 shape');
xlabel('$z$')
ylabel('$\theta(z)$')
legend('bvp4c','eig utility','power method',...
        'analytical','Location','southwest')
grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);


figure();
subplot(2,2,1);
plot(x,y_bvp4c_normalized(1,:),'bo');
title(sprintf('Buckling Mode shape %d bvp4c',1));
xlabel('$z$'); axis([0 1 -0.1 1.1])
ylabel('$\theta(z)$'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,2);
plot(x,y_eig,'ko');
title(sprintf('Buckling Mode shape %d. Matlab eig() result',1));
xlabel('$z$'); axis([0 1 -0.1 1.1])
ylabel('$theta(z)$'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,3);
plot(x,y_power,'+');
title(sprintf('Buckling Mode shape %d. Power method result',1));
xlabel('$z$'); axis([0 1 -0.1 1.1])
ylabel('$\theta(z)$'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

subplot(2,2,4);
plot(x,y_power,'r');
title(sprintf('Buckling Mode shape %d. analytical result',1));
xlabel('$z$'); axis([0 1 -0.1 1.1])
ylabel('$\theta(z)$'); grid;
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);

end
%=========================================================%
function calculate_critial_length(lam)

r       = 0.05; %meter  radius
g       = 9.81; %acc. due to gravity
density = 7500;   % kg/m^3
E       = 100*10^9;  %Pa
I0      = (1/4)*pi*(r)^4;
L       =  (lam^2*E*I0/(density*g*pi*r^2))^(1/3);
fprintf('Critical length is\n');
disp(L);
end
%=========================================================%
function initialize()
reset(0);
```

```
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');

format long g
end
```

2.4 HW 4

2.4.1 Problem 1

Important note on Matlab: I used Matlab 2016a to write all the software. In particular, the function `histcounts()` was used for binning as recommended by Matlab help pages. This function do not exist in Matlab 2014a and was added in Matlab 2014b.

Due: Thursday, March 17th, 2016

Note: For each of the problems below, you will be asked to run a large number of trials (10^6 or 10^7). During debugging, it may be useful to set this to a much lower number (maybe 10^3 or 10^4) until you are confident in your results. Once you are satisfied with your script, you can then increase the number of trials to the requested values to get a more statistically pleasing set of plots.

Note: Pre-allocating space for arrays is particularly important when you are running 10^n cases!

(1) (10 pts) An alternative representation of a peaked PDF is the Lorentzian shape:

$$f(x) = A\frac{1}{1+x^2},$$

Here x takes on values ranging from $\pm\infty$.

(a) Find the normalization constant A so that the probability of x taking on some value within its admissible range is 1.

(b) Use Matlab's rand utility to replicate this PDF by sampling its associated CDF and inverting it. Use 10^6 trials and plot your results in bins of width $\Delta x = 0.1$ over a range from $x = -10$ to $+10$.

Figure 2.19: problem 1 description

2.4.1.1 Part (a)

$$f(x) = A\frac{1}{1+x}$$

To normalize it, we first solve for A from

$$\int_{-\infty}^{\infty} f(x)\,dx = 1$$

Hence

$$\begin{aligned} 1 &= \int_{-\infty}^{\infty} A\frac{1}{1+x}dx \\ &= A\left[\arctan(x)\right]_{-\infty}^{\infty} \\ &= A\left(\frac{\pi}{2}+\frac{\pi}{2}\right) \end{aligned}$$

Therefore

$$\boxed{A = \frac{1}{\pi}}$$

Now we need to find cumulative probability distribution $F(x)$ and invert then it.

$$\begin{aligned} F(x) &= \int_{-\infty}^{x} f(x)\,dx \\ &= \int_{-\infty}^{x} \frac{1}{\pi}\frac{1}{1+x}dx \\ &= \frac{1}{\pi}\left(\arctan(x)\right)_{-\infty}^{x} \\ &= \frac{1}{\pi}\left(\arctan(x)+\frac{\pi}{2}\right) \end{aligned}$$

Hence

$$F(x) = \frac{1}{\pi}\arctan(x) + \frac{1}{2}$$

To invert,

$$\pi\left(F(x)-\frac{1}{2}\right)=\arctan(x)$$

Hence

$$\boxed{x=F^{-}1(x)=\tan\left(\pi\left(F(x)-\tfrac{1}{2}\right)\right)}$$

2.4.1.2 Part(b)

Matlab rand was used to generate number of samples. To see the effect, the sampling was increased from 10^2 to 10^6. The area under the pdf generated using sampling was normalized so that its area is one. The following plots show the result. The source code is included.

The algorithm used is the following

Algorithm 1 Algorithm to generate random variables from pdf using rand

```
procedure GENERATE_RV
    F(x) ← cumulative distribution
    N ← Number of trials
    for i ← 1, N do
        x(i) ← rand()
        data(i) ← F^{-1}(x(i))
    end for
    generate normalized histogram from data
end procedure
```

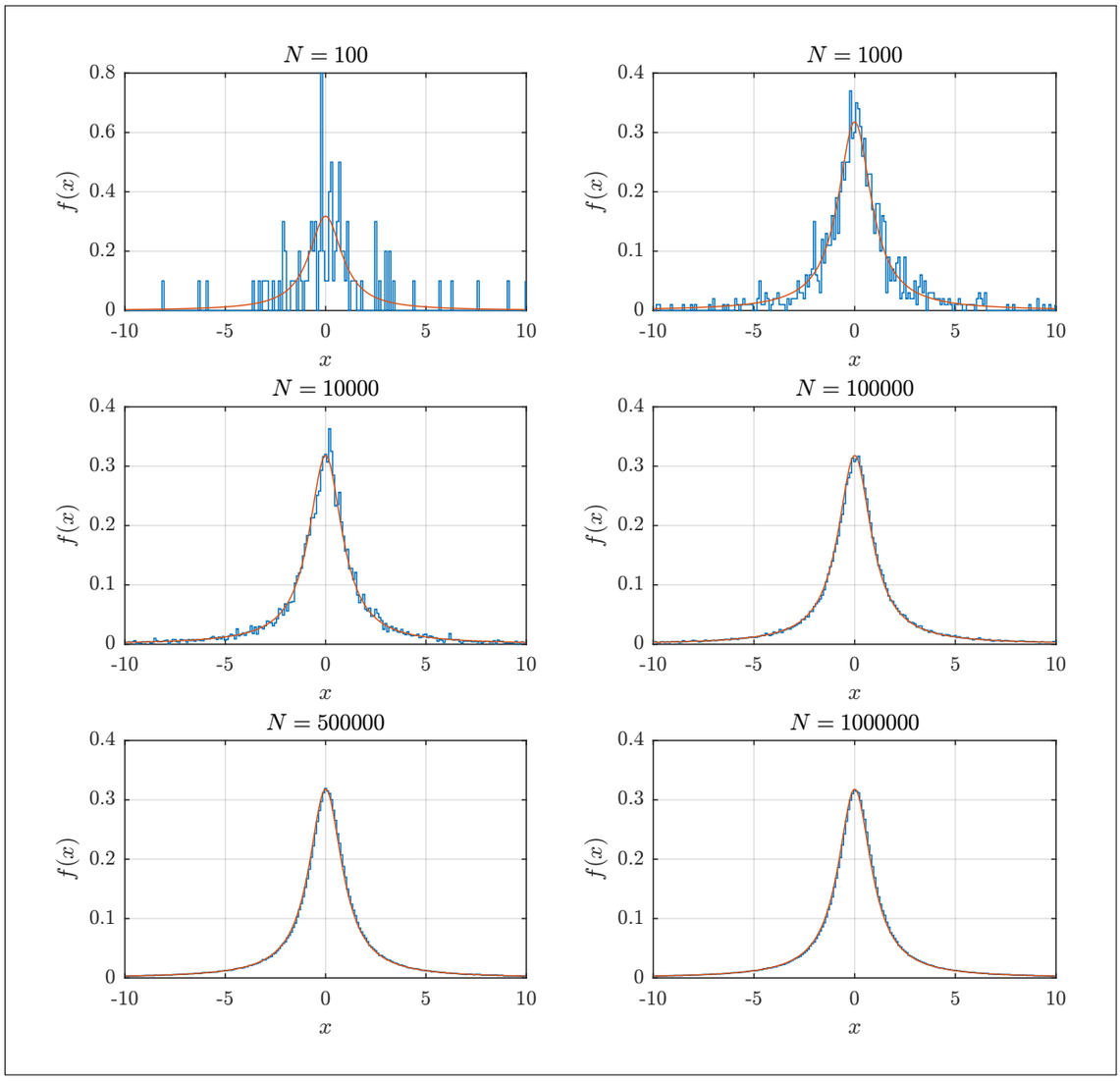


Figure 2.20: Showing how sampling improves with increased N

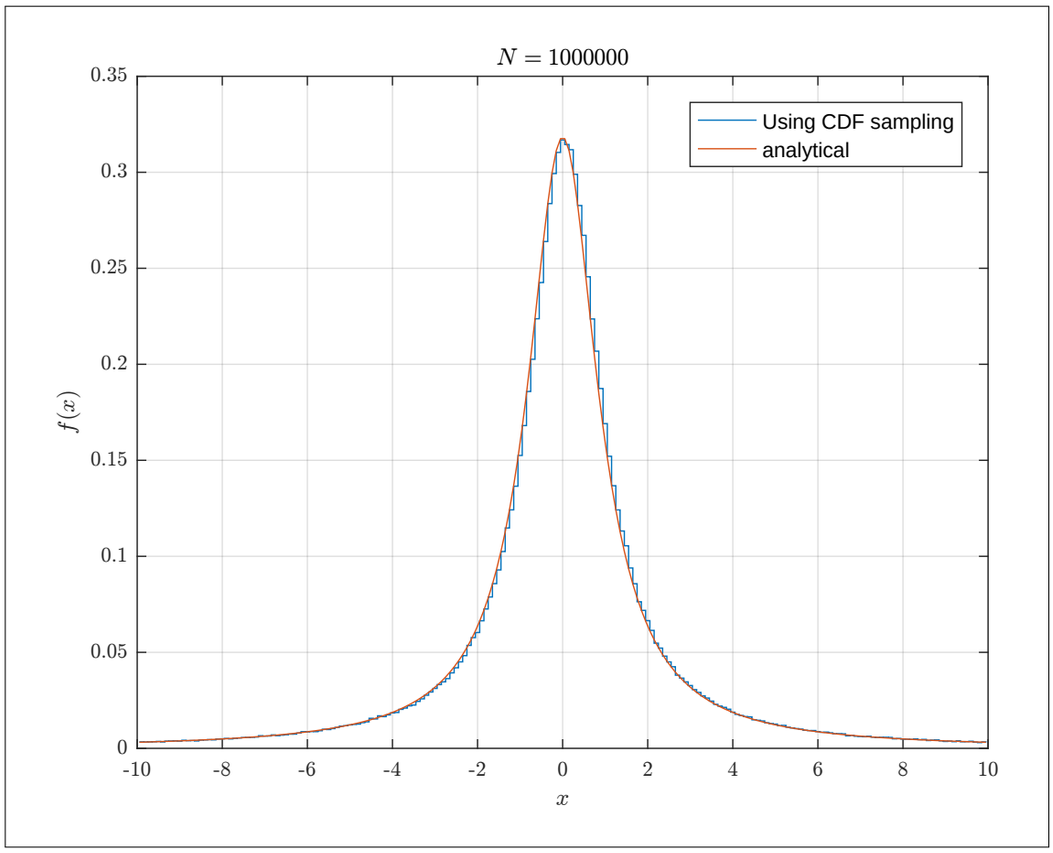


Figure 2.21: The case for $N = 10^6$ only.

```
function nma_HW4_problem_1()
%Problem 1, EMA 471, HW4
%by Nasser M. Abbasi
%
%
close all; clc;

N = [10^2 10^3 10^4 10^5 5*10^5 10^6];
for i=1:length(N)  %make a plot for each N
    subplot(3,2,i);
    process(N(i));
end

%do last one on its own
figure();
process(N(end));
legend('Using CDF sampling','analytical');
end

%==============================
%This function solve the problem for specific N. This is done
%to see how increasing N improves the process.
function process(N)
%Initialize RNG
rng('default');
rng(1);

%generate N random numbers from U(0,1)
xi  = rand(N,1);

%sample the CDF using the inverse of F(x). Report
%shows the analytical result
data  =  finv(xi);

%generate the bins using histcounts
bin_width = 0.1;
```

```
edges     = -10:bin_width:10;
x_bins    = histcounts( data(data<=10&data>=-10),edges);

%now comes the tricky part. We need to find the real area
%in order to normalize with. If we use the area from the
%above, which is from -10<x<10,  then this will not be
%accurate (it is a little smaller than the true area). To get
%the real data, we have to rebin the full data. This below is just
%to get the full area

edges     = min(data):bin_width:max(data);
x_bins2   = histcounts( data,edges);
area      = bin_width*sum(x_bins2);

%This below is another way to get the full area.
%h    = histogram(data,'BinWidth',bin_width,...
%                        'BinLimits',[min(data),max(data)]);
%area = bin_width*sum(h.Values);   %use this to normalize with

%Continue with our bins -10<=x<=10. Now normalize
%it and Plot the pdf generated and compare it with the
%analytical one
x    = -9.95:bin_width:9.95;  %use center of bins
stairs(x,x_bins/area);   %notice, we divide by the full area
hold on;

 %add the analytical pdf. This is normalized
plot(x,(1/pi)*(1./(1+x.^2)));
title(sprintf('$N=%d$',N),'interpreter','Latex','Fontsize',7);
xlabel('$x$','interpreter','Latex','Fontsize',7);
ylabel('$f(x)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
grid;

end
%==================================
%This is the inverse of the CDF. Derived by hand
%from inversing the analytical pdf given in the problem
function r= finv(y)
 r =  tan(pi*(y-1/2));
end
```

2.4.2 Problem 2

(2) (15 pts) The extent to which radiation causes cell damage depends strongly on its Linear Energy Transfer (LET) rate. Heavy charged particles have high LET values and cause far more damage than light charged particles with low LET values. The probability of cell survival can be described by the following two pdfs:

High LET radiation:

$$f_H(D) = \exp(-0.0187D)$$

Low LET radiation:

$$f_L(D) = \begin{cases} \exp(a_1 D + a_2 D^2 + a_3 D^3 + a_4 D^4), & D \le 600 \text{ rads} \\ A\exp(a_5 D), & D > 600 \text{ rads} \end{cases}$$

In the latter expression for $f_L(D)$, the coefficients have these values:

$$a_1 = -6.84806 \times 10^{-4}$$
$$a_2 = +4.87285 \times 10^{-6}$$
$$a_3 = -3.28988 \times 10^{-8}$$
$$a_4 = +2.66992 \times 10^{-11}$$
$$a_5 = -0.0073$$
$$A = 7.9839$$

Use a rejection algorithm to reproduce the probabilities of cell survival when cells are subjected to radiation doses up to 1000 rads. Use 1 rad bin widths and run your simulation for 10^7 trials for both the high and low LET cases. Plot your results on a semilog plot (`semilogy`) with the axes in x (D) from 0 to 1000 rads and in y (f) from 10^{-3} to 1. Note that the PDFs have already been normalized to have a maximum value of 1.

Figure 2.22: problem 2 description

The rejection method was used to first generate $f_H(D)$ and then was used again to generate $F_L(D)$. So the result shows each of these separately below.

For each case, number of trials was increased to see the effect. The following trials were used $\{10^4, 10^5, 10^6, 10^7\}$. The last amount is the one asked for in this problem. For each case, both the similogy plot is shown and the normal scale plot is also shown. These results shows the rejection method improves as more trials are used. The analytic pdf is also plotted against the generated one to compare.

Matlab source code is included. Implemented on Matlab 2016a.

The algorithm used is the following

Algorithm 2 Algorithm to generate random variables from pdf using rejection method

```
procedure GENERATE_RV
    f(x) ← probability distribution
    N ← Number of trials
    k ← 0
    for i ← 1,N do
        ζ1 ← rand()
        ζ2 ← rand()
        if ζ2 ≤ f(ζ1) then
            k ← k + 1
            count[k] ← ζ1
        end if
    end for
    generate normalized histogram from count
end procedure
```

2.4.2.1 High LET radiation

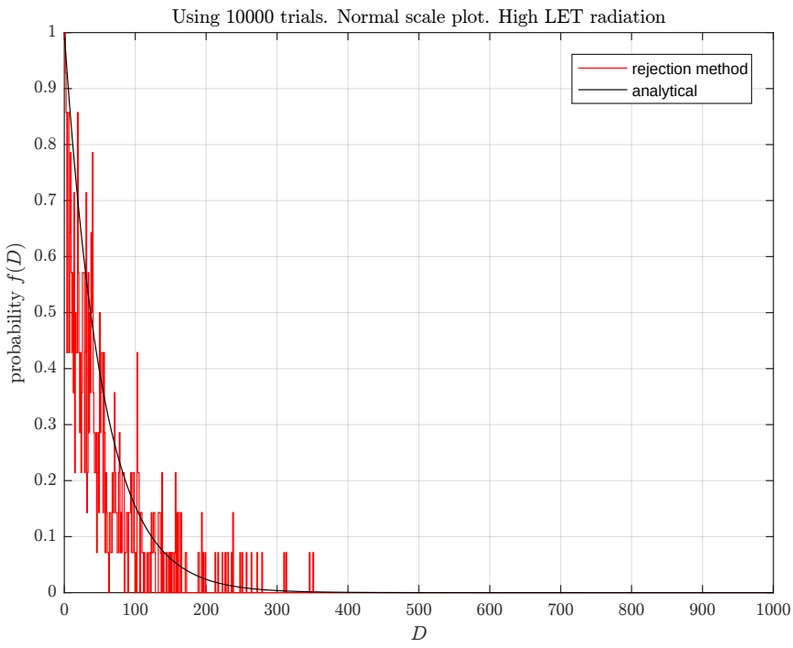


Figure 2.23: High LET radiation, 10^4 trials, normal plot

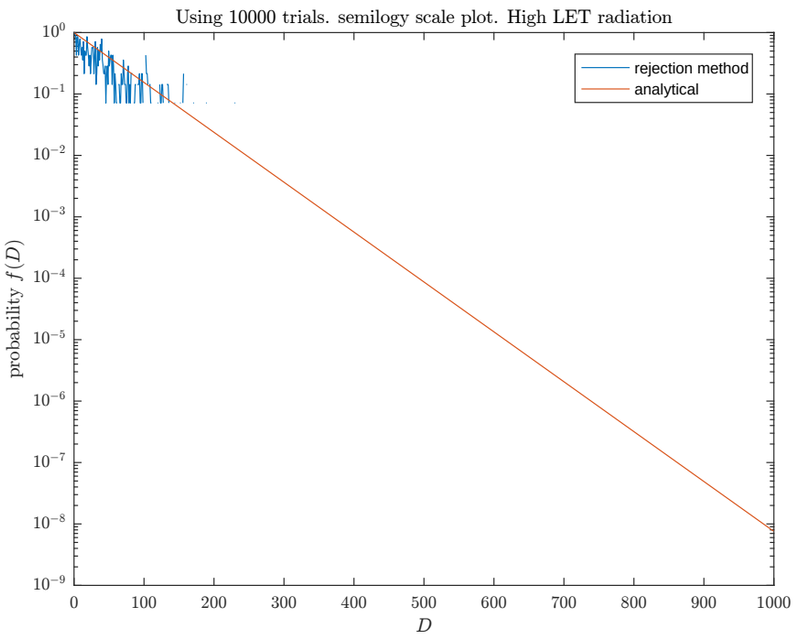


Figure 2.24: High LET radiation, 10^4 trials log plot

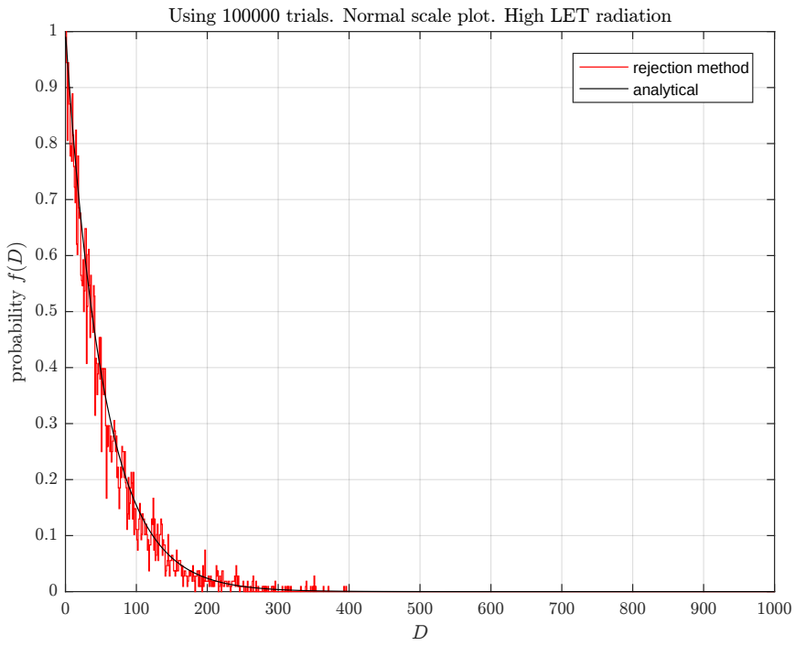


Figure 2.25: High LET radiation, 10^5 trials, normal plot

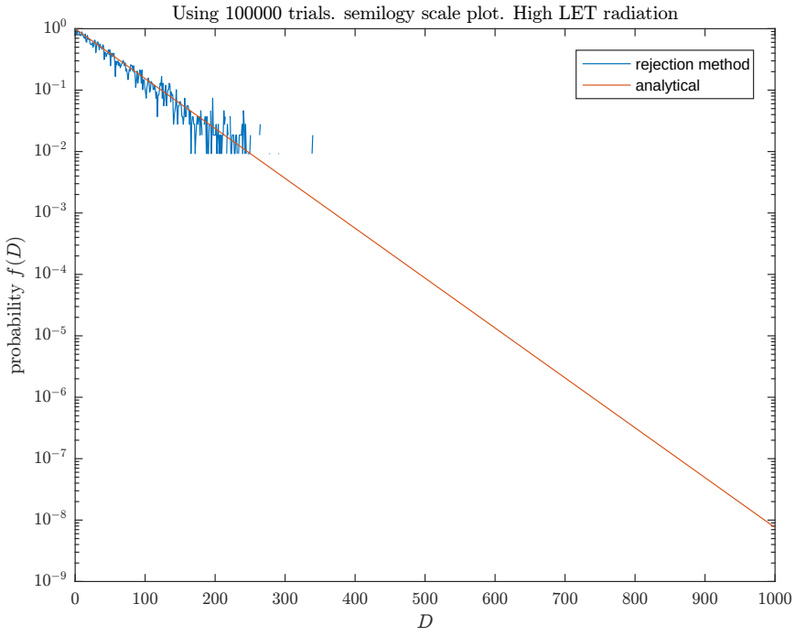


Figure 2.26: High LET radiation, 10^5 trials log plot

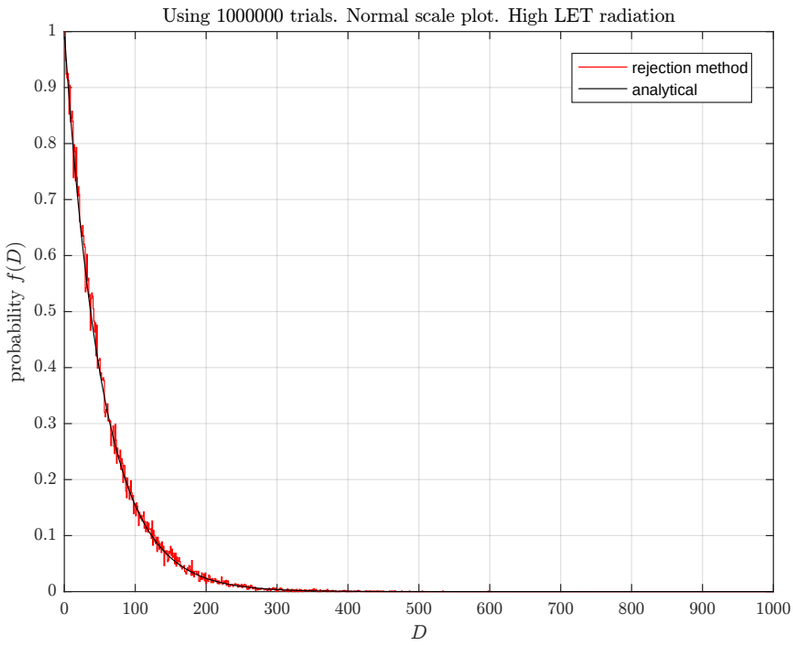


Figure 2.27: High LET radiation, 10^6 trials, normal plot

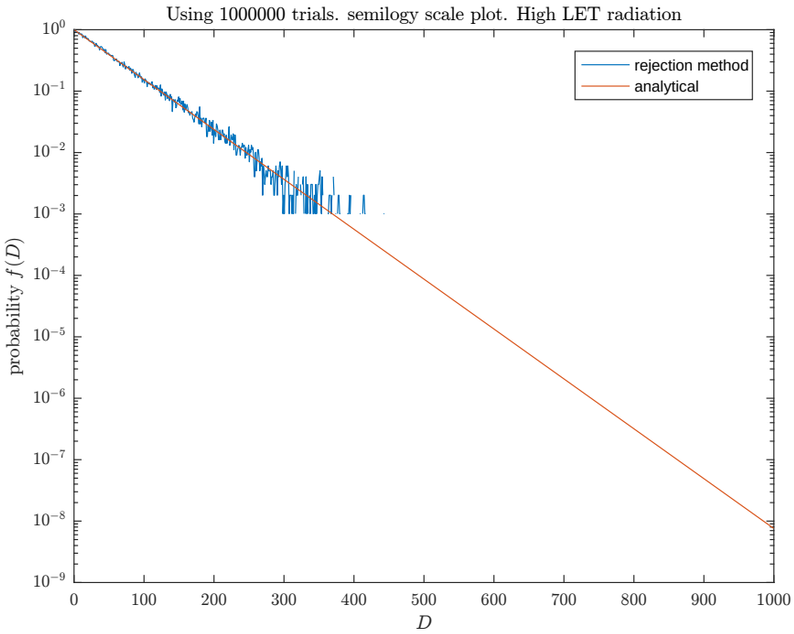


Figure 2.28: High LET radiation, 10^6 trials log plot

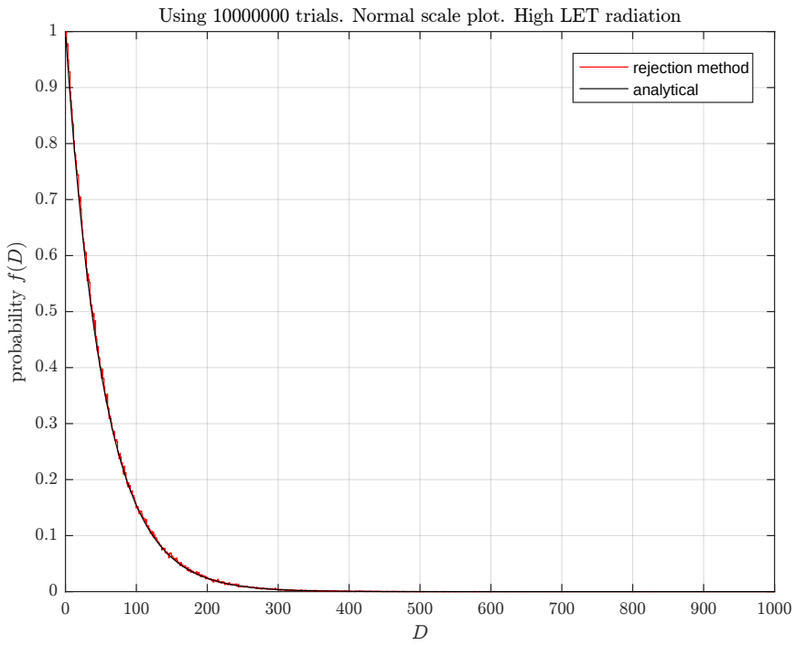


Figure 2.29: High LET radiation, 10^7 trials, normal plot

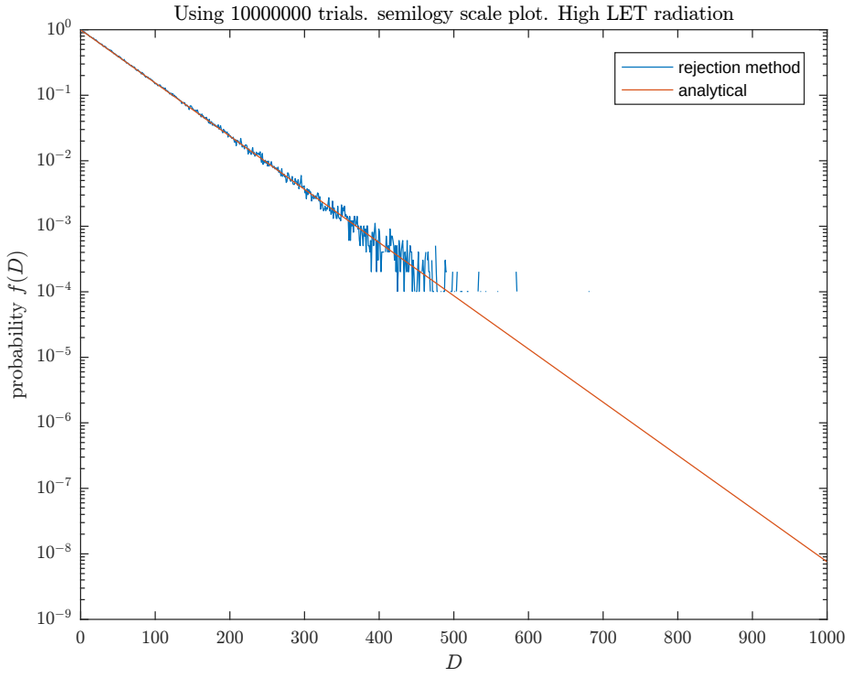


Figure 2.30: High LET radiation, 10^7 trials log plot

2.4.2.2 Low LET radiation

The above was repeated for the low LET radiation. The result is given below

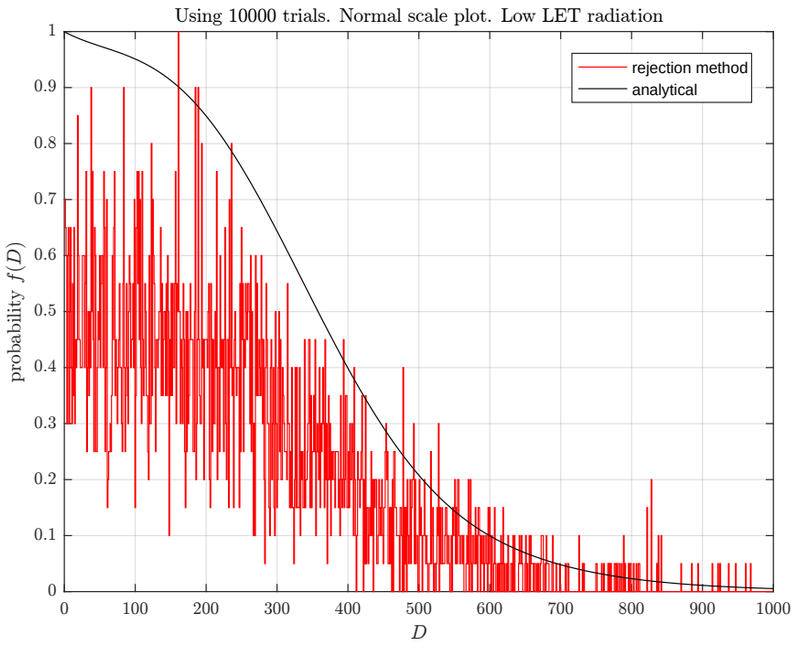


Figure 2.31: Low LET radiation, 10^4 trials, normal plot

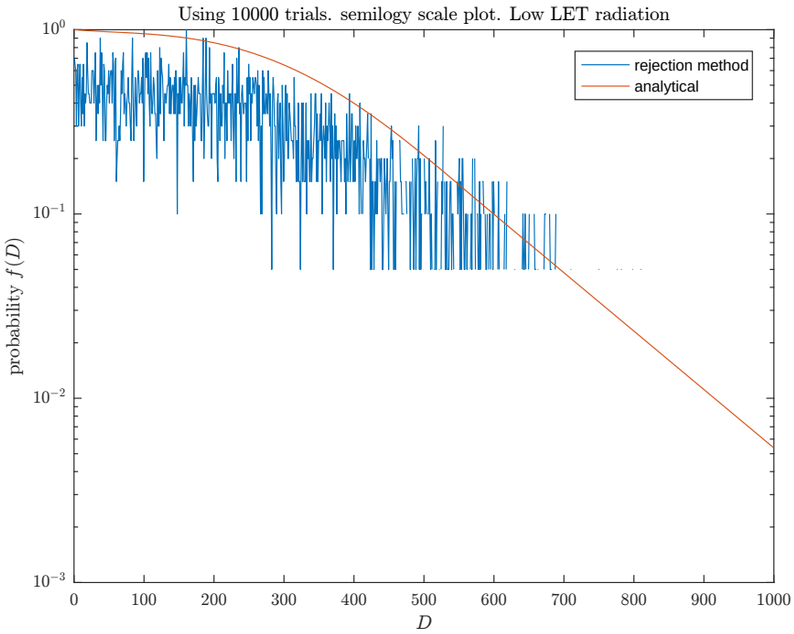


Figure 2.32: Low LET radiation, 10^4 trials log plot

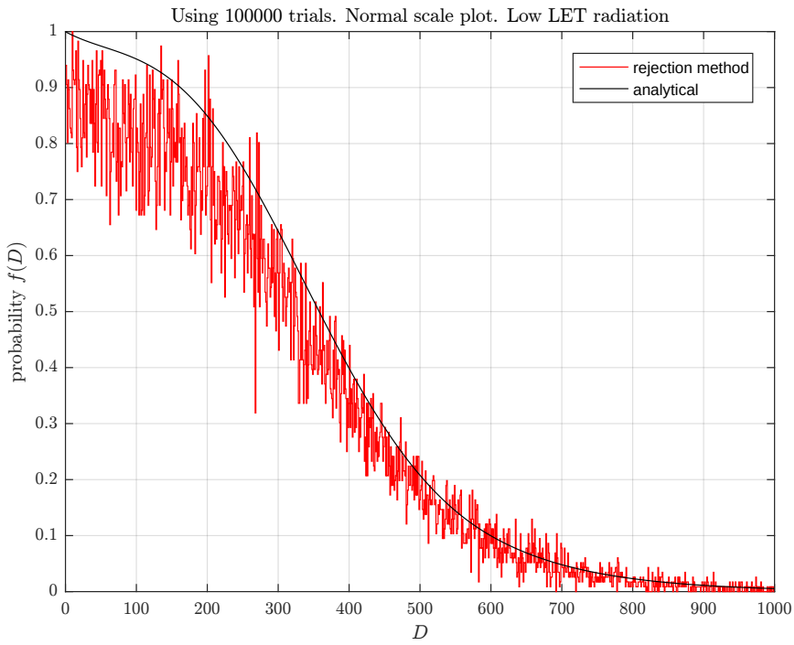


Figure 2.33: Low LET radiation, 10^5 trials, normal plot

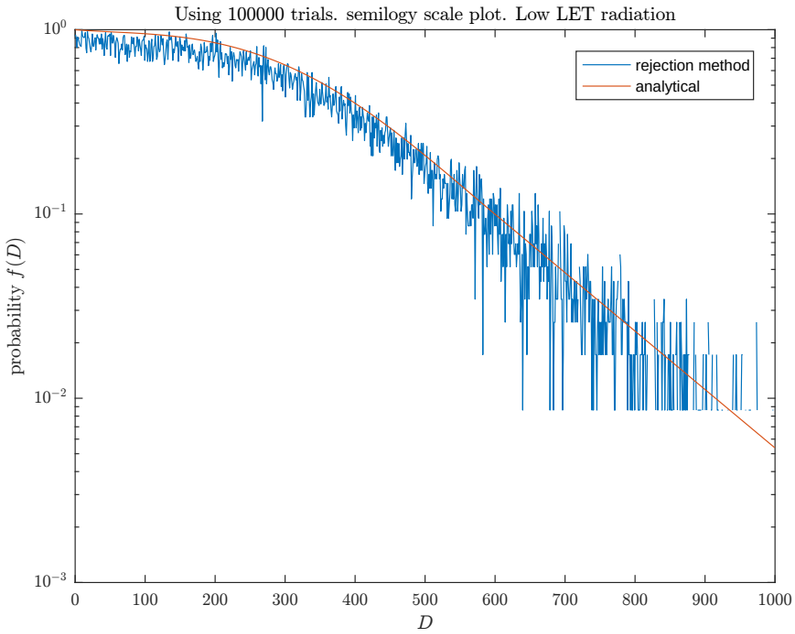


Figure 2.34: Low LET radiation, 10^5 trials log plot

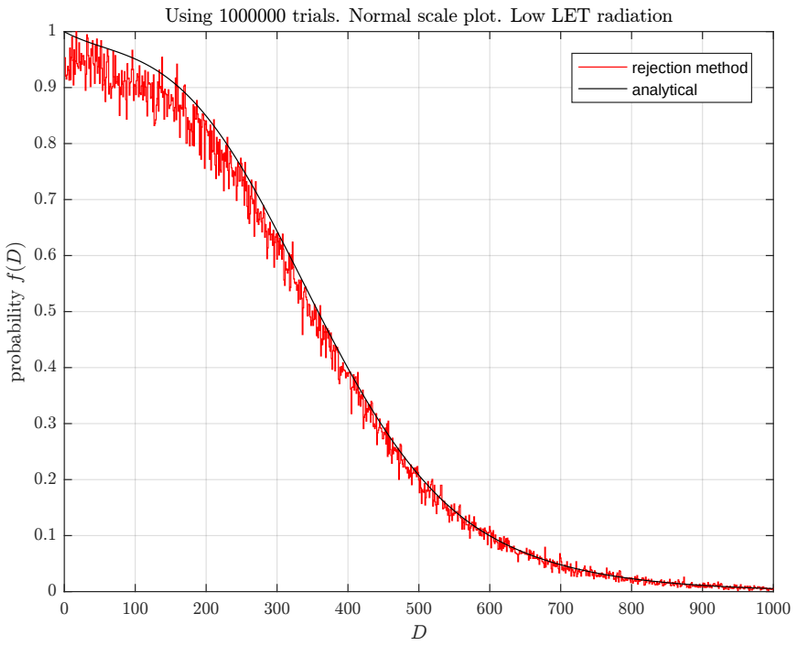


Figure 2.35: Low LET radiation, 10^6 trials, normal plot

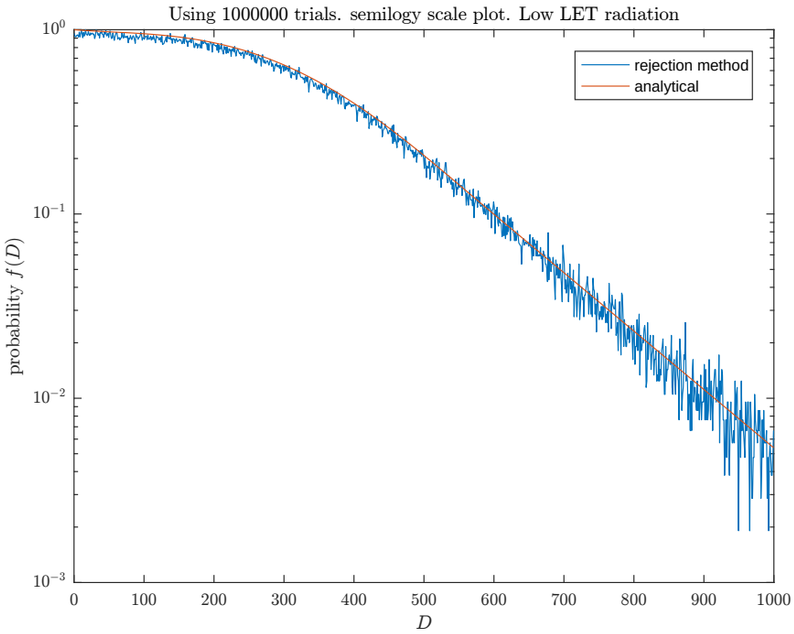


Figure 2.36: Low LET radiation, 10^6 trials log plot

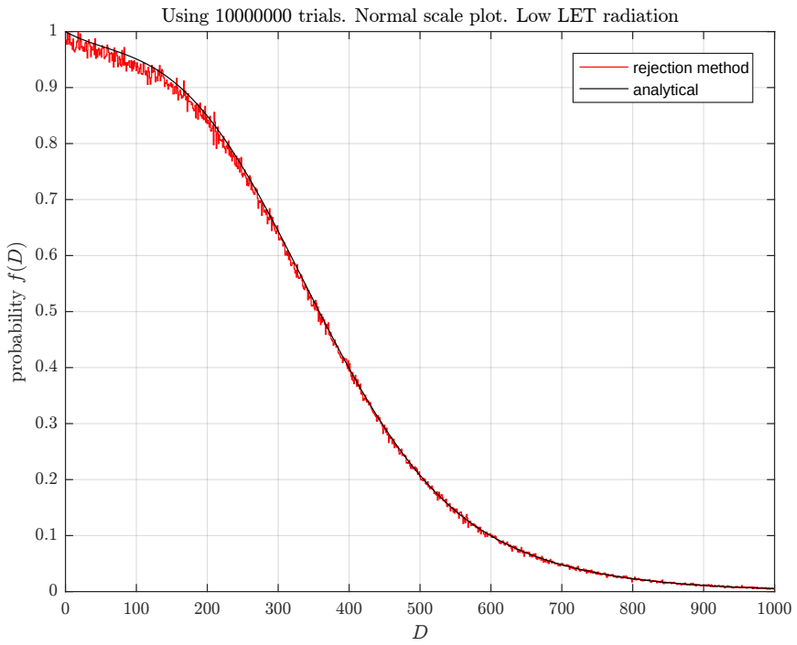


Figure 2.37: Low LET radiation, 10^7 trials, normal plot

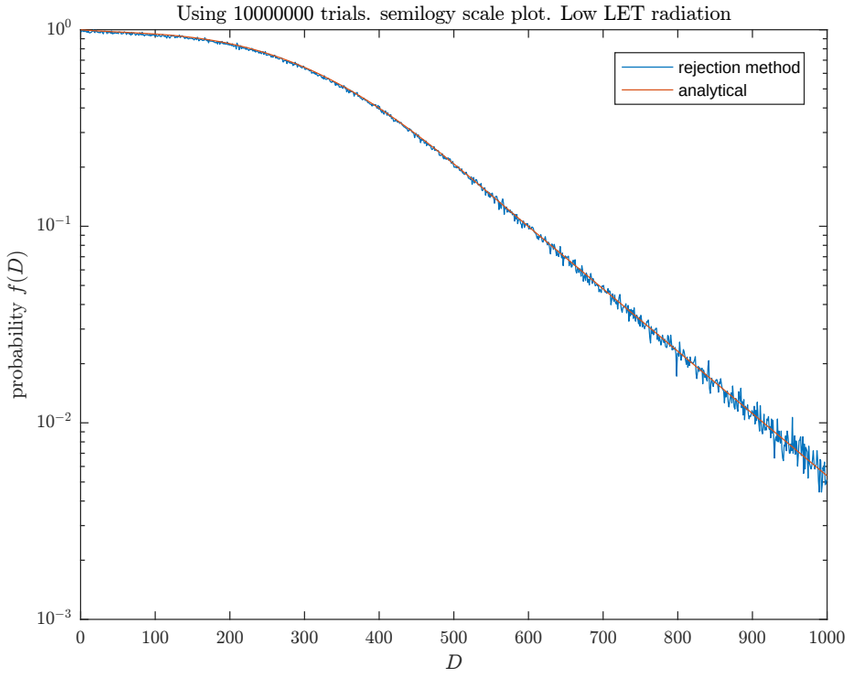


Figure 2.38: Low LET radiation, 10^7 trials log plot

```
function nma_HW4_problem_2()
%Problem 2, EMA 471, HW4
%by Nasser M. Abbasi
%
%
close all; clc;

ntrials = [10^4 10^5 10^6 10^7];
for i=1:length(ntrials)
    process(ntrials(i));
end
end
%==================================
function process(N)
%N: number of trials

```

```
%Initialize random number generator
rng('default');
rng(1);

MAX     = 1000;         %maximum rad to simulate
counts  = zeros(N,1);  %to save accepted trials into
k       = 0;           %counter for keeping accepted

for i=1:N
    x_1 = rand();
    x_2 = rand();
    D = x_1*MAX; %convert to actual D from 0..1000
    if x_2< exp(-0.0187*D)
       k = k + 1;
       counts(k) = D;
    end
end

generate_plots(N,counts(1:k),MAX);
end
%=======================
function generate_plots(number_of_trials,counts,MAX)
generate_normal_plots(number_of_trials,counts,MAX);
generate_log_plots(number_of_trials,counts,MAX);
end
%=======================
function generate_normal_plots(number_of_trials,counts,MAX)
figure;
bin_width = 1;
edges     = 0:bin_width:MAX;
x_bins    = histcounts( counts,edges);
x         = 0.5:bin_width:MAX-0.5;  %use center of bins
stairs(x,x_bins/max(x_bins),'r');   %notice, we divide by the full area
hold on;
plot(x,exp(-0.0187*x),'k');
grid;

title(sprintf( ...
     'Using %d trials. Normal scale plot. High LET radiation',...
        number_of_trials),'interpreter','Latex','Fontsize',7);
xlabel('$D$','interpreter','Latex','Fontsize',7);
ylabel('probability $f(D)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical');
print(gcf, '-dpdf', '-r600', ...
 sprintf('../images/%d_part_a_normal_plot.pdf',number_of_trials));

end
%================================
function generate_log_plots(number_of_trials,counts,MAX)
figure;
bin_width = 1;
edges     = 0:bin_width:MAX;
x_bins    = histcounts( counts,edges);
x         = 0.5:bin_width:MAX-0.5;  %use center of bins
semilogy(x,x_bins/max(x_bins));
hold on;
semilogy(x,exp(-0.0187*x));
%grid;

title(sprintf(...
   'Using %d trials. semilogy scale plot. High LET radiation',...
       number_of_trials),'interpreter','Latex','Fontsize',7);
```

```
xlabel('$D$','interpreter','Latex','Fontsize',7);
ylabel('probability $f(D)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical');
print(gcf, '-dpdf', '-r600', ...
  sprintf('../images/%d_part_a_log_plot.pdf',number_of_trials));

end
```

```
function nma_HW4_problem_2_second_part()
%Problem 2, Low Let radiation. EMA 471, HW4
%by Nasser M. Abbasi
%
%
close all; clc;

ntrials = [10^4 10^5 10^6 10^7];
for i=1:length(ntrials)
    process(ntrials(i));
end
end
%==================================
function process(N)
%N: number of trials

%Initialize random number generator
rng('default');
rng(1);

MAX     = 1000;         %maximum rad to simulate
counts  = zeros(N,1);   %to save accepted trials into
k       = 0;            %counter for keeping accepted

for i=1:N
    x_1 = rand();
    x_2 = rand();
    D = x_1*MAX; %convert to actual D from 0..1000

    if x_2 < pdf_low_let(D)
       k = k + 1;
       counts(k) = D;
    end
end

generate_plots(N,counts(1:k),MAX);
end
%=======================
function generate_plots(number_of_trials,counts,MAX)
analytical_pdf = zeros(MAX,1);
for i=1:MAX
  analytical_pdf(i) = pdf_low_let(i);
end

generate_normal_plots(number_of_trials,counts,MAX,analytical_pdf);
generate_log_plots(number_of_trials,counts,MAX,analytical_pdf);
end
%=======================
function generate_normal_plots(number_of_trials,counts, ...
                                        MAX,analytical_pdf)
figure;
bin_width = 1;
edges     = 0:bin_width:MAX;
x_bins    = histcounts( counts,edges);
x         = 0.5:bin_width:MAX-0.5;  %use center of bins
```

```
%notice, we divide by the full area
stairs(x,x_bins/max(x_bins),'r');
hold on;
plot(x,analytical_pdf,'k');
grid;

title(sprintf(...
     'Using %d trials. Normal scale plot. Low LET radiation',...
        number_of_trials),'interpreter','Latex','Fontsize',7);
xlabel('$D$','interpreter','Latex','Fontsize',7);
ylabel('probability $f(D)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical');
print(gcf, '-dpdf', '-r600', ...
 sprintf('../images/%d_part_b_normal_plot.pdf',number_of_trials));

end
%================================
function generate_log_plots(number_of_trials,counts,...
                                                MAX,analytical_pdf)
figure;
bin_width = 1;
edges     = 0:bin_width:MAX;
x_bins    = histcounts( counts,edges);
x         = 0.5:bin_width:MAX-0.5;  %use center of bins
semilogy(x,x_bins/max(x_bins));
hold on;
semilogy(x,analytical_pdf);
%grid;

title(sprintf(...
   'Using %d trials. semilogy scale plot. Low LET radiation',...
       number_of_trials),'interpreter','Latex','Fontsize',7);
xlabel('$D$','interpreter','Latex','Fontsize',7);
ylabel('probability $f(D)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical');
print(gcf, '-dpdf', '-r600', ...
   sprintf('../images/%d_part_b_log_plot.pdf',number_of_trials));

end
%============================
function r= pdf_low_let(D)
a1=-6.84806*10^(-4);
a2= 4.87285*10^(-6);
a3=-3.28988*10^(-8);
a4= 2.66992*10^(-11);
a5=-0.0073;
A=7.9839;

if D<=600
   r = exp(a1*D+a2*D^2+a3*D^3+a4*D^4);
else
   r = A*exp(a5*D);
end
end
```

2.4.3 Problem 3

(3) (15 pts) When populations are subjected to relatively high doses of radiation, there is an increased probability of members of the population developing certain kinds of cancer at times following exposure. The incidence of excess cancer has certain characteristics: there is a "latency period" where no excess incidence beyond the background rate is observed, followed by a relatively rapid rise to a "plateau period" where the cancer rate is definitely observed to be higher than (in excess of) the background rate. In the simplest representation of this excess incidence, the plateau period looks like a square wave, but in reality the transition between the latency period and the plateau period is smooth. In order to model this, consider the following normalized PDF:

$$f_{\text{excess}} = \frac{1}{2}\left[\tanh(x-l) - \tanh[0.5(x-(l+p))]\right]$$

Here x is the number of years following exposure, l is the latency period (years) and p is the plateau period (years). For pancreatic cancer, the latency and plateau periods are 15 and 30 years respectively. Replicate this normalized PDF with the rejection algorithm, sorting incidences of excess cancer into bins of width 0.1 years for 60 years following exposure. Run 10^7 trials and plot your results.

Note: As an aside, note that $f_{\text{excess}} = 1$ does not mean there is a 100% chance of getting cancer; it means that the incidence of excess cancers is 100% of the observed rate within the plateau period.

Figure 2.39: problem 3 description

The following trials were used $\{10^4, 10^5, 10^6, 10^7\}$. The last amount is the one asked for in this problem. For each case, analytic pdf is plotted against the generated one to compare. As more trials used, the approximation to the true pdf is improved.

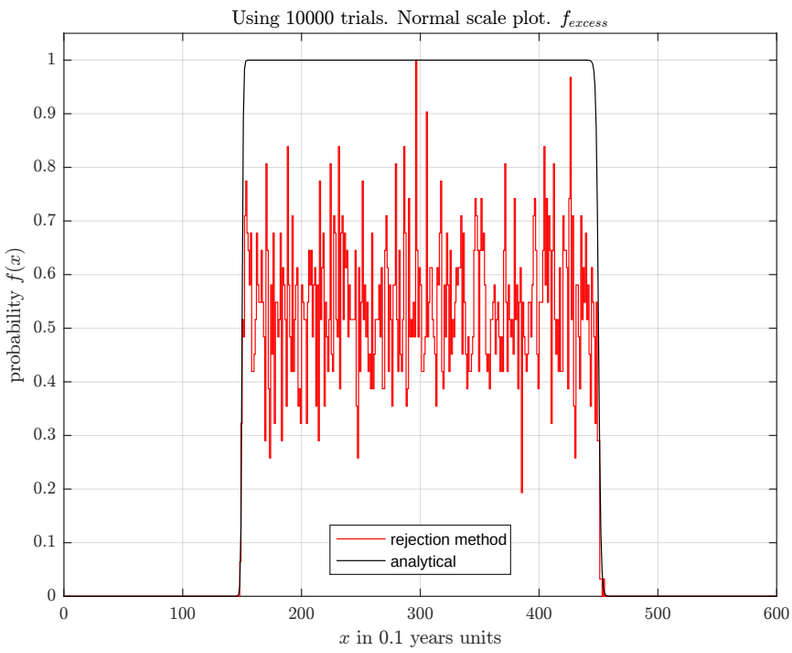


Figure 2.40: 10^4 trials, normal plot

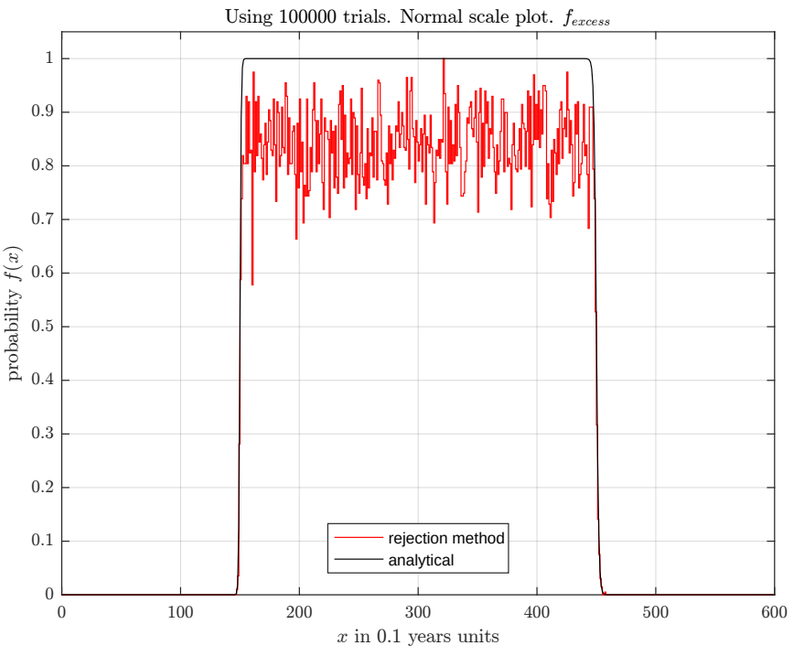


Figure 2.41: 10^5 trials, normal plot

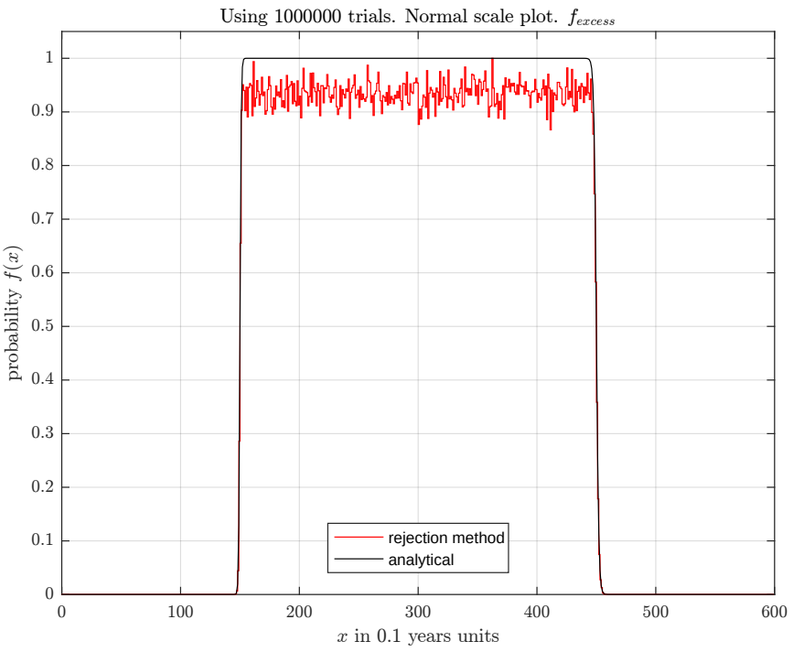


Figure 2.42: 10^6 trials, normal plot

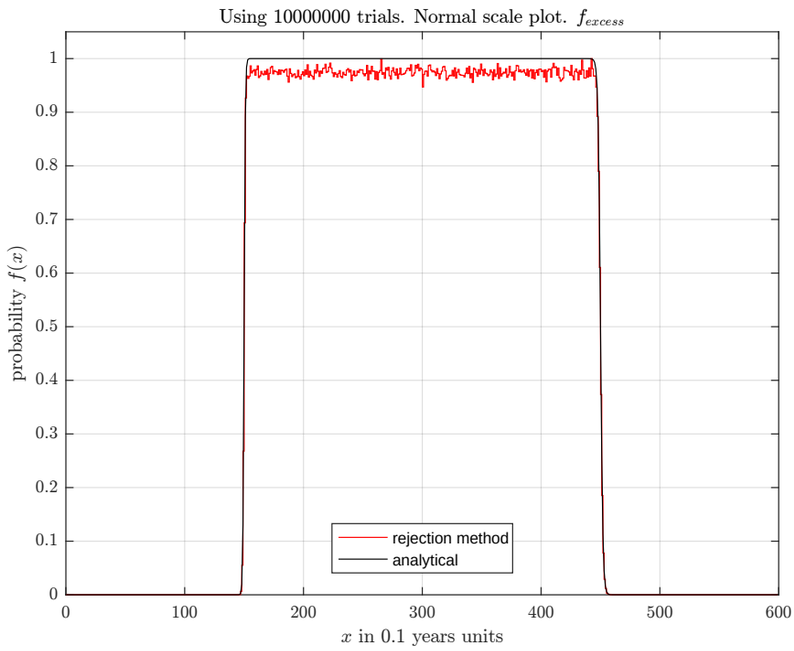


Figure 2.43: 10^7 trials, normal plot

The following plots adds 10^8 number of trials in order to see how close the generated pdf will be to the theoretical pdf in the plateau region (the region between 15 and 45 years). Generated pdf becomes closer to the theoretical pdf, but there always be a small gap at the top. With more trials, the generated pdf will converge to the theoretical pdf.

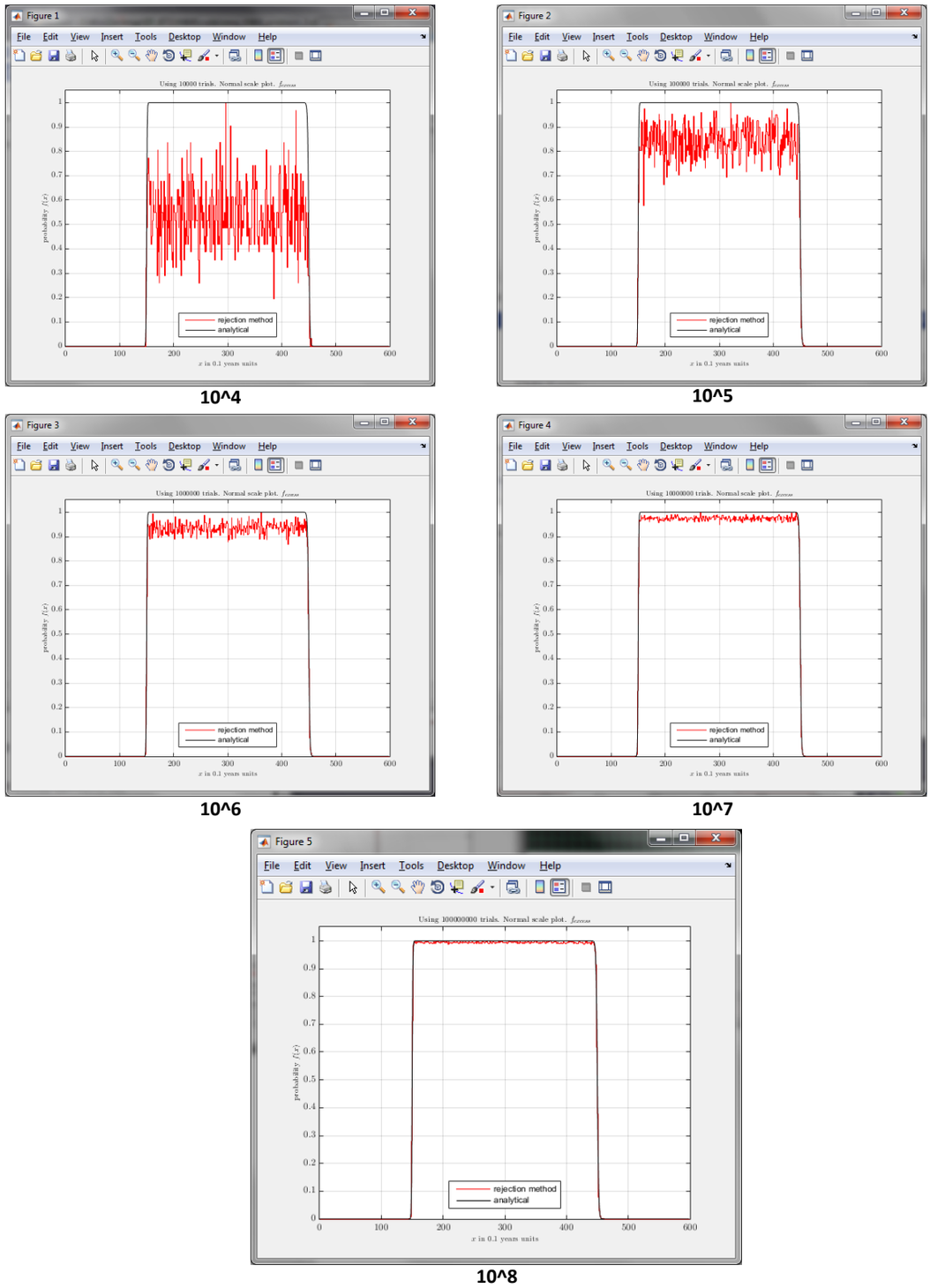


Figure 2.44: $\{10^4, 10^5, 10^6, 10^7, 10^8\}$ trials all on one plot.

```
function nma_HW4_problem_3()
%Problem 2, Low Let radiation. EMA 471, HW4
%by Nasser M. Abbasi
%
%
close all; clc;

ntrials = [10^4 10^5 10^6 10^7];
for i=1:length(ntrials)
    process(ntrials(i));
end
end
%==================================
function process(N)
%N: number of trials

%Initialize random number generator
rng('default');
rng(1);

bin_width = 0.1;
MAX       = 60/bin_width;  %60 years with 0.1 divisions
counts  = zeros(N,1);      %to save accepted trials into
k       = 0;               %counter for keeping accepted

for i=1:N
    z_1 = rand();
    z_2 = rand();
%convert to actual x from 0..600 (60 years, with .1 division)
    x = z_1*MAX;

    if z_2 < find_pdf_at(x,bin_width)
       k = k + 1;
       counts(k) = x;
    end
end

generate_plots(N,counts(1:k),MAX,bin_width);
end
%=======================
function generate_plots(number_of_trials,counts,MAX,bin_width)
analytical_pdf = find_pdf_at((1/2)* ...
                      bin_width:MAX-(1/2)*bin_width,bin_width);
generate_normal_plots(number_of_trials,counts,...
                                  MAX,analytical_pdf,bin_width);
%generate_log_plots(number_of_trials,counts,...
%                   MAX,analytical_pdf,bin_width);
end
%=======================
function generate_normal_plots(number_of_trials,counts,...
                                  MAX,analytical_pdf,bin_width)
figure;
edges     = 0:MAX;
x_bins    = histcounts(counts,edges);

%use center of bins
x         = (1/2)*bin_width:MAX-(1/2)*bin_width;

%notice, we divide by the full area
stairs(x,x_bins/max(x_bins),'r');
hold on;
plot(x,analytical_pdf,'k');
ylim([0 1.05]);
```

```
grid;

title(sprintf('Using %d trials. Normal scale plot. $f_{excess}$',...
        number_of_trials),'interpreter','Latex','Fontsize',7);
xlabel('$x$ in 0.1 years units','interpreter','Latex','Fontsize',7);
ylabel('probability $f(x)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical','Location','south');
print(gcf, '-dpdf', '-r600', ...
     sprintf('../images/%d_problem_3_normal_plot.pdf',...
             number_of_trials));

end
%================================
function generate_log_plots(number_of_trials,counts,MAX,...
                                        analytical_pdf,bin_width)
figure;
edges     = 0:MAX;
x_bins    = histcounts( counts,edges);

%use center of bins
x         = (1/2)*bin_width:MAX-(1/2)*bin_width;
semilogy(x,x_bins/max(x_bins));
hold on;
semilogy(x,analytical_pdf);
%grid;

title(sprintf('Using %d trials. semilogy scale plot. $f_{excess}$',...
       number_of_trials),'interpreter','Latex','Fontsize',7);
xlabel('$x$','interpreter','Latex','Fontsize',7);
ylabel('probability $f(D)$','interpreter','Latex','Fontsize',7);
set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
legend('rejection method','analytical');
print(gcf, '-dpdf', '-r600', ...
 sprintf('../images/%d_problem_3_log_plot.pdf',number_of_trials));

end
%============================
function r= find_pdf_at(x,bin_width)
L = 15/bin_width;  %Latency in years
p = 30/bin_width;  %plateau in years

 r = (1/2)*(tanh(x-L)-tanh(0.5*(x-(L+p))));

end
```

2.5 HW 5

2.5.1 Problem 1

EP 471 – Homework #5
Due: Thursday, April 7th, 2016

(1) (8 pts) It is stated without proof in Exercise 15 that Gaussian quadrature of order N produces an exact result when applied to the integration of a polynomial of order $2N-1$. Consider the following polynomials in the interval $0 \le x \le 4$:

(a) $f_1(x) = x^5$

(b) $f_2(x) = x^7$

Integrate (a) and (b) over the interval $0 \le x \le 4$ using Gaussian quadrature of $N = 3$ and 4 respectively and compare your results to the analytical values. Does the quadrature of the appropriate order produce an exact match?

Figure 2.45: problem 1 description

A polynomial $f(x)$ of order p is integrated exactly with the Gaussian quadrature method using $\frac{p+1}{2}$ number of Gaussian points.

Hence $f_1(x) = x^5$ needs $\frac{5+1}{2} = 3$ Gaussian points and $f_2(x) = x^7$ needs $\frac{7+1}{2} = 4$ Gaussian points for exact result.

The integral $\int_a^b f(x)\,dx$ is first converted to be in the domain $\{-1, +1\}$ as follows

$$\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{+1} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)dt$$

For a polynomial $f(x)$ of order $p = 3$, two Gaussian points are needed to evaluate the above integral exactly. Therefore the above integral simplifies to

$$\int_a^b f(x)\,dx = A\left(w_1 f(At_1 + B) + w_2 f(At_2 + B)\right)$$

Where

$$A = \frac{b-a}{2}$$
$$B = \frac{b+a}{2}$$

And w_i is the weight at location t_i. The weights and location of the weights are obtained from tables. For higher order polynomials, more points and weights are needed.

In general, using N points the integral is

$$\begin{aligned}\int_a^b f(x)\,dx &= \frac{b-a}{2}\sum_{i=1}^{N} w_i f\left(\frac{b-a}{2}t_i + \frac{a+b}{2}\right)\\ &= A\sum_{i=1}^{N} w_i f(At_i + B)\end{aligned} \tag{1}$$

The program nma_EMA_471_HW5_problem_1.m integrates the above two polynomials $f_1(x), f_2(x)$ using Gaussian quadrature method (1) using $N = 3$ and $N = 4$ points respectively and compares the result of each to the analytical solution. The following table shows the result.

Table 2.7: HW5, problem 1 result

function	analytical result	N	Gaussian quadrature result
$f_1(x) = x^4$	$\int_0^4 x^4\,dx = \frac{2048}{3} = 682.666666666667$	3	682.666666666667
$f_1(x) = x^7$	$\int_0^4 x^7\,dx = 8192$	4	8192

The result is exact. Note: The above Matlab program used the exact weights and points for Gaussian quadrature as given in https://en.wikipedia.org/wiki/Gaussian_quadrature

```
function nma_EMA_471_HW5_problem_1()
%Solves problem 1, HW5, EMA 471
%Nasser M. Abbasi

%reference  https://en.wikipedia.org/wiki/Gaussian_quadrature for points
%and weights. These are exact.

gauss_3_points=[-sqrt(3/5) , 5/9;    %point, weight   per row
                0          , 8/9;
                sqrt(3/5)  , 5/9];

gauss_4_points=[-sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36;
                -sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
                 sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
                 sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36];

f1=@(x) x.^5;
integrate(f1,0,4,gauss_3_points)

f1=@(x) x.^7;
integrate(f1,0,4,gauss_4_points)

end
%==================================
function the_sum = integrate(f,from,to,g)
%INPUT:   f  is handle to function to integrate
%         from,to these are lower and upper integral bounds
%         g  This is matrix of Gaussian quadrature. first column is points
%            second column is corresponding weights

A = (to-from)/2;
B = (to+from)/2;
i = 1:size(g,1);
the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) );   %vectored sum
end
```

2.5.2 Problem 2

(2) (12 pts) In one of our exercises we evaluated the following function by Gaussian quadrature:

$$I = \int_0^{2.5} \frac{\sin^2 x \cdot \sin(x^2)}{(1+x^2)^2} dx$$

using 2, 3 and 4 gauss points. In this case, the domain is large enough and the function changes sharply enough that we don't get very good agreement with Matlab's `quad` utility even for 4 gauss points. To get better agreement, we could use a larger number of Gauss points, or break the domain into pieces, $0 \le x \le 1.25$ and $1.25 \le x \le 2.5$, for example.

The following extends the table in Exercise 15 to 5, 6, 7 and 8 gauss points:

Number of terms, N	Values of t (t_i)	Weighting factor	Valid up to degree
5	0	0.56888889	9
	±0.53846931	0.47862867	
	±0.90617985	0.23692689	
6	±0.23861918	0.46791393	11
	±0.66120939	0.36076157	
	±0.93246951	0.17132449	
7	0	0.41795918	13
	±0.40584515	0.38183005	
	±0.74153119	0.27970539	
	±0.94910791	0.12948497	
8	±0.18343464	0.36268378	15
	±0.52553241	0.31370665	
	±0.79666648	0.22238103	
	±0.96028986	0.10122854	

(a) Keeping the original domain, $0 \le x \le 2.5$, how does the agreement with Matlab's `quad` utility improve with 5, 6, 7 and 8 point Gaussian quadrature?
(b) Breaking the domain into two pieces, $0 \le x \le 1.25$ and $1.25 \le x \le 2.5$, evaluate the integral using 5 and 6 gauss points in each subdomain. How does the agreement compare with results from Matlab's `quad` utility now?

Figure 2.46: problem 2 description

2.5.2.1 part a

The program nma_EMA_471_HW5_problem_2_part_a.m implements the first part of this problem.

The following table shows the result of the computation. It shows the result of the integral using Gaussian quadrature for different number of points with the relative error against Matlab's Quad (integral) command.

Number of points	relative error (percentage)	value of integral
2	79.845442	0.023377470178383
3	23.892753	0.143704430262801
4	3.060319	0.112441294428254
5	4.692010	0.121433298541329
6	0.019015	0.116013045399658
7	0.011820	0.116004700249995
8	0.001535	0.115989208171974

Table 2.8: Gaussian quadrature using different points $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2}\, dx$. Compared with Matlab Quad result of 0.115990989197426 for same integral

The following is a plot of the above data

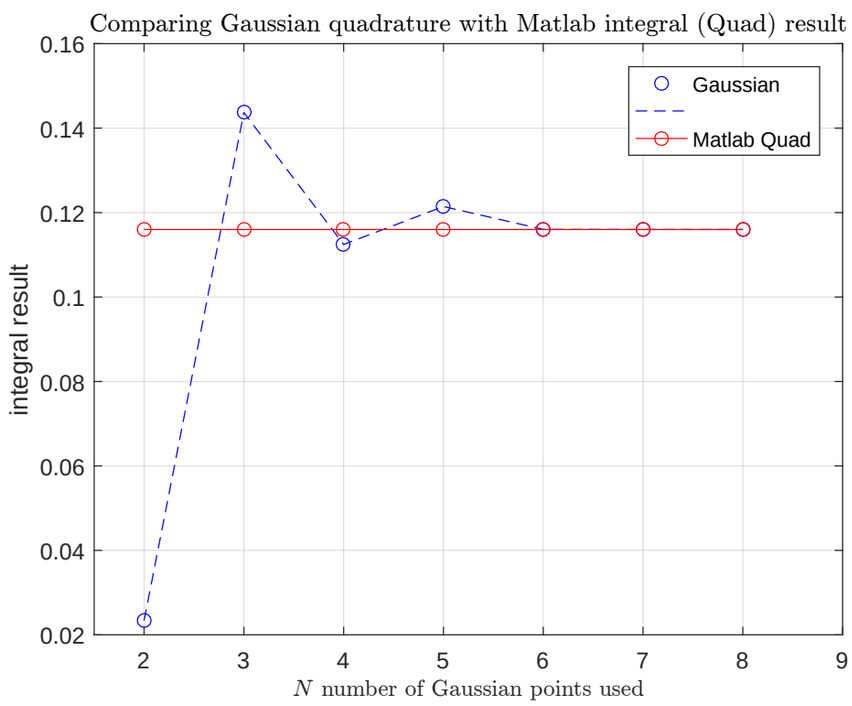


Figure 2.47: Comparing Gaussian quadrature with Matlab's integral result

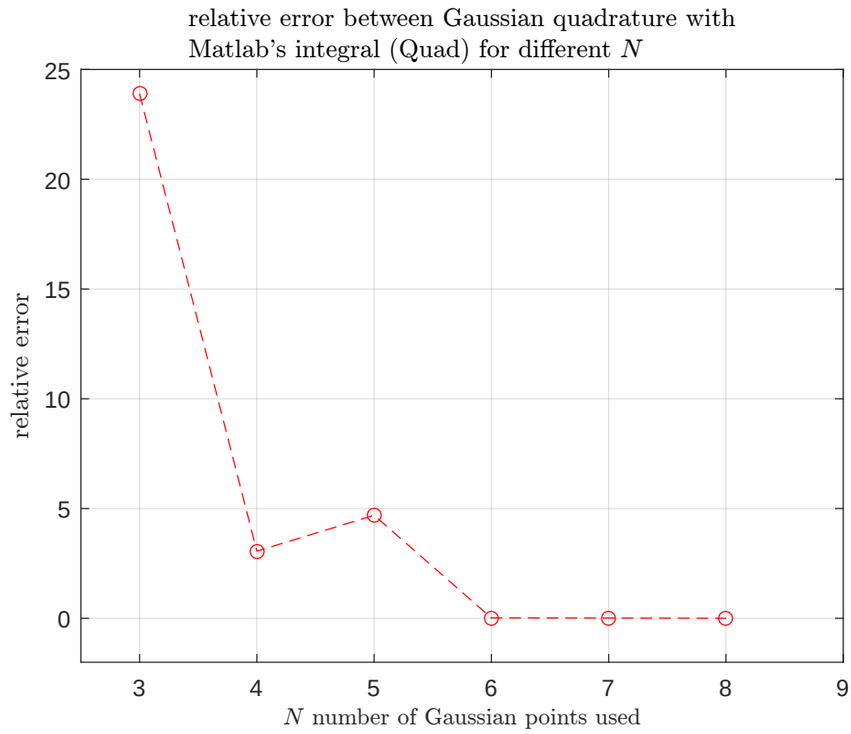


Figure 2.48: Relative error for different N values

2.5.2.2 part b

The program nma_EMA_471_HW5_problem_2_part_b.m implements the second part of this problem. By breaking the domain into 2 parts, the following table shows the result of the computation. It shows the result of the integral using Gaussian quadrature for 5 and 6 points with the relative error against Matlab's Quad (integral) command. The integration was done on each subdomain and the results added.

Number of points	relative error (percentage)	value of integral using Gaussian quadrature
5	1.231392	0.114562685084637
6	0.000303	0.115990636860983

Table 2.9: Gaussian quadrature using 5 and 6 points $\int_0^{1.25} \frac{\sin^2 x \sin x^2}{(1+x^2)^2}\,dx + \int_{1.25}^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2}\,dx$. Compared with Matlab integral result of $\int_0^{2.5} \frac{\sin^2 x \sin x^2}{(1+x^2)^2}\,dx = 0.115990989197426$

The above shows clearly that by breaking the domain into two smaller parts, and adding each result, the final result of Gaussian quadrature improved compared to part(a) where one large domain was used. This makes sense. Because we have effectively used more sampling points in part(b) compared to part(a) when looking at the whole domain.

This shows that, to obtain more accuracy using Gaussian quadrature, and still use the same number of points N, then we can break the domain into smaller regions, and use N on each region, and add the result obtained from each region.

To see the difference between part(a) and (b) more clearly, the following table shows the result for 5 and 6 points side by side from part(a) and part(b). The table below shows the relative error is much smaller for part(b).

Number of points	relative error part(b)	relative error part(a)
5	1.231392	4.692010
6	0.000303	0.019015

Table 2.10: Gaussian quadrature using 5 and 6 points. Comparing part(a) and part(b) relative error against Matlab's Quad

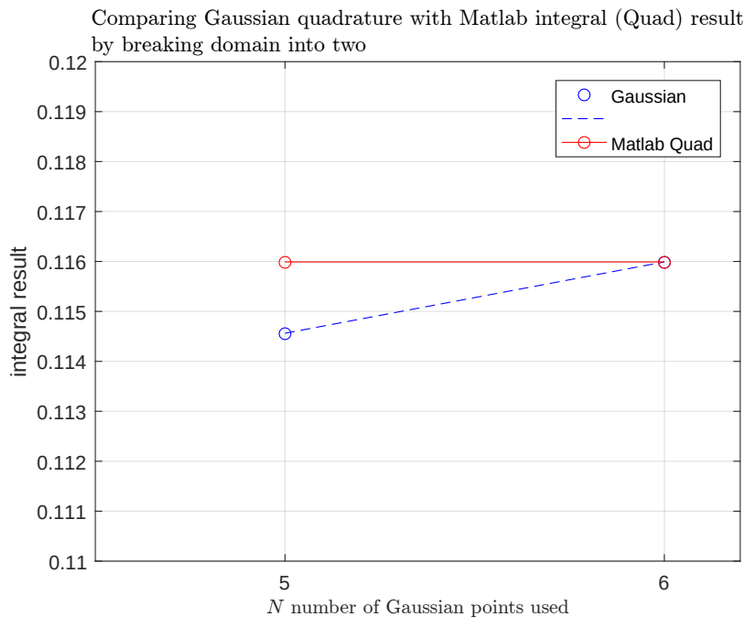


Figure 2.49: Comparing Gaussian quadrature with Matlab's integral result, part(b)

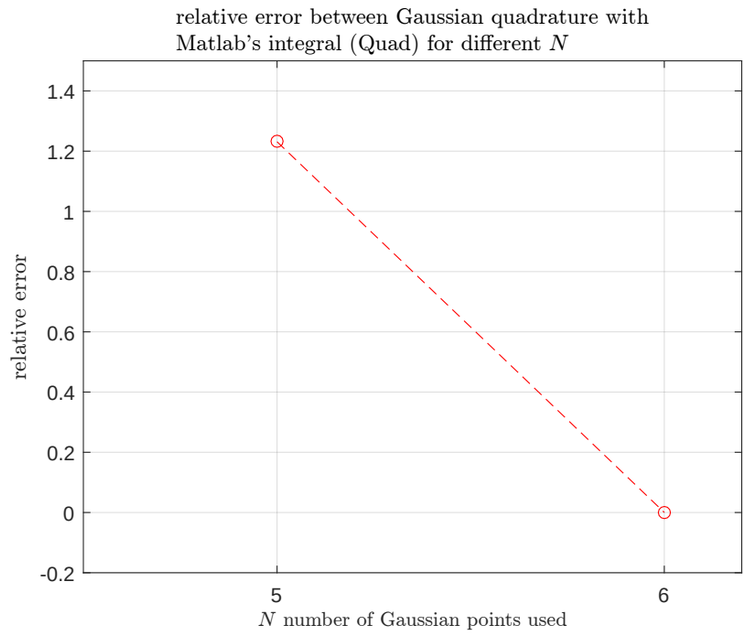


Figure 2.50: Relative error for different N values, part(b)

```
function nma_EMA_471_HW5_problem_2_part_a()
%Solves problem 2, part(a) HW5, EMA 471
%Nasser M. Abbasi

close all; clc;

gauss_2_points=[-0.57735027 , 1;    %point, weight   per row
                 0.57735027  , 1
               ];
```

```
10
11 gauss_3_points=[-sqrt(3/5) , 5/9;    %point, weight   per row
12                 0          , 8/9;
13                 sqrt(3/5)  , 5/9];
14
15 gauss_4_points=[-sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36;
16                 -sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
17                  sqrt(3/7- 2/7*sqrt(6/5)) , (18+sqrt(30))/36;
18                  sqrt(3/7+ 2/7*sqrt(6/5)) , (18-sqrt(30))/36];
19
20 gauss_5_points=[0                          , 128/225;
21             -(1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
22              (1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
23             -(1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900;
24              (1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900];
25
26 gauss_6_points=[0.238619186083197 , 0.467913934572691;
27                -0.238619186083197 , 0.467913934572691;
28                 0.661209386466265 , 0.360761573048139;
29                -0.661209386466265 , 0.360761573048139;
30                 0.932469514203152 , 0.171324492379170;
31                -0.932469514203152 , 0.171324492379170];
32
33 gauss_7_points=[0                 , 0.417959183673469;
34                 0.405845151377397 , 0.381830050505119;
35                -0.405845151377397 , 0.381830050505119;
36                 0.741531185599394 , 0.279705391489277;
37                -0.741531185599394 , 0.279705391489277;
38                 0.949107912342759 , 0.129484966168870;
39                -0.949107912342759 , 0.129484966168870];
40
41
42 gauss_8_points=[0.183434642495650 , 0.362683783378361;
43                -0.183434642495650 , 0.362683783378361;
44                 0.525532409916329 , 0.313706645877887;
45                -0.525532409916329 , 0.313706645877887;
46                 0.796666477413627 , 0.222381034453374;
47                -0.796666477413627 , 0.222381034453374;
48                 0.960289856497536 , 0.101228536290376;
49                -0.960289856497536 , 0.101228536290376];
50
51 f=@(x) sin(x).^2 .* sin(x.^2) ./ (1+x.^2).^2 ;
52 x_min     = 0;
53 x_max     = 2.5;
54 data      = zeros(7,4);
55 chk       =  integral(f,x_min,x_max);
56 data(:,1) = chk;
57
58 for i=1:size(data,1)
59     switch i
60       case 1
61            data(i,2) = integrate(f,x_min,x_max,gauss_2_points);
62            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
63            data(i,4) = 2;
64       case 2
65            data(i,2) = integrate(f,x_min,x_max,gauss_3_points);
66            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
67            data(i,4) = 3;
68       case 3
69            data(i,2) = integrate(f,x_min,x_max,gauss_4_points);
70            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
71            data(i,4) = 4;
72       case 4
```

```
            data(i,2) = integrate(f,x_min,x_max,gauss_5_points);
            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
            data(i,4) = 5;
       case 5
            data(i,2) = integrate(f,x_min,x_max,gauss_6_points);
            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
            data(i,4) = 6;
       case 6
            data(i,2) = integrate(f,x_min,x_max,gauss_7_points);
            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
            data(i,4) = 7;
       case 7
            data(i,2) = integrate(f,x_min,x_max,gauss_8_points);
            data(i,3) = 100*abs(chk - data(i,2))/abs(chk);
            data(i,4) = 8;
    end
end

figure;
plot(data(:,4),data(:,2),'bo',data(:,4),data(:,2),'b--');
hold on;
plot(data(:,4),data(:,1),'r-o');
xlim([1.5,9]);
xlabel('$N$ number of Gaussian points used', ...
        'interpreter','Latex','Fontsize',10);
ylabel('integral result');
title('Comparing Gaussian quadrature with Matlab integral (Quad) result',...
      'interpreter','Latex');
legend('Gaussian','','Matlab Quad');
grid;

figure;
plot(data(2:end,4),data(2:end,3),'ro',data(2:end,4),...
       data(2:end,3),'r--');
xlabel('$N$ number of Gaussian points used','interpreter',...
        'Latex','Fontsize',10);
ylabel('relative error','interpreter','Latex');
title({'relative error between Gaussian quadrature with',...
        'Matlab''s integral (Quad) for different $N$'},...
         'interpreter','Latex');
grid;
xlim([2.5,9]);
ylim([-2,25]);




end
%==================================
function the_sum = integrate(f,from,to,g)
%INPUT:   f  is handle to function to integrate
% from,to these are lower and upper integral bounds
% g  This is matrix of Gaussian quadrature. first column is points
% second column is corresponding weights

A = (to-from)/2;
B = (to+from)/2;
i = 1:size(g,1);
the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) );   %vectored sum
end
```

```
function nma_EMA_471_HW5_problem_2_part_b()
%Solves problem 2, part(b) HW5, EMA 471
%Nasser M. Abbasi

```

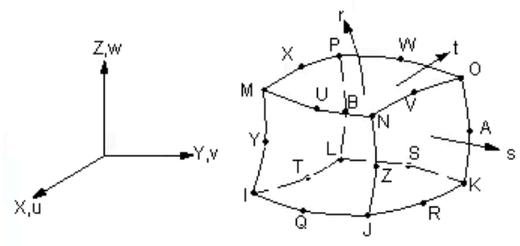
```
close all; clc;


gauss_5_points=[0                          , 128/225;
                -(1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
                 (1/3)*sqrt(5-2*sqrt(10/6)) , (322+13*sqrt(70))/900;
                -(1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900;
                 (1/3)*sqrt(5+2*sqrt(10/6)) , (322-13*sqrt(70))/900];

gauss_6_points=[0.238619186083197 , 0.467913934572691;
               -0.238619186083197 , 0.467913934572691;
                0.661209386466265 , 0.360761573048139;
               -0.661209386466265 , 0.360761573048139;
                0.932469514203152 , 0.171324492379170;
               -0.932469514203152 , 0.171324492379170];


f=@(x) sin(x).^2 .* sin(x.^2) ./ (1+x.^2).^2 ;
data      = zeros(2,4);
chk       =  integral(f,0,2.5);
data(:,1) = chk;

data(1,2) = integrate(f,0,1.25,gauss_5_points)+integrate(f,1.25,2.5,gauss_5_points);
data(1,3) = 100*abs(chk - data(1,2))/abs(chk);
data(1,4) = 5;

data(2,2) = integrate(f,0,1.25,gauss_6_points)+integrate(f,1.25,2.5,gauss_6_points);
data(2,3) = 100*abs(chk - data(2,2))/abs(chk);
data(2,4) = 6;

figure;
plot(data(:,4),data(:,2),'bo',data(:,4),data(:,2),'b--');
hold on;
plot(data(:,4),data(:,1),'r-o');
xlim([1.5,9]);
xlabel('$N$ number of Gaussian points used','interpreter','Latex','Fontsize',10);
ylabel('integral result');
title({'Comparing Gaussian quadrature with Matlab integral (Quad) result', ...
       'by breaking domain into two'},'interpreter','Latex');
legend('Gaussian','','Matlab Quad');
grid;
xlim([4.5,6.2]);
ylim([.11,.12]);
ax = gca;
ax.XTick = [5 6];


figure;
plot(data(:,4),data(:,3),'ro',data(:,4),data(:,3),'r--');
xlabel('$N$ number of Gaussian points used','interpreter','Latex','Fontsize',10);
ylabel('relative error','interpreter','Latex');
title({'relative error between Gaussian quadrature with',...
        'Matlab''s integral (Quad) for different $N$'},'interpreter','Latex');
grid;
xlim([4.5,6.2]);
ylim([-.2,1.5]);
ax = gca;
ax.XTick = [5 6];



end
%==================================
```

```
function the_sum = integrate(f,from,to,g)
%INPUT:   f  is handle to function to integrate
%         from,to these are lower and upper integral bounds
%         g  This is matrix of Gaussian quadrature. first column is points
%            second column is corresponding weights

A = (to-from)/2;
B = (to+from)/2;
i = 1:size(g,1);
the_sum = A * sum( g(i,2) .* f(A*g(i,1)+B) );   %vectored sum
end
```

2.5.3 Problem 3

(3) (20 pts) When using commercial software such as ANSYS, one can find the interpolation functions used for various element types in the Shape Functions section of the on-line Theory Manual. The most general 3D continuum elements are 20-node brick elements, and this figure from ANSYS lists the interpolation scheme:

Figure 12.17: 20-Node Brick Element



These shape functions are used for 20-node solid elements such as SOLID90:

$$
\begin{aligned}
u = \frac{1}{8}(&u_I(1-s)(1-t)(1-r)(-s-t-r-2) + u_J(1+s)(1-t)(1-r)(s-t-r-2)\\
&+ u_K(1+s)(1+t)(1-r)(s+t-r-2) + u_L(1-s)(1+t)(1-r)(-s+t-r-2)\\
&+ u_M(1-s)(1-t)(1+r)(-s-t+r-2) + u_N(1+s)(1-t)(1+r)(s-t+r-2)\\
&+ u_O(1+s)(1+t)(1+r)(s+t+r-2) + u_P(1-s)(1+t)(1+r)(-s+t+r-2))\\
&+ \frac{1}{4}(u_Q(1-s^2)(1-t)(1-r) + u_R(1+s)(1-t^2)(1-r)\\
&+ u_S(1-s^2)(1+t)(1-r) + u_T(1-s)(1-t^2)(1-r)\\
&+ u_U(1-s^2)(1-t)(1+r) + u_V(1+s)(1-t^2)(1+r)\\
&+ u_W(1-s^2)(1+t)(1+r) + u_X(1-s)(1-t^2)(1+r)\\
&+ u_Y(1-s)(1-t)(1-r^2) + u_Z(1+s)(1-t)(1-r^2)\\
&+ u_A(1+s)(1+t)(1-r^2) + u_B(1-s)(1+t)(1-r^2))
\end{aligned}
$$

Note that in this representation, "*r*", "*s*" and "*t*" have replaced "xi" (ξ), "eta" (η) and "zeta" (ζ) as the natural coordinate system variables. The interpolation shown above is for displacement degree-of-freedom *u*, but this same interpolation holds for the other degrees-of-freedom as well as for the coordinates (*x*,*y*,*z*) within the element domain. Calculation of the volume of this element would be accomplished through:

Figure 2.51: problem 3 description

03/28/16

$$V = \int dx\,dy\,dz = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \frac{\partial(x,y,z)}{\partial(r,s,t)}\; dr\; ds\; dt \cong \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} w_i w_j w_k \det J(r_i,s_j,t_k)$$

In the event that we need to find the response of a 3D structure to its own weight, we have to convert the continuously distributed weight density into a series of 20 discrete weights at each of the nodes. The work-equivalent finite element result is that the force at node i (i = I through B in the figure above) is found from:

$$F_i = \int \gamma(x,y,z) f_i(r,s,t)\,dx\,dy\,dz$$

$$= \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \gamma[x(r,s,t),y(r,s,t),z(r,s,t)] f_i(r,s,t) \frac{\partial(x,y,z)}{\partial(r,s,t)}\; dr\; ds\; dt$$

Here $\gamma = \rho g$ is the weight density and may be a function of position within the element. The f_i are the interpolation functions listed in the figure on the previous page. For example, for i = I,

$$f_i = f_I = \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)$$

Aside: The interpolation functions are defined so that they go to one when their associated node is approached from inside the element. All the other interpolation functions go to zero at the same location, so that the interpolated quantity goes to the nodal quantity. In the case of node I, you'll notice from the figure that this corresponds to $r = s = t = -1$.

Given the nodal coordinates listed below for nodes I through B (taken from one element in a mesh prepared using ANSYS), and given the spatial dependence of the mass density, find the z-component forces to be applied to each of the nodes so that the weight is applied to the element in a work-equivalent fashion (assumes **g** points in the –z direction).

Nodal coordinates (x, y, z) are: (all values in units of cm):

Node I:	1	0	0
Node J:	1.1	0	0
Node K:	1.09066	0	−0.086305
Node L:	0.99692	0	−0.078459
Node M:	1.0077	0.16559	0
Node N:	1.1069	0.17203	0
Node O:	1.1035	0.17202	−0.08663
Node P:	1.0046	0.16557	−0.07882
Node Q:	1.05	0	0
Node R:	1.0992	0	−0.043186
Node S:	1.0468	0	−0.082382
Node T:	0.9923	0	−0.039260
Node U:	1.0573	0.16881	0
Node V:	1.1061	0.17202	−0.043328
Node W:	1.0540	0.16880	−0.082725
Node X:	1.0069	0.16558	−0.039418
Node Y:	1.0051	0.082737	0
Node Z:	1.1046	0.085964	0
Node A:	1.1012	0.085938	−0.086550
Node B:	1.0020	0.082709	−0.078731

03/28/16

The mass density (units of g/cm^3) is due to a functionally graded material and has the functional form:

$$\rho = \rho_o(x^2 + z^2) \qquad ; \qquad \rho_o = 1$$

Figure 2.52: more problem 3 description

2.5.3.1 shape functions

The following are the shape functions

$$f_I = \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)$$
$$f_J = \frac{1}{8}(1-r)(s+1)(1-t)(-r+s-t-2)$$
$$f_K = \frac{1}{8}(1-r)(s+1)(t+1)(-r+s+t-2)$$
$$f_L = \frac{1}{8}(1-r)(1-s)(t+1)(-r-s+t-2)$$
$$f_M = \frac{1}{8}(r+1)(1-s)(1-t)(r-s-t-2)$$
$$f_N = \frac{1}{8}(r+1)(s+1)(1-t)(r+s-t-2)$$
$$f_O = \frac{1}{8}(r+1)(s+1)(t+1)(r+s+t-2)$$
$$f_P = \frac{1}{8}(r+1)(1-s)(t+1)(r-s+t-2)$$
$$f_Q = \frac{1}{4}(1-r)\left(1-s^2\right)(1-t)$$
$$f_R = \frac{1}{4}(1-r)(s+1)\left(1-t^2\right)$$
$$f_S = \frac{1}{4}(1-r)\left(1-s^2\right)(t+1)$$
$$f_T = \frac{1}{4}(1-r)(1-s)\left(1-t^2\right)$$
$$f_U = \frac{1}{4}(r+1)\left(1-s^2\right)(1-t)$$
$$f_V = \frac{1}{4}(r+1)(s+1)\left(1-t^2\right)$$
$$f_W = \frac{1}{4}(r+1)\left(1-s^2\right)(t+1)$$
$$f_X = \frac{1}{4}(r+1)(1-s)\left(1-t^2\right)$$
$$f_Y = \frac{1}{4}\left(1-r^2\right)(1-s)(1-t)$$
$$f_Z = \frac{1}{4}\left(1-r^2\right)(s+1)(1-t)$$
$$f_A = \frac{1}{4}\left(1-r^2\right)(s+1)(t+1)$$
$$f_B = \frac{1}{4}\left(1-r^2\right)(1-s)(t+1)$$

To obtain the Jacobian, we need to obtain the determinant of

$$\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{pmatrix}$$

Where

$$x(r,s,t) = \sum_{i=I}^{I=B} x_i f_i(r,s,t)$$
$$y(r,s,t) = \sum_{i=I}^{I=B} y_i f_i(r,s,t)$$
$$z(r,s,t) = \sum_{i=I}^{I=B} z_i f_i(r,s,t)$$

Once we determine $x(r,s,t), y(r,s,t), z(r,s,t)$ from the above, then we can determine the Jacobian determinant at each Gaussian point.

2.5.3.2 $x(s,t,r)$ terms

From the above sum, expanding $x(r,s,t)$ gives

$x_=x_I f_I + x_J f_J + x_K f_K + x_L f_L + x_M f_M + x_N f_N + x_O f_O + x_P f_P + x_Q f_Q + x_R f_R + x_S f_S + x_T f_T + x_U f_U + x_V f_V$

Substituting the values of f_i into the above results in

$$\begin{aligned}
x(r,s,t) =& x_I \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)+ \\
& x_J \frac{1}{8}(1-r)(s+1)(1-t)(-r+s-t-2)+ \\
& x_K \frac{1}{8}(1-r)(s+1)(t+1)(-r+s+t-2)+ \\
& x_L \frac{1}{8}(1-r)(1-s)(t+1)(-r-s+t-2)+ \\
& x_M \frac{1}{8}(r+1)(1-s)(1-t)(r-s-t-2)+ \\
& x_N \frac{1}{8}(r+1)(s+1)(1-t)(r+s-t-2)+ \\
& x_O \frac{1}{8}(r+1)(s+1)(t+1)(r+s+t-2)+ \\
& x_P \frac{1}{8}(r+1)(1-s)(t+1)(r-s+t-2)+ \\
& x_Q \frac{1}{4}(1-r)\left(1-s^2\right)(1-t)+ \\
& x_R \frac{1}{4}(1-r)(s+1)\left(1-t^2\right)+ \\
& x_S \frac{1}{4}(1-r)\left(1-s^2\right)(t+1)+ \\
& x_T \frac{1}{4}(1-r)(1-s)\left(1-t^2\right)+ \\
& x_U \frac{1}{4}(r+1)\left(1-s^2\right)(1-t)+ \\
& x_V \frac{1}{4}(r+1)(s+1)\left(1-t^2\right)+ \\
& x_W \frac{1}{4}(r+1)\left(1-s^2\right)(t+1)+ \\
& x_X \frac{1}{4}(r+1)(1-s)\left(1-t^2\right)+ \\
& x_Y \frac{1}{4}\left(1-r^2\right)(1-s)(1-t)+ \\
& x_Z \frac{1}{4}\left(1-r^2\right)(s+1)(1-t)+ \\
& x_A \frac{1}{4}\left(1-r^2\right)(s+1)(t+1)+ \\
& x_B \frac{1}{4}\left(1-r^2\right)(1-s)(t+1)
\end{aligned}$$

Taking partial derivative of the above w.r.t. r, s, t in turn gives the following

$$\begin{aligned}
\frac{\partial x}{\partial r} =& x_I\left(\frac{1}{8}(s-1)(t-1)(2r+s+t+1)\right)+\\
& x_J\left(\frac{1}{8}(s+1)(t-1)(-2r+s-t-1)\right)+\\
& x_K\left(-\frac{1}{8}(s+1)(t+1)(-2r+s+t-1)\right)+\\
& x_L\left(-\frac{1}{8}(s-1)(t+1)(2r+s-t+1)\right)+\\
& x_M\left(-\frac{1}{8}(s-1)(t-1)(-2r+s+t+1)\right)+\\
& x_N\left(-\frac{1}{8}(s+1)(t-1)(2r+s-t-1)\right)+\\
& x_O\left(\frac{1}{8}(s+1)(t+1)(2r+s+t-1)\right)+\\
& x_P\left(\frac{1}{8}(s-1)(t+1)(-2r+s-t+1)\right)+\\
& x_Q\left(-\frac{1}{4}\left(s^2-1\right)(t-1)\right)+\\
& x_R\left(\frac{1}{4}(s+1)\left(t^2-1\right)\right)+\\
& x_S\left(\frac{1}{4}\left(s^2-1\right)(t+1)\right)+\\
& x_T\left(-\frac{1}{4}(s-1)\left(t^2-1\right)\right)+\\
& x_U\left(\frac{1}{4}\left(s^2-1\right)(t-1)\right)+\\
& x_V\left(-\frac{1}{4}(s+1)\left(t^2-1\right)\right)+\\
& x_W\left(-\frac{1}{4}\left(s^2-1\right)(t+1)\right)+\\
& x_X\left(\frac{1}{4}(s-1)\left(t^2-1\right)\right)+\\
& x_Y\left(-\frac{1}{2}r(s-1)(t-1)\right)+\\
& x_Z\left(\frac{1}{2}r(s+1)(t-1)\right)+\\
& x_A\left(-\frac{1}{2}r(s+1)(t+1)\right)+\\
& x_B\left(\frac{1}{2}r(s-1)(t+1)\right)
\end{aligned}$$

In the Matlab implementation,the terms r, s, t in the above expression are the Gaussian integration points along the three directions. Similarly, we now find $\frac{\partial x}{\partial s}$ as above. This

results in

$$
\begin{aligned}
\frac{\partial x}{\partial s} =& x_I \left(\frac{1}{8}(r-1)(t-1)(r+2s+t+1) \right) + \\
& x_J \left(-\frac{1}{8}(r-1)(t-1)(r-2s+t+1) \right) + \\
& x_K \left(\frac{1}{8}(r-1)(t+1)(r-2s-t+1) \right) + \\
& x_L \left(-\frac{1}{8}(r-1)(t+1)(r+2s-t+1) \right) + \\
& x_M \left(\frac{1}{8}(r+1)(t-1)(r-2s-t-1) \right) + \\
& x_N \left(-\frac{1}{8}(r+1)(t-1)(r+2s-t-1) \right) + \\
& x_O \left(\frac{1}{8}(r+1)(t+1)(r+2s+t-1) \right) + \\
& x_P \left(-\frac{1}{8}(r+1)(t+1)(r-2s+t-1) \right) + \\
& x_Q \left(-\frac{1}{2}(r-1)s(t-1) \right) + \\
& x_R \left(\frac{1}{4}(r-1)\left(t^2-1\right) \right) + \\
& x_S \left(\frac{1}{2}(r-1)s(t+1) \right) + \\
& x_T \left(-\frac{1}{4}(r-1)\left(t^2-1\right) \right) + \\
& x_U \left(\frac{1}{2}(r+1)s(t-1) \right) + \\
& x_V \left(-\frac{1}{4}(r+1)\left(t^2-1\right) \right) + \\
& x_W \left(-\frac{1}{2}(r+1)s(t+1) \right) + \\
& x_X \left(\frac{1}{4}(r+1)\left(t^2-1\right) \right) + \\
& x_Y \left(-\frac{1}{4}\left(r^2-1\right)(t-1) \right) + \\
& x_Z \left(\frac{1}{4}\left(r^2-1\right)(t-1) \right) + \\
& x_A \left(-\frac{1}{4}\left(r^2-1\right)(t+1) \right) + \\
& x_B \left(\frac{1}{4}\left(r^2-1\right)(t+1) \right)
\end{aligned}
$$

Similarly, we now find $\frac{\partial x}{\partial t}$ as above. This results in

$$\begin{aligned}
\frac{\partial x}{\partial t} =& x_I\left(\frac{1}{8}(r-1)(s-1)(r+s+2t+1)\right)+\\
& x_J\left(-\frac{1}{8}(r-1)(s+1)(r-s+2t+1)\right)+\\
& x_K\left(\frac{1}{8}(r-1)(s+1)(r-s-2t+1)\right)+\\
& x_L\left(-\frac{1}{8}(r-1)(s-1)(r+s-2t+1)\right)+\\
& x_M\left(\frac{1}{8}(r+1)(s-1)(r-s-2t-1)\right)+\\
& x_N\left(-\frac{1}{8}(r+1)(s+1)(r+s-2t-1)\right)+\\
& x_O\left(\frac{1}{8}(r+1)(s+1)(r+s+2t-1)\right)+\\
& x_P\left(-\frac{1}{8}(r+1)(s-1)(r-s+2t-1)\right)+\\
& x_Q\left(-\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& x_R\left(\frac{1}{2}(r-1)(s+1)t\right)+\\
& x_S\left(\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& x_T\left(-\frac{1}{2}(r-1)(s-1)t\right)+\\
& x_U\left(\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& x_V\left(-\frac{1}{2}(r+1)(s+1)t\right)+\\
& x_W\left(-\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& x_X\left(\frac{1}{2}(r+1)(s-1)t\right)+\\
& x_Y\left(-\frac{1}{4}\left(r^2-1\right)(s-1)\right)+\\
& x_Z\left(\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& x_A\left(-\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& x_B\left(\frac{1}{4}\left(r^2-1\right)(s-1)\right)
\end{aligned}$$

2.5.3.3 $y(r,s,t)$ terms

We now repeat all the above to find $\frac{\partial y}{\partial r}, \frac{\partial y}{\partial s}$ and $\frac{\partial y}{\partial t}$. We first need to expand $y(r,s,t) = \sum_{i=I}^{I=B} y_i f_i(r,s,t)$, which gives

$$y = y_I f_I + y_J f_J + y_K f_K + y_L f_L + y_M f_M + y_N f_N + y_O f_O + y_P f_P + y_Q f_Q + y_R f_R + y_S f_S + y_T f_T + y_U f_U + y_V f_V + y$$

Expanding the above gives

$$\begin{aligned}
y(r,s,t) =& y_I \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)+ \\
& y_J \frac{1}{8}(1-r)(s+1)(1-t)(-r+s-t-2)+ \\
& y_K \frac{1}{8}(1-r)(s+1)(t+1)(-r+s+t-2)+ \\
& y_L \frac{1}{8}(1-r)(1-s)(t+1)(-r-s+t-2)+ \\
& y_M \frac{1}{8}(r+1)(1-s)(1-t)(r-s-t-2)+ \\
& y_N \frac{1}{8}(r+1)(s+1)(1-t)(r+s-t-2)+ \\
& y_O \frac{1}{8}(r+1)(s+1)(t+1)(r+s+t-2)+ \\
& y_P \frac{1}{8}(r+1)(1-s)(t+1)(r-s+t-2)+ \\
& y_Q \frac{1}{4}(1-r)\left(1-s^2\right)(1-t)+ \\
& y_R \frac{1}{4}(1-r)(s+1)\left(1-t^2\right)+ \\
& y_S \frac{1}{4}(1-r)\left(1-s^2\right)(t+1)+ \\
& y_T \frac{1}{4}(1-r)(1-s)\left(1-t^2\right)+ \\
& y_U \frac{1}{4}(r+1)\left(1-s^2\right)(1-t)+ \\
& y_V \frac{1}{4}(r+1)(s+1)\left(1-t^2\right)+ \\
& y_W \frac{1}{4}(r+1)\left(1-s^2\right)(t+1)+ \\
& y_X \frac{1}{4}(r+1)(1-s)\left(1-t^2\right)+ \\
& y_Y \frac{1}{4}\left(1-r^2\right)(1-s)(1-t)+ \\
& y_Z \frac{1}{4}\left(1-r^2\right)(s+1)(1-t)+ \\
& y_A \frac{1}{4}\left(1-r^2\right)(s+1)(t+1)+ \\
& y_B \frac{1}{4}\left(1-r^2\right)(1-s)(t+1)
\end{aligned}$$

Taking partial derivatives of the above w.r.t. r,s,t in turn, we see that it gives similar results to earlier ones, but the only difference is in the multipliers now being the y_i values of

coordinates instead of the x_i coordinates. This is reproduced again for completion

$$\begin{aligned}
\frac{\partial y}{\partial r} =& y_I\left(\frac{1}{8}(s-1)(t-1)(2r+s+t+1)\right)+\\
& y_J\left(\frac{1}{8}(s+1)(t-1)(-2r+s-t-1)\right)+\\
& y_K\left(-\frac{1}{8}(s+1)(t+1)(-2r+s+t-1)\right)+\\
& y_L\left(-\frac{1}{8}(s-1)(t+1)(2r+s-t+1)\right)+\\
& y_M\left(-\frac{1}{8}(s-1)(t-1)(-2r+s+t+1)\right)+\\
& y_N\left(-\frac{1}{8}(s+1)(t-1)(2r+s-t-1)\right)+\\
& y_O\left(\frac{1}{8}(s+1)(t+1)(2r+s+t-1)\right)+\\
& y_P\left(\frac{1}{8}(s-1)(t+1)(-2r+s-t+1)\right)+\\
& y_Q\left(-\frac{1}{4}\left(s^2-1\right)(t-1)\right)+\\
& y_R\left(\frac{1}{4}(s+1)\left(t^2-1\right)\right)+\\
& y_S\left(\frac{1}{4}\left(s^2-1\right)(t+1)\right)+\\
& y_T\left(-\frac{1}{4}(s-1)\left(t^2-1\right)\right)+\\
& y_U\left(\frac{1}{4}\left(s^2-1\right)(t-1)\right)+\\
& y_V\left(-\frac{1}{4}(s+1)\left(t^2-1\right)\right)+\\
& y_W\left(-\frac{1}{4}\left(s^2-1\right)(t+1)\right)+\\
& y_X\left(\frac{1}{4}(s-1)\left(t^2-1\right)\right)+\\
& y_Y\left(-\frac{1}{2}r(s-1)(t-1)\right)+\\
& y_Z\left(\frac{1}{2}r(s+1)(t-1)\right)+\\
& y_A\left(-\frac{1}{2}r(s+1)(t+1)\right)+\\
& y_B\left(\frac{1}{2}r(s-1)(t+1)\right)
\end{aligned}$$

Similarly, we now find $\frac{\partial y}{\partial s}$ as above. This results in

$$\begin{aligned}
\frac{\partial y}{\partial s} =& y_I\left(\frac{1}{8}(r-1)(t-1)(r+2s+t+1)\right)+\\
& y_J\left(-\frac{1}{8}(r-1)(t-1)(r-2s+t+1)\right)+\\
& y_K\left(\frac{1}{8}(r-1)(t+1)(r-2s-t+1)\right)+\\
& y_L\left(-\frac{1}{8}(r-1)(t+1)(r+2s-t+1)\right)+\\
& y_M\left(\frac{1}{8}(r+1)(t-1)(r-2s-t-1)\right)+\\
& y_N\left(-\frac{1}{8}(r+1)(t-1)(r+2s-t-1)\right)+\\
& y_O\left(\frac{1}{8}(r+1)(t+1)(r+2s+t-1)\right)+\\
& y_P\left(-\frac{1}{8}(r+1)(t+1)(r-2s+t-1)\right)+\\
& y_Q\left(-\frac{1}{2}(r-1)s(t-1)\right)+\\
& y_R\left(\frac{1}{4}(r-1)\left(t^2-1\right)\right)+\\
& y_S\left(\frac{1}{2}(r-1)s(t+1)\right)+\\
& y_T\left(-\frac{1}{4}(r-1)\left(t^2-1\right)\right)+\\
& y_U\left(\frac{1}{2}(r+1)s(t-1)\right)+\\
& y_V\left(-\frac{1}{4}(r+1)\left(t^2-1\right)\right)+\\
& y_W\left(-\frac{1}{2}(r+1)s(t+1)\right)+\\
& y_X\left(\frac{1}{4}(r+1)\left(t^2-1\right)\right)+\\
& y_Y\left(-\frac{1}{4}\left(r^2-1\right)(t-1)\right)+\\
& y_Z\left(\frac{1}{4}\left(r^2-1\right)(t-1)\right)+\\
& y_A\left(-\frac{1}{4}\left(r^2-1\right)(t+1)\right)+\\
& y_B\left(\frac{1}{4}\left(r^2-1\right)(t+1)\right)
\end{aligned}$$

We now find $\frac{\partial y}{\partial t}$ as above. This results in

$$\begin{aligned}
\frac{\partial y}{\partial t} =& y_I\left(\frac{1}{8}(r-1)(s-1)(r+s+2t+1)\right)+\\
& y_J\left(-\frac{1}{8}(r-1)(s+1)(r-s+2t+1)\right)+\\
& y_K\left(\frac{1}{8}(r-1)(s+1)(r-s-2t+1)\right)+\\
& y_L\left(-\frac{1}{8}(r-1)(s-1)(r+s-2t+1)\right)+\\
& y_M\left(\frac{1}{8}(r+1)(s-1)(r-s-2t-1)\right)+\\
& y_N\left(-\frac{1}{8}(r+1)(s+1)(r+s-2t-1)\right)+\\
& y_O\left(\frac{1}{8}(r+1)(s+1)(r+s+2t-1)\right)+\\
& y_P\left(-\frac{1}{8}(r+1)(s-1)(r-s+2t-1)\right)+\\
& y_Q\left(-\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& y_R\left(\frac{1}{2}(r-1)(s+1)t\right)+\\
& y_S\left(\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& y_T\left(-\frac{1}{2}(r-1)(s-1)t\right)+\\
& y_U\left(\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& y_V\left(-\frac{1}{2}(r+1)(s+1)t\right)+\\
& y_W\left(-\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& y_X\left(\frac{1}{2}(r+1)(s-1)t\right)+\\
& y_Y\left(-\frac{1}{4}\left(r^2-1\right)(s-1)\right)+\\
& y_Z\left(\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& y_A\left(-\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& y_B\left(\frac{1}{4}\left(r^2-1\right)(s-1)\right)
\end{aligned}$$

2.5.3.4 $z(r,s,t)$ terms

We now repeat all the above to find $\frac{\partial z}{\partial r}, \frac{\partial z}{\partial s}$ and $\frac{\partial z}{\partial t}$. These produce similar results to the above, but will have z_i as multipliers. We first need to expand $z(r,s,t) = \sum_{i=I}^{I=B} z_i f_i(r,s,t)$, which gives

$$z_{=} z_I f_I + z_J f_J + z_K f_K + z_L f_L + z_M f_M + z_N f_N + z_O f_O + z_P f_P + z_Q f_Q + z_R f_R + z_S f_S + z_T f_T + z_U f_U + z_V f_V + z_W$$

Expanding the above gives

$$
\begin{aligned}
z(r,s,t) = & z_I \frac{1}{8}(1-r)(1-s)(1-t)(-r-s-t-2)+ \\
& z_J \frac{1}{8}(1-r)(s+1)(1-t)(-r+s-t-2)+ \\
& z_K \frac{1}{8}(1-r)(s+1)(t+1)(-r+s+t-2)+ \\
& z_L \frac{1}{8}(1-r)(1-s)(t+1)(-r-s+t-2)+ \\
& z_M \frac{1}{8}(r+1)(1-s)(1-t)(r-s-t-2)+ \\
& z_N \frac{1}{8}(r+1)(s+1)(1-t)(r+s-t-2)+ \\
& z_O \frac{1}{8}(r+1)(s+1)(t+1)(r+s+t-2)+ \\
& z_P \frac{1}{8}(r+1)(1-s)(t+1)(r-s+t-2)+ \\
& z_Q \frac{1}{4}(1-r)\left(1-s^2\right)(1-t)+ \\
& z_R \frac{1}{4}(1-r)(s+1)\left(1-t^2\right)+ \\
& z_S \frac{1}{4}(1-r)\left(1-s^2\right)(t+1)+ \\
& z_T \frac{1}{4}(1-r)(1-s)\left(1-t^2\right)+ \\
& z_U \frac{1}{4}(r+1)\left(1-s^2\right)(1-t)+ \\
& z_V \frac{1}{4}(r+1)(s+1)\left(1-t^2\right)+ \\
& z_W \frac{1}{4}(r+1)\left(1-s^2\right)(t+1)+ \\
& z_X \frac{1}{4}(r+1)(1-s)\left(1-t^2\right)+ \\
& z_Y \frac{1}{4}\left(1-r^2\right)(1-s)(1-t)+ \\
& z_Z \frac{1}{4}\left(1-r^2\right)(s+1)(1-t)+ \\
& z_A \frac{1}{4}\left(1-r^2\right)(s+1)(t+1)+ \\
& z_B \frac{1}{4}\left(1-r^2\right)(1-s)(t+1)
\end{aligned}
$$

Taking partial derivative of the above w.r.t. r, s, t in turns gives the following

$$
\begin{aligned}
\frac{\partial z}{\partial r} =& z_I \left(\frac{1}{8}(s-1)(t-1)(2r+s+t+1)\right) + \\
& z_J \left(\frac{1}{8}(s+1)(t-1)(-2r+s-t-1)\right) + \\
& z_K \left(-\frac{1}{8}(s+1)(t+1)(-2r+s+t-1)\right) + \\
& z_L \left(-\frac{1}{8}(s-1)(t+1)(2r+s-t+1)\right) + \\
& z_M \left(-\frac{1}{8}(s-1)(t-1)(-2r+s+t+1)\right) + \\
& z_N \left(-\frac{1}{8}(s+1)(t-1)(2r+s-t-1)\right) + \\
& z_O \left(\frac{1}{8}(s+1)(t+1)(2r+s+t-1)\right) + \\
& z_P \left(\frac{1}{8}(s-1)(t+1)(-2r+s-t+1)\right) + \\
& z_Q \left(-\frac{1}{4}\left(s^2-1\right)(t-1)\right) + \\
& z_R \left(\frac{1}{4}(s+1)\left(t^2-1\right)\right) + \\
& z_S \left(\frac{1}{4}\left(s^2-1\right)(t+1)\right) + \\
& z_T \left(-\frac{1}{4}(s-1)\left(t^2-1\right)\right) + \\
& z_U \left(\frac{1}{4}\left(s^2-1\right)(t-1)\right) + \\
& z_V \left(-\frac{1}{4}(s+1)\left(t^2-1\right)\right) + \\
& z_W \left(-\frac{1}{4}\left(s^2-1\right)(t+1)\right) + \\
& z_X \left(\frac{1}{4}(s-1)\left(t^2-1\right)\right) + \\
& z_Y \left(-\frac{1}{2}r(s-1)(t-1)\right) + \\
& z_Z \left(\frac{1}{2}r(s+1)(t-1)\right) + \\
& z_A \left(-\frac{1}{2}r(s+1)(t+1)\right) + \\
& z_B \left(\frac{1}{2}r(s-1)(t+1)\right)
\end{aligned}
$$

Similarly, $\frac{\partial z}{\partial s}$ results in

$$\begin{aligned}
\frac{\partial z}{\partial s} =& z_I \left(\frac{1}{8}(r-1)(t-1)(r+2s+t+1)\right) + \\
& z_J \left(-\frac{1}{8}(r-1)(t-1)(r-2s+t+1)\right) + \\
& z_K \left(\frac{1}{8}(r-1)(t+1)(r-2s-t+1)\right) + \\
& z_L \left(-\frac{1}{8}(r-1)(t+1)(r+2s-t+1)\right) + \\
& z_M \left(\frac{1}{8}(r+1)(t-1)(r-2s-t-1)\right) + \\
& z_N \left(-\frac{1}{8}(r+1)(t-1)(r+2s-t-1)\right) + \\
& z_O \left(\frac{1}{8}(r+1)(t+1)(r+2s+t-1)\right) + \\
& z_P \left(-\frac{1}{8}(r+1)(t+1)(r-2s+t-1)\right) + \\
& z_Q \left(-\frac{1}{2}(r-1)s(t-1)\right) + \\
& z_R \left(\frac{1}{4}(r-1)\left(t^2-1\right)\right) + \\
& z_S \left(\frac{1}{2}(r-1)s(t+1)\right) + \\
& z_T \left(-\frac{1}{4}(r-1)\left(t^2-1\right)\right) + \\
& z_U \left(\frac{1}{2}(r+1)s(t-1)\right) + \\
& z_V \left(-\frac{1}{4}(r+1)\left(t^2-1\right)\right) + \\
& z_W \left(-\frac{1}{2}(r+1)s(t+1)\right) + \\
& z_X \left(\frac{1}{4}(r+1)\left(t^2-1\right)\right) + \\
& z_Y \left(-\frac{1}{4}\left(r^2-1\right)(t-1)\right) + \\
& z_Z \left(\frac{1}{4}\left(r^2-1\right)(t-1)\right) + \\
& z_A \left(-\frac{1}{4}\left(r^2-1\right)(t+1)\right) + \\
& z_B \left(\frac{1}{4}\left(r^2-1\right)(t+1)\right)
\end{aligned}$$

And $\frac{\partial z}{\partial t}$ gives

$$\begin{aligned}
\frac{\partial z}{\partial t} =& z_I\left(\frac{1}{8}(r-1)(s-1)(r+s+2t+1)\right)+\\
& z_J\left(-\frac{1}{8}(r-1)(s+1)(r-s+2t+1)\right)+\\
& z_K\left(\frac{1}{8}(r-1)(s+1)(r-s-2t+1)\right)+\\
& z_L\left(-\frac{1}{8}(r-1)(s-1)(r+s-2t+1)\right)+\\
& z_M\left(\frac{1}{8}(r+1)(s-1)(r-s-2t-1)\right)+\\
& z_N\left(-\frac{1}{8}(r+1)(s+1)(r+s-2t-1)\right)+\\
& z_O\left(\frac{1}{8}(r+1)(s+1)(r+s+2t-1)\right)+\\
& z_P\left(-\frac{1}{8}(r+1)(s-1)(r-s+2t-1)\right)+\\
& z_Q\left(-\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& z_R\left(\frac{1}{2}(r-1)(s+1)t\right)+\\
& z_S\left(\frac{1}{4}(r-1)\left(s^2-1\right)\right)+\\
& z_T\left(-\frac{1}{2}(r-1)(s-1)t\right)+\\
& z_U\left(\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& z_V\left(-\frac{1}{2}(r+1)(s+1)t\right)+\\
& z_W\left(-\frac{1}{4}(r+1)\left(s^2-1\right)\right)+\\
& z_X\left(\frac{1}{2}(r+1)(s-1)t\right)+\\
& z_Y\left(-\frac{1}{4}\left(r^2-1\right)(s-1)\right)+\\
& z_Z\left(\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& z_A\left(-\frac{1}{4}\left(r^2-1\right)(s+1)\right)+\\
& z_B\left(\frac{1}{4}\left(r^2-1\right)(s-1)\right)
\end{aligned}$$

Finally now we can determine the Jacobian and its determinant using the above expressions. This is done in the Matlab code provided. The following Jacobian Matrix is evaluated at each Gaussian integration point then its determinant is found using `det()` command.

$$\begin{pmatrix}
\frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r}\\
\frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s}\\
\frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t}
\end{pmatrix}$$

2.5.3.5 results

The first step was to obtain estimate of the volume in order to verify that the volume calculation was valid and that the Jacobian was correct.

An independent small piece of code was written to plot the 3D shape and obtain its volume using a build-in function in the computer algebra program Mathematica. This is a plot of the 3D shape generated and below it is the code used to generate the plot, with the volume found shown in the title.

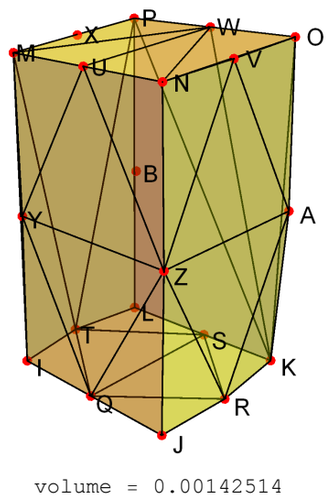
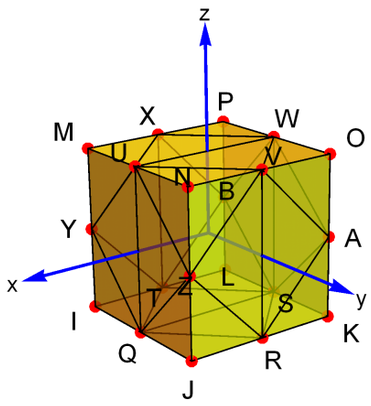


Figure 2.53: 3D plot of the volume in physical coordinates

```
xI=1;xJ=1.1;xK=1.09066;xL=0.99692;xM=1.0077;
xN=1.1069;xO=1.1035;xP=1.0046;xQ=1.05;xR=1.0992;
xS=1.0468;xT=0.9923;xU=1.0573;xV=1.1061;xW=1.0540;
xX=1.0069;xY=1.0051;xZ=1.1046;xA=1.1012;xB=1.0020;
xCoordinates={xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,xW,xX,xY,xZ,xA,xB};
yI=0;yJ=0;yK=0;yL=0;yM=0.16559;
yN=0.17203;yO=0.17202;yP=0.16557;yQ=0;yR=0;
yS=0;yT=0;yU=0.16881;yV=0.17202;yW=0.16880;
yX=0.16558;yY=0.082737;yZ=0.085964;yA=0.085938;yB=0.082709;
yCoordinates={yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,yX,yY,yZ,yA,yB};
zI=0;zJ=0;zK=-0.086305;zL=-0.078459;zM=0;
zN=0;zO=-0.08663;zP=-0.07882;zQ=0;zR=-0.043186;
zS=-0.082382;zT=-0.039260;zU=0;zV=-0.043328;zW=-0.082725;
zX=-0.039418;zY=0;zZ=0;zA=-0.086550;zB=-0.078731;
zCoordinates={zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,zY,zZ,zA,zB};
data3D=Table[{xCoordinates[[i]],yCoordinates[[i]],
          zCoordinates[[i]]},{i,1,Length@yCoordinates}];
nodes={"I","J","K","L","M","N","O","P","Q",
       "R","S","T","U","V","W","X","Y","Z","A","B"};
Needs["TetGenLink`"];
{pts,surface}=TetGenConvexHull[data3D];
c=First@Last@Reap@Do[Sow@{nodes[[i]],data3D[[i]]},{i,1,Length[nodes]}];
Labeled[Graphics3D[{
   {Red,PointSize[0.02],Point[data3D]},
   {Yellow,Opacity[.4],EdgeForm[{Thin,Black}],
    GraphicsComplex[data3D,Polygon[surface]]},
   {Text[Style[#[[1]],14],1.01*#[[2]]]&/@c}
},Boxed->False,Axes->False,SphericalRegion->True,
    ImageSize->300,ImageMargins->5],
    Row[{"volume = ",RegionMeasure@ConvexHullMesh[data3D]}]]
```

.

We now know the volume should be 0.0042514cm^3 from the above independent verification. The Matlab code was now implemented, and the volume was verified to be the same. Also, a separate test was run to verify that $||J|| = 1$ for a test 3D volume which was aligned along the same orientation as the natural coordinates as shown below.




```
volume = 8.
```

Figure 2.54: 3D plot of the aligned volume used for verification

The code used to plot the above is

```
xI=1;xJ=1.1;xK=1.09066;xL=0.99692;xM=1.0077; xN=1.1069;xO=1.1035;xP=1.0046;
xQ=1.05;xR=1.0992;xS=1.0468;xT=0.9923;xU=1.0573;xV=1.1061;xW=1.0540;
xX=1.0069;xY=1.0051;xZ=1.1046;xA=1.1012;xB=1.0020;
xCoordinates={xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,xW,xX,xY,xZ,xA,xB};
yI=0;yJ=0;yK=0;yL=0;yM=0.16559; yN=0.17203;yO=0.17202;yP=0.16557;yQ=0;yR=0;
yS=0;yT=0;yU=0.16881;yV=0.17202;yW=0.16880; yX=0.16558;yY=0.082737;yZ=0.085964;
yA=0.085938;yB=0.082709;
yCoordinates={yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,yX,yY,yZ,yA,yB};
zI=0;zJ=0;zK=-0.086305;zL=-0.078459;zM=0; zN=0;zO=-0.08663;zP=-0.07882;zQ=0;
zR=-0.043186; zS=-0.082382;zT=-0.039260;zU=0;zV=-0.043328;zW=-0.082725;
zX=-0.039418;zY=0;zZ=0;zA=-0.086550;zB=-0.078731;
zCoordinates={zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,zY,zZ,zA,zB};
data3D=Table[{xCoordinates[[i]],yCoordinates[[i]],zCoordinates[[i]]},{i,1,
    Length@yCoordinates}];
nodes={"I","J","K","L","M","N","O","P","Q",
       "R","S","T","U","V","W","X","Y","Z","A","B"};
Needs["TetGenLink`"];
{pts,surface}=TetGenConvexHull[data3D];
c=First@Last@Reap@Do[Sow@{nodes[[i]],data3D[[i]]},{i,1,Length[nodes]}];
Labeled[Graphics3D[{
   {Red,PointSize[0.02],Point[data3D]},
   {Yellow,Opacity[.4],EdgeForm[{Thin,Black}],
    GraphicsComplex[data3D,Polygon[surface]]},
   {Text[Style[#[[1]],14],1.01*#[[2]]]&/@c}
},Boxed->False,Axes->False,SphericalRegion->True,
    ImageSize->300,ImageMargins->5],
    Row[{"volume = ",RegionMeasure@ConvexHullMesh[data3D]}]]
```

.

This test also passed in Matlab and gave a volume of 8cm^3 as expected. Here is the small code segment in Matlab which verifies the above.

```
wt(1) = 5/9;        wt(2) = 8/9;   wt(3) = 5/9;
gs(1) = -sqrt(3/5); gs(2) = 0;     gs(3) = sqrt(3/5) ;

%set nodal coordinates as cube of side 2, centered
%with natural coordinates origin
xI=1;       xJ=1;       xK=-1;         xL=-1;        xM=1;
```

```
xN=1;       xO=-1;      xP=-1;        xQ=1;        xR=0;
xS=-1;      xT=0;     xU=1;         xV=0;        xW=-1;
xX=0;       xY=1;      xZ=1;         xA=-1;       xB=-1;

xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV,...
             xW,xX,xY,xZ,xA,xB];
yI=-1;       yJ=1;        yK=1;           yL=-1;       yM=-1;
yN=1;        yO=1;        yP=-1;          yQ=0;        yR=1;
yS=0;        yT=-1;       yU=0;           yV=1;        yW=0;
yX=-1;       yY=-1;       yZ=1;           yA=1;        yB=-1;
yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,...
             yX,yY,yZ,yA,yB];

zI=-1;         zJ=-1;     zK=-1;    zL=-1;     zM=1;
zN=1;          zO=1;      zP=1;     zQ=-1;     zR=-1;
zS=-1;         zT=-1;     zU=1;     zV=1;      zW=1;
zX=1;          zY=0;      zZ=0;     zA=0;      zB=0;
zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,...
              zW,zX,zY,zZ,zA,zB];

%to collect sum of integrals at each node
the_sum = zeros(20,1);

%find the volume first, to use for verification.
for i=1:3
    s = gs(i);
    for j=1:3
        t = gs(j);
        for k=1:3
          r = gs(k);
          J = get_jacobian(r,s,t,xCoordinates,yCoordinates,zCoordinates);
          detJ = det(J);
          fprintf('|J|= %3.3f at Gaussian point [r=%3.3f,s=%3.3f,t=%3.3f]\n',
   detJ,r,s,t);
          for ii=1:length(xCoordinates)
              the_sum(ii)=the_sum(ii)+ wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
          end
        end
    end
end

fprintf('volume for test is [%3.3f] (is it 8?)\n',sum(the_sum));
end
```

.

The output of the above is given below, with the rest of the program output.

The program nma_EMA_471_HW5_problem_3.m implements the main solution to this problem and included in the zip file. It runs both the Jacobian verification and the load calculations after that.

The main loop of the Matlab function iterates over three indices i,j,k from 1 to 3 each. In the inner most loop, it finds the determinant of the Jacobian, the mass density, and evaluates the shape function at the Gaussian points, then sums the result. At the end it prints the work-equivalent conversion for each of the twenty nodes. The following shows the main core of the program

```
for i=1:3
    s = gs(i);
    for j=1:3
        t = gs(j);
        for k=1:3
            r = gs(k);
```

```
            J = get_jacobian(r,s,t,xCoordinates,...
                             yCoordinates,zCoordinates);
            detJ = det(J);
            mg = find_mass_density(r,s,t,xCoordinates,zCoordinates);

            for ii=1:length(xCoordinates)
                the_sum(ii)=the_sum(ii)+...
                        wt(i)*wt(j)*wt(k)*mg*g*f(ii,r,s,t)*detJ;
            end
        end
    end
end
```

.

The final result is in the following table

Corner	work-equivalent load (z direction) $\frac{\text{gram-cm}}{\text{sec}^2}$	Newton units
I	−0.1097835	−0.000001934
J	−0.1336398	−0.000001990
K	−0.1159590	−0.000002
L	−0.1143340	−0.000001928
M	−0.1373666	−0.000001927
N	−0.1611689	−0.000001984
O	−0.1418730	−0.00000198
P	−0.1160910	−0.000001924
Q	0.1701952	0.000002614
R	0.1812529	0.000002739
S	0.0808510	0.000002592
T	0.0741811	0.000002513
U	0.2242564	0.000002589
V	0.2374809	0.000002728
W	0.1905987	0.000002589
X	0.1802555	0.000002476
Y	0.1723161	0.000002482
Z	0.2334560	0.000002734
A	0.1929484	0.000002710
B	0.0986489	0.000002487

Table 2.11: work-equivalent conversion at each corner, problem 3

We also see that the load on the corners is negative while on the middle nodes it is positive. This agrees with what one would expect as per class notes on the 8-node element. Only difference is that this is a 3D element.

The following is the console output from running the above program. It is implemented using Matlab 2016a. It starts with the Jacobian verification then it will run the main task next only if the verification passes.

```
>>nma_EMA_471_HW5_problem_3()

starting verification of Jacobian....
|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=-0.775]
```

```
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=-0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.000,t=0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=-0.775]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=0.000]
|J|= 1.000 at Gaussian point [r=-0.775,s=0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.000,s=0.775,t=0.775]
|J|= 1.000 at Gaussian point [r=0.775,s=0.775,t=0.775]
volume for test is [8.000] (is it 8?)
!! passed Jacobian test. Will run main program now
volume is 0.001426 cm^3

load at corner I = -0.1934002 [gram-cm/sec^2] = -0.000001934 N
load at corner J = -0.1990333 [gram-cm/sec^2] = -0.000001990 N
load at corner K = -0.2000363 [gram-cm/sec^2] = -0.000002000 N
load at corner L = -0.1928213 [gram-cm/sec^2] = -0.000001928 N
load at corner M = -0.1927113 [gram-cm/sec^2] = -0.000001927 N
load at corner N = -0.1984265 [gram-cm/sec^2] = -0.000001984 N
load at corner O = -0.1980087 [gram-cm/sec^2] = -0.000001980 N
load at corner P = -0.1923718 [gram-cm/sec^2] = -0.000001924 N
load at corner Q = 0.2614284 [gram-cm/sec^2] = 0.000002614 N
load at corner R = 0.2739156 [gram-cm/sec^2] = 0.000002739 N
load at corner S = 0.2592383 [gram-cm/sec^2] = 0.000002592 N
load at corner T = 0.2513418 [gram-cm/sec^2] = 0.000002513 N
load at corner U = 0.2589067 [gram-cm/sec^2] = 0.000002589 N
load at corner V = 0.2727979 [gram-cm/sec^2] = 0.000002728 N
load at corner W = 0.2588535 [gram-cm/sec^2] = 0.000002589 N
load at corner X = 0.2475648 [gram-cm/sec^2] = 0.000002476 N
load at corner Y = 0.2481743 [gram-cm/sec^2] = 0.000002482 N
load at corner Z = 0.2733760 [gram-cm/sec^2] = 0.000002734 N
load at corner A = 0.2710102 [gram-cm/sec^2] = 0.000002710 N
load at corner B = 0.2487254 [gram-cm/sec^2] = 0.000002487 N
>>
```

```
function nma_EMA_471_HW5_problem_3()
%Solves problem 3, HW5, EMA 471
%Nasser M. Abbasi

close all; clc;

status = do_jacobian_test();
if ~status
```

```
    error('failed jacobian test. Internal code error\n');
else
   fprintf('!! passed Jacobian test. Will run main program now\n\n');
   do_main_program();
end

end
%=====================================
function status = do_jacobian_test()
%This function checks that |J|=1 at each Gaussian point.
%This verifies the code is ok before
%running the main program. This also checkes that volume is
% 2*2*2=8 cm^3 since we are using a cube with nodal coordinates
% with side length = 2 cm and it is aligned along the natural
%coordinates and centered at the natural coordinates origin also.
%

status = true;

wt(1) = 5/9;        wt(2) = 8/9;   wt(3) = 5/9;
gs(1) = -sqrt(3/5); gs(2) = 0;     gs(3) = sqrt(3/5) ;

%set nodal coordinates as cube of side 2,
%centered with natural coordinates origin
xI=1;       xJ=1;      xK=-1;        xL=-1;       xM=1;
xN=1;       xO=-1;      xP=-1;        xQ=1;        xR=0;
xS=-1;      xT=0;     xU=1;         xV=0;        xW=-1;
xX=0;       xY=1;      xZ=1;         xA=-1;       xB=-1;

xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,xV, ...
              xW,xX,xY,xZ,xA,xB];

yI=-1;       yJ=1;        yK=1;           yL=-1;       yM=-1;
yN=1;        yO=1;        yP=-1;          yQ=0;        yR=1;
yS=0;        yT=-1;       yU=0;           yV=1;        yW=0;
yX=-1;       yY=-1;       yZ=1;           yA=1;        yB=-1;
yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,...
              yX,yY,yZ,yA,yB];

zI=-1;         zJ=-1;     zK=-1;    zL=-1;     zM=1;
zN=1;          zO=1;      zP=1;     zQ=-1;     zR=-1;
zS=-1;         zT=-1;     zU=1;     zV=1;      zW=1;
zX=1;          zY=0;      zZ=0;     zA=0;      zB=0;
zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,...
              zX,zY,zZ,zA,zB];

the_sum = zeros(20,1);  %to collect sum of integrals at each node

%find the volume first, to use for verification.
format short;
format compact;
fprintf('starting verification of Jacobian....\n');
for i=1:3
    s = gs(i);

    for j=1:3
        t = gs(j);

        for k=1:3
            r = gs(k);

            J = get_jacobian(r,s,t,xCoordinates,...
                             yCoordinates,zCoordinates);
```

```
            detJ = det(J);
            fprintf('|J|= %3.3f at Gaussian point [r=%3.3f,s=%3.3f,t=%3.3f]\n',...
                detJ,r,s,t);
            if detJ <=0
                status = false;  %FAILED TEST
                return;
            end
            for ii=1:length(xCoordinates)
                the_sum(ii)=the_sum(ii)+ ...
                             wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
            end
        end
    end
end

fprintf('volume for test is [%3.3f] (is it 8?)\n',sum(the_sum));

end

%====================================
function do_main_program()
wt(1) = 5/9;        wt(2) = 8/9;   wt(3) = 5/9;
gs(1) = -sqrt(3/5); gs(2) = 0;     gs(3) = sqrt(3/5) ;

xI=1;      xJ=1.1;      xK=1.09066;    xL=0.99692;   xM=1.0077;
xN=1.1069; xO=1.1035;   xP=1.0046;     xQ=1.05;      xR=1.0992;
xS=1.0468; xT=0.9923;   xU=1.0573;     xV=1.1061;    xW=1.0540;
xX=1.0069; xY=1.0051;   xZ=1.1046;     xA=1.1012;    xB=1.0020;

xCoordinates=[xI,xJ,xK,xL,xM,xN,xO,xP,xQ,xR,xS,xT,xU,...
              xV,xW,xX,xY,xZ,xA,xB];

yI=0;       yJ=0;        yK=0;           yL=0;        yM=0.16559;
yN=0.17203; yO=0.17202;  yP=0.16557;     yQ=0;        yR=0;
yS=0;       yT=0;        yU=0.16881;     yV=0.17202;  yW=0.16880;
yX=0.16558; yY=0.082737; yZ=0.085964;    yA=0.085938; yB=0.082709;
yCoordinates=[yI,yJ,yK,yL,yM,yN,yO,yP,yQ,yR,yS,yT,yU,yV,yW,...
              yX,yY,yZ,yA,yB];

zI=0;          zJ=0;          zK=-0.086305;    zL=-0.078459;   zM=0;
zN=0;          zO=-0.08663;   zP=-0.07882;     zQ=0;           zR=-0.043186;
zS=-0.082382; zT=-0.039260; zU=0;             zV=-0.043328;   zW=-0.082725;
zX=-0.039418; zY=0;          zZ=0;            zA=-0.086550;   zB=-0.078731;
zCoordinates=[zI,zJ,zK,zL,zM,zN,zO,zP,zQ,zR,zS,zT,zU,zV,zW,zX,...
             zY,zZ,zA,zB];

the_sum = zeros(20,1);
g       = 9.81*100; %acceleration g in cm per sec^2

%find the volume first, to use for verification.
for i=1:3
    s = gs(i);

    for j=1:3
        t = gs(j);

        for k=1:3
            r = gs(k);

            J = get_jacobian(r,s,t,xCoordinates,...
                             yCoordinates,zCoordinates);
            detJ = det(J);
            if detJ <=0
```

```
135                error('code internal error, invalid jacobian det. %7.6f\n',detJ);
136            end
137            for ii=1:length(xCoordinates)
138                the_sum(ii)=the_sum(ii)+ ...
139                            wt(i)*wt(j)*wt(k)*f(ii,r,s,t)*detJ;
140            end
141        end
142    end
143 end
144
145 fprintf('volume is %7.6f cm^3\n', sum(the_sum));
146
147 for i=1:3
148     s = gs(i);
149
150     for j=1:3
151         t = gs(j);
152
153         for k=1:3
154             r = gs(k);
155
156             J = get_jacobian(r,s,t,xCoordinates,...
157                                       yCoordinates,zCoordinates);
158             detJ = det(J);
159             if detJ <=0
160                error('code internal error, invalid jacobian det. %7.6f\n',detJ);
161             end
162             mg = find_mass_density(r,s,t,xCoordinates,zCoordinates);
163
164             for ii=1:length(xCoordinates)
165                 the_sum(ii)=the_sum(ii)+...
166                           wt(i)*wt(j)*wt(k)*mg*g*f(ii,r,s,t)*detJ;
167             end
168
169         end
170     end
171 end
172
173 map_node={'I','J','K','L','M','N','O','P','Q','R','S',...
174                  'T','U','V',...
175                  'W','X','Y','Z','A','B'};
176
177 for i=1:length(xCoordinates)
178   fprintf('load at corner %c = %9.7f [gram-cm/sec^2] = %10.9f N\n',...
179      map_node{i},the_sum(i),the_sum(i)*10^(-3)*10^(-2));
180 end
181 end
182 %=================================================
183 function mass_density = find_mass_density(r,s,t,...
184                                       xCoordinates,zCoordinates)
185
186         p0 = 1;
187         X  = 0;
188         for i=1:20
189             X = X + xCoordinates(i)*f(i,r,s,t);
190         end
191
192         Z = 0;
193         for i=1:20
194             Z = Z + zCoordinates(i)*f(i,r,s,t);
195         end
196
197         mass_density = p0*(X^2+Z^2);
```

```
end
%==================================
function the_shape_function=f(idx,r,s,t)
switch idx
    case 1 %I
        the_shape_function=(1/8)*(1-r)*(1-s)*(1-t)*(-r-s-t-2);
    case 2 %J
        the_shape_function=(1/8)*(1-r)*(s+1)*(1-t)*(-r+s-t-2);
    case 3 %K
        the_shape_function=(1/8)*(1-r)*(s+1)*(t+1)*(-r+s+t-2);
    case 4 %L
        the_shape_function=(1/8)*(1-r)*(1-s)*(t+1)*(-r-s+t-2);
    case 5 %M
        the_shape_function=(1/8)*(r+1)*(1-s)*(1-t)*(r-s-t-2);
    case 6 %N
        the_shape_function=(1/8)*(r+1)*(s+1)*(1-t)*(r+s-t-2);
    case 7 %O
        the_shape_function=(1/8)*(r+1)*(s+1)*(t+1)*(r+s+t-2);
    case 8 %P
        the_shape_function=(1/8)*(r+1)*(1-s)*(t+1)*(r-s+t-2);
    case 9 %Q
        the_shape_function=(1/4)*(1-r)*(1-s^2)*(1-t);
    case 10 %R
        the_shape_function=(1/4)*(1-r)*(s+1)*(1-t^2);
    case 11 %S
        the_shape_function=(1/4)*(1-r)*(1-s^2)*(t+1);
    case 12 %T
        the_shape_function=(1/4)*(1-r)*(1-s)*(1-t^2);
    case 13 %U
        the_shape_function=(1/4)*(1+r)*(1-s^2)*(1-t);
    case 14 %V
        the_shape_function=(1/4)*(r+1)*(s+1)*(1-t^2);
    case 15 %W
        the_shape_function=(1/4)*(r+1)*(1-s^2)*(t+1);
    case 16 %X
        the_shape_function=(1/4)*(r+1)*(1-s)*(1-t^2);
    case 17 %Y
        the_shape_function=(1/4)*(1-r^2)*(1-s)*(1-t);
    case 18 %Z
        the_shape_function=(1/4)*(1-r^2)*(s+1)*(1-t);
    case 19 %A
        the_shape_function=(1/4)*(1-r^2)*(s+1)*(t+1);
    case 20 %B
        the_shape_function=(1/4)*(1-r^2)*(1-s)*(t+1);
end
end

%------------------------------ internal function
function the_result=dds(c,r,s,t)
%find dx/ds or dy/ds or dz/ds. These all have same
%form, except for the multiplier c, which is the nodal
%coordinates, passed in.
the_result=c(1)*((1/8)*(r-1)*(t-1)*(r+2*s+t+1))+...
    c(2)*(-(1/8)*(r-1)*(t-1)*(r-2*s+t+1))+...
    c(3)*((1/8)*(r-1)*(t+1)*(r-2*s-t+1))+...
    c(4)*(-(1/8)*(r-1)*(t+1)*(r+2*s-t+1))+...
    c(5)*((1/8)*(r+1)*(t-1)*(r-2*s-t-1))+...
    c(6)*(-(1/8)*(r+1)*(t-1)*(r+2*s-t-1))+...
    c(7)*((1/8)*(r+1)*(t+1)*(r+2*s+t-1))+...
    c(8)*(-(1/8)*(r+1)*(t+1)*(r-2*s+t-1))+...
    c(9)*(-(1/2)*(r-1)*s*(t-1))+...
    c(10)*((1/4)*(r-1)*(t^2-1))+...
    c(11)*((1/2)*(r-1)*s*(t+1))+...
```

```
    c(12)*(-(1/4)*(r-1)*(t^2-1))+...
    c(13)*((1/2)*(r+1)*s*(t-1))+...
    c(14)*(-(1/4)*(r+1)*(t^2-1))+...
    c(15)*(-(1/2)*(r+1)*s*(t+1))+...
    c(16)*((1/4)*(r+1)*(t^2-1))+...
    c(17)*(-(1/4)*(r^2-1)*(t-1))+...
    c(18)*((1/4)*(r^2-1)*(t-1))+...
    c(19)*(-(1/4)*(r^2-1)*(t+1))+...
    c(20)*((1/4)*(r^2-1)*(t+1));
end
%------------------------------ internal function
function the_result=ddt(c,r,s,t)
%find dx/dt or dy/dt or dz/dt. These all have same form,
%except for the multiplier c, which is the nodal coordinates,
%passed in.

the_result=c(1)*((1/8)*(r-1)*(s-1)*(r+s+2*t+1))+...
    c(2)*(-(1/8)*(r-1)*(s+1)*(r-s+2*t+1))+...
    c(3)*((1/8)*(r-1)*(s+1)*(r-s-2*t+1))+...
    c(4)*(-(1/8)*(r-1)*(s-1)*(r+s-2*t+1))+...
    c(5)*((1/8)*(r+1)*(s-1)*(r-s-2*t-1))+...
    c(6)*(-(1/8)*(r+1)*(s+1)*(r+s-2*t-1))+...
    c(7)*((1/8)*(r+1)*(s+1)*(r+s+2*t-1))+...
    c(8)*(-(1/8)*(r+1)*(s-1)*(r-s+2*t-1))+...
    c(9)*(-(1/4)*(r-1)*(s^2-1))+...
    c(10)*((1/2)*(r-1)*(s+1)*t)+...
    c(11)*((1/4)*(r-1)*(s^2-1))+...
    c(12)*(-(1/2)*(r-1)*(s-1)*t)+...
    c(13)*((1/4)*(r+1)*(s^2-1))+...
    c(14)*(-(1/2)*(r+1)*(s+1)*t)+...
    c(15)*(-(1/4)*(r+1)*(s^2-1))+...
    c(16)*((1/2)*(r+1)*(s-1)*t)+...
    c(17)*(-(1/4)*(r^2-1)*(s-1))+...
    c(18)*((1/4)*(r^2-1)*(s+1))+...
    c(19)*(-(1/4)*(r^2-1)*(s+1))+...
    c(20)*((1/4)*(r^2-1)*(s-1));
end
%------------------------------ internal function
function the_result=ddr(c,r,s,t)
%find dx/dr or dy/dr or dz/dr. These all have same form,
%except for the multiplier c, which is the nodal coordinates,
%passed in.

the_result=c(1)*((1/8)*(s-1)*(t-1)*(2*r+s+t+1))+...
    c(2)*((1/8)*(s+1)*(t-1)*(-2*r+s-t-1))+...
    c(3)*(-(1/8)*(s+1)*(t+1)*(-2*r+s+t-1))+...
    c(4)*(-(1/8)*(s-1)*(t+1)*(2*r+s-t+1))+...
    c(5)*(-(1/8)*(s-1)*(t-1)*(-2*r+s+t+1))+...
    c(6)*(-(1/8)*(s+1)*(t-1)*(2*r+s-t-1))+...
    c(7)*((1/8)*(s+1)*(t+1)*(2*r+s+t-1))+...
    c(8)*((1/8)*(s-1)*(t+1)*(-2*r+s-t+1))+...
    c(9)*(-(1/4)*(s^2-1)*(t-1))+...
    c(10)*((1/4)*(s+1)*(t^2-1))+...
    c(11)*((1/4)*(s^2-1)*(t+1))+...
    c(12)*(-(1/4)*(s-1)*(t^2-1))+...
    c(13)*((1/4)*(s^2-1)*(t-1))+...
    c(14)*(-(1/4)*(s+1)*(t^2-1))+...
    c(15)*(-(1/4)*(s^2-1)*(t+1))+...
    c(16)*((1/4)*(s-1)*(t^2-1))+...
    c(17)*(-(1/2)*r*(s-1)*(t-1))+...
    c(18)*((1/2)*r*(s+1)*(t-1))+...
    c(19)*(-(1/2)*r*(s+1)*(t+1))+...
    c(20)*((1/2)*r*(s-1)*(t+1));
```

```
end
%=========================================
function J=get_jacobian(r,s,t,xCoordinates,yCoordinates,zCoordinates)

J = [ddr(xCoordinates,r,s,t),ddr(yCoordinates,r,s,t),...
     ddr(zCoordinates,r,s,t);
     dds(xCoordinates,r,s,t),dds(yCoordinates,r,s,t),...
     dds(zCoordinates,r,s,t);
     ddt(xCoordinates,r,s,t),ddt(yCoordinates,r,s,t),...
     ddt(zCoordinates,r,s,t)
     ];
end
```

2.6 HW 6

2.6.1 Problem 1

(1) (10 pts) Consider the following elliptic partial differential equation:

$$2\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial y^2} - \frac{\partial u}{\partial x} - \frac{\partial u}{\partial y} - 5u = -5,$$

Solve this problem over the unit square, $0 \le x \le 1$, $0 \le y \le 1$, subject to $u = 0$ on all boundaries. Approximate the first derivative by a centrally-differenced expression. You may use either a direct solution (simultaneous system of equations) or an iterative solution. Find the largest value of u within the domain.

Figure 2.55: problem 1 description

We need to solve $2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) - \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) - 5u = 5$ on the unit square. The discretized algebraic equation resulting from approximating this PDE using standard 5 point Laplacian and centered difference for the first derivatives is given by

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} &= \frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right)\\ \frac{\partial u}{\partial x} &= \frac{1}{2h}\left(U_{i+1,j} - U_{i-1,j}\right)\\ \frac{\partial u}{\partial y} &= \frac{1}{2h}\left(U_{i,j+1} - U_{i,j-1}\right)\end{aligned}$$

which has local truncation error $O\left(h^2\right)$. Therefore the PDE becomes

$$2\left(\frac{1}{h^2}\left(U_{i-1,j} + U_{i+1,j} + U_{i,j-1} + U_{i,j+1} - 4U_{i,j}\right)\right) - \frac{1}{2h}\left(U_{i+1,j} - U_{i-1,j} + U_{i,j+1} - U_{i,j-1}\right) - 5U_{i,j} = -5$$

$$U_{i-1,j}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{i+1,j}\left(\frac{2}{h^2} - \frac{1}{2h}\right) + U_{i,j-1}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{i,j+1}\left(\frac{2}{h^2} - \frac{1}{2h}\right) + U_{i,j}\left(\frac{-8}{h^2} - 5\right) = -5$$

2.6.1.1 Numbering system and grid updating

The numbering system used for the grid is the following. The indices for the unknown $U_{i,j}$ are numbered row wise, left to right, bottom to top. This follows the standard Cartesian coordinates system.

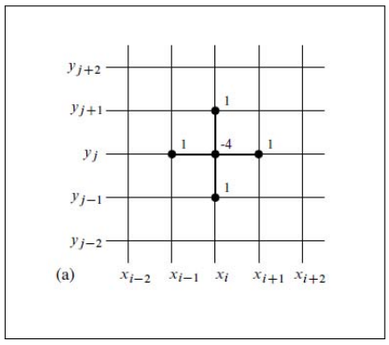


Figure 2.56: problem 1 grid

Lower case n is used to indicate the number of unknowns along one dimension, and upper case N is used to indicate the total number of unknowns. For example, if we use a grid with 5 points on each side, we obtain

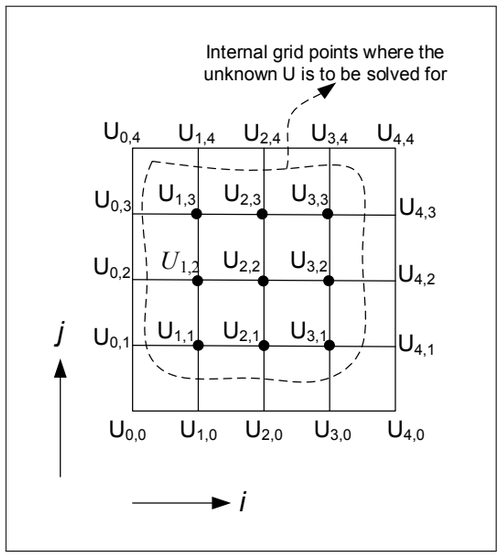


Figure 2.57: Example for $N = 3$

In the diagram above, $n = 3$ is the number of unknowns on each one row or each column, and since there are 3 internal rows, there will be 9 unknowns in total, all are located on internal grid points. There are a total of 25 grid points, 16 of which are on the boundaries (given as zero) and 9 are internal (which we need to solve for).

2.6.1.2 Structure of $Au = f$

We will derive the first few rows of the A matrix to see the structure. For $i = 1, j = 1$

$$U_{0,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,1}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,0}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{1,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,1}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 1$

$$U_{1,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,1}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,0}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,1}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 3, j = 1$

$$U_{2,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{4,1}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,0}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,1}\left(\frac{-8}{h^2}-5\right)=-5$$

The above completes one row (it is the bottom row in the grid) but it goes as the first row in the A matrix. Since boundary conditions are zero, then we can eliminate $U_{0,1}, U_{4,1}, U_{1,0}, U_{2,0}, U_{3,0}$ from the first row. Hence the above becomes

For $i = 1, j = 1$

$$U_{2,1}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,1}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 1$

$$U_{1,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,1}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,1}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 3, j = 1$

$$U_{2,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,1}\left(\frac{-8}{h^2}-5\right)=-5$$

Looking at the second row in the grid (the second row from the bottom up), we find

For $i = 1, J = 2$

$$U_{0,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{1,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,2}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 2$

$$U_{1,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,2}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 3, j = 2$

$$U_{2,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{4,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,2}\left(\frac{-8}{h^2}-5\right)=-5$$

Removing boundary conditions entries the above becomes

For $i = 1, J = 2$

$$U_{2,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{1,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,2}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 2$

$$U_{1,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,2}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,2}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 3, j = 2$

$$U_{2,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,1}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,2}\left(\frac{-8}{h^2}-5\right)=-5$$

And finally for the third row from the bottom up, (this will be the last row in the A matrix) we have

For $i = 1, J = 3$

$$U_{0,3}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{1,4}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,3}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 3$

$$U_{1,3}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{2,4}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{2,3}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 3, j = 3$

$$U_{2,3}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{4,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{3,4}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{3,3}\left(\frac{-8}{h^2}-5\right)=-5$$

Removing boundary conditions entries the above becomes

For $i = 1, J = 3$

$$U_{2,3}\left(\frac{2}{h^2}-\frac{1}{2h}\right)+U_{1,2}\left(\frac{2}{h^2}+\frac{1}{2h}\right)+U_{1,3}\left(\frac{-8}{h^2}-5\right)=-5$$

For $i = 2, j = 3$

$$U_{1,3}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{3,3}\left(\frac{2}{h^2} - \frac{1}{2h}\right) + U_{2,2}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{2,3}\left(\frac{-8}{h^2} - 5\right) = -5$$

For $i = 3, j = 3$

$$U_{2,3}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{3,2}\left(\frac{2}{h^2} + \frac{1}{2h}\right) + U_{3,3}\left(\frac{-8}{h^2} - 5\right) = -5$$

Therefor the $Au = f$ structure is the following

$$\begin{pmatrix}
\left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & 0 & 0 & 0 & 0 \\
\left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & 0 & 0 & 0 \\
0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right) & 0 & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & 0 & 0 \\
\left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & 0 & \left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & 0 \\
0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 \\
0 & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right) & 0 & 0 & \left(\frac{2}{h^2}-\frac{1}{2h}\right) \\
0 & 0 & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & 0 & \left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) & 0 \\
0 & 0 & 0 & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right) & \left(\frac{2}{h^2}-\frac{1}{2h}\right) \\
0 & 0 & 0 & 0 & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & 0 & \left(\frac{2}{h^2}+\frac{1}{2h}\right) & \left(\frac{-8}{h^2}-5\right)
\end{pmatrix}
\begin{pmatrix} U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ U_{1,3} \\ U_{2,3} \\ U_{3,3} \end{pmatrix} =
\begin{pmatrix} -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \end{pmatrix}$$

To simplify, let $c_1 = \left(\frac{-8}{h^2} - 5\right), c_2 = \left(\frac{2}{h^2} - \frac{1}{2h}\right), c_3 = \left(\frac{2}{h^2} + \frac{1}{2h}\right)$) since these repeat everywhere.

$$\begin{pmatrix}
c_1 & c_2 & 0 & c_2 & 0 & 0 & 0 & 0 & 0 \\
c_3 & c_1 & c_2 & 0 & c_2 & 0 & 0 & 0 & 0 \\
0 & c_3 & c_1 & 0 & 0 & c_2 & 0 & 0 & 0 \\
c_3 & 0 & 0 & c_1 & c_2 & 0 & c_2 & 0 & 0 \\
0 & c_3 & 0 & c_3 & c_1 & c_2 & 0 & c_2 & 0 \\
0 & 0 & c_3 & 0 & c_3 & c_1 & 0 & 0 & c_2 \\
0 & 0 & 0 & c_3 & 0 & 0 & c_1 & c_2 & 0 \\
0 & 0 & 0 & 0 & c_3 & 0 & c_3 & c_1 & c_2 \\
0 & 0 & 0 & 0 & 0 & c_3 & 0 & c_3 & c_1
\end{pmatrix}
\begin{pmatrix} U_{1,1} \\ U_{2,1} \\ U_{3,1} \\ U_{1,2} \\ U_{2,2} \\ U_{3,2} \\ U_{1,3} \\ U_{2,3} \\ U_{3,3} \end{pmatrix} =
\begin{pmatrix} -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \\ -5 \end{pmatrix}$$

So now we see the structure of $Au = f$. For the number of unknowns being n in one row, we have the following layout

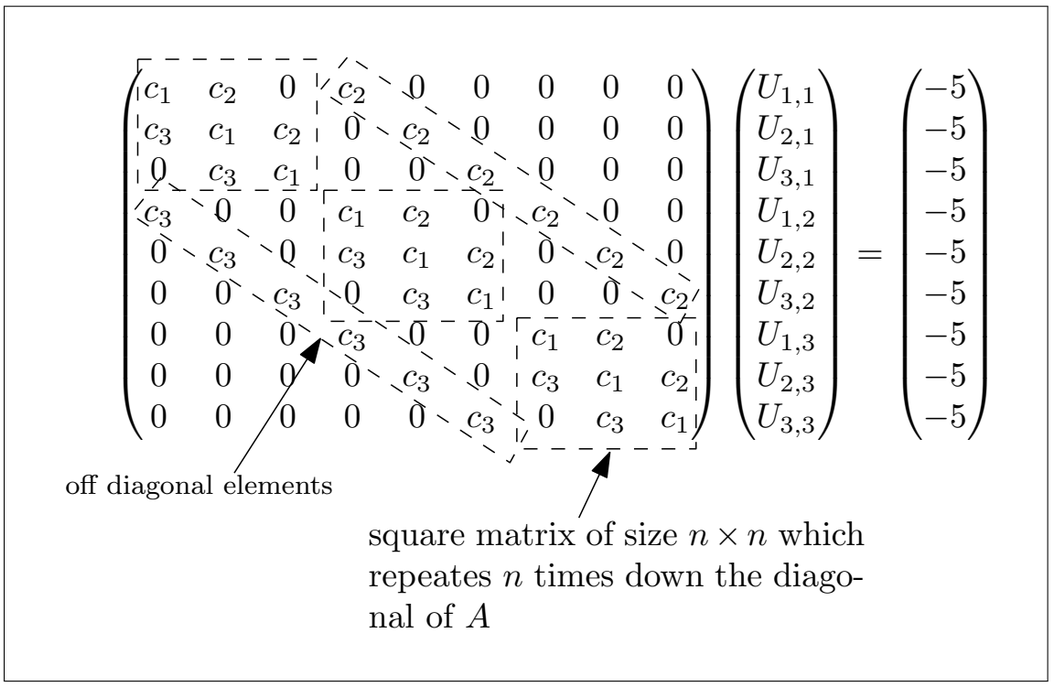


Figure 2.58: A matrix structure

The program `nma_EMA_471_HW6_problem_1.m` solves this $Au = f$ using the direct method and plots the solution. Maximum value found was

$$\boxed{u_{\max} = 0.1609}$$

Here is plot of the solution

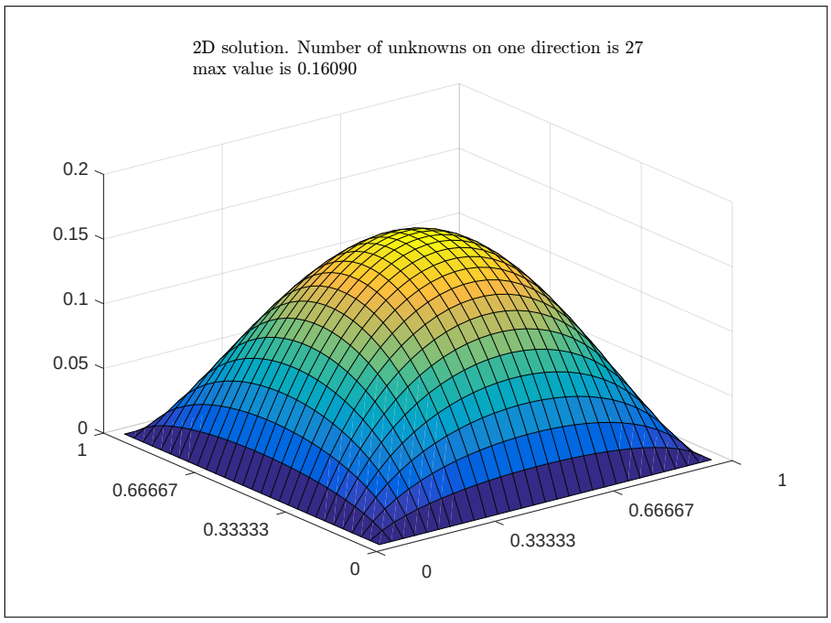


Figure 2.59: Solution plot

```
function nma_EMA_471_HW6_problem_1
%solution to probnlem 1, HW6, EMA 471

N  = 27;  %number of unknowns in one direction. change
%grid size as needed
h  = 1/(N+1); %grid spacing

c1 = (-8/h^2-5);
c2 = (2/h^2-1/(2*h));
c3 = (2/h^2+1/(2*h));
A  = lap2d(N,c1,c3,c2);   %make the A matrix
```

```
12
13 f  = -5*ones(N^2,1);  %RHS
14 u  = A\f; %direct solver
15 u  = reshape(u,N,N);
16 U  = zeros(N+2,N+2); %put the zero BC back in to plot
17 U(2:end-1,2:end-1) = u;  %put the solution into the larger grid
18 figure;
19 surf(U);
20
21 title({sprintf('2D solution. Number of unknowns on one direction is %d',N), ...
22        sprintf('max value is $%6.5f$',max(U(:)))},...
23               'Interpreter', 'latex','fontsize',10);
24
25 %relabel ticks for 0..1 in both directions
26 r = get(gca,'XTickLabel');
27 set(gca,'XTickLabel',num2str((0:1/(length(r)-1):1)'));
28 r = get(gca,'YTickLabel');
29 set(gca,'YTickLabel',num2str((0:1/(length(r)-1):1)'));
30
31 %found a bug in Matlab!
32 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
33 end
34 %==================================
35 function L2 = lap2d(n,middle,left,right)
36 %function to construct the 2D A matrix for this problem
37 e  = ones(n,1);
38 B  = [e*left e*middle/2 e*right];
39 L  = spdiags(B,[-1 0 1],n,n);
40 I  = speye(n);
41 Lm = kron(I,L); %does the central diagonal
42 Lo = kron(L,I); %does the off diagonal
43 L2 = Lm+Lo;
44 end
```

2.6.2 Problem 2

(2) (15 pts) If heat conduction takes place within a body that is simultaneously immersed in a fluid, there are two heat removal mechanisms. One particular energy balance gives:

$$\nabla^2 T - 20T = -200, \qquad \nabla^2 T = \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}$$

Solve this problem over the unit square, $0 \le x \le 1$, $0 \le y \le 1$ subject to insulated boundary conditions at $x = 0$ and $y = 0$ (temperature gradients normal to the boundary are zero) and $T = 0$ at $x = 1$ and $y = 1$. Find the peak temperature in the medium.

Figure 2.60: problem 2 description

On the left edge we have (where $i = 0$) we have

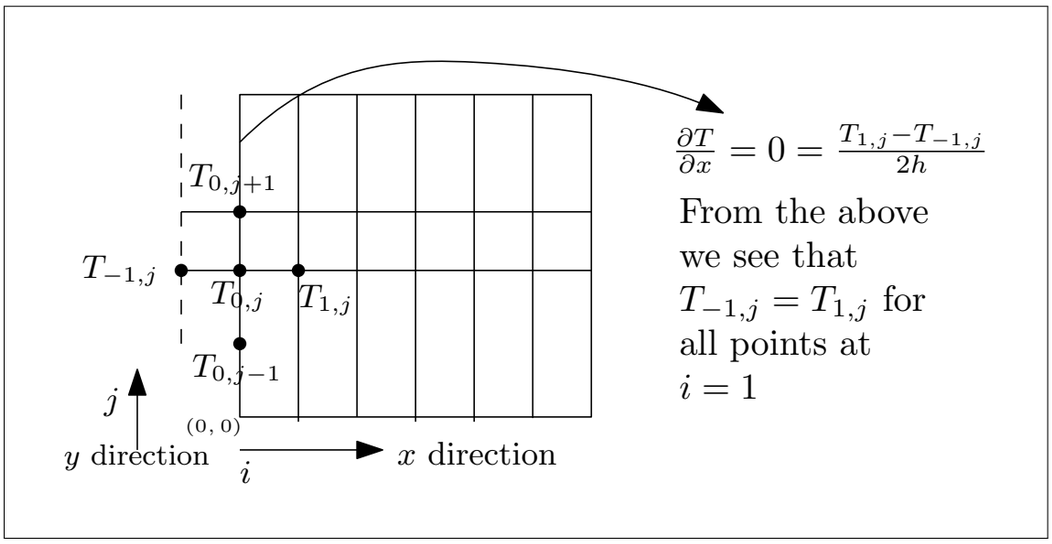


Figure 2.61: Left edge conditions

And on the bottom edge where $j = 0$ we have

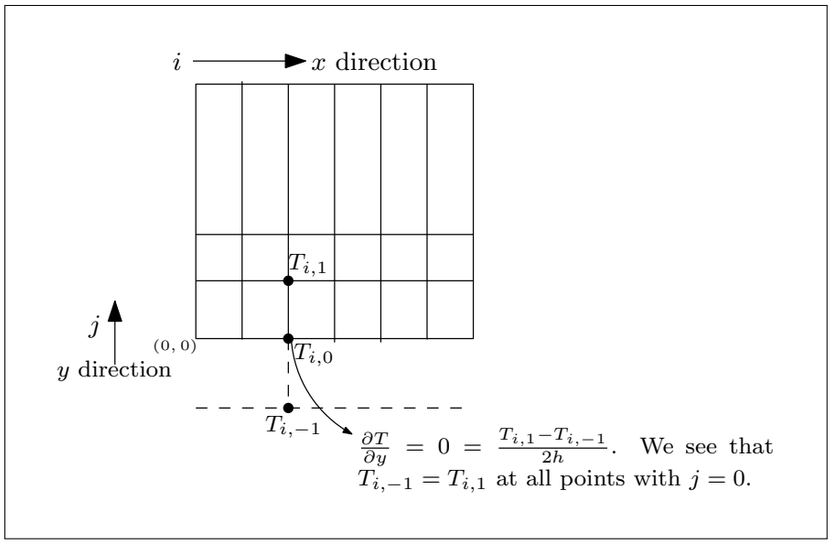


Figure 2.62: bottom edge conditions

Using the above relations, then at node $i = 0$,

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{1,j} - 2T_{0,j} + T_{-1,j}}{h^2}$$

Using the relation we found earlier, which said that $T_{-1,j} = T_{1,j}$, the above becomes

$$\frac{\partial^2 T}{\partial x^2} = \frac{2T_{1,j} - 2T_{0,j}}{h^2}$$

Therefore, the differential equation at $i = 0$ and for all j becomes

$$\begin{aligned} \nabla^2 T - 20T &= -200 \\ \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} - 20T &= -200 \\ \frac{2T_{1,j} - 2T_{0,j}}{h^2} + \frac{T_{0,j+1} - 2T_{0,j} + T_{0,j-1}}{h^2} - 20T_{0,j} &= -200 \end{aligned} \tag{1}$$

The above is what we will use on the left edge, for $j = 1 \cdots N$ where N is the number of internal nodes. We now find the PDE on the lower edge in similar way. On the bottom edge, where $j = 0$, we have

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,1} - 2T_{i,0} + T_{i,-1}}{h^2}$$

Using the relation we found earlier, which said that $T_{i,-1} = T_{i,1}$, then the above becomes

$$\frac{\partial^2 T}{\partial y^2} = \frac{2T_{i,1} - 2T_{i,0}}{h^2}$$

Therefore, the differential equation at $j = 0$ and for all i becomes

$$\begin{aligned} \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} - 20T &= -200 \\ \frac{T_{i+1,0} - 2T_{i,0} + T_{i-1,0}}{h^2} + \frac{2T_{i,1} - 2T_{i,0}}{h^2} - 20T_{i,0} &= -200 \end{aligned} \tag{2}$$

The above is what we will use on the bottom edge, for $i = 1 \cdots N$ where N is the number of internal nodes. Now that we found the PDE on the left and on the right edge, we write the PDE on the internal nodes, which is the standard form

$$\begin{aligned} \frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} - 20T &= -200 \\ \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{h^2} - 20T_{i,j} &= -200 \end{aligned} \tag{3}$$

Using (1,2,3) equations, we now find the $Ax = f$ form. Let us assume that $N = 3$, so our grid is the following

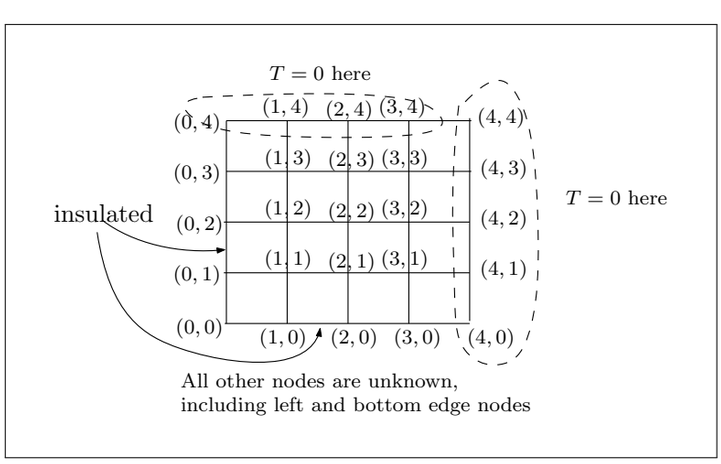


Figure 2.63: Example grid used to find the A matrix

For $i = 0, j = 0$, this is special node. We can either use the condition for the left edge to handle it, or the condition for the bottom edge, or use the average of the adjacent nodes. Let us use the bottom edge condition for it. Hence at this node the PDE is from (2)

$$\begin{aligned} \frac{T_{1,0} - 2T_{0,0} + T_{-1,0}}{h^2} + \frac{2T_{0,1} - 2T_{0,0}}{h^2} - 20T_{0,0} &= -200 \\ \frac{2T_{1,0} - 2T_{0,0}}{h^2} + \frac{2T_{0,1} - 2T_{0,0}}{h^2} - 20T_{0,0} &= -200 \\ 2T_{1,0} - 2T_{0,0} + 2T_{0,1} - 2T_{0,0} - 20h^2 T_{0,0} &= -200h^2 \\ 2T_{1,0} + T_{0,0}\left(-4 - 20h^2\right) + 2T_{0,1} &= -200h^2 \end{aligned} \tag{0,0}$$

For $i = 1, j = 0$, from (2)

$$\begin{aligned}\frac{T_{2,0}-2T_{1,0}+T_{0,0}}{h^2}+\frac{2T_{1,1}-2T_{1,0}}{h^2}-20T_{1,0}&=-200\\ T_{2,0}+T_{1,0}\left(-4-20h^2\right)+T_{0,0}+2T_{1,1}&=-200h^2\end{aligned} \tag{1,0}$$

And for $i=2,j=0$

$$\begin{aligned}\frac{T_{3,0}-2T_{2,0}+T_{1,0}}{h^2}+\frac{2T_{2,1}-2T_{2,0}}{h^2}-20T_{2,0}&=-200\\ T_{3,0}-4T_{2,0}+T_{1,0}+2T_{2,1}-20h^2T_{2,0}&=-200h^2\\ T_{3,0}+T_{2,0}\left(-4-20h^2\right)+T_{1,0}+2T_{2,1}&=-200h^2\end{aligned} \tag{2,0}$$

And for $i=3,j=0$

$$\begin{aligned}\frac{T_{4,0}-2T_{3,0}+T_{2,0}}{h^2}+\frac{2T_{3,1}-2T_{3,0}}{h^2}-20T_{3,0}&=-200\\ T_{4,0}-4T_{3,0}+T_{2,0}+2T_{3,1}-20h^2T_{3,0}&=-200h^2\\ T_{4,0}+T_{3,0}\left(-4-20h^2\right)+T_{2,0}+2T_{3,1}&=-200h^2\end{aligned}$$

But $T_{4,0}$ is know, since it is on the right edge and is zero there. Hence the above becomes

$$T_{3,0}\left(-4-40h\right)+T_{2,0}+2T_{3,1}=-200h^2 \tag{3,0}$$

We now move to the second grid line from the bottom. On the first node, which is $i=0,j=1$, we use the left edge PDE, which is (1). Hence

$$\begin{aligned}\frac{2T_{1,1}-2T_{0,1}}{h^2}+\frac{T_{0,2}-2T_{0,1}+T_{0,0}}{h^2}-20T_{0,1}&=-200\\ 2T_{1,1}-4T_{0,1}+T_{0,2}+T_{0,0}-20h^2T_{0,1}&=-200h^2\\ 2T_{1,1}+T_{0,1}\left(-4-20h^2\right)+T_{0,2}+T_{0,0}&=-200h^2\end{aligned} \tag{0,1}$$

And for $i=1,j=1$, this is an internal node, so we use (3)

$$\begin{aligned}\frac{T_{2,1}-2T_{1,1}+T_{0,1}}{h^2}+\frac{T_{1,2}-2T_{1,1}+T_{1,0}}{h^2}-20T_{1,1}&=-200\\ T_{2,1}-4T_{1,1}+T_{0,1}+T_{1,2}+T_{1,0}-20h^2T_{1,1}&=-200h^2\\ T_{2,1}+T_{1,1}\left(-4-20h^2\right)+T_{0,1}+T_{1,2}+T_{1,0}&=-200h^2\end{aligned} \tag{1,1}$$

And for $i=2,j=1$, this is an internal node, so we use (3)

$$\begin{aligned}\frac{T_{3,1}-2T_{2,1}+T_{1,1}}{h^2}+\frac{T_{2,2}-2T_{2,1}+T_{2,0}}{h^2}-20T_{2,1}&=-200\\ T_{3,1}+T_{2,1}\left(-4-20h^2\right)+T_{1,1}+T_{2,2}+T_{2,0}&=-200h^2\end{aligned} \tag{2,1}$$

And for $i=3,j=1$, this is an internal node, so we use (3) and set $T_{4,1}=0$ since known

$$\begin{aligned}\frac{T_{4,1}-2T_{3,1}+T_{2,1}}{h^2}+\frac{T_{3,2}-2T_{3,1}+T_{3,0}}{h^2}-20T_{3,1}&=-200\\ T_{3,1}\left(-4-20h^2\right)+T_{2,1}+T_{3,2}+T_{3,0}&=-200h^2\end{aligned} \tag{3,1}$$

For node $i=4,j=1$, this is a known value for T there, so we skip it. Going to the next grid row above, for $i=0,j=2$, this is a left edge node, so we use (1)

$$\frac{2T_{1,2} - 2T_{0,2}}{h^2} + \frac{T_{0,3} - 2T_{0,2} + T_{0,1}}{h^2} - 20T_{0,2} = -200$$
$$2T_{1,2} + T_{0,2}\left(-4 - 20h^2\right) + T_{0,3} + T_{0,1} = -200h^2 \tag{0,2}$$

For $i = 1, j = 2$, this is an internal node, so we use (3)

$$\frac{T_{2,2} - 2T_{1,2} + T_{0,2}}{h^2} + \frac{T_{1,3} - 2T_{1,2} + T_{1,1}}{h^2} - 20T_{1,2} = -200$$
$$T_{2,2} + T_{1,2}\left(-4 - 20h^2\right) + T_{0,2} + T_{1,3} + T_{1,1} = -200h^2 \tag{1,2}$$

For $i = 2, j = 2$, this is an internal node, so we use (3)

$$\frac{T_{3,2} - 2T_{2,2} + T_{1,2}}{h^2} + \frac{T_{2,3} - 2T_{2,2} + T_{2,1}}{h^2} - 20T_{2,2} = -200$$
$$T_{3,2} + T_{2,2}\left(-4 - 20h^2\right) + T_{1,2} + T_{2,3} + T_{2,1} = -200h^2 \tag{2,2}$$

And for $i = 3, j = 2$, this is an internal node, so we use (3) and set $T_{4,2} = 0$ since known

$$\frac{T_{4,2} - 2T_{3,2} + T_{2,2}}{h^2} + \frac{T_{3,3} - 2T_{3,2} + T_{3,1}}{h^2} - 20T_{3,2} = -200$$
$$T_{3,2}\left(-4 - 20h^2\right) + T_{2,2} + T_{3,3} + T_{3,1} = -200h^2 \tag{3,2}$$

For node $i = 4, j = 2$, this is a known value for T there, so we skip it. Going to the next grid row above, for $i = 0, j = 3$, this is a left edge node, so we use (1)

$$\frac{2T_{1,3} - 2T_{0,3}}{h^2} + \frac{T_{0,4} - 2T_{0,3} + T_{0,2}}{h^2} - 20T_{0,3} = -200$$
$$2T_{1,3} + T_{0,3}\left(-4 - 20h^2\right) + T_{0,4} + T_{0,2} = -200h^2$$

But $T_{0,4}$ is on the top edge, which is known and is zero, therefore the above is

$$2T_{1,3} + T_{0,3}\left(-4 - 20h^2\right) + T_{0,2} = -200h^2 \tag{0,3}$$

For node $i = 1, j = 3$, this is an internal node, so we use (3)

$$\frac{T_{2,3} - 2T_{1,3} + T_{0,3}}{h^2} + \frac{T_{1,4} - 2T_{1,3} + T_{1,2}}{h^2} - 20T_{1,3} = -200$$
$$T_{2,3} + T_{1,3}\left(-4 - 20h^2\right) + T_{0,3} + T_{1,4} + T_{1,2} = -200h^2$$

But $T_{1,4}$ is on the top edge, which is known and is zero, therefore the above is

$$T_{2,3} + T_{1,3}\left(-4 - 20h^2\right) + T_{0,3} + T_{1,2} = -200h^2 \tag{1,3}$$

And on node $i = 2, j = 3$, this is an internal node, so we use (3)

$$\frac{T_{3,3} - 2T_{2,3} + T_{1,3}}{h^2} + \frac{T_{2,4} - 2T_{2,3} + T_{2,2}}{h^2} - 20T_{2,3} = -200$$
$$T_{3,3} + T_{2,3}\left(-4 - 20h^2\right) + T_{1,3} + T_{2,4} + T_{2,2} = -200h^2$$

But $T_{2,4}$ is on the top edge, which is known and is zero, therefore the above is

$$T_{3,3} + T_{2,3}\left(-4 - 20h^2\right) + T_{1,3} + T_{2,2} = -200h^2 \tag{2,3}$$

Finally, for node $i = 3, j = 3$

$$\frac{T_{4,3} - 2T_{3,3} + T_{2,3}}{h^2} + \frac{T_{3,4} - 2T_{3,3} + T_{3,2}}{h^2} - 20T_{3,3} = -200$$
$$T_{4,3} + T_{3,3}\left(-4 - 20h^2\right) + T_{2,3} + T_{3,4} + T_{3,2} = -200h^2$$

But $T_{3,4}$ and $T_{4,3}$ are known and is zero, therefore the above is

$$T_{3,3}\left(-4 - 20h^2\right) + T_{2,3} + T_{3,2} = -200h^2 \tag{3,3}$$

Now we able to see the form of the matrix A by writing the equations for this small grid. There are 16 unknowns. Let $\left(-4 - 20h^2\right) = \alpha$ then

$$\begin{array}{c}
\text{bottom edge} \\
\text{bottom edge} \\
\text{bottom edge} \\
\text{bottom edge} \\
\text{left edge} \\
\text{internal nodes} \\
\text{internal nodes} \\
\text{internal nodes} \\
\text{left edge} \\
\text{internal nodes} \\
\text{internal nodes} \\
\text{internal nodes} \\
\text{left edge} \\
\text{row below top edge} \\
\text{row below top edge} \\
\text{row below top edge}
\end{array}
\begin{bmatrix}
\alpha & 2 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & \alpha & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & \alpha & 1 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & \alpha & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & \alpha & 2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha & 2 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & \alpha
\end{bmatrix}
\begin{bmatrix} T_{0,0} \\ T_{1,0} \\ T_{2,0} \\ T_{3,0} \\ T_{0,1} \\ T_{1,1} \\ T_{2,1} \\ T_{3,1} \\ T_{0,2} \\ T_{1,2} \\ T_{2,2} \\ T_{3,2} \\ T_{0,3} \\ T_{1,3} \\ T_{2,3} \\ T_{3,3} \end{bmatrix}
=
\begin{bmatrix} -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \\ -200h^2 \end{bmatrix}$$

The above was implemented in the matlab program `nma_EMA_471_HW6_problem_2.m`. The maximum value of T found was

9.577

Here is the 3D plot and a contour plot.

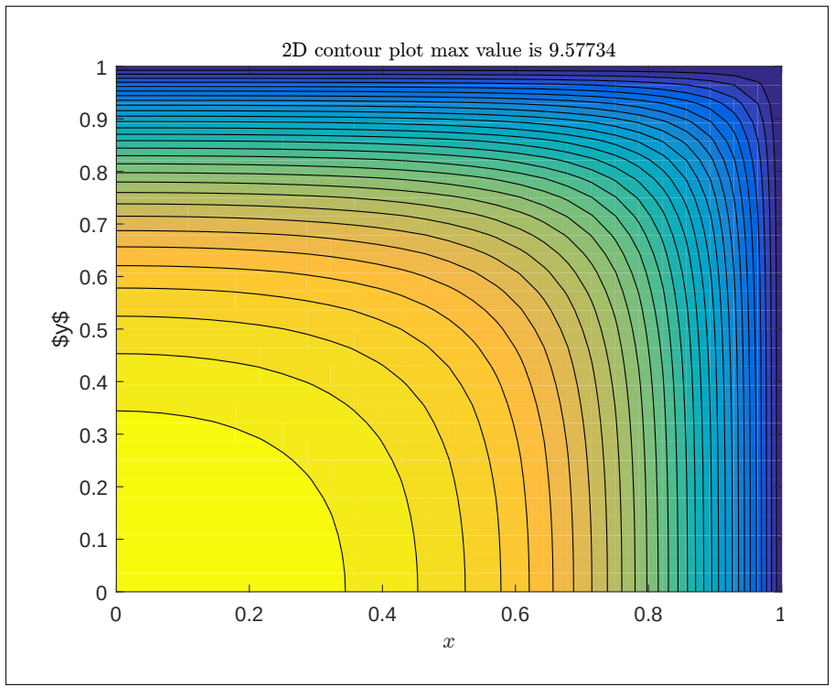


Figure 2.64: contour plot

And this is the 3D plot.

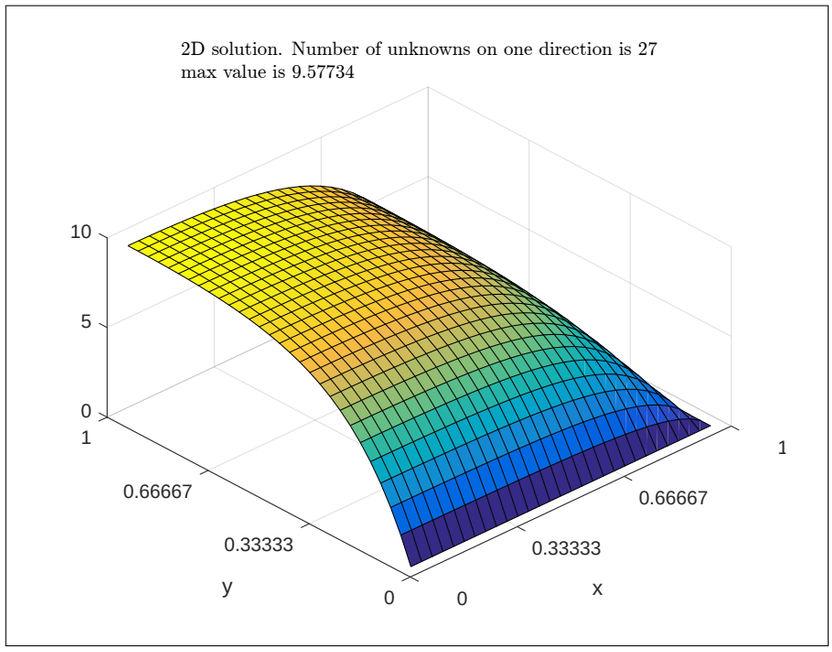


Figure 2.65: 3D plot

I have also solved this in Mathematica, using the finite element solver build into NDSOlve. I also obtained the same result. Here is the contour plot and the 3D plot.

```
Clear[u, x, y];
r = NDSolveValue[{
   D[u[x, y], {x, 2}] + D[u[x, y], {y, 2}] - 20*u[x, y] == - 200 +
     NeumannValue[0, x == 0] + NeumannValue[0, y == 0],
      DirichletCondition[u[x, y] == 0, x == 1],
   DirichletCondition[u[x, y] == 0, y == 1 ]}, u, {x, 0, 1}, {y, 0, 1},
    Method -> {"FiniteElement",
    "MeshOptions" -> {"BoundaryMeshGenerator" -> "Continuation"}}]

ContourPlot[r[x, y], {x, 0, 1}, {y, 0, 1}]
Plot3D[r[x, y], {x, 0, 1}, {y, 0, 1}, Mesh -> All]
```

.

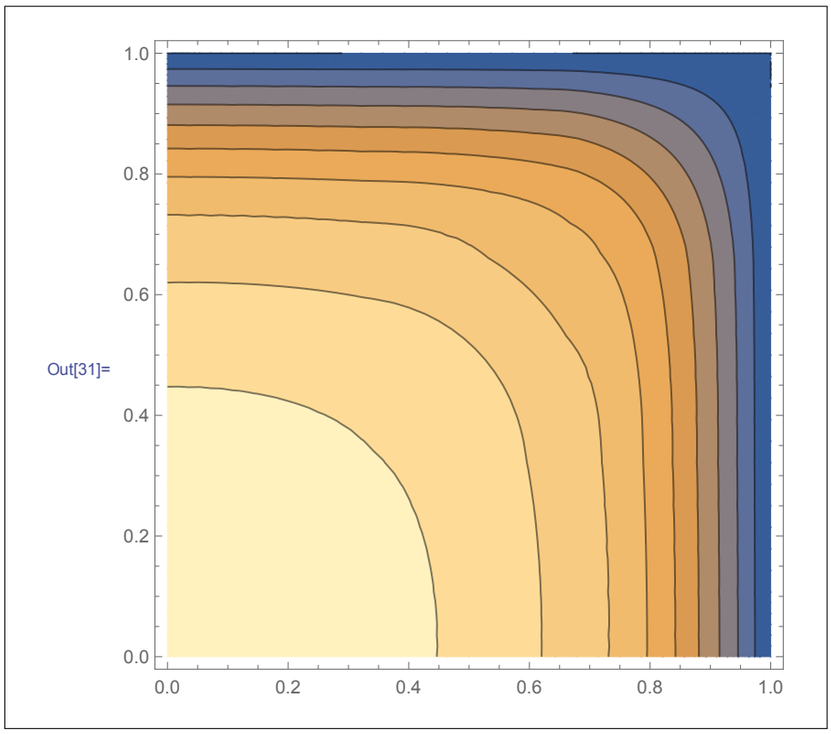


Figure 2.66: Contour plot using Mathematica

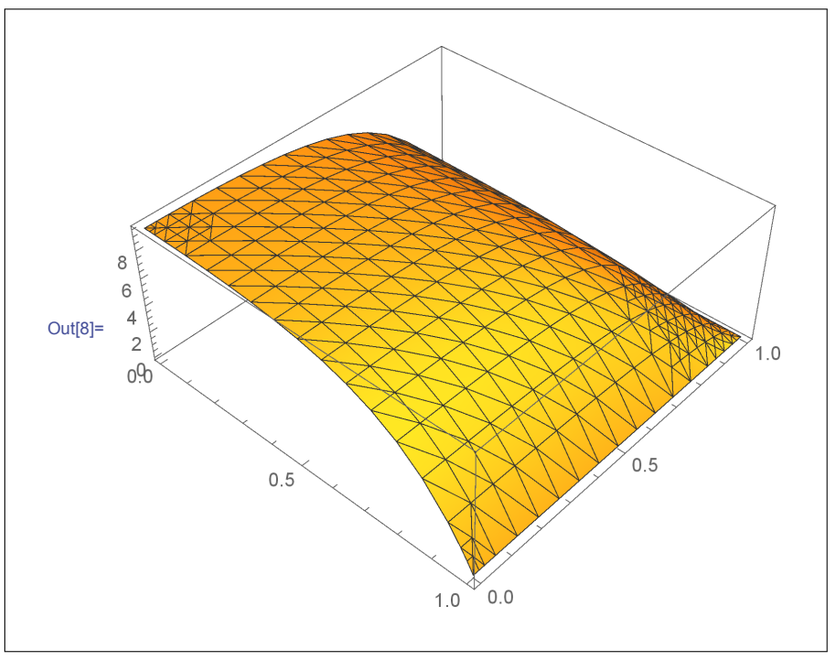


Figure 2.67: 3D plot using Mathematica

```
function nma_EMA_471_HW6_problem_2
%solution to probnlem 2, HW6, EMA 471

close all;
n  = 27;  %number of internal nodes in one direction.
%change grid size as needed
h  = 1/(n+1); %grid spacing
A  = make_A_matrix(h,n);
f  = -200*h^2*ones((n+1)^2,1);  %RHS
u  = A\f; %direct solver

u  = reshape(u,n+1,n+1)'; %change it for natural setting
u  = [u;zeros(1,n+1)];    %add zero boundary conditions
u  = [u zeros(n+2,1)];
u  = flipud(u); %so it looks right
figure;
surf(u);
```

```
18
19 title({sprintf('2D solution. Number of unknowns on one direction is %d',n), ...
20     sprintf('max value is $%6.5f$',max(u(:)))},...
21     'Interpreter', 'latex','fontsize',10);
22
23 %relable ticks for 0..1 in both directions
24 r = get(gca,'XTickLabel');
25 set(gca,'XTickLabel',num2str((0:1/(length(r)-1):1)'));
26 r = get(gca,'YTickLabel');
27 set(gca,'YTickLabel',num2str((0:1/(length(r)-1):1)'));
28 xlabel('$x$','Interpreter', 'latex');
29 ylabel('$y$','Interpreter', 'latex');
30
31 %found a bug in Matlab!
32 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
33
34 %do contour plot also
35 x     = 0:h:1;
36 y     = 0:h:1;
37 [X,Y] = meshgrid(x,y);
38
39 figure
40 contourf(X,Y,flipud(u),30);
41 xlabel('$x$', 'Interpreter', 'latex');
42 ylabel('$y$','Interpreter', 'latex');
43 title({sprintf('2D contour plot max value is $%6.5f$',...
44        max(u(:)))},...
45     'Interpreter', 'latex','fontsize',10);
46 end
47 %==================================
48 function A = make_A_matrix(h,n)
49 %h is grid spacing.
50 %n is number of internal nodes on any one grid direction.
51 %please see report for details.
52
53 a = -4-20*h^2;
54 A = zeros((n+1)^2,(n+1)^2);
55
56 %make T(0,0), lower corner node
57 A(1,1)   = a;
58 A(1,2)   = 2;
59 A(1,n+2) = 2;
60
61 %rest of bottom edge nodes. These are node on lower edge, where
62 %it is insulated
63 for i = 2:n
64     A(i,i)     = a;
65     A(i,i-1)   = 1;
66     A(i,i+1)   = 1;
67     A(i,i+n+1) = 2;
68 end
69 %last node on botton edge. special handling
70 i          = i+1;
71 A(i,i-1)   = 1;
72 A(i,i)     = a;
73 A(i,i+n+1) = 2;
74
75 %now make left edge rows. First special
76 for i = n+2:n+1:((n+1)^2-n)
77     A(i,i-(n+1)) = 1;
78
79     if i==n+2
80        A(i,i)     = a;
```

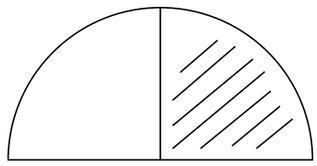
```
       A(i,i+1)   = 2;
       A(i,i+n+1) = 1;
    else
      A(i,i)      = a;
      A(i,i+1)    = 2;
      if i<((n+1)^2-n)
          A(i,i+n+1) = 1;
      end
    end
end

%now make middle rows
for i = n+3: n+1: ((n+1)^2-2*n)
   for k = i:i+n-1
            A(k,k-(n+1)) = 1;
            A(k,k-1)     = 1;
            A(k,k)       = a;
            if k<(i+n-1)
               A(k,k+1) = 1;
            end
            A(k,k+n+1) = 1;
   end
end

%now do the top edge
for i = (n+1)^2-(n-1):(n+1)^2
    A(i,i-(n+1))  = 1;
    A(i,i-1)      = 1;
    A(i,i)        = a;
    if i<(n+1)^2
        A(i,i+1) = 1;
    end
end
end
```

2.6.3 Problem 3

(3) (15 pts) The Laplacian operator can be represented in other coordinate systems. In a cylindrical system, it takes the form:

$$\nabla^2 T = \frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2}$$

Solve the problem, $\nabla^2 T = -q'''/k$, over a semicircular domain that is defined by $0 \le r \le 1$ and $0 \le \theta \le \pi$ where the right hand side of the equation is –200 when $0 \le \theta \le \pi/2$ and is equal to 0 when $\pi/2 < \theta \le \pi$. The heat source exists over just half the plate as shown below:



The extension of finite difference methods to this problem is straightforward at all places except the origin where $r \to 0$. Instead of using the Laplacian operator directly at the origin,

we can instead write an energy balance based on energy produced in a small control volume around the origin. When we do this, the result is:

$$T_0 = \frac{0.5T_1 + T_2 + ... + T_{N-1} + 0.5T_N}{N-1} + \frac{q'''\Delta r^2}{8k}$$

Here, N is the total number of circumferential nodes one radial mesh spacing from the origin, with T_1 and T_N being the nodal temperatures at $\theta = 0$ and $\theta = \pi$ respectively, and the other T_i's being nodal temperatures between these limits. T_0 is the temperature at $r = 0$. For example, if $N = 5$, this corresponds to an angular mesh spacing of $\pi/4$. (I'm not recommending an angular mesh spacing this coarse; it's just for illustration.) In the limit that the radial mesh spacing goes to zero, this states that the temperature at the origin is a kind of simple average of its nearest neighbors, where the temperatures at the edges receive a half-weighting relative to those in the interior. Since the mesh spacing isn't actually infinitely fine, we'll have the term on the far right. Since, for this specific problem, q'''/k is 200, the term on the far right is actually $25\Delta r^2$. For all nodes other than the one at the origin, we write finite-difference equations based on the Laplacian operator.

One other issue to consider is the treatment of those nodes at $\theta = \pi/2$, a line that straddles the region where there is a heat source and where there is none. The term on the right hand side of the heat conduction equation should be –100 for those nodes.

Solve this problem over the domain subject to the conditions $T = 0$ at $r = 1$ and an insulated condition $\partial T/\partial\theta = 0$ at $\theta = 0$ and $\theta = \pi$. Report the temperature at the origin, T_0.

Reminder: If you have not done so already, send me an email about what you are planning to do for your project. Take some time to think about it and make sure you run it past me if not one of the default projects. You don't want to be in the position of leaving this to the last minute. We will allocate some in-class time for you to work on these as well.

Figure 2.68: problem 3 description

The grid numbering used is the following

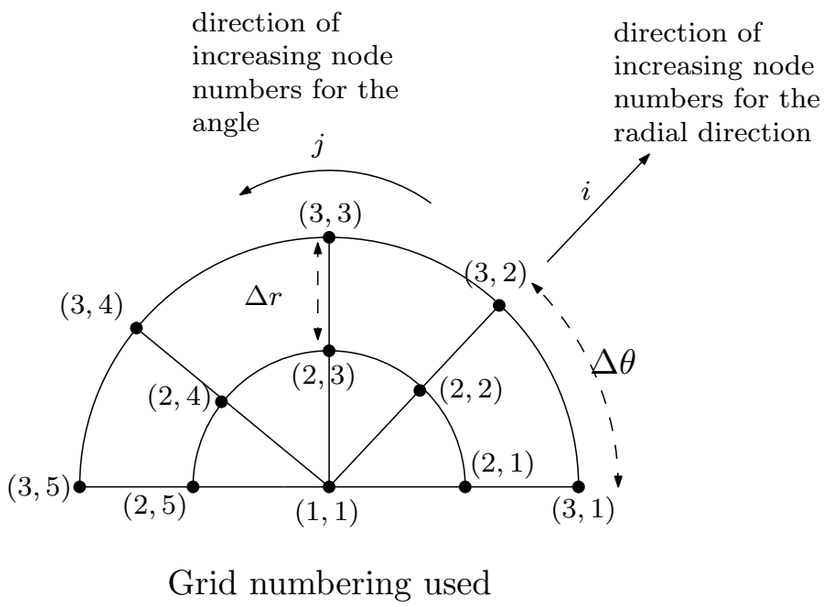


Figure 2.69: Grid numbering for problem 3

The PDE is, for $0 \le \theta \le \frac{\pi}{2}$

$$\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} = -200$$

And for $\frac{\pi}{2} < \theta \le \pi$

$$\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} = 0$$

For the nodes with $\theta = \frac{\pi}{2}$ we will use

$$\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} = -100$$

Using centered difference, and using T_{ij} to mean $T_{r_i,\theta j}$ then the above can be written as

$$\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} = -200 \quad (1)$$

Where $i = 1 \cdots N_r$ where N_r is the number of grid points in the radial direction, which is 3 in the diagram above. The following diagram shows the boundary conditions to use.

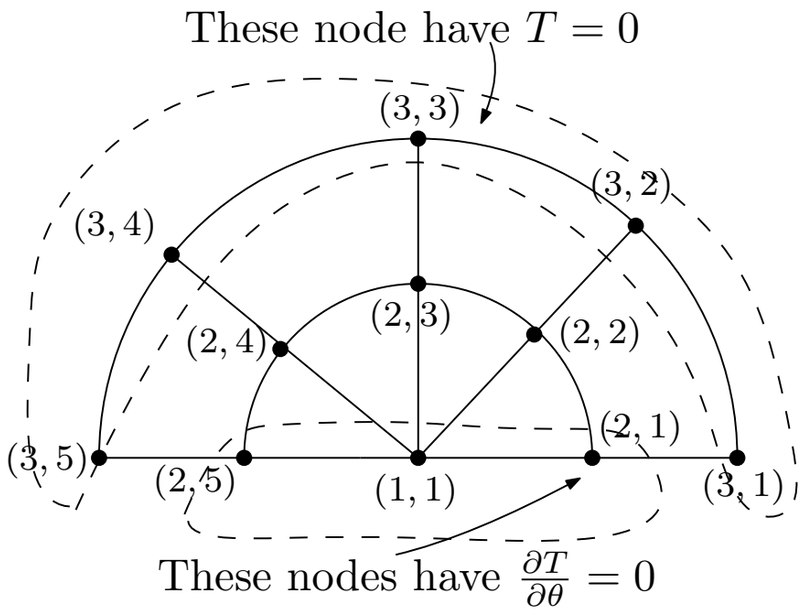


Figure 2.70: boundary conditions for problem 3

We now find the A matrix in order to solve the problem using the direct method. We assume $N_\theta = 7$ and $N_r = 5$ for the purpose of seeing what the A structure is

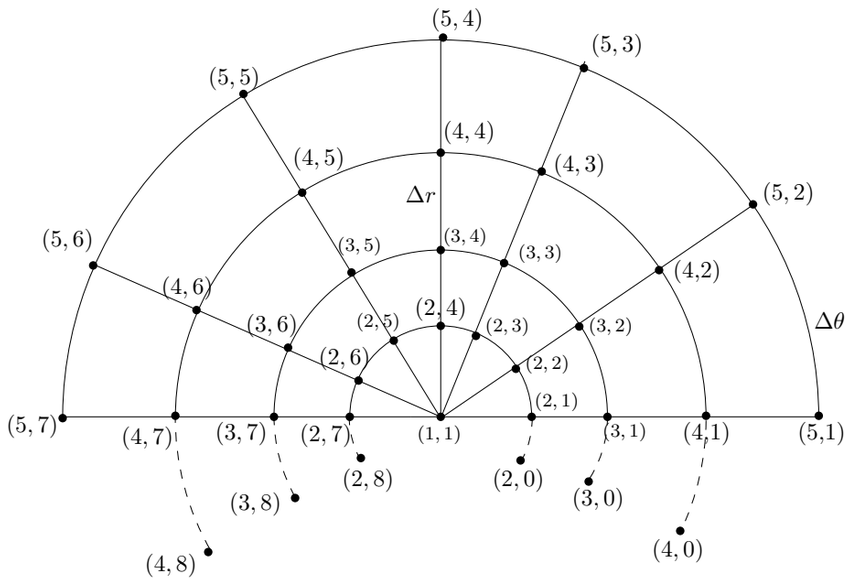


Figure 2.71: Example grid using $N_r = 5$ and $N_\theta = 7$ problem 3

For node (1,1), this is the special node. which is

$$T_{1,1} = \frac{0.5T_{2,1} + T_{2,2} + T_{2,3} + \cdots + T_{2,N-1} + 0.5T_{2,N_\theta}}{N_\theta - 1} + 25\Delta r^2$$

Where N_θ is the number of grid point in the angular direction. For example, in the diagram given above, $N_\theta = 7$, hence

$$T_{1,1} - \frac{0.5T_{2,1} + T_{2,2} + T_{2,3} + T_{2,4} + T_{2,5} + T_{2,6} + 0.5T_{2,7}}{6} = 25\Delta r^2$$

$$T_{1,1} - \frac{1}{12}T_{2,1} - \frac{1}{6}T_{2,2} - \frac{1}{6}T_{2,3} - \frac{1}{6}T_{2,4} - \frac{1}{6}T_{2,5} - \frac{1}{6}T_{2,6} - \frac{1}{12}T_{2,7} = 25\Delta r^2 \tag{1,1}$$

For node (2,1), this is an insulated node. Hence by introducing an imaginary node as

shown above, then on this node, the PDE is

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -200 \\
\frac{T_{1,1} - 2T_{21} + T_{3,1}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,1} - T_{1,1}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,0} - 2T_{2,1} + T_{2,2}}{\Delta \theta^2} &= -200
\end{aligned}$$

But due to insulation, then $\frac{\partial T}{\partial \theta} = \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta\theta} = 0$ which means $T_{i,j+1} = T_{i,j-1}$, or at this node $T_{2,0} = T_{2,2}$, hence the above becomes

$$\frac{T_{1,1} - 2T_{21} + T_{3,1}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,1} - T_{1,1}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{2T_{2,2} - 2T_{2,1}}{\Delta \theta^2} = -200$$

Collecting terms

$$\begin{aligned}
T_{1,1}\left(\frac{1}{\Delta r^2} - \frac{1}{2\Delta r^2}\right) + T_{2,1}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta \theta^2}\right) + T_{3,1}\left(\frac{1}{\Delta r^2} + \frac{1}{2\Delta r^2}\right) + T_{2,2}\left(\frac{2}{\Delta r^2 \Delta \theta^2}\right) &= -200 \\
T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{2,1}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta \theta^2}\right) + T_{3,1}\left(\frac{3}{2\Delta r^2}\right) + T_{2,2}\left(\frac{2}{\Delta r^2 \Delta \theta^2}\right) &= -200 \quad (2,1)
\end{aligned}$$

At node (3,1)

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -200 \\
\frac{T_{2,1} - 2T_{3,1} + T_{4,1}}{\Delta r^2} + \frac{1}{2(\Delta r)}\frac{T_{4,1} - T_{2,1}}{2\Delta r} + \frac{1}{[2(\Delta r)]^2}\frac{T_{3,0} - 2T_{3,1} + T_{3,2}}{\Delta \theta^2} &= -200
\end{aligned}$$

But due to insulation, then $\frac{\partial T}{\partial \theta} = \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta\theta} = 0$ which means $T_{i,j+1} = T_{i,j-1}$, or at this node $T_{3,0} = T_{3,2}$, hence the above becomes

$$\frac{T_{2,1} - 2T_{3,1} + T_{4,1}}{\Delta r^2} + \frac{1}{2(\Delta r)}\frac{T_{4,1} - T_{2,1}}{2\Delta r} + \frac{1}{[2(\Delta r)]^2}\frac{2T_{3,2} - 2T_{3,1}}{\Delta \theta^2} = -200$$

Collecting terms

$$\begin{aligned}
T_{2,1}\left(\frac{1}{\Delta r^2} - \frac{1}{4\Delta r^2}\right) + T_{3,1}\left(-\frac{2}{\Delta r^2} - \frac{2}{[2(\Delta r)]^2 \Delta \theta^2}\right) + T_{4,1}\left(\frac{1}{\Delta r^2} + \frac{1}{4\Delta r^2}\right) + T_{3,2}\left(\frac{2}{[2(\Delta r)]^2 \Delta \theta^2}\right) &= -200 \\
T_{2,1}\left(\frac{3}{4\Delta r^2}\right) + T_{3,1}\left(-\frac{2}{\Delta r^2} - \frac{1}{2\Delta r^2 \Delta \theta^2}\right) + T_{4,1}\left(\frac{5}{4\Delta r^2}\right) + T_{3,2}\left(\frac{1}{2\Delta r^2 \Delta \theta^2}\right) &= -200 \quad (3,1)
\end{aligned}$$

At node (4,1)

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -200 \\
\frac{T_{3,1} - 2T_{4,1} + T_{5,1}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{T_{5,1} - T_{3,1}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,0} - 2T_{4,1} + T_{4,2}}{\Delta \theta^2} &= -200
\end{aligned}$$

But due to insulation, then $\frac{\partial T}{\partial \theta} = \frac{T_{i,j+1} - T_{i,j-1}}{2\Delta\theta} = 0$ which means $T_{i,j+1} = T_{i,j-1}$, or at this node $T_{4,0} = T_{4,2}$, hence the above becomes (and also $T_{5,1} = 0$ since on boundary)

$$\frac{T_{3,1} - 2T_{4,1}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{-T_{3,1}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{-2T_{4,1} + 2T_{4,2}}{\Delta\theta^2} = -200$$

Collecting terms

$$T_{3,1}\left(\frac{5}{6\Delta r^2}\right) + T_{4,1}\left(-\frac{2}{\Delta r^2} - \frac{2}{9\Delta r^2\Delta\theta^2}\right) + T_{4,2}\left(\frac{2}{9\Delta r^2\Delta\theta^2}\right) = -200 \tag{4,1}$$

The above *completes half of the bottom grid row.* Now we move to the next grid at, one $\Delta\theta$ above.

At node $(2,2)$, this is an internal node.

$$\begin{aligned}\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200\\ \frac{T_{1,2} - 2T_{22} + T_{3,2}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,2} - T_{1,2}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,1} - 2T_{22} + T_{2,3}}{\Delta\theta^2} &= -200\end{aligned}$$

Collecting terms

$$T_{1,2}\left(\frac{1}{\Delta r^2} - \frac{1}{2\Delta r^2}\right) + T_{22}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2\Delta\theta^2}\right) + T_{3,2}\left(\frac{1}{\Delta r^2} + \frac{1}{2\Delta r^2}\right) + T_{2,1}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) + T_{2,3}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) = -200$$

But $T_{1,2} = T_{1,1}$. This is the same node. Hence the above becomes

$$T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{22}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2\Delta\theta^2}\right) + T_{3,2}\left(\frac{3}{2\Delta r^2}\right) + T_{2,1}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) + T_{2,3}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) = -200 \tag{2,2}$$

At node $(3,2)$, this is an internal node

$$\begin{aligned}\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200\\ \frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta\theta^2} &= -200\\ \frac{T_{2,2} - 2T_{3,2} + T_{4,2}}{\Delta r^2} + \frac{1}{2\Delta r}\frac{T_{4,2} - T_{2,2}}{2\Delta r} + \frac{1}{[2(\Delta r)]^2}\frac{T_{3,1} - 2T_{3,2} + T_{3,3}}{\Delta\theta^2} &= -200\\ T_{2,2}\left(\frac{3}{4\Delta r^2}\right) + T_{3,2}\left(\frac{-2}{\Delta r^2} - \frac{1}{2\Delta r^2\Delta\theta^2}\right) + T_{4,2}\left(\frac{5}{4\Delta r^2}\right) + T_{3,1}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right) + T_{3,3}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right) &= -200\end{aligned} \tag{3,2}$$

At node $(4,2)$,this is an internal node

$$\begin{aligned}\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -200\\ \frac{T_{3,2} - 2T_{4,2} + T_{5,2}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{T_{5,2} - T_{3,2}}{2\Delta r} + \frac{1}{[3(\Delta r)]^2}\frac{T_{4,1} - 2T_{4,2} + T_{4,3}}{\Delta\theta^2} &= -200\end{aligned}$$

But $T_{5,2} = 0$ since on boundary, hence

$$\frac{T_{3,2}-2T_{4,2}}{\Delta r^2}+\frac{1}{3\Delta r}\frac{-T_{3,2}}{2\Delta r}+\frac{1}{[3(\Delta r)]^2}\frac{T_{4,1}-2T_{4,2}+T_{4,3}}{\Delta\theta^2}=-200$$

$$T_{3,2}\left(\frac{1}{\Delta r^2}-\frac{1}{6\Delta r^2}\right)+T_{4,2}\left(\frac{-2}{\Delta r^2}-\frac{2}{9\Delta r^2\Delta\theta^2}\right)+T_{4,1}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)+T_{4,3}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)=-200$$

$$T_{3,2}\left(\frac{5}{6\Delta r^2}\right)+T_{4,2}\left(\frac{-2}{\Delta r^2}-\frac{2}{9\Delta r^2\Delta\theta^2}\right)+T_{4,1}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)+T_{4,3}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)=-200 \quad (4,2)$$

This *completes first internal grid line on the right half.* Now we move another $\Delta\theta$ anti-clock wise and process the central line.

At node $(2,3)$

$$\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}=-200$$

$$\frac{T_{i-1,j}-2T_{ij}+T_{i+1,j}}{\Delta r^2}+\frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1}-2T_{ij}+T_{i,j+1}}{\Delta\theta^2}=-200$$

$$\frac{T_{1,3}-2T_{2,3}+T_{3,3}}{\Delta r^2}+\frac{1}{\Delta r}\frac{T_{3,3}-T_{1,3}}{2\Delta r}+\frac{1}{\Delta r^2}\frac{T_{2,2}-2T_{2,3}+T_{2,4}}{\Delta\theta^2}=-200$$

$$T_{1,1}\left(\frac{1}{2\Delta r^2}\right)+T_{2,3}\left(\frac{-2}{\Delta r^2}-\frac{2}{\Delta r^2\Delta\theta^2}\right)+T_{3,3}\left(\frac{3}{2\Delta r^2}\right)+T_{2,2}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right)+T_{2,4}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right)=-200 \quad (2,3)$$

At node $(3,3)$

$$\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}=-200$$

$$\frac{T_{i-1,j}-2T_{ij}+T_{i+1,j}}{\Delta r^2}+\frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1}-2T_{ij}+T_{i,j+1}}{\Delta\theta^2}=-200$$

$$\frac{T_{2,3}-2T_{3,3}+T_{4,3}}{\Delta r^2}+\frac{1}{2\Delta r}\frac{T_{4,3}-T_{2,3}}{2\Delta r}+\frac{1}{4\Delta r^2}\frac{T_{3,2}-2T_{3,3}+T_{3,4}}{\Delta\theta^2}=-200$$

$$T_{2,3}\left(\frac{3}{4\Delta r^2}\right)+T_{3,3}\left(\frac{-2}{\Delta r^2}-\frac{1}{2\Delta r^2\Delta\theta^2}\right)+T_{4,3}\left(\frac{5}{4\Delta r^2}\right)+T_{3,2}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right)+T_{3,4}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right)=-200 \quad (3,3)$$

At node $(4,3)$

$$\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}=-200$$

$$\frac{T_{i-1,j}-2T_{ij}+T_{i+1,j}}{\Delta r^2}+\frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1}-2T_{ij}+T_{i,j+1}}{\Delta\theta^2}=-200$$

$$\frac{T_{3,3}-2T_{4,3}+T_{5,3}}{\Delta r^2}+\frac{1}{3\Delta r}\frac{T_{5,3}-T_{3,3}}{2\Delta r}+\frac{1}{9\Delta r^2}\frac{T_{4,2}-2T_{4,3}+T_{4,4}}{\Delta\theta^2}=-200$$

But $T_{5,3}=0$ since at boundary, hence

$$\frac{T_{3,3}-2T_{4,3}}{\Delta r^2}+\frac{1}{3\Delta r}\frac{-T_{3,3}}{2\Delta r}+\frac{1}{9\Delta r^2}\frac{T_{4,2}-2T_{4,3}+T_{4,4}}{\Delta\theta^2}=-200$$

$$T_{3,3}\left(\frac{5}{6\Delta r^2}\right)+T_{4,3}\left(\frac{-2}{\Delta r^2}-\frac{2}{9\Delta r^2\Delta\theta^2}\right)+T_{4,2}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)+T_{4,4}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right)=-200 \quad (4,3)$$

Now we move to the central line. On this line the PDE is $\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}=-100$.

Hence at node $(2,4)$ we have

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -100 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -100 \\
\frac{T_{1,4} - 2T_{2,4} + T_{3,4}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,4} - T_{1,4}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,3} - 2T_{2,4} + T_{2,5}}{\Delta \theta^2} &= -100 \\
T_{1,4}\left(\frac{1}{\Delta r^2} - \frac{1}{2\Delta r^2}\right) + T_{2,4}\left(-\frac{2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta \theta^2}\right) + T_{3,4}\left(\frac{1}{\Delta r^2} + \frac{1}{2\Delta r^2}\right) + T_{2,3}\left(\frac{1}{\Delta r^2 \Delta \theta^2}\right) + T_{2,5}\left(\frac{1}{\Delta r^2 \Delta \theta^2}\right) &= -100
\end{aligned}$$

But $T_{1,4} = T_{1,1}$. This is the same node. Hence the above becomes

$$T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{2,4}\left(-\frac{2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta \theta^2}\right) + T_{3,4}\left(\frac{3}{2\Delta r^2}\right) + T_{2,3}\left(\frac{1}{\Delta r^2 \Delta \theta^2}\right) + T_{2,5}\left(\frac{1}{\Delta r^2 \Delta \theta^2}\right) = -100 \tag{2,4}$$

At node $(3,4)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -100 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -100 \\
\frac{T_{2,4} - 2T_{3,4} + T_{4,4}}{\Delta r^2} + \frac{1}{2\Delta r}\frac{T_{4,4} - T_{2,4}}{2\Delta r} + \frac{1}{4\Delta r^2}\frac{T_{3,3} - 2T_{3,4} + T_{3,5}}{\Delta \theta^2} &= -100 \\
T_{2,4}\left(\frac{3}{4\Delta r^2}\right) + T_{3,4}\left(\frac{-2}{\Delta r^2} - \frac{1}{2\Delta r^2 \Delta \theta^2}\right) + T_{4,4}\left(\frac{5}{4\Delta r^2}\right) + T_{3,3}\left(\frac{1}{4\Delta r^2 \Delta \theta^2}\right) + T_{3,5}\left(\frac{1}{4\Delta r^2 \Delta \theta^2}\right) &= -100
\end{aligned} \tag{3,4}$$

At node $(4,4)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= -100 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= -100 \\
\frac{T_{3,4} - 2T_{4,4} + T_{5,4}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{T_{5,4} - T_{3,4}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,3} - 2T_{4,4} + T_{4,5}}{\Delta \theta^2} &= -100
\end{aligned}$$

But $T_{5,4} = 0$ since at B.C. hence

$$\begin{aligned}
\frac{T_{3,4} - 2T_{4,4}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{-T_{3,4}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,3} - 2T_{4,4} + T_{4,5}}{\Delta \theta^2} &= -100 \\
T_{3,4}\left(\frac{5}{6\Delta r^2}\right) + T_{4.4}\left(\frac{-2}{\Delta r^2} + \frac{-2}{9\Delta r^2 \Delta \theta^2}\right) + T_{4,3}\left(\frac{1}{9\Delta r^2 \Delta \theta^2}\right) + T_{4,5}\left(\frac{1}{9\Delta r^2 \Delta \theta^2}\right) &= -100
\end{aligned} \tag{4,4}$$

This completes the central line, now we move $\Delta\theta$ anti-clock wise and process the next grid line.

Node $(2,5)$ is in the left side, where the PDE is $\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} = 0$. Hence

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta\theta^2} &= 0 \\
\frac{T_{1,5} - 2T_{2,5} + T_{3,5}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,5} - T_{1,5}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,4} - 2T_{2,5} + T_{2,6}}{\Delta\theta^2} &= 0 \\
T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{2,5}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta\theta^2}\right) + T_{3,5}\left(\frac{3}{2\Delta r^2}\right) + T_{2,4}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) + T_{2,6}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) &= 0 \quad (2,5)
\end{aligned}$$

At node $(3,5)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta\theta^2} &= 0 \\
\frac{T_{2,5} - 2T_{3,5} + T_{4,5}}{\Delta r^2} + \frac{1}{2\Delta r}\frac{T_{4,5} - T_{2,5}}{2\Delta r} + \frac{1}{4\Delta r^2}\frac{T_{3,4} - 2T_{3,5} + T_{3,6}}{\Delta\theta^2} &= 0 \\
T_{2,5}\left(\frac{3}{4\Delta r^2}\right) + T_{3,5}\left(\frac{-2}{\Delta r^2} - \frac{1}{2\Delta r^2\Delta\theta^2}\right) + T_{4,5}\left(\frac{5}{4\Delta r^2}\right) + T_{3,4}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right) + T_{3,6}\left(\frac{1}{4\Delta r^2\Delta\theta^2}\right) &= 0 \quad (3,5)
\end{aligned}$$

At node $(4,5)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta\theta^2} &= 0 \\
\frac{T_{3,5} - 2T_{4,5} + T_{5,5}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{T_{5,5} - T_{3,5}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,4} - 2T_{4,5} + T_{4,6}}{\Delta\theta^2} &= 0
\end{aligned}$$

But $T_{5,5} = 0$ since at boundary

$$\begin{aligned}
\frac{T_{3,5} - 2T_{4,5}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{-T_{3,5}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,4} - 2T_{4,5} + T_{4,6}}{\Delta\theta^2} &= 0 \\
T_{3,5}\left(\frac{5}{6\Delta r^2}\right) + T_{4,5}\left(\frac{-2}{\Delta r^2} - \frac{2}{9\Delta r^2\Delta\theta^2}\right) + T_{4,4}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right) + T_{4,6}\left(\frac{1}{9\Delta r^2\Delta\theta^2}\right) &= 0 \quad (4,5)
\end{aligned}$$

At node $(2,6)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta\theta^2} &= 0 \\
\frac{T_{1,6} - 2T_{2,6} + T_{3,6}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,6} - T_{1,6}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,5} - 2T_{2,6} + T_{2,7}}{\Delta\theta^2} &= 0 \\
T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{2,6}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2\Delta\theta^2}\right) + T_{3,6}\left(\frac{3}{2\Delta r^2}\right) + T_{2,5}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) + T_{2,7}\left(\frac{1}{\Delta r^2\Delta\theta^2}\right) &= 0 \quad (2,6)
\end{aligned}$$

At node $(3,6)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= 0 \\
\frac{T_{2,6} - 2T_{3,6} + T_{4,6}}{\Delta r^2} + \frac{1}{2\Delta r}\frac{T_{4,6} - T_{2,6}}{2\Delta r} + \frac{1}{4\Delta r^2}\frac{T_{3,5} - 2T_{3,6} + T_{3,7}}{\Delta \theta^2} &= 0 \\
T_{2,6}\left(\frac{3}{4\Delta r^2}\right) + T_{3,6}\left(\frac{-2}{\Delta r^2} - \frac{1}{2\Delta r^2 \Delta \theta^2}\right) + T_{4,6}\left(\frac{5}{4\Delta r^2}\right) + T_{3,5}\left(\frac{1}{4\Delta r^2 \Delta \theta^2}\right) + T_{3,7}\left(\frac{1}{4\Delta r^2 \Delta \theta^2}\right) &= 0
\end{aligned} \tag{3,6}$$

At node $(4,6)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= 0 \\
\frac{T_{3,6} - 2T_{4,6} + T_{5,6}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{T_{5,6} - T_{3,6}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,5} - 2T_{4,6} + T_{4,7}}{\Delta \theta^2} &= 0
\end{aligned}$$

But $T_{5,6} = 0$ since at boundary hence

$$\begin{aligned}
\frac{T_{3,6} - 2T_{4,6}}{\Delta r^2} + \frac{1}{3\Delta r}\frac{-T_{3,6}}{2\Delta r} + \frac{1}{9\Delta r^2}\frac{T_{4,5} - 2T_{4,6} + T_{4,7}}{\Delta \theta^2} &= 0 \\
T_{3,6}\left(\frac{5}{6\Delta r^2}\right) + T_{4,6}\left(\frac{-2}{\Delta r^2} - \frac{2}{9\Delta r^2 \Delta \theta^2}\right) + T_{4,5}\left(\frac{1}{9\Delta r^2 \Delta \theta^2}\right) + T_{4,7}\left(\frac{1}{9\Delta r^2 \Delta \theta^2}\right) &= 0
\end{aligned} \tag{4,6}$$

We now move to the bottom grid line at $\theta = \pi$, where it is insulated.

At node $(2,7)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{1,7} - 2T_{2,7} + T_{3,7}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,7} - T_{1,7}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{T_{2,6} - 2T_{2,7} + T_{2,8}}{\Delta \theta^2} &= 0
\end{aligned}$$

But $T_{2,8} = T_{2,5}$ due to insulation and $T_{1,7} = T_{1,1}$ since same node, hence above becomes

$$\begin{aligned}
\frac{T_{1,1} - 2T_{2,7} + T_{3,7}}{\Delta r^2} + \frac{1}{\Delta r}\frac{T_{3,7} - T_{1,1}}{2\Delta r} + \frac{1}{\Delta r^2}\frac{2T_{2,6} - 2T_{2,7}}{\Delta \theta^2} &= 0 \\
T_{1,1}\left(\frac{1}{2\Delta r^2}\right) + T_{2,7}\left(\frac{-2}{\Delta r^2} - \frac{2}{\Delta r^2 \Delta \theta^2}\right) + T_{3,7}\left(\frac{3}{2\Delta r^2}\right) + T_{2,6}\left(\frac{2}{\Delta r^2 \Delta \theta^2}\right) &= 0
\end{aligned} \tag{2,7}$$

At node $(3,7)$

$$\begin{aligned}
\frac{\partial^2 T}{\partial r^2} + \frac{1}{r}\frac{\partial T}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T}{\partial \theta^2} &= 0 \\
\frac{T_{i-1,j} - 2T_{ij} + T_{i+1,j}}{\Delta r^2} + \frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j} - T_{i-1,j}}{2\Delta r} + \frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1} - 2T_{ij} + T_{i,j+1}}{\Delta \theta^2} &= 0 \\
\frac{T_{2,7} - 2T_{3,7} + T_{4,7}}{\Delta r^2} + \frac{1}{2\Delta r}\frac{T_{4,7} - T_{2,7}}{2\Delta r} + \frac{1}{4\Delta r^2}\frac{T_{3,6} - 2T_{3,7} + T_{3,8}}{\Delta \theta^2} &= 0
\end{aligned}$$

But $T_{3,8} = T_{3,6}$ due to insulation, hence

$$\frac{T_{2,7}-2T_{3,7}+T_{4,7}}{\Delta r^2}+\frac{1}{2\Delta r}\frac{T_{4,7}-T_{2,7}}{2\Delta r}+\frac{1}{4\Delta r^2}\frac{2T_{3,6}-2T_{3,7}}{\Delta\theta^2}=0$$

$$T_{2,7}\left(\frac{3}{4\Delta r^2}\right)+T_{3,7}\left(\frac{-2}{\Delta r^2}-\frac{1}{2\Delta r^2\Delta\theta^2}\right)+T_{4,7}\left(\frac{5}{4\Delta r^2}\right)+T_{3,6}\left(\frac{1}{2\Delta r^2\Delta\theta^2}\right)=0 \tag{3,7}$$

At node (4,7)

$$\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}=0$$

$$\frac{T_{i-1,j}-2T_{i,j}+T_{i+1,j}}{\Delta r^2}+\frac{1}{(i-1)(\Delta r)}\frac{T_{i+1,j}-T_{i-1,j}}{2\Delta r}+\frac{1}{[(i-1)(\Delta r)]^2}\frac{T_{i,j-1}-2T_{ij}+T_{i,j+1}}{\Delta\theta^2}=0$$

$$\frac{T_{3,7}-2T_{4,7}+T_{5,7}}{\Delta r^2}+\frac{1}{3\Delta r}\frac{T_{5,7}-T_{3,7}}{2\Delta r}+\frac{1}{9\Delta r^2}\frac{T_{4,6}-2T_{4,7}+T_{4,8}}{\Delta\theta^2}=0$$

But $T_{4,8}=T_{4,6}$ due to insulation and $T_{5,7}=0$ since on boundary, hence

$$\frac{T_{3,7}-2T_{4,7}}{\Delta r^2}+\frac{1}{3\Delta r}\frac{-T_{3,7}}{2\Delta r}+\frac{1}{9\Delta r^2}\frac{2T_{4,6}-2T_{4,7}}{\Delta\theta^2}=0$$

$$T_{3,7}\left(\frac{5}{6\Delta r^2}\right)+T_{4,7}\left(\frac{-2}{\Delta r^2}-\frac{2}{9\Delta r^2\Delta\theta^2}\right)+T_{4,6}\left(\frac{2}{9\Delta r^2\Delta\theta^2}\right)=0 \tag{4,7}$$

Now we are able to see the A matrix structure. The number of unknowns is 22. Let $\alpha_1=\left(\frac{-2}{\Delta r^2}-\frac{1}{2\Delta r^2\Delta\theta^2}\right),\alpha_2=\left(\frac{-2}{\Delta r^2}-\frac{2}{\Delta r^2\Delta\theta^2}\right),\alpha_3=\left(\frac{-2}{\Delta r^2}-\frac{2}{9\Delta r^2\Delta\theta^2}\right),\beta=\frac{1}{\Delta r^2},\gamma=\frac{1}{\Delta r^2\Delta\theta^2}$ then the above equations can now be written as $Ax=f$

$$\begin{pmatrix}
1 & -\frac{1}{12} & 0 & 0 & -\frac{1}{6} & 0 & 0 & -\frac{1}{6} & 0 & 0 & -\frac{1}{6} & 0 & 0 & -\frac{1}{6} & 0 & 0 & -\frac{1}{6} & 0 & 0 & -\frac{1}{12} & 0 & 0\\
\frac{1}{2}\beta & \alpha_2 & \frac{3}{2}\beta & 0 & 2\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{2}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & \frac{5}{6}\beta & \alpha_3 & 0 & 0 & \frac{2}{9}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{1}{2}\beta & \gamma & 0 & 0 & \alpha_2 & \frac{3}{2}\beta & 0 & \gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & \frac{1}{4}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{4}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \frac{1}{9}\gamma & 0 & \frac{5}{4}\beta & \alpha_3 & 0 & 0 & \frac{1}{9}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{1}{2}\beta & 0 & 0 & 0 & \gamma & 0 & 0 & \alpha_2 & \frac{3}{2}\beta & 0 & \gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & \frac{1}{4}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{4}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9}\gamma & 0 & \frac{5}{6}\beta & \alpha_3 & 0 & 0 & \frac{1}{9}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{1}{2}\beta & 0 & 0 & 0 & 0 & 0 & 0 & \gamma & 0 & 0 & \alpha_2 & \frac{3}{2}\beta & 0 & \gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{4}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9}\gamma & 0 & \frac{5}{6}\beta & \alpha_3 & 0 & 0 & \frac{1}{9}\gamma & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{1}{2}\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma & 0 & 0 & \alpha_3 & \frac{3}{2}\beta & 0 & \gamma & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{4}\gamma & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9}\gamma & 0 & \frac{5}{6}\beta & \alpha_3 & 0 & 0 & \frac{1}{9}\gamma & 0 & 0 & 0 & 0\\
\frac{1}{2}\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \gamma & 0 & \alpha_3 & \frac{3}{2}\beta & 0 & \gamma & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0 & \frac{1}{4}\gamma & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{9}\gamma & 0 & \frac{5}{6}\beta & \alpha_3 & 0 & 0 & \frac{1}{9}\gamma & 0\\
\frac{1}{2}\beta & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2\gamma & 0 & 0 & \alpha_2 & \frac{3}{2}\beta & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2}\gamma & 0 & \frac{3}{4}\beta & \alpha_1 & \frac{5}{4}\beta & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{2}{9}\gamma & 0 & \frac{5}{6}\beta & \alpha_3 & 0
\end{pmatrix}
\begin{pmatrix} T_{11}\\ T_{21}\\ T_{31}\\ T_{41}\\ T_{22}\\ T_{32}\\ T_{42}\\ T_{23}\\ T_{33}\\ T_{43}\\ T_{24}\\ T_{34}\\ T_{44}\\ T_{25}\\ T_{35}\\ T_{45}\\ T_{26}\\ T_{36}\\ T_{46}\\ T_{27}\\ T_{37}\\ T_{47} \end{pmatrix}
=
\begin{pmatrix} 25\Delta r^2\\ -200\\ -200\\ -200\\ -200\\ -200\\ -200\\ -200\\ -200\\ -200\\ -100\\ -100\\ -100\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0\\ 0 \end{pmatrix}$$

We now can see the pattern to use. The above is solved using $Ax=f$ in Matlab.

The solution below. The temperature at origin was found to be 4.737 degrees and the maximum was 6.666 degrees.

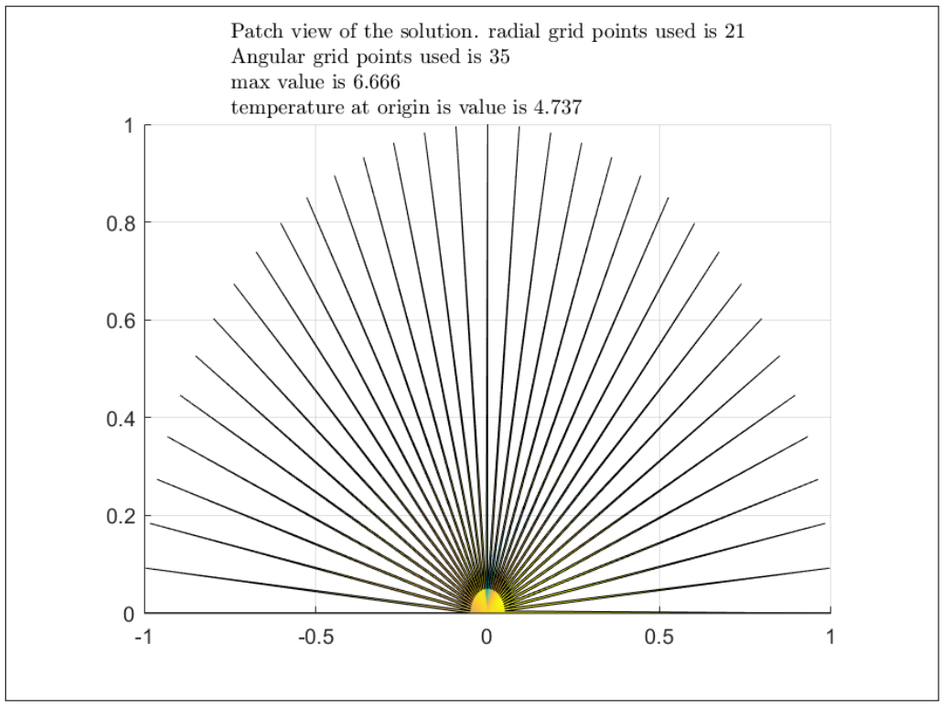


Figure 2.72: problem 3 solution using Matlab patch command

```
function nma_EMA_471_HW6_problem_3
%solution to problem 3, HW6, EMA 471
%EMA 471
%
%Please see report for derivation of the A matrix for
%this problem This uses the direct method, not iterative. Was
%very time consuming to make the A matrix. An iterative method
%would have been simpler
%
%version completed after midnight. 4/21/2016

close all; clc;
Nr        = 21;  %grid points in radial direction
Na        = 35;  %grid points in angular direction
n         = (Nr-2)*Na+1;  %number of unknowns
delR      = 1/(Nr-1);  %grid spacing radial direction
delAngle  = pi/(Na-1); %grid spacing angular direction in radians
A         = make_A_matrix(Nr,Na,n,delR,delAngle);
f         = ones(n,1);    %RHS.
f(1)      = 25*delR^2;    %this where the origin maps to
f(2:(Nr-2)*(Na-1)/2+1)  = -200;  %right side of disk
f((Nr-2)*(Na-1)/2:(Nr-2)*(Na-1)/2+(Nr-2))  = -100;  %center line

%left side of disk
f((Nr-2)*(Na-1)/2:(Nr-2)*(Na-1)/2+(Nr-2)+1:end) = 0;

u = A\f;  %direct (All this hard work, just to make this one call)

fprintf('T_0 = %6.4f\n',u(1));

%find X,Y coordinates for patch command. I could use cart2pol()
%also, but need to patch the coordinates in the process to add
%the boundary conditions, which has known solution of zero,
%and loop seemed easier

X   = zeros((Nr-1)*Na+1,1);  %X coordinates
Y   =   X;
sol = X;
```

```
for i=1:Nr
    if i==1   %this is origin
        sol(i) = u(i);
        X(i)   = 0;
        Y(i)   = 0;
    else
         for j = 0:Na-1

             %watch out. coordinate index is different
             %than solution index, since we are
             %adding boundary conditions
             coord_idx = i+j*(Nr-1);

             if i==Nr
                 sol_idx = i+j*(Nr-1);
                 sol(sol_idx)=0;  %B.C.
             else
                 sol_idx = i+j*(Nr-2);
                 sol(sol_idx)=u(sol_idx);
             end
             X(coord_idx)=(i-1)*delR*cos(j*delAngle);
             Y(coord_idx)=(i-1)*delR*sin(j*delAngle);
         end
     end
end

%make patch plot and print the solution
patch(X,Y,sol);
grid;

title({sprintf('Patch view of the solution. radial grid points used is %d',Nr), ...
    sprintf('Angular grid points used is %d',Na),...
    sprintf('max value is $%6.3f$',max(sol)),...
    sprintf('temperature at origin is value is $%6.3f$',sol(1))},...
    'Interpreter', 'latex','fontsize',10);

end
%===================================
function A = make_A_matrix(Nr,Na,n,delR,delAngle)
%please see report for details. This is complicated A matrix due to
%the geometry

a1    = (-2/delR^2-1/(2*delR^2*delAngle^2));
a2    = (-2/delR^2-2/(delR^2*delAngle^2));
a3    = (-2/delR^2-2/(9*delR^2*delAngle^2));
b     =  1/delR^2;
gamma = 1/(delR^2*delAngle^2);
z     = 1;  %this is used to count the angular jumps.
            %Needed to sync with

A = zeros(n);

%make first line. This is the bottom edge, right size of disk
A(1,1)   = 1;
A(1,2)   = -1/(2*(Na-1));
for j = Nr:(Nr-2):(Nr+(Nr-2)*(Na-3))
    A(1,j) = -1/(Na-1);
end
A(1,(Nr-2)*Na-1) = -1/(2*(Na-1));


for i=2:(Nr-2):2+(Nr-2)*(Na-1)
    if i==2 || i==2+(Nr-2)*(Na-1)   %these are the lower edges
```

```
        if i == 2 %first edge, at theta=0
            zz = 1; %to help me find where I am in the matrix
            for k=i:i+Nr-3 %process each radial line
                           %(internal nodes)
                if k==i %first in block
                    A(k,1)=(1/2)*b;
                    A(k,2)=a2;
                    A(k,3)=(3/2)*b;
                    A(k,Nr)=2*gamma;
                elseif k==i+Nr-3 %last
                    A(k,Nr-2)=(5/6)*b;
                    A(k,Nr-1)=a3;
                    A(k,Nr-1+Nr-2)=2/9*gamma;
                else
                    zz=zz+1;
                    A(k,zz)=(3/5)*b;
                    A(k,zz+1)=a1;
                    A(k,zz+2)=5/4*b;
                    A(k,zz+Nr-2)=1/2*gamma;
                end
            end
        else %last edge, at theta=pi
            zz=0;
            for k=i:i+Nr-3 %process each radial line (internal)
                z0=2+(Nr-2)*(Na-1); %where last edge node starts
                if k==i %first in block
                    A(k,z0)=a2;
                    A(k,z0-Nr-2)=2*gamma;
                    A(k,z0+1)=3/2*b;
                elseif k==i+Nr-3 %last
                    zz=zz+1;
                    A(k,z0+zz)=a3;
                    A(k,z0+zz-1)=5/6*b;
                    A(k,z0+zz-(Nr-2))=2/9*gamma;
                else
                    zz=zz+1;
                    A(k,z0+zz)=a1;
                    A(k,z0+zz-1)=(3/5)*b;
                    A(k,z0+zz+1)=5/4*b;
                    A(k,z0+zz-(Nr-2))=1/2*gamma;
                end
            end
        end
    else  %internal radial lines
        for k=i:i+Nr-3 %process each radial line (internal nodes)
            if k==i %first in block
                z = z + 1;
                A(k,1)=(1/2)*b;
                A(k,z)=gamma;
                A(k,z+(Nr-2))=a2;
                A(k,z+1+(Nr-2))=3/2*b;
                A(k,z+2*(Nr-2))=gamma;
            elseif k==i+Nr-3 %last
                z = z +1;
                A(k,z)=1/9*gamma;
                A(k,z+(Nr-3))=(5/4)*b;
                A(k,z+1+(Nr-3))=a3;
                A(k,z+2*(Nr-2))=1/8*gamma;
            else
                z=z+1; %to tag where entries start
                A(k,z)=(1/4)*gamma;
                A(k,z+(Nr-3))=(3/4)*b;
                A(k,z+1+(Nr-3))=a1;
```

```
                A(k,z+2+(Nr-3))=5/4*b;
                A(k,z+2*(Nr-2))=(1/4)*gamma;
            end
        end
    end
end
end
```

2.7 HW 7

2.7.1 Problem 1

EP 471 – Homework #7 Solns: Parabolic PDEs
Due: Thursday, May 5th, 2016

Use Matlab's pdepe utility for both problems.

(1) (20 pts) When a body experiences very high temperatures, radiative heating/cooling becomes an important mechanism for heat transfer. Consider the case of a slab of steel, six inches thick, that has been heated in a furnace to a uniform temperature of 2350 °F. The steel is removed from the furnace, and we'd like to know how fast it cools. Specifically, we'd like to know how long it takes for the centerline temperature of the steel to fall below 1000 °F.

Take the material properties of the steel as constant (k = 25 Btu/hr-ft-°F, ρ = 500 lbm/ft^3, and c = 0.12 Btu/lbm-°F). Perform two simulations. In the first, ignore radiative heat transfer and use a convective heat transfer coefficient to ambient air (T_∞ = 60 °F) of 50 Btu/hr-ft^2-°F. In the second, include radiative heat transfer. In addition to the convective cooling, radiative cooling includes a term of the form:

$$q'' = \varepsilon\sigma(T_s^4 - T_\infty^4)$$

Here ε is the emissivity (dimensionless, take equal to 0.8). The parameter σ is the Stefan-Boltzmann constant, equal to 0.171×10^{-8} Btu/hr-ft^2-°R^4. Here T_s is the surface temperature of the steel and T_∞ is again the ambient air temperature. Note that in this radiative heat transfer expression, temperatures must be expressed in absolute values (°R, not °F). Calculate and plot the evolution of the temperature profile in the steel and report the times required for the centerline temperature to fall below 1000 °F without and with the effect of radiative cooling.

Figure 2.73: problem 1 description

The PDE to solve is the parabolic PDE

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2}$$

There is no source term. We consider only convective heat loss in the first part of the problem, then for the second part, add the radiative heat loss.

Before we solve this problem, it is a good idea to write down all the units. This table summarizes this

term	name	dimensional	SI units	Imperial units	SI convert
$\alpha = \frac{k}{(\rho)(c_p)}$	diffusivity	$\frac{L^2}{T}$	$\frac{\text{meter}^2}{\text{sec}}$	$\frac{\text{ft}^2}{\text{sec}}$	0.3048
k	conductivity	$\frac{ML}{T^3\Theta}$	$\frac{\text{Watt}}{\text{meter-Kelvin}}$	$\frac{\text{Btu}}{\text{hr-ft-F}^o}$	1.73
ρ	mass density	$\frac{M}{L^3}$	$\frac{\text{kg}}{\text{meter}^3}$	$\frac{\text{lbm}}{\text{ft}^3}$	16.019
c_p	specific heat	$\frac{L^2M}{T^2\Theta}$	$\frac{\text{joule}}{\text{kg-Kelvin}}$	$\frac{\text{Btu}}{\text{lbm-F}^o}$	4188
σ	Stefan-Boltzmann		$\frac{\text{Watt}}{\text{meter}^2\ \text{Kevin}^4}\left(5.670367\times10^{-8}\right)$	$\frac{\text{Btu}}{\text{hr-ft}^2\text{-R}^4}\left(0.171\times10^{-8}\right)$	

And to convert F^o to C^o us $C^o = (F^o - 32)\frac{5}{9}$. To convert from F^o to absolute R^o (Rankine) add 459.67. Hence $R^o = F^o + 459.67$.

The Matlab program `nma_HW_7_problem_1.m` solves both parts. Summary of result

	time to cool to 1000 F (minutes)
convective only loss	20.18
convective and radiative loss	17.26 (85.5% of above time)

The following shows the plot for the first part (convective heat loss only).

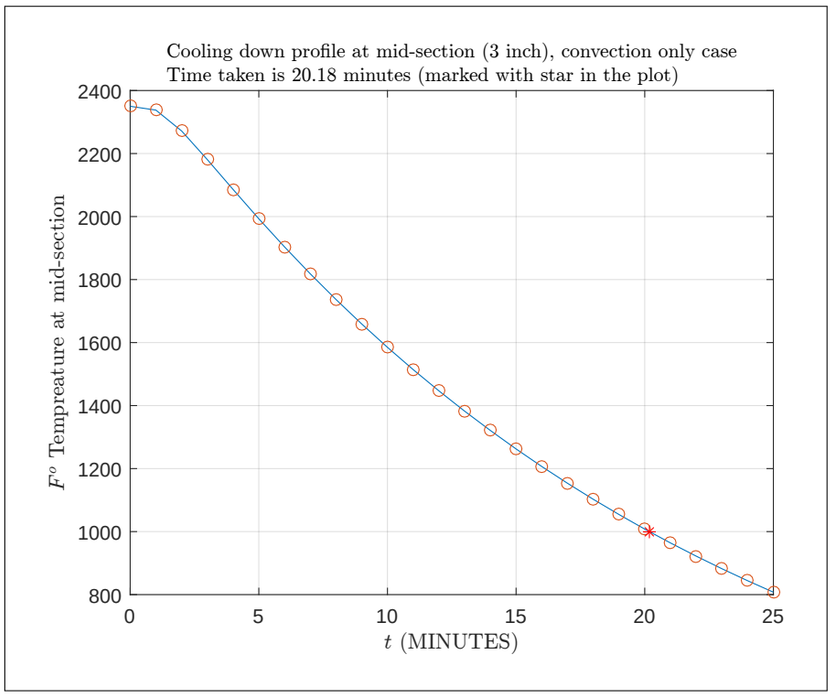


Figure 2.74: First part of first problem. Convective only heat loss

The plot below shows the profile of the heat loss across the whole section. Time is increasing going down in this figure. From initial time $t = 0$ to $t = 25$ minutes.

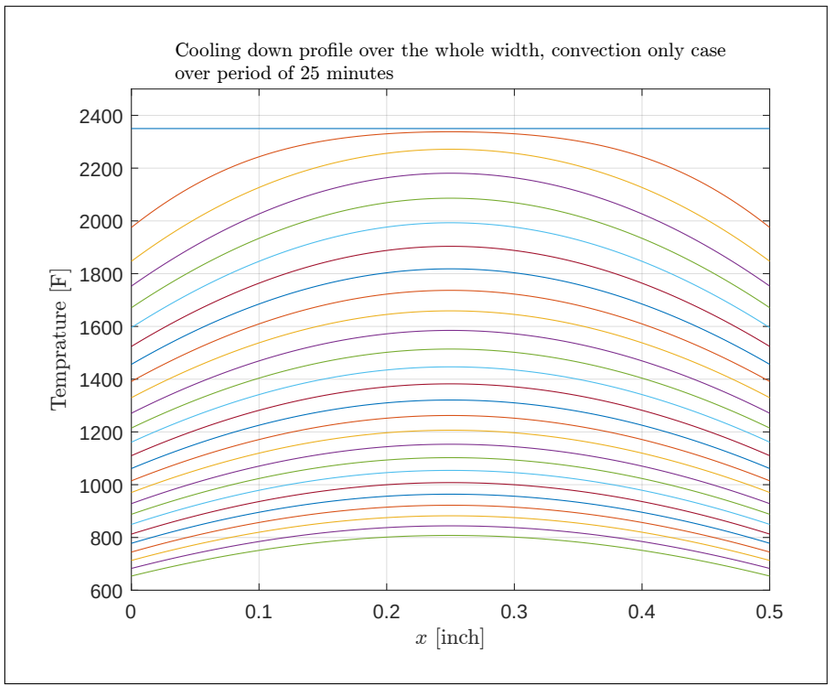


Figure 2.75: problem 1, first part. Profile across the whole section

Now radiative heat loss was added. It was found that steel will cool faster, by about 85.5 percent of the time it took without radiative heat.

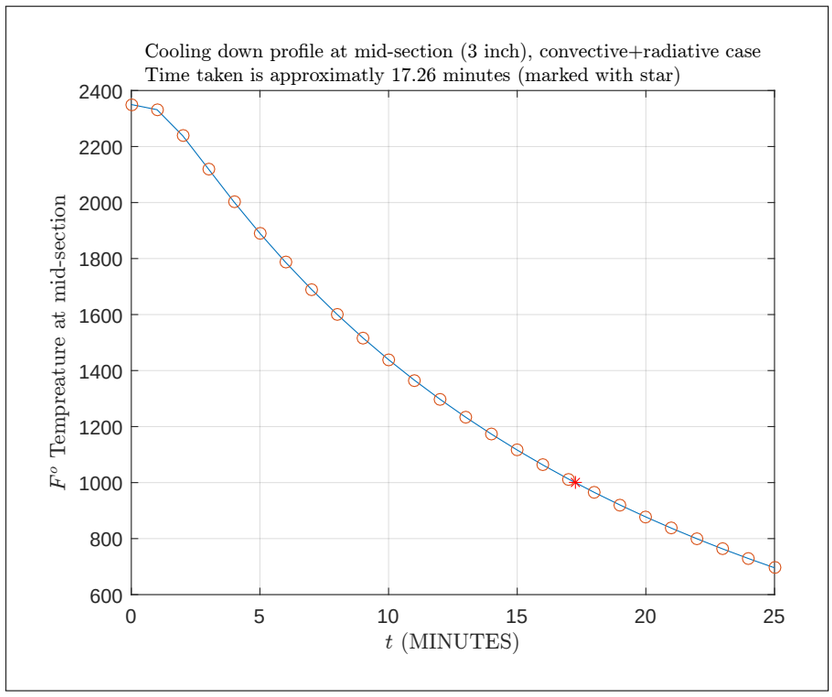


Figure 2.76: Second part of first problem. Convective and radiative heat loss

A plot is given below which compare both cases on same plot to make it more clear to see the difference.

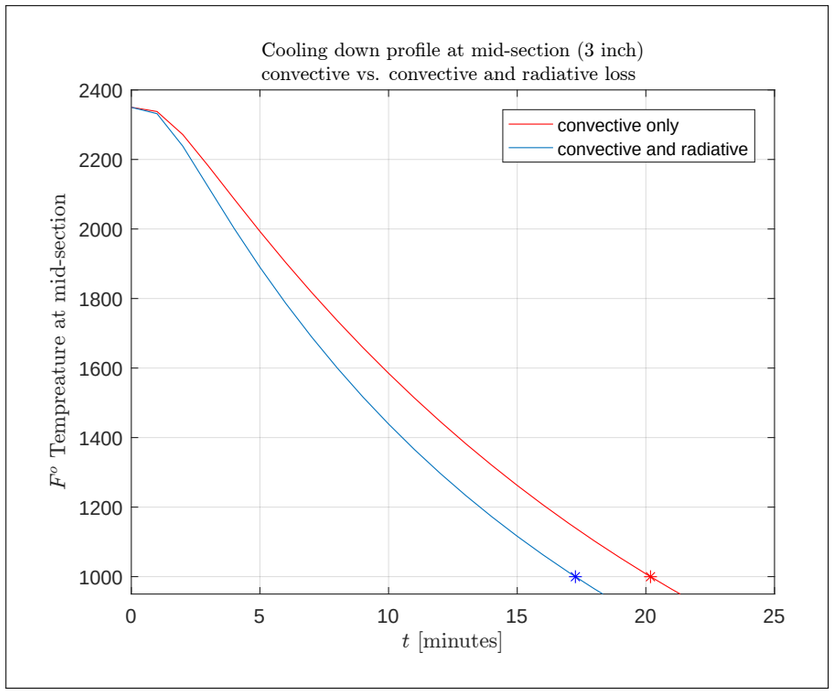


Figure 2.77: Both parts one and two on same plot, comparing heat loss at mid-section

```
function nma_HW_7_problem_1()
%solves problem 1, HW 7. EMA 471
%adopted from trans_heat_cond_pdepe.m
%
clear; clc; close all;

% parameters of interest:
% thermal diffusivity is declared as a global variable
global g_alpha;

 % thermal conductivity is declared as a global variable
global g_k;
```

```
global g_h;        % heat transfer coefficients to ambient air
global g_T_inf;    % ambient (asymptotic) temperature

g_k     = 25;    % thermal conductivity of steel, [Btu/(hr-ft-F)]
c_p     = 0.12;  % specific heat capacity of steel, [Btu/(lbm-F)]
rho     = 500;   % density of steel, [lbm/ft^3]
g_h     = 50;    % convective heat transfer coefficient to air
                 % [Btu/(hr-ft^2-F)]
g_T_inf = 60;    % ambient temperature, [F]
g_alpha = g_k/(rho*c_p); % thermal diffusivity [ft^2/hr]

[soln_pde_1,x,t] = process_part_one();
[soln_pde_2,~,~] = process_part_two();

% plots of numerical solution as points:
figure;
x_idx = round(length(x)/2);
plot(t*60,soln_pde_1(:,x_idx),'r');
hold on;
plot(t*60,soln_pde_2(:,x_idx));
title({'Cooling down profile at mid-section (3 inch)', ...
       'convective vs. convective and radiative loss'},...
             'Interpreter', 'latex','fontsize',10);
xlabel('$t$ [minutes]', 'Interpreter', 'latex');
ylabel('$F^o$ Tempreature at mid-section','Interpreter', 'latex');
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
legend('convective only','convective and radiative');
grid;
ylim([1000-50,2400]);
hold on;
plot(20.18,1000,'r*');
plot(17.26,1000,'b*');

end
%=========================================
function [soln_pde,x,t] = process_part_one()
%This part for the convective heat transfer loss only

x = linspace(0,0.5,100);  %units in feet, so this is 6 inch
t  = 0:1/60:25/60;   %units in hrs. So this is up to 25 minutes
                     %this time scale was found to be enough
                     %to see the cooling to 1000 F.
m = 0;               %for Matlab pdepe since this
                     %is catersian coordinates

soln_pde = pdepe(m,@pde_trans_heat,@pde_trans_ic,...
                @pde_trans_bc_part_1,x,t);

figure;
x_idx = round(length(x)/2);  %to find mid-point

 %change time to minutes
plot(t*60,soln_pde(:,x_idx),t*60,soln_pde(:,x_idx),'o');
grid;
title({'Cooling down profile at mid-section (3 inch), convection only case',...
 'Time taken is $20.18$ minutes (marked with star in the plot)'},...
  'Interpreter', 'latex','fontsize',10);

xlabel('$t$ (MINUTES)', 'Interpreter', 'latex');
ylabel('$F^o$ Tempreature at mid-section','Interpreter', 'latex');
hold on;
plot(20.18,1000,'r*');
%set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
```

```
76
77  figure;
78  for i=1:length(t)
79      plot(x,soln_pde(i,:));
80      hold on;
81  end
82  title({'Cooling down profile over the whole width, convection only case',...
83         'over period of 25 minutes'},...
84        'Interpreter', 'latex','fontsize',10);
85  xlabel('$x$ [inch]','Interpreter', 'latex');
86  ylabel('Temprature [F]','Interpreter', 'latex');
87  %set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
88  grid;
89  ylim([600,2500]);
90  end
91  %=======================================
92  function [soln_pde,x,t] = process_part_two()
93  %This part for the convective heat
94  %transfer loss + radiative heat loss
95
96  x = linspace(0,0.5,100); % units feet
97  t  = 0:1/60:25/60;
98  m = 0;
99
100 soln_pde = pdepe(m,@pde_trans_heat,@pde_trans_ic,...
101                 @pde_trans_bc_part_2,x,t);
102
103 figure;
104 x_idx=round(length(x)/2);
105 plot(t*60,soln_pde(:,x_idx),t*60,soln_pde(:,x_idx),'o');
106 grid;
107 title({'Cooling down profile at mid-section (3 inch), convective+radiative case',...
108   'Time taken is approximatly $17.26$ minutes (marked with star)'},...
109               'Interpreter', 'latex','fontsize',10);
110 xlabel('$t$ (MINUTES)', 'Interpreter', 'latex');
111 ylabel('$F^o$ Tempreature at mid-section','Interpreter', 'latex');
112 hold on;
113 plot(17.26,1000,'r*');
114 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
115
116 figure;
117 for i=1:length(t)
118     plot(x,soln_pde(i,:));
119     hold on;
120 end
121 title({'Cooling down profile over the whole width, convective+radiative case',...
122        'over period of 25 minutes'},...
123       'Interpreter', 'latex','fontsize',10);
124 xlabel('$x$ [inch]','Interpreter', 'latex');
125 ylabel('Temprature [F]','Interpreter', 'latex');
126 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
127 grid;
128 end
129 %=======================================
130 function [c,f,s] = pde_trans_heat(~,~,~,DTDx)
131 global g_alpha;    % thermal diffusivity. global variable
132
133 %THESE COMMENT FROM CLASS CODE
134 % because k is a constant in this problem we pull it out of
135 % the divergence operator and divide both sides by k.
136 % Then our "capacity" for this problem is
137 % rho*c_p/k = 1/alpha.  Note that if we had
138 % to leave k inside the operator, the capacity would
```

```
% just be rho*c_p
c = 1/g_alpha;

% again, because k has been pulled out of the operator,
% the flux in this problem is just dT/dx.  If it were
% left inside, the flux would be k*dT/dx
f = DTDx;

s = 0; % the source term is the power density/thermal conductivity
end
%=======================================
function [pL,qL,pr,qr] = pde_trans_bc_part_1(~,TL,~,Tr,~)
%case one, convective heat loss only
global g_h g_k g_T_inf;

pL =  g_h*(TL - g_T_inf);
qL = -g_k;

pr = g_h*(Tr - g_T_inf);
qr = g_k;
end
%=============================================
function [pL,qL,pr,qr] = pde_trans_bc_part_2(~,TL,~,Tr,~)
%case where have convective coefficient and radiative
global g_h g_k g_T_inf;
Stefan_Blotzmann = 0.171*10^(-8);
emissivity       = 0.8;
c0               = 459.67;  %convert from F to R

pL = g_h*(TL - g_T_inf)+emissivity*Stefan_Blotzmann*...
                                  ((TL+c0)^4-(g_T_inf+c0)^4);
qL = -g_k;
pr = g_h*(Tr - g_T_inf)+emissivity*Stefan_Blotzmann*...
                                   ((Tr+c0)^4-(g_T_inf+c0)^4);
qr = g_k;
end
%=======================================
function T_0 = pde_trans_ic(~)
T_0 = 2350; %initial temp, [F]
end
%--------------------------------------------------------------------------
```

2.7.2 Problem 2

(2) (20 pts) A hollow tube 20 cm in length is filled with air containing 2% of ethyl alcohol vapors. At the bottom of the tube ($x = 20$ cm) is a pool of alcohol which evaporates into the stagnant gas above. The concentration at this end should be assumed constant and equal to 10%. At the top of the tube ($x = 0$ cm), the alcohol vapors dissipate to the outside air, so the concentration is essentially zero. Considering only the effects of molecular diffusion, determine the spatial evolution of the alcohol concentration as a function of time. In particular, plot the time-dependent evolution of the concentration at $x = 4$, 8, 12 and 16 cm. How long does it take for the concentration at $x = 16$ cm to reach 7.5%? The governing equation for the concentration takes the form:

$$\frac{\partial c}{\partial t} = D\frac{\partial^2 c}{\partial x^2}$$

Here D is the diffusion coefficient and has a value of 0.119 cm^2/s.

Figure 2.78: problem 2 description

In this problem, the independent variable is the concentration of alcohol, called $c(x,t)$. The initial conditions is that $c(x,0) = 0.02$ and the boundary conditions is the left side (where $x = 0$) is $c(L,t) = 0$ and on the right side (where $x = 20$ cm), then $c(R,t) = 0.1$.

This is a parabolic pde. It is solved using pdepe. The program `nma_HW7_problem_2.m` solves this and the following plot shows the result. It took

486	seconds

For the concentration at $x = 16$ cm to reach 7.5%. The first plot below show all profiles on same plot, then each one on a separate plot

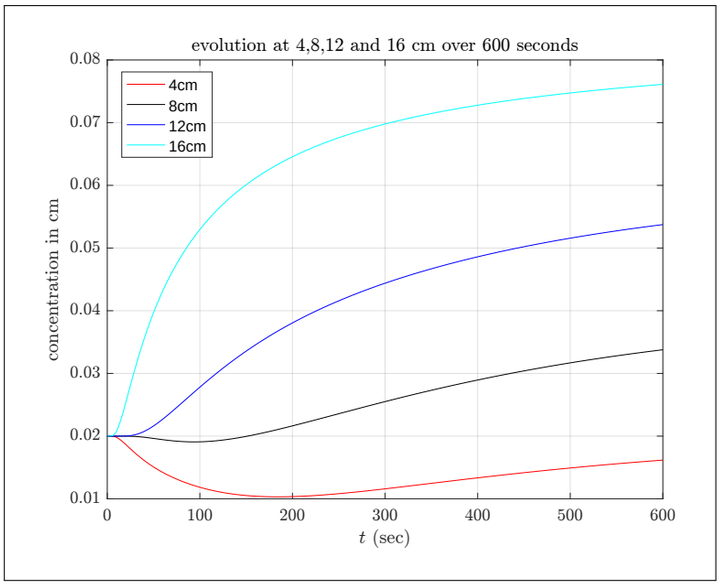


Figure 2.79: profile of concentrationn at 4, 8, 12, 16 cm combined in one plot

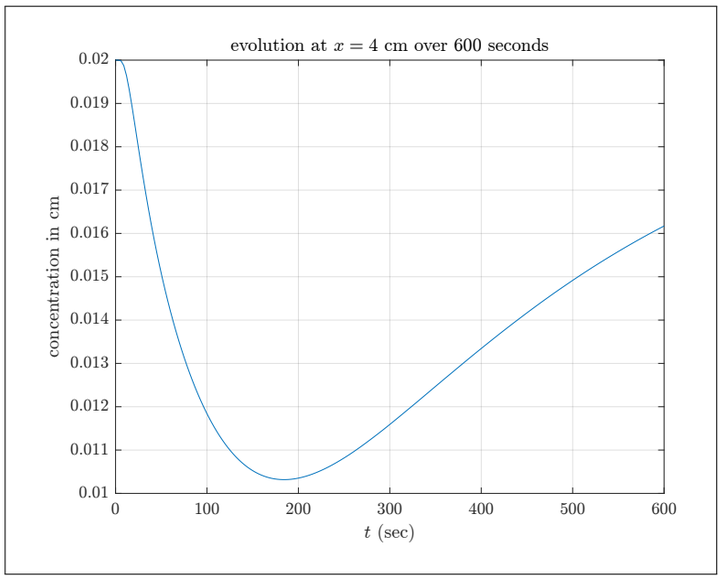


Figure 2.80: profile of concentrationn at 4 cm

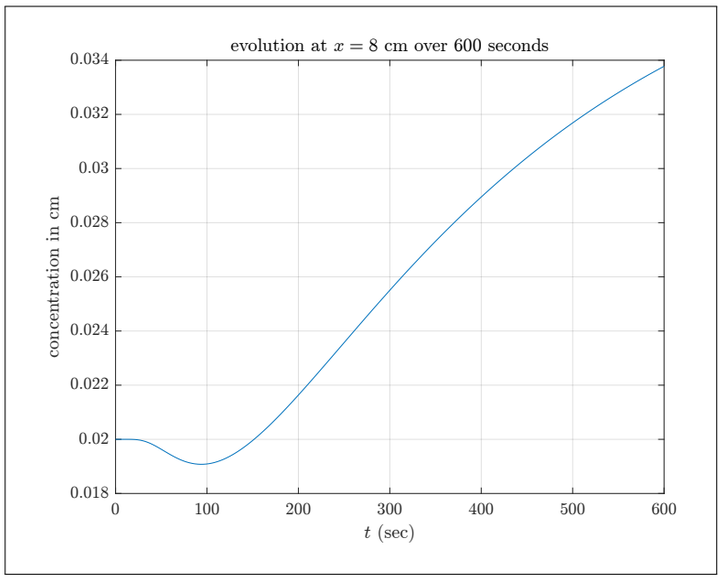


Figure 2.81: profile of concentrationn at 8 cm

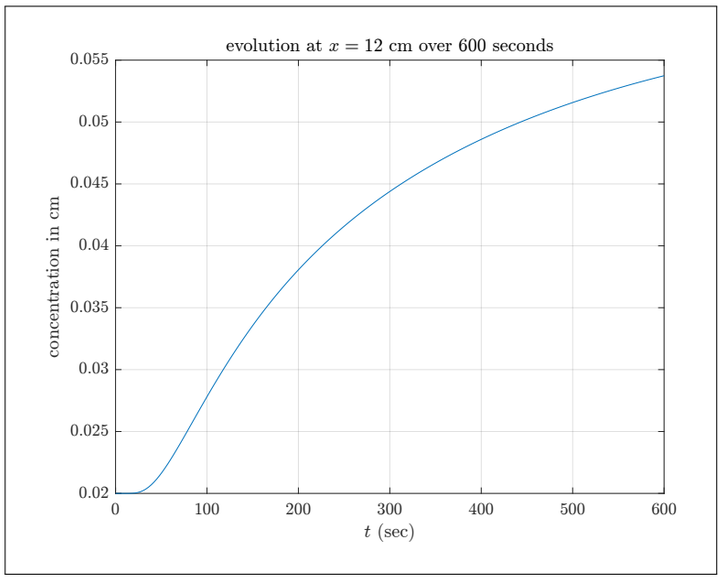


Figure 2.82: profile of concentrationn at 12 cm

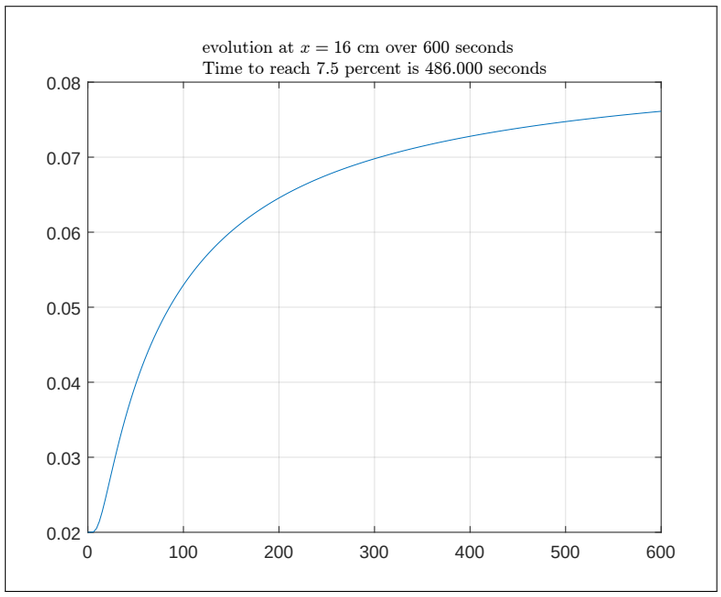


Figure 2.83: profile of concentrationn at 16 cm

```
function nma_HW_7_problem_2()
%solves problem 2, HW 7. EMA 471
%
clear; clc; close all;

% parameters of interest:
global g_D;     % diffusion coefficient declared as a global
g_D = 0.119;    % cm^2/sec

x  = 0:0.1:20;
t  = linspace(0,600,length(x));
m = 0;

soln_pde = pdepe(m,@pde_trans_heat,@pde_trans_ic, ...
                 @pde_trans_bc,x,t);

plot_at(4,t,x,soln_pde);
plot_at(8,t,x,soln_pde);
plot_at(12,t,x,soln_pde);
plot_at(16,t,x,soln_pde);

%plot all on same figure now

figure;
idx = x==4;
plot(t,soln_pde(:,idx),'r');
hold on;
idx = x==8;
plot(t,soln_pde(:,idx),'k');
idx = x==12;
plot(t,soln_pde(:,idx),'b');
idx = x==16;
plot(t,soln_pde(:,idx),'c');
grid;
legend('4cm','8cm','12cm','16cm');
title(sprintf('evolution at 4,8,12 and 16 cm over $%d$ seconds',...
               t(end)),...
               'Interpreter', 'latex','fontsize',10);
  xlabel('$t$ (sec)', 'Interpreter', 'latex');
  ylabel('concentration in cm','Interpreter', 'latex');
  set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
end
```

```
%==================================
function plot_at(x0,t,x,soln_pde)

idx = x==x0;
figure;
plot(t,soln_pde(:,idx));
if x0==16
   I=find(round(soln_pde(:,idx),3)==0.075);
   title({sprintf('evolution at $x=%d$ cm over $%d$ seconds',...
     x0,t(end)),...
     sprintf('Time to reach $7.5$ percent is $%5.3f$ seconds',...
             t(I(1))),...
     },'Interpreter', 'latex','fontsize',10);
else
  title(sprintf('evolution at $x=%d$ cm over $%d$ seconds',...
                x0,t(end)),...
               'Interpreter', 'latex','fontsize',10);
  xlabel('$t$ (sec)', 'Interpreter', 'latex');
  ylabel('concentration in cm','Interpreter', 'latex');
  set(gca,'TickLabelInterpreter', 'Latex','fontsize',10);
end
grid;


end
%======================================
function [c,f,s] = pde_trans_heat(~,~,~,DTDx)
global g_D; % diffusion coefficient declared as a global variable

c = 1/g_D;
f = DTDx;
s = 0;
end
%======================================
function [pL,qL,pr,qr] = pde_trans_bc(~,TL,~,Tr,~)
pL =  TL;
qL = 0;

pr = Tr - 0.1;
qr = 0;
end
%======================================
function T_0 = pde_trans_ic(~)
T_0 = 0.02; %initial concentration
end
%------------------------------------------------------------------------
```

Chapter 3

Project overview

Local contents

3.1 Guidelines

04/04/16

EP 471 -- Engineering Problem Solving II
Project Ideas/Descriptions
Project Due Date: Thursday, May 12th, 2016

The purpose of the class project is to give you an opportunity to solve a fairly difficult problem of your choice using some of the methods we've encountered this semester. You can do something in support of another class or project, explore something of interest, or choose one of the default ideas. (See the associated default project descriptions: ep_default_project_ema_fea, ep_default_project_nucl_power or ep_default_project_rad_sci.) Whatever you choose, the problem should have a sufficiently high degree-of-difficulty so that you can see the advantage of using Matlab to do something that would clearly be prohibitive using pencil and paper.

Those of you in the nuclear area may want to do something in support of a transport or radiation interactions course. Those in the mechanics area can select a vibrations or dynamics problem from one of the 500-level classes. These courses typically contain problems that are quite difficult with pencil and paper but are easily amenable to solution using some of the computational methods we've discussed.

Whatever you choose, you must be able to describe the idea in a level of detail comparable to what's in one of the default documents and ***have my approval*** to proceed. (No approval is necessary if you choose one of the default projects.) This includes the problem definition, what method(s) you intend to use, and how you intend to evaluate the integrity of your results. If you are tempted to choose something relatively simple (a variation of a problem that can be solved with the ode45 solver examined at the beginning of the semester, for example), be advised that I would consider that bailing out and would knock down the technical content of your project for low degree-of-difficulty.

The project is worth 100 points, with the grading broken down as follows:

- 25% is allocated to the quality of your written report. This includes the problem statement and a discussion of your results.
- 75% is allocated to the technical content of your project, including some kind of independent check of the results. The latter could include a check against another simulation, or checking the results of your scripts under some limiting condition for which there is an analytical or semi-empirical solution.

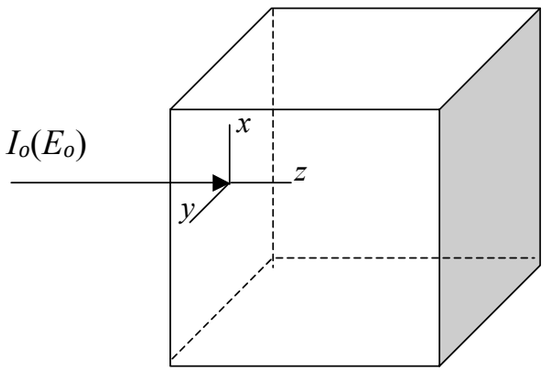
3.2 default project

4/4/16

EP 471 -- Engineering Problem Solving II
Default Project: NE Majors/Radiation Science Track

Monte Carlo simulation of shielding problem

In NEEP 408, you typically do some "back-of-the-envelope" shielding problems where (for instance) a mono-energetic, mono-directional beam is incident on one side of a slab, and you wish to find the build-up flux on the other side so you can estimate a dose or dose-equivalent.



In the sketch above, the *x* and *y* directions are intended to extend to infinity, so that the only finite dimension is in the direction of the shield thickness, *z*. Suppose we have an incident beam of 1 MeV photons on one side of an iron slab, and the shield is ten mean-free-paths thick (on the basis of the initial 1 MeV energy). The purpose of this problem is to use Monte Carlo analysis to determine a spectrum on the far side of the shield and therefore provide an alternative to the simplistic build-up flux and build-up factors used in NEEP 408.

You already have many of the elements you'll need for this problem from the Monte Carlo exercises we've encountered in class. You'll need a single loop that runs for the number of particles you're going to sample (let's say 10^7). The form of the loop should consist of a decision-making process that works like this:

(1) Determine the penetration depth. If this is negative (particle has departed backwards through the incident surface) or is positive and exceeds the shield thickness (particle has successfully penetrated the shield), terminate this particle's history and record results. (See also the section of text on the Moodle web page regarding evaluation of flux from Monte Carlo tallies.)
(2) Based on the current unit vector describing the particle's orientation, determine where the next interaction occurs. (This is based on the exponential attenuation PDF we examined in class.) Decide whether or not to continue based on the additional penetration, as outlined in step (1).

4/4/16

(3) Determine whether the interaction is a scattering event (particle continues to survive) or an absorption event (history is terminated). The fraction of RNG-space occupied by the absorption events increases as the photon energy decreases. You will need a semi-empirical model of the photoelectric effect's energy dependence in order to estimate this. There is a closed-form expression for Compton scattering, so its energy dependence is known. This part of the process is quite simple. Having generated a random number ξ, the particle is declared absorbed if $\xi < \mu_a/(\mu_s + \mu_a)$, otherwise it survives.

(4) If the event is declared to be a scattering event, determine two scattering angles, an azimuthal angle (equally likely in all directions) and a polar angle (rejection technique, as in class). Define a new particle orientation based on these new angles. The new orientation is defined from the old orientation through the rotational transformation,

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_{\text{new}} = \begin{bmatrix} \cos\theta\cos\varphi & -\sin\theta & \cos\theta\sin\varphi \\ \sin\theta\cos\varphi & \cos\theta & \sin\theta\sin\varphi \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}_{\text{old}}$$

Here θ is the azimuthal angle defined off the local x-axis and φ is the polar angle defined off the local z-axis. From the figure on the previous page, the initial unit vector describing particle orientation is [0 0 1].

(5) Determine the new photon energy from the polar scattering angle in step (4) and return to step (1), adjusting cross-sections based on new photon energy.

Every case will terminate in either an absorption event in the shield or a penetration through the shield (forward or backward). Once you've completed the loop, you should have records of the number and energy of the penetrating particles. You can use the `hist` utility to plot the spectrum of energies on the far side of the shield. Integrate the product of particle energy and absorption coefficient to determine the dose. Compare your result to a "back-of-the-envelope" calculation from NEEP 408.

Tallies

Since the histories are random, x also is a random variable. It does not in general coincide with the random variables that are sampled in producing the histories: the scattering angles, positions, and distances between collisions. Rather, x most often is related to scalar flux, current distribution, escape probability, or one of the other dependent variables that are sought from the solution of the transport equation. Our task then is to ask which of the properties that we have available from the simulation of random particle histories should be tallied in order to calculate the scalar flux, current, or other parameters of interest.

For scalar flux the two most widely used tallies result from the relationship between collision density and scalar flux and from the definition of scalar flux in terms of total neutron track length, both discussed in Chapter 1. Suppose that we want to calculate the average scalar flux $\bar{\phi}$ in some volume $\tilde{V}$ where the total cross section is $\tilde{\sigma}$. Then since $\tilde{\sigma}\bar{\phi}$ is the collision density; $\bar{c}$, the mean number of collisions in V per unit time, is

$$\bar{c} = \tilde{V}\tilde{\sigma}\bar{\phi}. \tag{7-48}$$

Hence for the Monte Carlo simulation we may write

$$\bar{\phi} = \frac{1}{\tilde{V}\tilde{\sigma}}\bar{c}, \tag{7-49}$$

where $\bar{c}$ is the mean number of collisions, normalized to one source particle. The random variable whose mean we want to calculate is thus $\bar{c}$, the mean number of collisions per neutron history in $\tilde{V}$. If we normalize our calculation to a source strength of one neutron, we then have a sample estimate of $\hat{c}$:

$$\hat{c} = \frac{1}{N}\sum_n c_n, \tag{7-50}$$

where c_n is the number of collisions made in $\tilde{V}$ during the nth history. Our sample estimate of the scalar flux is then

$$\hat{\phi} = \frac{1}{\tilde{V}\tilde{\sigma}}\frac{1}{N}\sum_n c_n. \tag{7-51}$$

A shortcoming of this estimate of the scalar flux lies in the fact that only particles that collide in $\tilde{V}$ will contribute to the collision estimator $\hat{\phi}$. We

next discuss the path length estimator for which every particle that passes through $\tilde{V}$ contributes, whether or not a collision occurs. Recall from Chapter 1 that the scalar flux may be defined as the total track length traversed by all particles per unit volume per unit time. Hence

$$\bar{\phi} = \frac{1}{\tilde{V}} \bar{l}, \tag{7-52}$$

where $\bar{l}$ is the mean track length normalized to one source particle. If we have a Monte Carlo simulation of N particles, we therefore estimate the mean value $\bar{l}$ of the random variable l by

$$\hat{l} = \frac{1}{N} \sum_n l_n, \tag{7-53}$$

where l_n is the track length in $\tilde{V}$ of the nth particle. Note that l_n may consist of more than one contribution since a single particle may pass through the volume $\tilde{V}$ more than once. From Eqs. 7-52 and 7-53 we then have as our path length flux estimator

$$\hat{\phi} = \frac{1}{\tilde{V}} \frac{1}{N} \sum_n l_n. \tag{7-54}$$

We would also like to be able to estimate particle currents, for if currents can be determined, escape probabilities and other particle balance properties follow immediately. Suppose we want to calculate the mean value of the current crossing surface $\tilde{A}$ in the $\hat{n}$ direction,

$$\tilde{A}\bar{J} = \tilde{A}\left(\bar{J}_+ - \bar{J}_-\right), \tag{7-55}$$

where $\bar{J}_+$ and $\bar{J}_-$ are the mean values of the partial currents in the positive and negative directions. We may write

$$\tilde{A}\bar{J}_+ = \bar{p}^+ \quad \text{and} \quad \tilde{A} \cdot \bar{J} = \bar{p}^-, \tag{7-56}$$

where $\bar{p}^{\pm}$ are the mean numbers of particles passing through the surface per second in the positive and negative directions respectively. These quantities can be estimated from our Monte Carlo sample as

$$\hat{p}^{\pm} = \frac{1}{N} \sum_n p_n^{\pm} \tag{7-57}$$

where p_n^+ and p_n^- are the number of passages through the surface $\tilde{A}$ made

by the nth particle history in the positive and negative direction respectively. We have

$$\hat{J}_+ = \frac{1}{\bar{A}} \frac{1}{N} \sum_n p_n^+, \qquad \hat{J}_- = \frac{1}{\bar{A}} \frac{1}{N} \sum_n p_n^-, \tag{7-58}$$

and

$$\hat{J} = \hat{J}_+ - \hat{J}_-. \tag{7-59}$$

is an approximation to the net current.

It is also possible to calculate the average scalar flux over a surface by using the relationship among angular flux, scalar flux, and current, which is given in Chapter 1. It may be shown that the value of $\bar{\phi}$ on $\bar{A}$ may be estimated from

$$\hat{\phi} = \frac{1}{\bar{A}} \frac{1}{N} \sum_n \zeta_n \tag{7-60}$$

where ζ_n is the number of crossings of the nth neutron, each weighted by $|1/\mu|$, where $\mu = \hat{\Omega} \cdot \hat{n}$ is the direction cosine of the particle with respect to the positive normal to the surface. Thus if there were I crossings in the nth history,

$$\zeta_n = \sum_{i=1}^{I} \left| \frac{1}{\mu_i} \right|. \tag{7-61}$$

A difficulty arises if one must calculate the flux over a very small volume or surface, as is the case, for example, when it is desired to determine the response of a "point" detector in a system. The foregoing tallies become useless, since if the volume is very small no histories are likely to collide in it, or even to pass through it. In such circumstances there are two alternatives. One may use an adjoint Monte Carlo calculation in which particles are emitted from the detector volume,[2,13] or one may resort to one of the more subtle tallying techniques for the estimate of the flux at a point.[7,14–16] We defer discussion of the use of the adjoint equation until Section 7-6; estimates of the flux at a point and some of the tallies are treated in Section 7-7.

Before proceeding, it is important to note that it is common practice to normalize Monte Carlo results to a source of one particle, while in steady-state deterministic transport calculations the source is normally given in terms of particles per second. Thus the foregoing tallies have units of cm^{-2}

rather than particles/cm^2/sec as in deterministic calculations. For steady-state calculations, however, the correspondence is clear. The results of the Monte Carlo calculation are just multiplied by the number of particles produced per second; the magnitudes of the tallies are then correct, and the units become particles/cm^2/sec.

7-4 ERROR ESTIMATES

In the preceding section we indicated how to use a number of random variables to estimate the scalar flux and current in analog Monte Carlo calculations. The question now arises as to how much error the sample estimate $\hat{\phi}$ or $\hat{J}$ is likely to have in relation to the true values of the mean $\bar{\phi}$ or $\bar{J}$. To make an estimation of the statistical uncertainty of our results, we must go back to the properties of a random variable, introduce the concepts of expectation values and variance, and utilize the central limit theorem to arrive at an error estimate.

In the following discussion we designate the random variables c, l, p^+, p^-, ζ used in the preceding section to estimate flux or current as x. For a particular simulation in principle there exists a probability density function $f(x)$ for each of these estimators. Of course we can never determine this function exactly unless the problem is so simple that it can be solved analytically; otherwise an infinite number of particle histories would be required. Estimating the properties of $f(x)$, however, leads in turn to an estimate of the error in $\hat{x}$.

The functional dependence of $f(x)$ on x may have different forms depending on which of the estimators is under consideration. For example, the collision and surface crossing estimators c_n and p_n^+ and p_n^- can take on only integer values. Hence the probability density function has the form

$$f(x) = \sum_i p_i \delta(x - i), \tag{7-62}$$

where

$$\sum_i p_i = 1, \tag{7-63}$$

while the coefficients p_i determine the distribution. A special case of Eq. 7-62 is the binomial estimator

$$f(x) = p_0 \delta(x) + [1 - p_0]\delta(x - 1), \tag{7-64}$$

where the estimator can take on only values of zero and one. This would be

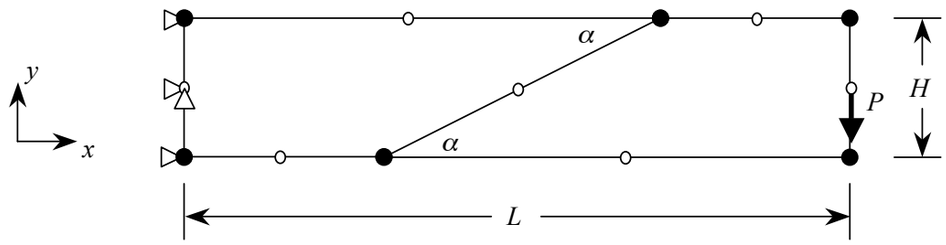
3.3 EMA project

4/4/16

EP 471 -- Engineering Problem Solving II
Default Project: EMA Majors

Assembly of Planar FE Problem

In EMA 405, we typically illustrate the element assembly process with 1D elements (bars or beams). The purpose of this project is to give you an idea of what's involved in assembling a 2D FE continuum problem (plane stress) from first principles. One of the problems we typically investigate is a two-Q8-element mesh distortion problem, where the geometry looks like this:



For the specific illustration in EMA 405, the numbers are these: $E = 1 \times 10^4$, $\nu = 0.3$, $L = 10$, $H = 2$, and $P = 20$. The angle α is a measure of the mesh distortion (departures of angles between adjacent edges from 90°) and is a parameter to be varied. The expected vertical displacement at the node where force P is applied, based on beam theory, is 1.031. You are to assemble a global stiffness matrix from the two element stiffness matrices and solve for the deflections at the nodes. You can compare your results to a comparable ANSYS FE model.

The global stiffness matrix is assembled from element stiffness matrices and embodies internal strain energy. Briefly, it starts out from integrating volumetric strain energy over the volume of the element. If we had a simple, linear spring, initially undeformed, and stretched it a distance x, the energy stored in that spring would be

$$\text{Energy} = \frac{1}{2} x \cdot kx = \frac{1}{2} kx^2$$

In a continuum problem in 2D, we have three modes of deformation: normal strain in x, normal strain in y and shear strain (xy). Instead of displacement × force [energy], we use strain × stress [N/m² or N·m/m³], the latter being energy density. The energy in the material is:

$$\text{Energy} = \int (\text{energy density})\, dV = \int \frac{1}{2} (\varepsilon_x \sigma_x + \varepsilon_y \sigma_y + \gamma_{xy} \tau_{xy})\, dV$$

If we define column vectors,

4/4/16

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} \qquad \boldsymbol{\sigma} = \begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix}$$

then the energy stored in the deformation could be written as energy = $\int \frac{1}{2} \boldsymbol{\varepsilon}^{\mathrm{T}} \boldsymbol{\sigma} \, dV$. If we have a linear, elastic problem in plane stress, then the stress is related to the strain through $\boldsymbol{\sigma} = \mathbf{E}\boldsymbol{\varepsilon}$, where

$$\mathbf{E} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{bmatrix}$$

The final issue involves relating the strain vector to the nodal displacements (described below). After doing this and performing a mathematical operation analogous to taking a derivative, we're left with an *element stiffness matrix* that takes the form:

$$k = \int \mathbf{B}^{\mathrm{T}} \mathbf{E} \mathbf{B} \, dV = \int_{-1}^{+1} \int_{-1}^{+1} \mathbf{B}^{\mathrm{T}} \mathbf{E} \mathbf{B} \det J \, d\xi \, d\eta$$

and where **B** is called the strain-displacement matrix. Here we have also assumed unit thickness in the out-of-plane direction. (The ½ disappears when we "require the functional to be stationary." This is like taking the derivative of a function and setting it equal to zero to find the value where the function is minimized. In this case we are finding a best function that minimizes potential energy instead of finding a best value that minimizes a function.)

One more issue merits discussion before proceeding, and this is the role of the Jacobian determinant in the calculation of an element stiffness matrix. Our Jacobian matrix is:

$$\mathbf{J} \equiv \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}$$

It is clear that det **J** appears in the integrand, but **J** is also important in establishing our strain-displacement matrices. Our strains involve displacement gradients:

$$\varepsilon_x = \frac{\partial u}{\partial x} \equiv u_{,x} \qquad \varepsilon_y = \frac{\partial v}{\partial y} \equiv v_{,y} \qquad \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \equiv u_{,y} + v_{,x}$$

4/4/16

However, the integral and problem formulation take place in the natural coordinate system of the element (ξ,η) and not the global coordinate system (x,y), so we have to turn the gradients in global variables into gradients in local variables. Given some quantity ϕ, we can use the chain rule to write a gradient in one variable in terms of another:

$$\frac{\partial \phi}{\partial \xi} = \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial \xi}$$
$$\frac{\partial \phi}{\partial \eta} = \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial \phi}{\partial y}\frac{\partial y}{\partial \eta}$$

or

$$\begin{bmatrix} \phi_{,\xi} \\ \phi_{,\eta} \end{bmatrix} = \begin{bmatrix} x_{,\xi} & y_{,\xi} \\ x_{,\eta} & y_{,\eta} \end{bmatrix} \begin{bmatrix} \phi_{,x} \\ \phi_{,y} \end{bmatrix} = \mathbf{J} \begin{bmatrix} \phi_{,x} \\ \phi_{,y} \end{bmatrix}$$

Gradients in the global variables can be related to gradients in the local variables through:

$$\begin{bmatrix} \phi_{,x} \\ \phi_{,y} \end{bmatrix} = \mathbf{\Gamma} \begin{bmatrix} \phi_{,\xi} \\ \phi_{,\eta} \end{bmatrix} \qquad \text{where} \qquad \mathbf{\Gamma} \equiv \mathbf{J}^{-1} = \frac{1}{\det J} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix}$$

Now we have what we need to formulate our strain-displacement matrix **B**. From the top of the previous page, we see that the strain is a 3 × 1 column vector. We have to relate this to the local degrees-of-freedom (u and v displacements at each corner and midside node), which is an 16 × 1 column vector, $[\, u_1 \;\; v_1 \;\; u_2 \;\; v_2 \;\; \ldots \;\; u_8 \;\; v_8 \,]^T$. So the relationship:

$$\boldsymbol{\varepsilon} = \mathbf{B}\mathbf{u}$$

requires **B** to be a 3 × 16 matrix. It is formulated as the product of three matrices of size 3 × 4, 4 × 4 and 4 × 16 respectively, each of which is shown below. First, from the definition of the strain components, we have:

$$\boldsymbol{\varepsilon} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}}_{\mathbf{B}_1} \begin{bmatrix} u_{,x} \\ u_{,y} \\ v_{,x} \\ v_{,y} \end{bmatrix}$$

Then, from the conversion of global gradients to local gradients, we have:

4/4/16

$$\begin{bmatrix} u_{,x} \\ u_{,y} \\ v_{,x} \\ v_{,y} \end{bmatrix} = \underbrace{\begin{bmatrix} \Gamma_{11} & \Gamma_{12} & 0 & 0 \\ \Gamma_{21} & \Gamma_{22} & 0 & 0 \\ 0 & 0 & \Gamma_{11} & \Gamma_{12} \\ 0 & 0 & \Gamma_{21} & \Gamma_{22} \end{bmatrix}}_{\mathbf{B}_2} \begin{bmatrix} u_{,\xi} \\ u_{,\eta} \\ v_{,\xi} \\ v_{,\eta} \end{bmatrix}$$

Finally, from differentiating the interpolation functions, we have

$$\begin{bmatrix} u_{,\xi} \\ u_{,\eta} \\ v_{,\xi} \\ v_{,\eta} \end{bmatrix} = \underbrace{\begin{bmatrix} f_{1,\xi} & 0 & f_{2,\xi} & 0 & \cdots & \cdots & f_{8,\xi} & 0 \\ f_{1,\eta} & 0 & f_{2,\eta} & 0 & \cdots & \cdots & f_{8,\eta} & 0 \\ 0 & f_{1,\xi} & 0 & f_{2,\xi} & \cdots & \cdots & 0 & f_{8,\xi} \\ 0 & f_{1,\eta} & 0 & f_{2,\eta} & \cdots & \cdots & 0 & f_{8,\eta} \end{bmatrix}}_{\mathbf{B}_3} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ \vdots \\ u_8 \\ v_8 \end{bmatrix}$$

so $\mathbf{B} = \mathbf{B}_1\mathbf{B}_2\mathbf{B}_3$. When we perform the integration, as shown in

$$k = \int \mathbf{B}^{\mathrm{T}}\mathbf{E}\mathbf{B}\, dV = \int_{-1}^{+1}\int_{-1}^{+1} \mathbf{B}^{\mathrm{T}}\mathbf{E}\mathbf{B} \det J \, d\xi \, d\eta$$

we do this term-by-term. Since $\mathbf{B}$ is 3 × 16, $\mathbf{B}^{\mathrm{T}}\mathbf{E}\mathbf{B}$ is 16 × 16, and there are 256 integrations to perform. In reality, we don't have to do this many, because you can see by inspection that $\mathbf{B}^{\mathrm{T}}\mathbf{E}\mathbf{B}$ is symmetric, so we can perform calculations on the 136 upper triangular entries and then fill the off-diagonal entries in the lower triangular part from $k_{ji} = k_{ij}$.

One of the things you'll have to pay attention to is the node-numbering scheme and the appropriate assembly of your individual element stiffness matrices into the global stiffness matrix. There are only 26 degrees-of-freedom in the problem, but you need to keep in mind that your element stiffness matrices won't remain contiguous when inserted into the global stiffness matrix.

The code segment below provides a way to map the element degrees-of-freedom into the global stiffness matrix. This presumes that we have defined the element topology for each element into an array called `elem_map_nodes`. If the nodes are numbered from top-to-bottom, left-to-right, the topology for the two elements (and the contents of the array `elem_map_nodes`) would be:

4/4/16

```
 9  11   3   1   10  7   2   6
11  13   5   3   12  8   4   7
```

Since there are two degrees-of-freedom per node (displacements u and v), we define an array `elem_map_dofs` from `elem_map_nodes` using:

```
elem_map_dofs = zeros(N_elem,16);
for i = 1:N_elem
    for j = 1:8
        elem_map_dofs(i,2*j-1) = 2*elem_map_nodes(i,j) - 1;
        elem_map_dofs(i,2*j)   = 2*elem_map_nodes(i,j);
    end
end
```

We then assemble the global stiffness matrix from the element stiffness matrices by mapping the corresponding element degrees-of-freedom into the global degrees-of-freedom:

```
  % Map element stiffness matrix into global stiffness matrix:
    for m = 1:16
        for n = 1:16
            global_m = elem_map_dofs(i,m);
            global_n = elem_map_dofs(i,n);
            K(global_m,global_n) = K(global_m,global_n) + k_el(m,n);
        end
        R(global_m,1) = R(global_m,1) + r_el(m,1);
    end
```

After assembling the global stiffness matrix, you need to apply boundary conditions. This involves crossing out the row and columns of the constrained degrees-of-freedom *before* submitting the system of equations to a linear solver. After taking out the rows and columns of the constrained degrees-of-freedom, you'll have a 22 × 22 system:

$$\mathbf{KD = R}$$

Once you have displacements, you can post-process the results to get stress components. You can generate an element-by-element contour plot so you can see stress discontinuities across element boundaries. This requires finding a 3 × 3 array of stress components at the corner and midside locations as well as at the center of the element. The process is a bit more complicated than it first appears, because it turns out that stresses are not as accurate at the extremes of the element boundaries as they are at the gauss points. For elements of this type, an element contour plot consists of two steps:

(1) calculating stresses at internal gauss points, and then..

(2) extrapolating those results to element boundaries.

Implementing such a process will give you stress contour plots that you can compare directly to ANSYS output.

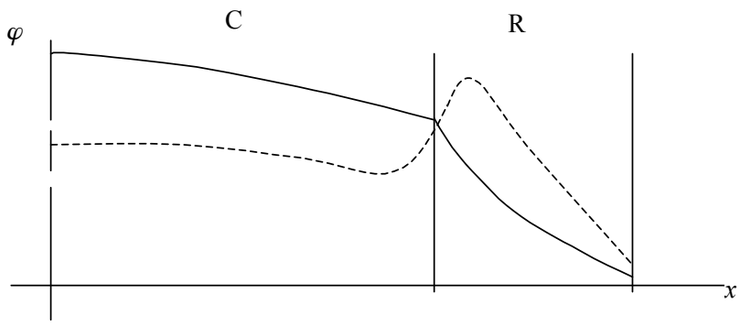
3.4 NP project

4/4/16

EP 471 -- Engineering Problem Solving II
Default Project: NE Majors/Power Track

Calculation of Critical, Two Group Flux and Adjoint Flux Profiles

In a previous exercise (Exercise 11), we found the critical flux shape in a one-speed reflected reactor. It is more likely that we want a two-group model (thermal reactors) or sometimes four or more groups (fast reactors) to describe how the flux varies with energy as well as position. The profile below continues from Exercise 11, where the solid line represents the fast flux distribution and the dashed line represents the thermal flux distribution.



In addition, we often want the *adjoint* flux, also called the neutron importance. Distributions of flux and adjoint flux *together* help us determine the reactivity effects of different changes to the reactor (the effect of inserting a control rod, also called rod worth, depends on both distributions). The purpose of this project is for you to extend the criticality calculations of Exercise 11 to find the flux and adjoint flux distributions in a one-dimensional reflected reactor using a two-group treatment.

The equations provided below are relevant to slab (Cartesian) geometry. The default problem is suggested for a cylindrical geometry based on a HTGR with inner graphite core, external annular graphite reflector and annular core (pebble bed geometry, for instance). The modification of these equations for cylindrical geometry requires a different treatment of the Laplacian operator (leakage), but that is a relatively straightforward modification.

Governing Equations:

Flux/Core Region:

$$-D_{C1}\frac{d^2\varphi_{C1}}{dx^2}+\Sigma_{RC1}\varphi_{C1}=\frac{1}{k}\chi_1 S'''$$

$$-D_{C2}\frac{d^2\varphi_{C2}}{dx^2}+\Sigma_{RC2}\varphi_{C2}=\frac{1}{k}\chi_2 S'''+\Sigma_{s12C}\varphi_{C1}$$

4/4/16

Flux/Reflector Region:

$$-D_{R1}\frac{d^2\varphi_{R1}}{dx^2}+\Sigma_{RR1}\varphi_{R1}=0$$

$$-D_{R2}\frac{d^2\varphi_{R2}}{dx^2}+\Sigma_{RR2}\varphi_{R2}=\Sigma_{s12R}\varphi_{R1}$$

where

$$S'''=\nu\Sigma_{fC1}\varphi_{C1}+\nu\Sigma_{fC2}\varphi_{C2}$$

Governing Equations:

Adjoint Flux/Core Region:

$$-D_{C1}\frac{d^2\varphi^*_{C1}}{dx^2}+\Sigma_{RC1}\varphi^*_{C1}=\frac{1}{k^*}\nu\Sigma_{f1}S'''^*+\Sigma_{s12C}\varphi^*_{C2}$$

$$-D_{C2}\frac{d^2\varphi^*_{C2}}{dx^2}+\Sigma_{RC2}\varphi^*_{C2}=\frac{1}{k^*}\nu\Sigma_{f2}S'''^*$$

Adjoint Flux/Reflector Region:

$$-D_{R1}\frac{d^2\varphi^*_{R1}}{dx^2}+\Sigma_{RR1}\varphi^*_{R1}=\Sigma_{s12R}\varphi^*_{R2}$$

$$-D_{R2}\frac{d^2\varphi^*_{R2}}{dx^2}+\Sigma_{RR2}\varphi^*_{R2}=0$$

where

$$S'''^*=\chi_1\varphi^*_{C1}+\chi_2\varphi^*_{C2}$$

The star (*) is used to distinguish the adjoint flux from the flux. It turns out that for the solution of interest to us (the fundamental mode shape) it is possible to prove that the eigenvalue for the adjoint equations (k^*) is identical to the eigenvalue for the flux equations (k). We can either just update one of these (k, for instance) and use it in place of k^*, or we can update both k and k^* and use the fact that they should be equal when we're finished as a check on our calculations.

The specific procedure outlined in Exercise 11 is modified as follows for the two-group solutions of the flux shapes: (superscript (n) denotes iteration n):

(1) Begin with an initial eigenvector $\varphi_1^{(0)}=\varphi_2^{(0)}$ for both fast and thermal groups and multiplication factor $k^{(0)}=1$.
(2) The right-hand-side of the equation in the *core* region can then be evaluated as an initial source:

$$S'''^{(0)}=\nu\Sigma_{fC1}\varphi^{(0)}_{C1}+\nu\Sigma_{fC2}\varphi^{(0)}_{C2}$$

4/4/16

(The source strength in the reflector region is zero because there's no nuclear fuel there.)

(3) Solve the following equations by establishing appropriate matrices and vectors ($\mathbf{A}_1$ and $\mathbf{b}_1$) and ($\mathbf{A}_2$ and $\mathbf{b}_2$), and solving for updated flux shapes, $\varphi_1^{(1)}, \varphi_2^{(1)}$. Note that since the thermal flux depends on the fast flux but not vice versa in this approach, we can solve sequentially for the fast flux, and then the thermal flux (also note in the equations below $\chi_1 = 1$ and $\chi_2 = 0$):

Fast flux:

$$-D_{C1}\frac{d^2\varphi_{C1}^{(1)}}{dx^2} + \Sigma_{RC1}\varphi_{C1}^{(1)} = \frac{1}{k^{(0)}}S'''^{(0)}$$

$$-D_{R1}\frac{d^2\varphi_{R1}^{(1)}}{dx^2} + \Sigma_{RR1}\varphi_{R1}^{(1)} = 0$$

Thermal flux:

$$-D_{C2}\frac{d^2\varphi_{C2}^{(1)}}{dx^2} + \Sigma_{RC2}\varphi_{C2}^{(1)} = \frac{1}{k}\chi_2 S'''^{(0)} + \Sigma_{s12C}\varphi_{C1}^{(1)}$$

$$-D_{R2}\frac{d^2\varphi_{R2}^{(1)}}{dx^2} + \Sigma_{RR2}\varphi_{R2}^{(1)} = \Sigma_{s12R}\varphi_{R1}^{(1)}$$

(4) Calculate a new source strength $S'''^{(1)} = \nu\Sigma_{fC1}\varphi_{C1}^{(1)} + \nu\Sigma_{fC2}\varphi_{C2}^{(1)}$ and then update the eigenvalue using:

$$k^{(1)} = k^{(0)}\frac{\int S'''^{(1)}dx}{\int S'''^{(0)}dx}$$

(5) Compare old and new eigenvalues and decide whether or not to continue based on relative agreement to within some tolerance. If continuing, the right-hand-side of the core region is updated based on the first iterate, and we solve for the second iterate of flux from:

Fast flux:

$$-D_{C1}\frac{d^2\varphi_{C1}^{(2)}}{dx^2} + \Sigma_{RC1}\varphi_{C1}^{(2)} = \frac{1}{k^{(1)}}S'''^{(1)}$$

$$-D_{R1}\frac{d^2\varphi_{R1}^{(2)}}{dx^2} + \Sigma_{RR1}\varphi_{R1}^{(2)} = 0$$

Thermal flux:

$$-D_{C2}\frac{d^2\varphi_{C2}^{(2)}}{dx^2} + \Sigma_{RC2}\varphi_{C2}^{(2)} = \frac{1}{k}\chi_2 S'''^{(1)} + \Sigma_{s12C}\varphi_{C1}^{(2)}$$

$$-D_{R2}\frac{d^2\varphi_{R2}^{(2)}}{dx^2} + \Sigma_{RR2}\varphi_{R2}^{(2)} = \Sigma_{s12R}\varphi_{R1}^{(2)}$$

This process continues until convergence is reached.

4/4/16

For the adjoint equations, we have a similar procedure. In fact, the **A**-matrices, based on the left-hand-sides of the equations, look exactly the same for flux and adjoint flux calculations. The specific procedure is:

(1) Begin with an initial eigenvector $\varphi_1^{*(0)} = \varphi_2^{*(0)}$ for both fast and thermal groups and multiplication factor $k^{*(0)} = 1$.
(2) The right-hand-side of the equation in the *core* region can then be evaluated as an initial source:

$$S'''^{*(0)} = \chi_1 \varphi_{C1}^{*(0)} + \chi_2 \varphi_{C2}^{*(0)}$$

(3) Solve the following equations by establishing appropriate matrices and vectors ($\mathbf{A}_1$ and $\mathbf{b}_1$) and ($\mathbf{A}_2$ and $\mathbf{b}_2$), and solving for updated adjoint flux shapes, $\varphi_1^{*(1)}, \varphi_2^{*(1)}$. Note that, in the case of the adjoint, the fast adjoint flux depends on the thermal adjoint flux, but not vice versa, opposite of what we encountered in solving for the neutron flux shapes. We can solve sequentially for the thermal adjoint flux, and then the fast adjoint flux:

Thermal adjoint flux:

$$-D_{C2}\frac{d^2\varphi_{C2}^{*(1)}}{dx^2} + \Sigma_{RC2}\varphi_{C2}^{*(1)} = \frac{1}{k^{*(0)}}\nu\Sigma_{fC2}S'''^{*(0)}$$

$$-D_{R2}\frac{d^2\varphi_{R2}^{*(1)}}{dx^2} + \Sigma_{RR2}\varphi_{R2}^{*(1)} = 0$$

Fast adjoint flux:

$$-D_{C1}\frac{d^2\varphi_{C1}^{*(1)}}{dx^2} + \Sigma_{RC1}\varphi_{C1}^{*(1)} = \frac{1}{k^{*(0)}}\nu\Sigma_{fC1}S'''^{*(0)} + \Sigma_{s12C}\varphi_{C2}^{*(1)}$$

$$-D_{R1}\frac{d^2\varphi_{R1}^{*(1)}}{dx^2} + \Sigma_{RR1}\varphi_{R1}^{*(1)} = \Sigma_{s12R}\varphi_{R2}^{*(1)}$$

(4) Calculate a new source strength $S'''^{*(1)} = \chi_1 \varphi_{C1}^{*(1)} + \chi_2 \varphi_{C2}^{*(1)}$ and then update the eigenvalue using:

$$k^{*(1)} = k^{*(0)}\frac{\int S'''^{*(1)}dx}{\int S'''^{*(0)}dx}$$

(5) Compare old and new eigenvalues and decide whether or not to continue based on relative agreement to within some tolerance. If continuing, the right-hand-side of the core region is updated based on the first iterate, and we solve for the second iterate of flux from:

4/4/16

Thermal adjoint flux:

$$-D_{C2}\frac{d^2\varphi_{C2}^{*(2)}}{dx^2}+\Sigma_{RC2}\varphi_{C2}^{*(2)}=\frac{1}{k^{*(1)}}\nu\Sigma_{fC2}S'''^{*(1)}$$

$$-D_{R2}\frac{d^2\varphi_{R2}^{*(2)}}{dx^2}+\Sigma_{RR2}\varphi_{R2}^{*(2)}=0$$

Fast adjoint flux:

$$-D_{C1}\frac{d^2\varphi_{C1}^{*(2)}}{dx^2}+\Sigma_{RC1}\varphi_{C1}^{*(2)}=\frac{1}{k^{*(1)}}\nu\Sigma_{fC1}S'''^{*(1)}+\Sigma_{s12C}\varphi_{C2}^{*(2)}$$

$$-D_{R1}\frac{d^2\varphi_{R1}^{*(2)}}{dx^2}+\Sigma_{RR1}\varphi_{R1}^{*(2)}=\Sigma_{s12R}\varphi_{R2}^{*(2)}$$

This process continues until convergence is reached.

Specifics: In a PBMR, the geometry is somewhat different than shown on the first page of this handout. Now we have three regions in cylindrical geometry: an inner reflector, an annular core region, and an outer reflector. The inner and outer reflector regions can be treated as pure graphite with identical properties, given as reflector properties below. Find the neutron multiplication factor assuming the radius of the inner reflector is 100 cm, the radius of the outer core is 200 cm and the outer boundary of the outer reflector is 250 cm. Use the following two-group constants for the regions:

Core: $\nu\Sigma_{f1}=0.0017\ \text{cm}^{-1}$ $\nu\Sigma_{f2}=0.0245\ \text{cm}^{-1}$
$\Sigma_{R1}=0.025\ \text{cm}^{-1}$ $\Sigma_{R2}=0.024\ \text{cm}^{-1}$ $\Sigma_{s12}=0.0233\ \text{cm}^{-1}$
$D_1=1.427\ \text{cm}$ $D_2=1.302\ \text{cm}$

Reflector: $\Sigma_{R1}=0.0402\ \text{cm}^{-1}$ $\Sigma_{R2}=0.000241\ \text{cm}^{-1}$ $\Sigma_{s12}=0.0401\ \text{cm}^{-1}$
$D_1=0.876\ \text{cm}$ $D_2=0.876\ \text{cm}$

Find the flux and adjoint flux shapes for the two-group representation as well as the eigenvalue $k = k^*$.

Reminder: Reactor physics problems require tighter convergence tolerances than other types of eigenvalue problems. We have previously been using `tol` = 1e-6. For this problem use `tol` = 1e-9.

Chapter 4

my project report

Local contents

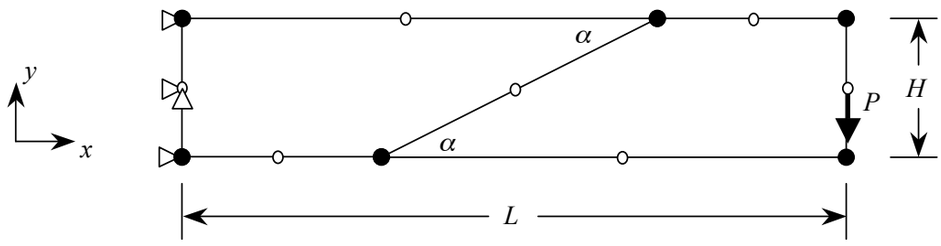
4.1 Description of the problem, geometry and element description

4.1.1 Problem statement

For completion of the report, the problem statement is given below taken from the project handout.

Assembly of Planar FE Problem

In EMA 405, we typically illustrate the element assembly process with 1D elements (bars or beams). The purpose of this project is to give you an idea of what's involved in assembling a 2D FE continuum problem (plane stress) from first principles. One of the problems we typically investigate is a two-Q8-element mesh distortion problem, where the geometry looks like this:



For the specific illustration in EMA 405, the numbers are these: $E = 1 \times 10^4$, $\nu = 0.3$, $L = 10$, $H = 2$, and $P = 20$. The angle α is a measure of the mesh distortion (departures of angles between adjacent edges from 90°) and is a parameter to be varied. The expected vertical displacement at the node where force P is applied, based on beam theory, is 1.031. You are to assemble a global stiffness matrix from the two element stiffness matrices and solve for the deflections at the nodes. You can compare your results to a comparable ANSYS FE model.

The global stiffness matrix is assembled from element stiffness matrices and embodies internal strain energy. Briefly, it starts out from integrating volumetric strain energy over the volume of the element. If we had a simple, linear spring, initially undeformed, and stretched it a distance x, the energy stored in that spring would be

$$\text{Energy} = \frac{1}{2} x \cdot kx = \frac{1}{2} kx^2$$

In a continuum problem in 2D, we have three modes of deformation: normal strain in x, normal strain in y and shear strain (xy). Instead of displacement × force [energy], we use strain × stress [N/m² or N·m/m³], the latter being energy density. The energy in the material is:

Figure 4.1: EMA project problem description

The problem described above was solved for the following values of the angle α (in degrees)

$$\{0,15,30,45,50,55\}$$

Where α is the distortion angle between the first and the second elements as shown in the above diagram. The method of finite elements was implemented in Matlab to solve for the displacements at the nodes. 4 Gaussian points were used for the integration step (this is also called the 2×2 integration rule). After the global stiffness matrix was assembled from the two elements stiffness matrices, the system equations given by $KD = F$ were solved using the direct linear system solver.

4.1.2 Geometry, nodes and element description

Two elements were used each having 8 nodes. 4 of these nodes are at the corners, and the other 4 are at the mid point of the element edges. The following diagram shows the global

coordinates of the elements nodes. The origin of the global coordinate system is at the lower left corner as shown in the diagram below.

L is the overall length of the beam, which is 10 meters, and h is the height of the beam (which is 2 meters).

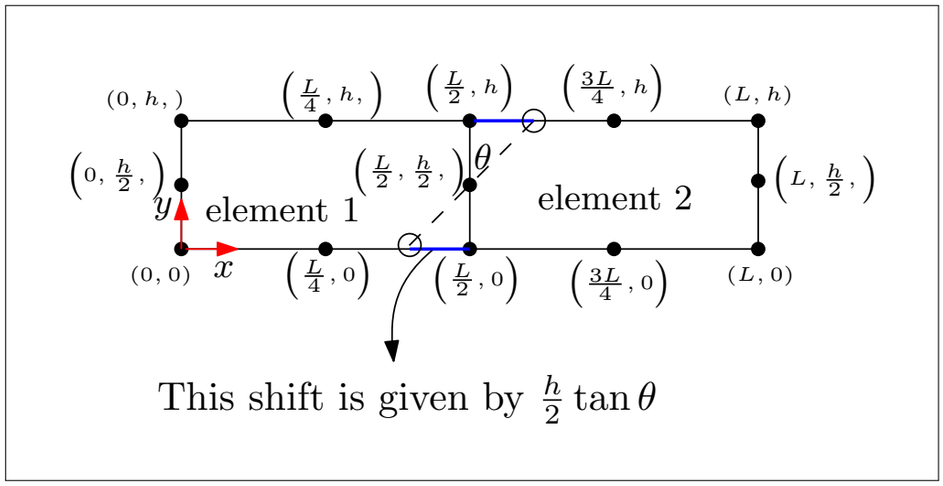


Figure 4.2: Global node coordinates using general coordinates, showing the distortion angle

The vertical and horizontal displacement of each node was solved for using the finite elements method for each of the different values of the angle α and the vertical displacement at the right bottom edge of the beam was compared to the expected theoretical value in order to see the effect of the element distortion on the accuracy of the finite element result using the element selected.

The idea is that a good finite element should produce the same displacement at its nodes regardless of how it was fitted to the physical region in place. This report was to determine if the element selected would still produce good results when deformed. The first step was to map each element local node number to a global node number. Local element numbers go from 1 to 8 since there are only 8 nodes per element, but the global node numbers enumerates over all the nodes in all the elements.

Local element node numbering is made in the standard anti clock wise direction by numbering the corner nodes first from 1 to 4, followed by numbering the middle nodes also in the anti clock wise sense from 5 to 8.

The following diagram shows the mapping between the local element node numbers and the global node numbers. The top diagram shows the global node numbers and the lower diagram shows each element local node numbers.

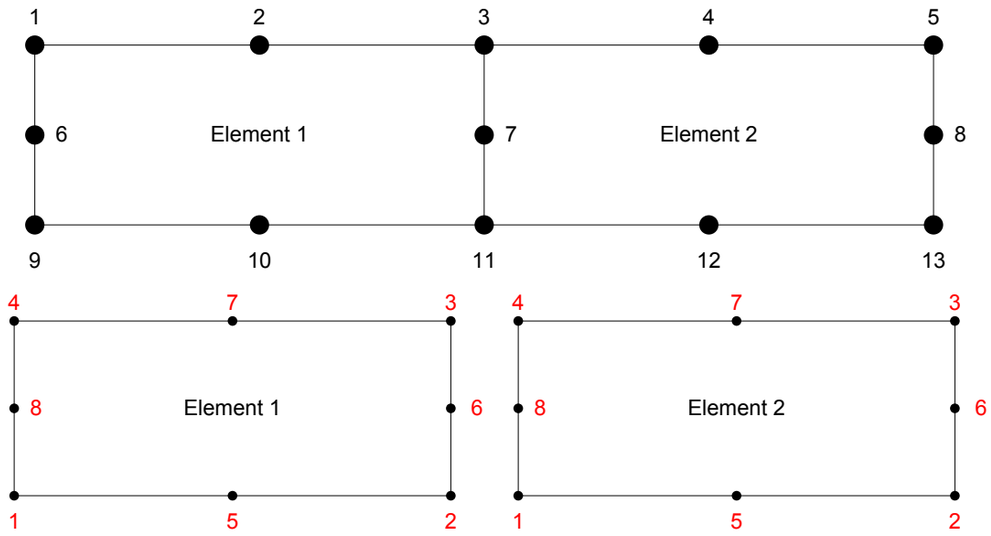


Figure 4.3: Global and element node numbering used for project EMA

Based on the the above, the table `elem_map_nodes` was constructed. This table gives the global node number (the entries inside the table) for an element number (the row of the table) and the node number within that specific element (the column in the table). For example, for element 1 with local node number 1 the table above shows that the global node number is 9.

element # \ element node #	1	2	3	4	5	6	7	8
1	9	11	3	1	10	7	2	6
2	11	13	5	3	12	8	4	7

Table 4.1: elem_map_nodes table. Mapping element node to global nodes

There is also a need to lookup the global coordinates $\{x_i, y_i\}$ given the global node number. This is needed when finding the Jacobian.

The following table called `global_coordinates_tbl` was constructed for this purpose. In this table, $H = 2, L = 10$ and $\Delta = \tan\theta\frac{H}{2}$ is the amount of shift in meters of the global node 3 and 11. These are the only two nodes which shift location when changing the angle α. When α is zero, there will be no distortion of the elements.

global node # \ global node coordinates	x_i	y_i
1	0	H
2	$\frac{L}{4}$	H
3	$\frac{L}{2}+\Delta$	H
4	$\frac{3}{4}L$	H
5	L	H
6	0	$\frac{H}{2}$
7	$\frac{L}{2}$	$\frac{H}{2}$
8	L	$\frac{H}{2}$
9	0	0
10	$\frac{L}{4}$	0
11	$\frac{L}{2}-\Delta$	0
12	$\frac{3}{4}L$	0
13	L	0

Table 4.2: global_coordinates_tbl. Mapping global node number to global coordinates

There are two degrees of freedom at each node. These are u, v, representing the horizontal and vertical displacement of a node. Hence there is a need for a lookup table called `elem_map_dofs` which gives the degree of freedom number of each element's local node. This table is used for assembling the global stiffness matrix.

Using the method in the project handout, the following table was generated.

element # \ element #	node 1		node 2		node 3		node 4		node 5		node 6		node 7		no
1	17	18	21	22	5	6	1	2	19	20	13	14	3	4	11
2	21	22	25	26	9	10	5	6	23	24	15	16	7	8	13

Table 4.3: elem_map_dof table. Mapping local element DOF to global stiffness matrix locations

The following diagram, generated in the Matlab program, gives the degree of freedom number corresponding to each element node (u, v). These DOF numbers represent the position of the unknowns in the solution of the $KD = F$. This means that there are a total of 26 degrees of freedom initially. However, due to boundary conditions constraints, the total number of degrees of freedom reduces to 22.

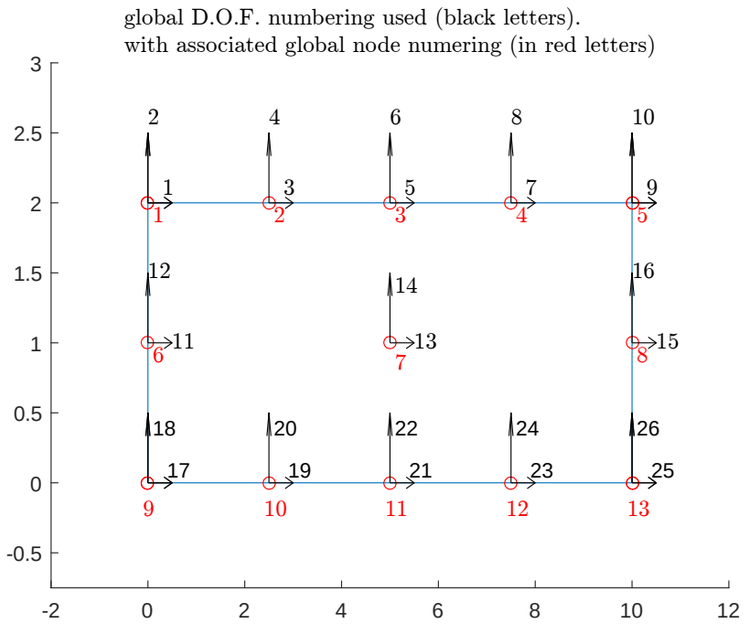


Figure 4.4: global DOF numbering used

The above was a general description of the problem and the geometry and data structures used in the Matlab implementation. Next is a discussion of the analytical derivation and the post processing stage which starts after solving for the displacements, followed by discussion of how stress was calculated at the element nodes from the stress value at the four Gaussian points.

4.2 Theory and analytical derivation

4.2.1 Shape functions

Since an 8 node element is used, then there will be 8 shape functions. In ANSYS, the element used is called `PLANE183`. This element is a serendipity element, which means it has nodes only on the edges and no node in the middle of the element.

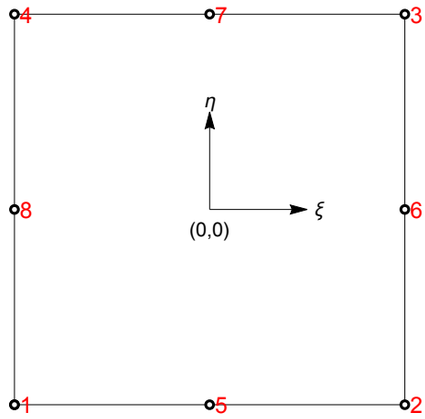


Figure 4.5: 8-node element used for the finite elements

The following are the 8 shape functions used

$$f_1 = -\frac{1}{4}\left(1-\eta^2\right)(1-\xi) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}(1-\eta)(1-\xi)$$
$$f_2 = -\frac{1}{4}\left(1-\eta^2\right)(\xi+1) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}(1-\eta)(\xi+1)$$
$$f_3 = -\frac{1}{4}\left(1-\eta^2\right)(\xi+1) - \frac{1}{4}(\eta+1)\left(1-\xi^2\right) + \frac{1}{4}(\eta+1)(\xi+1)$$
$$f_4 = -\frac{1}{4}\left(1-\eta^2\right)(1-\xi) - \frac{1}{4}(\eta+1)\left(1-\xi^2\right) + \frac{1}{4}(\eta+1)(1-\xi)$$
$$f_5 = \frac{1}{2}(1-\eta)\left(1-\xi^2\right)$$
$$f_6 = \frac{1}{2}\left(1-\eta^2\right)(\xi+1)$$
$$f_7 = \frac{1}{2}(\eta+1)\left(1-\xi^2\right)$$
$$f_8 = \frac{1}{2}\left(1-\eta^2\right)(1-\xi)$$

The global coordinates x, y are now expressed as functions of the natural coordinates ξ, η using

$$x(\xi,\eta) = \sum_{i=1}^{M} x_i f_i(\xi,\eta)$$
$$y(\xi,\eta) = \sum_{i=1}^{M} y_i f_i(\xi,\eta)$$

Where M is the number of nodes of each element (which is 8) and x_i, y_i are the global coordinates of these nodes. Expanding the above gives the result below.

The result below is shown for some element number k. In this expansion, x_i^k means the global x coordinate of the i^{th} node in the k^{th} element.

Similarly, y_i^k means the global y coordinate of the i^{th} node in the k^{th} element. These are read in the Matlab code using the `global_coordinates_tbl` table, which gives the global x, y coordinates of each global node and by using `elem_map_nodes` table to map the element

node number to the global node number.

$$x(\xi,\eta) = x_1^k\left(-\frac{1}{4}\left(1-\eta^2\right)(1-\xi) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}(1-\eta)(1-\xi)\right) + x_2^k\left(-\frac{1}{4}\left(1-\eta^2\right)(\xi+1) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}\right.$$

$$y(\xi,\eta) = y_1^k\left(-\frac{1}{4}\left(1-\eta^2\right)(1-\xi) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}(1-\eta)(1-\xi)\right) + y_2^k\left(-\frac{1}{4}\left(1-\eta^2\right)(\xi+1) - \frac{1}{4}(1-\eta)\left(1-\xi^2\right) + \frac{1}{4}\right.$$

4.2.2 Finding the Jacobian

The Jacobian is evaluated at each Gaussian integration point during the integration step. It has the form

$$J = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{pmatrix}$$

Each of the above derivatives is evaluated and used in the Matlab code.

$$\frac{\partial x}{\partial \xi} = x_1\left(\frac{1}{4}\left(1-\eta^2\right) + \frac{1}{2}(1-\eta)\xi + \frac{\eta-1}{4}\right) + x_2\left(\frac{1}{4}\left(\eta^2-1\right) + \frac{1}{2}(1-\eta)\xi + \frac{1-\eta}{4}\right) + x_3\left(\frac{1}{4}\left(\eta^2-1\right) + \frac{1}{2}(\eta+1)\xi + \frac{\eta+1}{4}\right)$$

$$\frac{\partial y}{\partial \xi} = y_1\left(\frac{1}{4}\left(1-\eta^2\right) + \frac{1}{2}(1-\eta)\xi + \frac{\eta-1}{4}\right) + y_2\left(\frac{1}{4}\left(\eta^2-1\right) + \frac{1}{2}(1-\eta)\xi + \frac{1-\eta}{4}\right) + y_3\left(\frac{1}{4}\left(\eta^2-1\right) + \frac{1}{2}(\eta+1)\xi + \frac{\eta+1}{4}\right)$$

$$\frac{\partial x}{\partial \eta} = x_1\left(\frac{1}{2}\eta(1-\xi) + \frac{1}{4}\left(1-\xi^2\right) + \frac{\xi-1}{4}\right) + x_2\left(\frac{1}{2}\eta(\xi+1) + \frac{1}{4}\left(1-\xi^2\right) + \frac{1}{4}(-\xi-1)\right) + x_3\left(\frac{1}{2}\eta(\xi+1) + \frac{1}{4}\left(\xi^2-1\right) + \frac{\xi}{}\right.$$

$$\frac{\partial y}{\partial \eta} = y_1\left(\frac{1}{2}\eta(1-\xi) + \frac{1}{4}\left(1-\xi^2\right) + \frac{\xi-1}{4}\right) + y_2\left(\frac{1}{2}\eta(\xi+1) + \frac{1}{4}\left(1-\xi^2\right) + \frac{1}{4}(-\xi-1)\right) + y_3\left(\frac{1}{2}\eta(\xi+1) + \frac{1}{4}\left(\xi^2-1\right) + \frac{\xi}{}\right.$$

The above is now used to find the Jacobian and its determinant and also find Γ and the matrix B_2. To find the matrix B_3 the derivatives of each shape function is taken w.r.t. ξ and η.

This below gives the result of this computation

$$\begin{aligned}
\frac{\partial f_1}{\partial \xi} &= \frac{1}{4}\left(1-\eta^2\right)+\frac{1}{2}(1-\eta)\xi+\frac{\eta-1}{4}\\
\frac{\partial f_1}{\partial \eta} &= \frac{1}{2}\eta(1-\xi)+\frac{1}{4}\left(1-\xi^2\right)+\frac{\xi-1}{4}\\
\frac{\partial f_2}{\partial \xi} &= \frac{1}{4}\left(\eta^2-1\right)+\frac{1}{2}(1-\eta)\xi+\frac{1-\eta}{4}\\
\frac{\partial f_2}{\partial \eta} &= \frac{1}{2}\eta(\xi+1)+\frac{1}{4}\left(1-\xi^2\right)+\frac{1}{4}(-\xi-1)\\
\frac{\partial f_3}{\partial \xi} &= \frac{1}{4}\left(\eta^2-1\right)+\frac{1}{2}(\eta+1)\xi+\frac{\eta+1}{4}\\
\frac{\partial f_3}{\partial \eta} &= \frac{1}{2}\eta(\xi+1)+\frac{1}{4}\left(\xi^2-1\right)+\frac{\xi+1}{4}\\
\frac{\partial f_4}{\partial \xi} &= \frac{1}{4}\left(1-\eta^2\right)+\frac{1}{2}(\eta+1)\xi+\frac{1}{4}(-\eta-1)\\
\frac{\partial f_4}{\partial \eta} &= \frac{1}{2}\eta(1-\xi)+\frac{1}{4}\left(\xi^2-1\right)+\frac{1-\xi}{4}\\
\frac{\partial f_5}{\partial \xi} &= -(1-\eta)\xi\\
\frac{\partial f_5}{\partial \eta} &= \frac{1}{2}\left(\xi^2-1\right)\\
\frac{\partial f_6}{\partial \xi} &= \frac{1}{2}\left(1-\eta^2\right)\\
\frac{\partial f_6}{\partial \eta} &= -\eta(\xi+1)\\
\frac{\partial f_7}{\partial \xi} &= -(\eta+1)\xi\\
\frac{\partial f_7}{\partial \eta} &= \frac{1}{2}\left(1-\xi^2\right)\\
\frac{\partial f_8}{\partial \xi} &= \frac{1}{2}\left(\eta^2-1\right)\\
\frac{\partial f_8}{\partial \eta} &= -\eta(1-\xi)
\end{aligned}$$

With the above, the matrix B_3 matrix was calculated giving

$$B = B_1 B_2 B3$$

And calculate the element stiffness matrix given by

$$\begin{aligned}
k_{\text{elem}} &= \int B^T E B \, dV\\
&= \int_{-1}^{+1}\int_{-1}^{+1} B^T E B |J| \, dV
\end{aligned}$$

The elements stiffness matrices k_{elem} are then combined to make the global stiffness matrix K and then the system $KD = F$ was solved for the unknowns D which are the nodal displacements in the x and y direction.

In the above F is the load vector, which in this problem contains only one non-zero entry, which is the vertical load of $-20N$ at the middle of the right edge of the beam.

In the above, the matrix B_1 is

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 1 & 1 & 0 \end{bmatrix}$$

And the matrix B_2 is

$$B_2 = \begin{bmatrix} \Gamma_{11} & \Gamma_{12} & 0 & 0 \\ \Gamma_{21} & \Gamma_{22} & 0 & 0 \\ 0 & 0 & \Gamma_{11} & \Gamma_{12} \\ 0 & 0 & \Gamma_{21} & \Gamma_{22} \end{bmatrix}$$

Where Γ is the inverse of the Jacobian matrix $\Gamma = J^{-1}$. And the matrix B_3 is

$$B_3 = \begin{bmatrix} \frac{\partial f_1}{\partial \xi} & 0 & \frac{\partial f_2}{\partial \xi} & 0 & \dots & \frac{\partial f_8}{\partial \xi} & 0 \\ \frac{\partial f_1}{\partial \eta} & 0 & \frac{\partial f_2}{\partial \eta} & 0 & \dots & \frac{\partial f_8}{\partial \eta} & 0 \\ 0 & \frac{\partial f_1}{\partial \xi} & 0 & \frac{\partial f_2}{\partial \xi} & \dots & 0 & \frac{\partial f_8}{\partial \xi} \\ 0 & \frac{\partial f_1}{\partial \eta} & 0 & \frac{\partial f_2}{\partial \eta} & \dots & 0 & \frac{\partial f_8}{\partial \eta} \end{bmatrix}$$

The following diagram shows the internal structure of the global stiffness matrix, found using the `spy()` command in Matlab. It shows the bands along the diagonals and illustrates how sparse the matrix is.

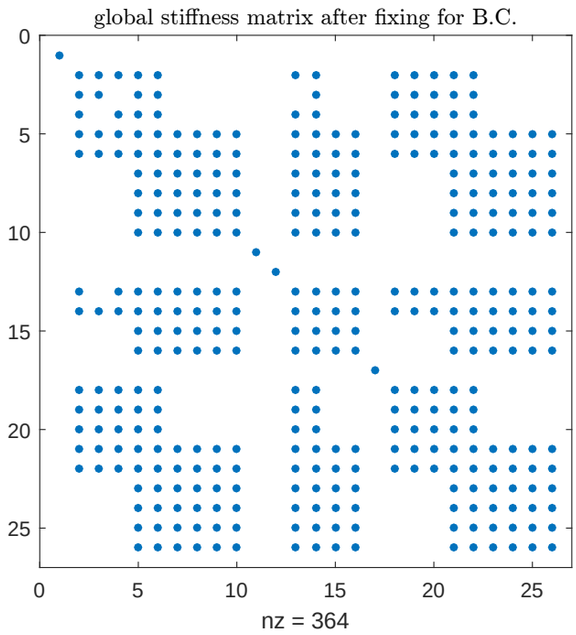


Figure 4.6: Global stiffness matrix spy() output showing the bands

The above concludes the solve stage. The next stage is the post processing, where stress calculations are performed.

4.2.3 Stress recovery

This is a discussion of how the stress at the elements nodes was found. Initially the direct method was used to find the stress at any point in the element.

This method was found to be accurate as long as there was no distortion. Once the angle was increased, this method did not produce stress results which agreed with ANSYS. Therefore, this method was not used, and instead a new implementation was made based on the extrapolation method.

This method is described in reference [1], pages 230-232. This method is more complicated that the direct method, but it is much more accurate. It uses two coordinates systems. The original natural coordinates system of the element (ξ, η), which extends from $-1 \dots 1$ across the length and height of the element, and a new coordinates systems called (r, s) which extends across what is called the Gaussian element.

Therefore, when $\xi = \frac{1}{\sqrt{3}}$ the value of r is one. And when $\eta = \frac{1}{\sqrt{3}}$ then $s = 1$ also.

The following diagram shows this layout more clearly.

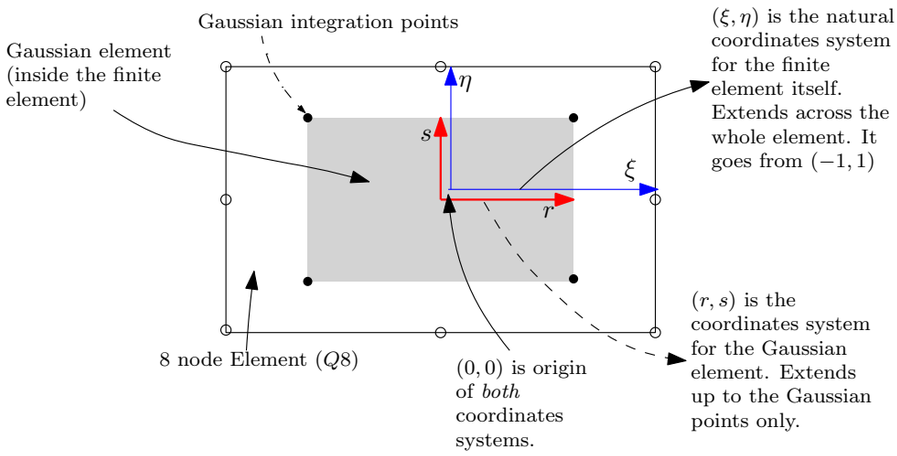


Figure 4.7: Stress recovery using r,s coordinates system inside ξ,η

Therefore, the relation between (r,s) coordinates system and (ξ,η) coordinates system is as follows

$$r = \xi\sqrt{3}$$
$$s = \eta\sqrt{3}$$

The following diagram shows the mapping at two different points for illustration

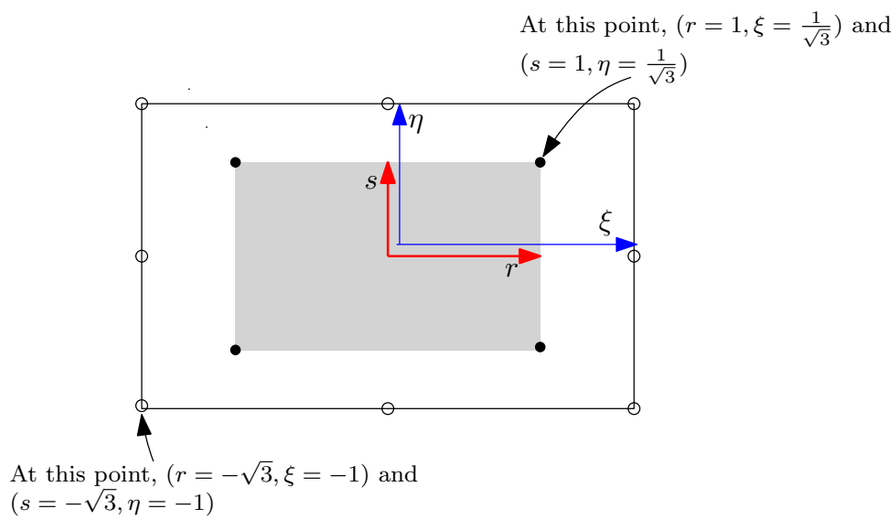


Figure 4.8: Stress recovery using r,s coordinates system inside ξ,η

Now that the mapping between ξ,η and (r,s) is determined, the next step was to find (r,s) at each point where the stress needs to be found at by extrapolating the stress value from the 4 Gaussian points to that point of interest. The reason for doing all of the above, is because (r,s) are used when evaluating the N_i shape functions described below, and not the original (ξ,η) values.

Therefore, for each point where the stress needs to be found, say point p, its coordinates in the (r,s) system are found first, and then the following extrapolation formula is applied

$$\sigma_p = \sum_{i=1}^{4} N_i \sigma_i$$

Where σ_i is the stress at the Gaussian point (which was found using the direct method based on the full 8 shape functions of the main element).

In the above, N_i are the shape functions used for extrapolation. These are not the same shape functions used in the original element. These shape functions are based on 4 node Gaussian element and are given by

$$
\begin{aligned}
N_1 &= \frac{1}{4}(1-r)(1-s) \\
N_2 &= \frac{1}{4}(1+r)(1-s) \\
N_3 &= \frac{1}{4}(1+r)(1+s) \\
N_4 &= \frac{1}{4}(1-r)(1+s)
\end{aligned}
$$

For example, to find the stress at point $(\xi,\eta) = (0,-1)$, the first step is to determine this point's (r_p, s_p) coordinates. Since $r_p = \sqrt{3}\xi$ then $r_p = 0$ and since $s_p = \sqrt{3}\eta$ then $s_p = -\sqrt{3} =$. Applying the extrapolation formula above gives

$$
\begin{aligned}
\sigma_p &= \left(\frac{1}{4}(1-r_p)(1-s_p)\right)\sigma_1 + \left(\frac{1}{4}(1+r_p)(1-s_p)\right)\sigma_2 + \left(\frac{1}{4}(1+r_p)(1+s_p)\right)\sigma_3 + \left(\frac{1}{4}(1-r_p)(1+s_p)\right)\sigma_4 \\
&= \left(\frac{1}{4}(1+\sqrt{3})\right)\sigma_1 + \left(\frac{1}{4}(1+\sqrt{3})\right)\sigma_2 + \left(\frac{1}{4}(1-\sqrt{3})\right)\sigma_3 + \left(\frac{1}{4}(1-\sqrt{3})\right)\sigma_4 \\
&= 0.6830127\sigma_1 + 0.6830127\sigma_2 - 0.1830127\sigma_3 - 0.1830127\sigma_4
\end{aligned}
$$

Since the stresses at four Gaussian points σ_i are known, the stress at the point p is now found from the above. The above method was found to produce more accurate result for σ_p than using the direct method to find σ_p and the result found for the stress at the nodes agreed with those found by ANSYS.

The following diagram shows the (r,s) coordinates of all the element points used to calculate the stresses at using this method. A total of 13 points was used per element. These are the 8 nodes of the element, and also the center of the element and the Gaussian points themselves giving a total of 13 points. These are used to generate the stress contour. This was done for both elements. The generated stress contour agreed with ANSYS results.

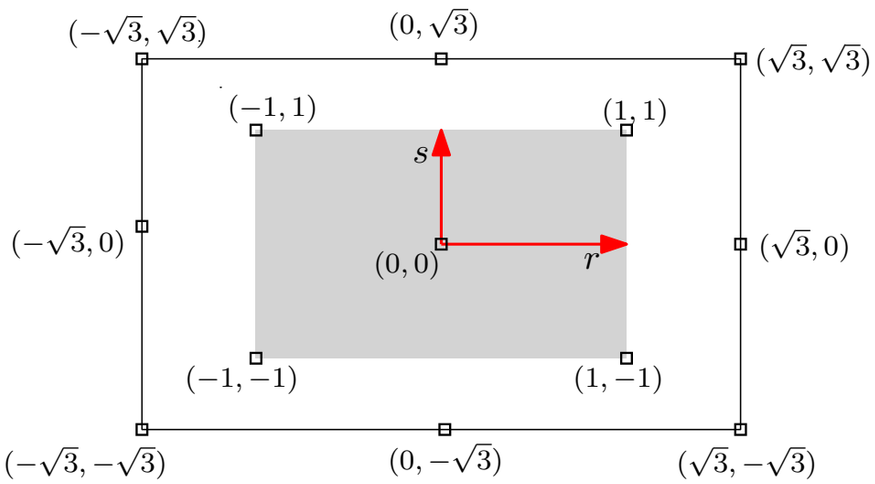


Figure 4.9: Location of points used for stress recovery in the r,s coordinate system

4.3 Results

The results are listed by the angle θ used to distort the elements with. For each angle, the deflection and σ_x stress contour and tables are generated using Matlab and also using ANYS to compare with side by side.

The following angles are used (in degrees) $\{0, 15, 30, 45, 50, 55\}$. When trying to use 60 degrees distortion, ANSYS complained and gave number of computational error messages relating to the element shape. It is not clear why ANSYS did not accept such large angle, since the Matlab implementation worked. But since ANSYS did not produce result for this case, the angle 55 degrees was the maximum distortion used for both Matlab and ANSYS.

For each angle, a short summary of the result in the form of a table is first given that compares Matlab and ANSYS result. This short summary contains only the deflection at the bottom right corner, which is node 13. After this, the full result and stress plots are given.

4.3.1 No element distortion. zero angle

4.3.1.1 summary of result

	x (meter)	y (meter)
Matlab	−0.15	−1.02895
ANSYS	−0.15	−1.0289

Table 4.4: Short summary of test case zero degree distortion

4.3.1.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.004650
2	0.065794	−0.096522
3	0.112725	−0.329300
4	0.140794	−0.655828
5	0.150000	−1.028950
6	0.000000	0.000000
7	−0.000000	−0.327050
8	−0.000000	−1.029100
9	0.000000	−0.004650
10	−0.065794	−0.096522
11	−0.112725	−0.329300
12	−0.140794	−0.655828
13	−0.150000	−1.028950

Table 4.5: Matlab result. nodal solutions, angle [0] degree

The following figure shows the deformation found

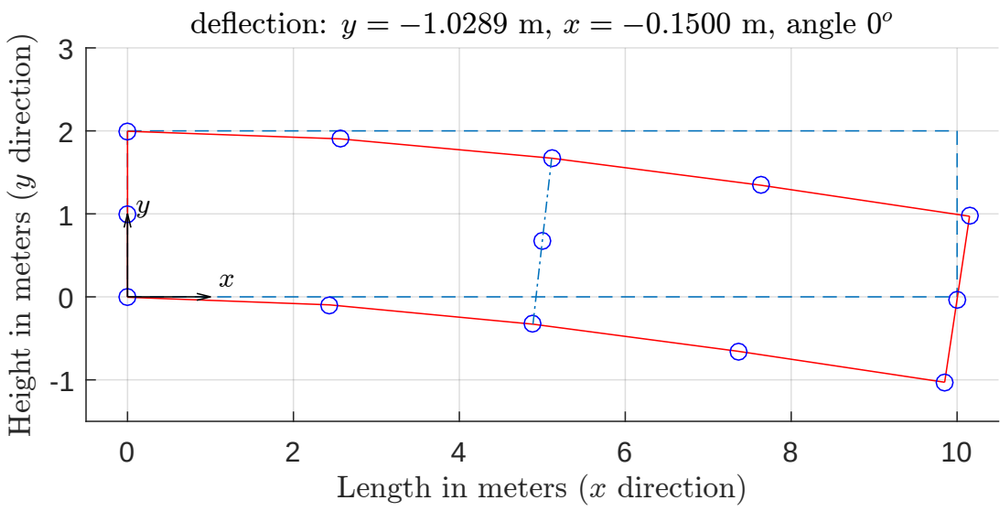


Figure 4.10: deflection found using 2 elements using Matlab, zero degree

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	–300.000
11	5.0000	0.0000	–150.000
3	5.0000	2.0000	150.000
1	0.0000	2.0000	300.000
10	2.5000	0.0000	–225.000
7	5.0000	1.0000	–0.000
2	2.5000	2.0000	225.000
6	0.0000	1.0000	–0.000
center	2.5000	1.0000	–0.000
Gauss point 1	1.0566	0.4226	–154.904
Gauss point 2	3.9434	0.4226	–104.904
Gauss point 3	3.9434	1.5774	104.904
Gauss point 4	1.0566	1.5774	154.904

Table 4.6: Matlab result. direct stress σ_x at each node, First element, angle [0] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	5.0000	0.0000	–150.000
13	10.0000	0.0000	–0.000
5	10.0000	2.0000	0.000
3	5.0000	2.0000	150.000
12	7.5000	0.0000	–75.000
8	10.0000	1.0000	0.000
4	7.5000	2.0000	75.000
7	5.0000	1.0000	0.000
center	7.5000	1.0000	0.000
Gauss point 1	6.0566	0.4226	–68.301
Gauss point 2	8.9434	0.4226	–18.301
Gauss point 3	8.9434	1.5774	18.301
Gauss point 4	6.0566	1.5774	68.301

Table 4.7: Matlab result. direct stress at each node, Second element, angle [0] degree

The following shows the direct stress contour generated in Matlab

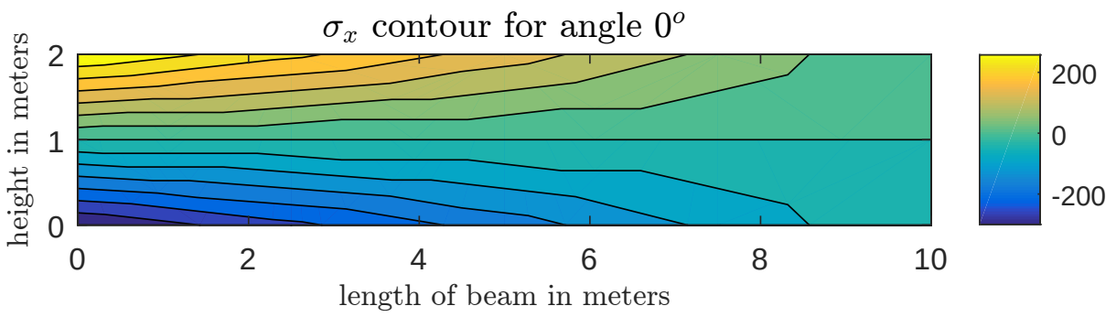


Figure 4.11: Contour of direct stress found using 2 elements using Matlab, zero degree

4.3.1.3 ANSYS result

```
1
2  PRINT U    NODAL SOLUTION PER NODE
3
4   ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
5
6   LOAD STEP=     1  SUBSTEP=     1
7    TIME=    1.0000      LOAD CASE=   0
8
9   THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
10
11     NODE       UX            UY            UZ            USUM
12        1   0.0000     -0.46500E-002  0.0000       0.46500E-002
13        2  0.65794E-001-0.96522E-001  0.0000       0.11681
14        3  0.11272     -0.32930       0.0000       0.34806
15        4  0.14079     -0.65583       0.0000       0.67077
16        5  0.15000      -1.0289       0.0000        1.0398
17        6   0.0000       0.0000       0.0000        0.0000
18        7  0.12540E-013-0.32705       0.0000       0.32705
19        8  0.14395E-013 -1.0291       0.0000        1.0291
20        9   0.0000     -0.46500E-002  0.0000       0.46500E-002
21       10 -0.65794E-001-0.96522E-001  0.0000       0.11681
22       11 -0.11272     -0.32930       0.0000       0.34806
23       12 -0.14079     -0.65583       0.0000       0.67077
24       13 -0.15000      -1.0289       0.0000        1.0398
25
26  MAXIMUM ABSOLUTE VALUES
27  NODE          5             8             0            13
28  VALUE   0.15000      -1.0291       0.0000       1.0398
29
30  /OUTPUT FILE= ansys_stress_solution_0.txt
```

The following figure shows the deformation found

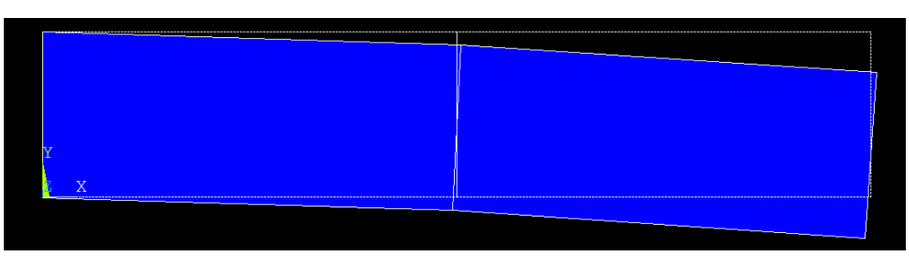


Figure 4.12: deflection found using 2 elements using ANSYS, zero degree

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```
1
2  PRINT S    ELEMENT SOLUTION PER ELEMENT
```

```
 ***** POST1 ELEMENT NODAL STRESS LISTING *****

  LOAD STEP=     1  SUBSTEP=     1
   TIME=    1.0000      LOAD CASE=   0

  THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


  ELEMENT=       1        PLANE183
    NODE     SX           SY           SZ           SXY          SYZ
    SXZ
       9  -300.00       3.0000       0.0000      -10.000       0.0000
    0.0000
      11  -150.00     -0.18598E-010  0.0000      -10.000       0.0000
    0.0000
       3   150.00      0.21138E-010  0.0000      -10.000       0.0000
    0.0000
       1   300.00      -3.0000       0.0000      -10.000       0.0000
    0.0000

  ELEMENT=       2        PLANE183
    NODE     SX           SY           SZ           SXY          SYZ
    SXZ
      11  -150.00      0.36168E-010  0.0000      -10.000       0.0000
    0.0000
      13 -0.45564E-010 -3.0000       0.0000      -10.000       0.0000
    0.0000
       5  0.45869E-010  3.0000       0.0000      -10.000       0.0000
    0.0000
       3   150.00     -0.51808E-010  0.0000      -10.000       0.0000
    0.0000
```

The following shows the direct stress contour generated in ANSYS

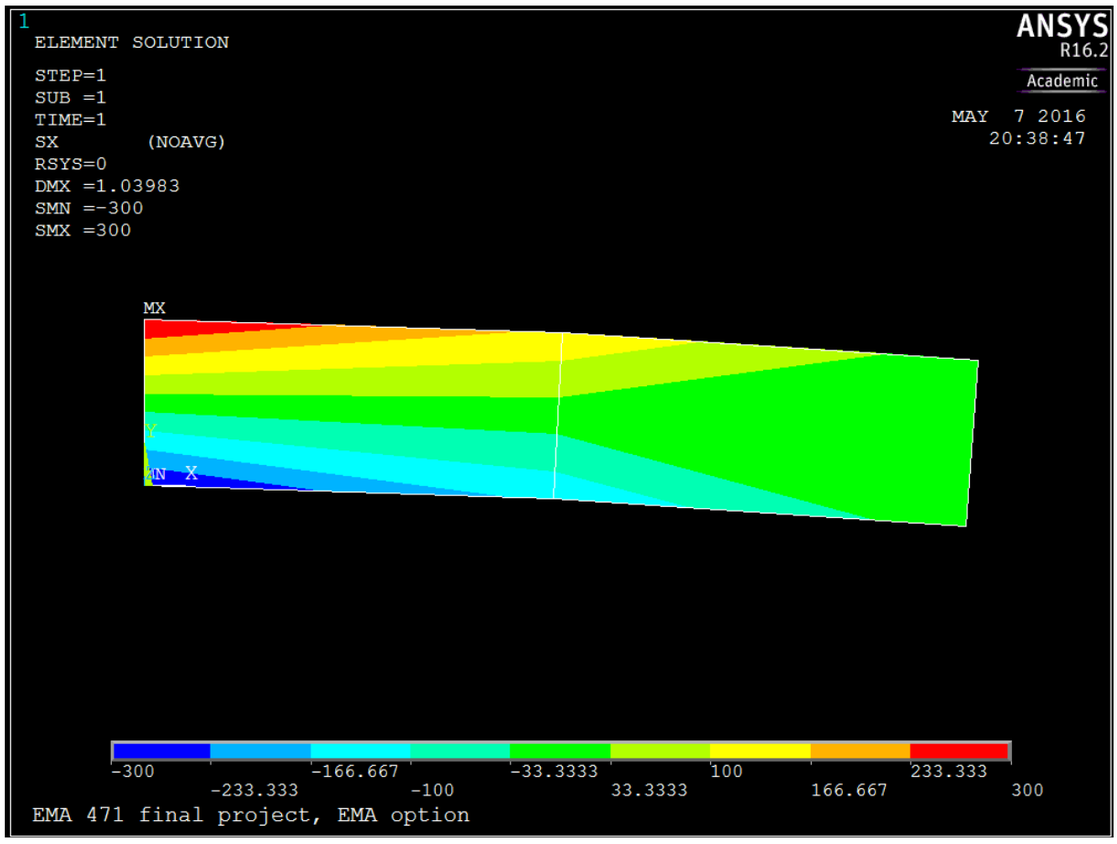


Figure 4.13: Contour of direct stress found using 2 elements using ANSYS, zero degree

4.3.2 15 degrees distortion

4.3.2.1 summary of result

	x (meter)	y (meter)
Matlab	−0.150262	−1.02555
ANSYS	−0.15026	−1.0256

Table 4.8: Short summary of test case 15 degrees distortion

4.3.2.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.008808
2	0.066221	−0.096603
3	0.115141	−0.356575
4	0.138750	−0.653161
5	0.146026	−1.012233
6	0.000000	0.000000
7	0.000596	−0.326056
8	0.001059	−1.018939
9	0.000000	−0.000457
10	−0.064844	−0.096198
11	−0.108540	−0.300195
12	−0.139066	−0.648279
13	−0.150262	−1.025550

Table 4.9: Matlab result. nodal solutions, angle [15] degree

The following figure shows the deformation found

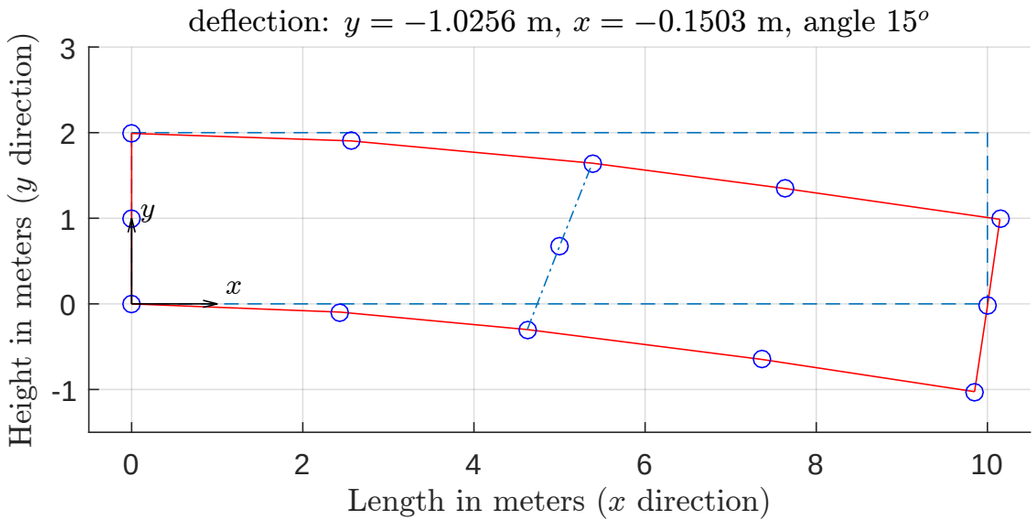


Figure 4.14: deflection found using 2 elements using Matlab, 15 degrees

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	−299.049
11	4.7321	0.0000	−148.195
3	5.2679	2.0000	144.369
1	0.0000	2.0000	300.912
10	2.5000	0.0000	−223.622
7	5.0000	1.0000	−1.913
2	2.5000	2.0000	222.641
6	0.0000	1.0000	0.932
center	2.5000	1.0000	−0.491
Gauss point 1	1.0566	0.4226	−154.111
Gauss point 2	3.9434	0.4226	−104.520
Gauss point 3	3.9434	1.5774	101.897
Gauss point 4	1.0566	1.5774	154.772

Table 4.10: Matlab result. direct stress σ_x at each node, First element, angle [15] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	4.7321	0.0000	−146.183
13	10.0000	0.0000	−0.994
5	10.0000	2.0000	2.746
3	5.2679	2.0000	142.734
12	7.5000	0.0000	−73.588
8	10.0000	1.0000	0.876
4	7.5000	2.0000	72.740
7	5.0000	1.0000	−1.724
center	7.5000	1.0000	−0.424
Gauss point 1	6.0566	0.4226	−67.181
Gauss point 2	8.9434	0.4226	−18.150
Gauss point 3	8.9434	1.5774	18.803
Gauss point 4	6.0566	1.5774	64.831

Table 4.11: Matlab result. direct stress at each node, Second element, angle [15] degree

The following shows the direct stress contour generated in Matlab

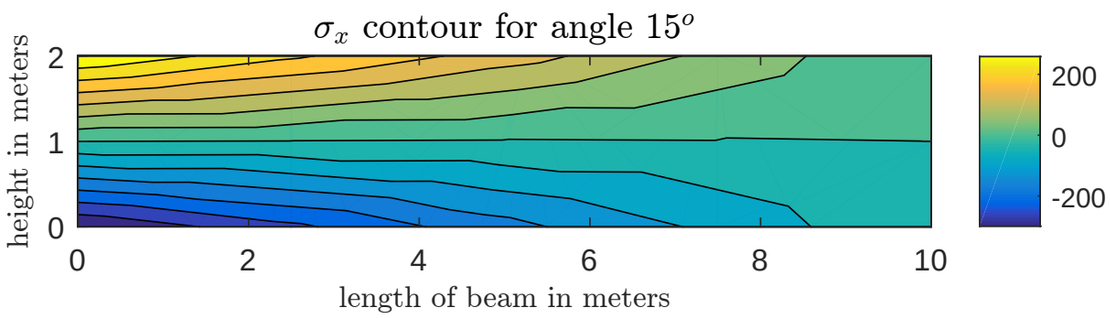


Figure 4.15: Contour of direct stress found using 2 elements using Matlab, 15 degrees

4.3.2.3 ANSYS result

```
1
2  PRINT U    NODAL SOLUTION PER NODE
3
4   ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
5
6   LOAD STEP=     1  SUBSTEP=     1
7    TIME=    1.0000      LOAD CASE=   0
8
9   THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
10
11    NODE      UX           UY           UZ          USUM
12       1   0.0000    -0.88076E-02  0.0000     0.88076E-02
13       2  0.66221E-01-0.96603E-01  0.0000     0.11712
14       3  0.11514    -0.35658      0.0000     0.37470
15       4  0.13875    -0.65316      0.0000     0.66774
16       5  0.14603     -1.0122      0.0000      1.0227
17       6   0.0000      0.0000      0.0000      0.0000
18       7  0.59649E-03-0.32606      0.0000     0.32606
19       8  0.10590E-02 -1.0189      0.0000      1.0189
20       9   0.0000    -0.45680E-03  0.0000     0.45680E-03
```

```
      10 -0.64844E-01-0.96198E-01  0.0000      0.11601
      11 -0.10854    -0.30019      0.0000      0.31921
      12 -0.13907    -0.64828      0.0000      0.66303
      13 -0.15026     -1.0256      0.0000       1.0365

 MAXIMUM ABSOLUTE VALUES
 NODE         13          13           0          13
 VALUE  -0.15026     -1.0256      0.0000      1.0365

 /OUTPUT FILE= ansys_stress_solution_15.txt
```

The following figure shows the deformation found

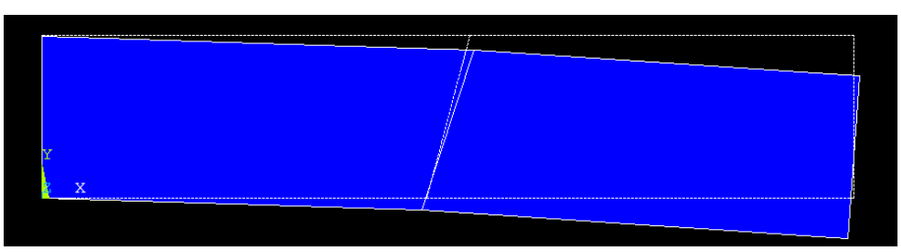


Figure 4.16: deflection found using 2 elements using ANSYS, 15 degrees

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```

 PRINT S    ELEMENT SOLUTION PER ELEMENT

 ***** POST1 ELEMENT NODAL STRESS LISTING *****

  LOAD STEP=     1  SUBSTEP=     1
   TIME=    1.0000      LOAD CASE=   0

  THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


  ELEMENT=       1        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
       9  -299.05      9.0115      0.0000     -31.003      0.0000      0.0000
      11  -148.20     -8.2859      0.0000     -10.894      0.0000      0.0000
       3   144.37     -11.096      0.0000     -11.079      0.0000      0.0000
       1   300.91      4.9358      0.0000      11.245      0.0000      0.0000

  ELEMENT=       2        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
      11  -146.18      1.3738      0.0000      4.0905      0.0000      0.0000
      13 -0.99365     -3.0444      0.0000     -16.666      0.0000      0.0000
       5   2.7463     0.33483      0.0000     -3.5896      0.0000      0.0000
       3   142.73      7.0677      0.0000     -23.081      0.0000      0.0000
```

The following shows the direct stress contour generated in ANSYS

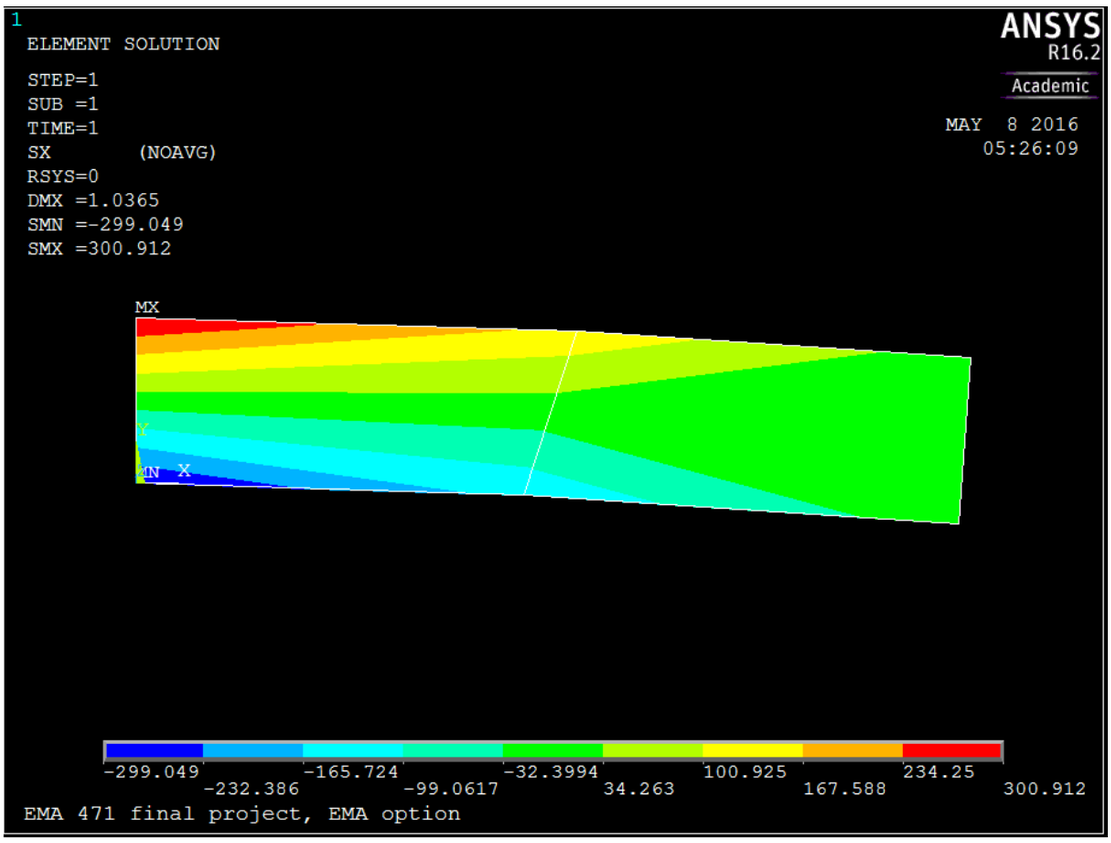


Figure 4.17: Contour of direct stress found using 2 elements using ANSYS, 15 degrees

4.3.3 30 degrees distortion

4.3.3.1 summary of result

	x (meter)	y (meter)
Matlab	−0.146118	−0.998214
ANSYS	−0.14612	−0.99821

Table 4.12: Short summary of test case 30 degrees distortion

4.3.3.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.013054
2	0.065955	−0.096602
3	0.115155	−0.384450
4	0.132363	−0.638148
5	0.137945	−0.972694
6	0.000000	0.000000
7	0.001094	−0.322637
8	0.002043	−0.985167
9	0.000000	0.003878
10	−0.063350	−0.095375
11	−0.102487	−0.265899
12	−0.132998	−0.628986
13	−0.146118	−0.998214

Table 4.13: Matlab result. nodal solutions, angle [30] degree

The following figure shows the deformation found

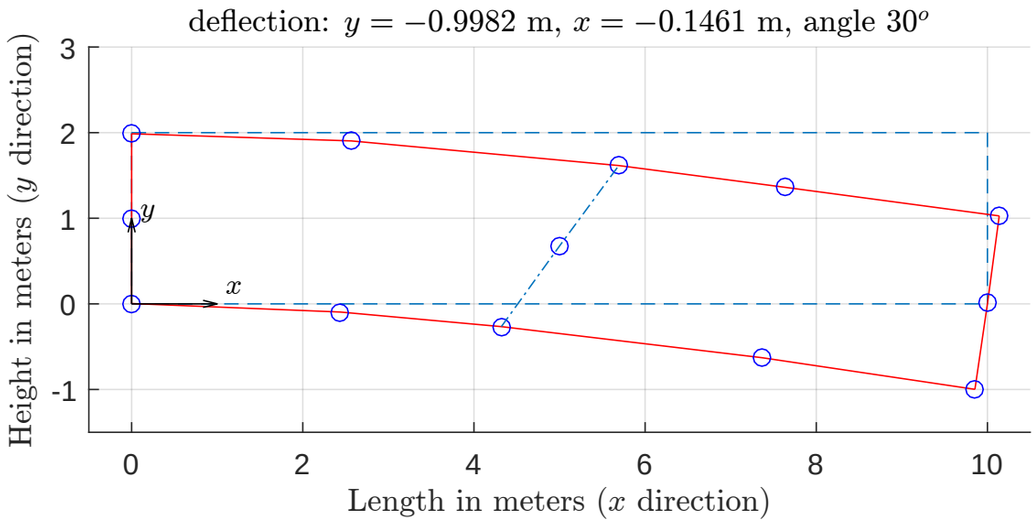


Figure 4.18: deflection found using 2 elements using Matlab, 30 degrees

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	−297.609
11	4.4226	0.0000	−138.871
3	5.5774	2.0000	128.858
1	0.0000	2.0000	302.166
10	2.5000	0.0000	−218.240
7	5.0000	1.0000	−5.007
2	2.5000	2.0000	215.512
6	0.0000	1.0000	2.279
center	2.5000	1.0000	−1.364
Gauss point 1	1.0566	0.4226	−152.145
Gauss point 2	3.9434	0.4226	−101.010
Gauss point 3	3.9434	1.5774	94.076
Gauss point 4	1.0566	1.5774	153.623

Table 4.14: Matlab result. direct stress σ_x at each node, First element, angle [30] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	4.4226	0.0000	−129.964
13	10.0000	0.0000	−6.206
5	10.0000	2.0000	9.722
3	5.5774	2.0000	123.499
12	7.5000	0.0000	−68.085
8	10.0000	1.0000	1.758
4	7.5000	2.0000	66.611
7	5.0000	1.0000	−3.232
center	7.5000	1.0000	−0.737
Gauss point 1	6.0566	0.4226	−60.856
Gauss point 2	8.9434	0.4226	−18.386
Gauss point 3	8.9434	1.5774	19.792
Gauss point 4	6.0566	1.5774	56.500

Table 4.15: Matlab result. direct stress at each node, Second element, angle [30] degree

The following shows the direct stress contour generated in Matlab

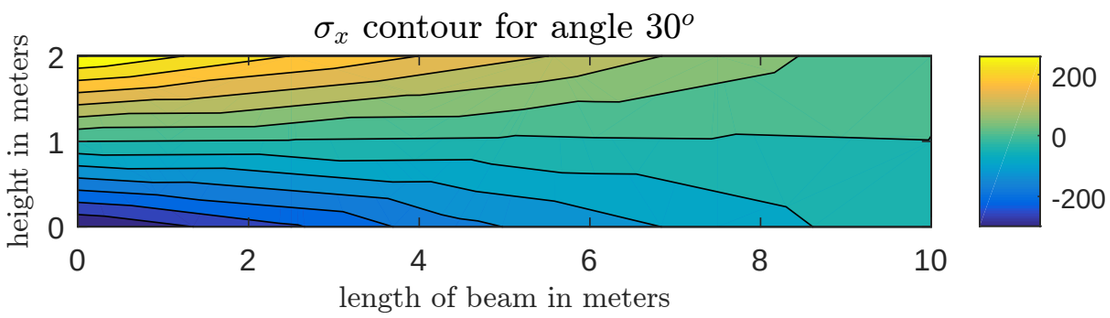


Figure 4.19: Contour of direct stress found using 2 elements using Matlab, 30 degrees

4.3.3.3 ANSYS result

```
1
2  PRINT U    NODAL SOLUTION PER NODE
3
4   ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
5
6   LOAD STEP=     1  SUBSTEP=     1
7    TIME=    1.0000      LOAD CASE=   0
8
9   THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
10
11    NODE      UX           UY           UZ          USUM
12       1   0.0000    -0.13054E-01  0.0000     0.13054E-01
13       2  0.65955E-01-0.96602E-01  0.0000     0.11697
14       3  0.11515    -0.38445      0.0000     0.40133
15       4  0.13236    -0.63815      0.0000     0.65173
16       5  0.13795    -0.97269      0.0000     0.98243
17       6   0.0000      0.0000      0.0000      0.0000
18       7  0.10941E-02-0.32264      0.0000     0.32264
19       8  0.20432E-02-0.98517      0.0000     0.98517
20       9   0.0000     0.38778E-02  0.0000     0.38778E-02
```

```
      10 -0.63350E-01-0.95375E-01  0.0000     0.11450
      11 -0.10249    -0.26590      0.0000     0.28497
      12 -0.13300    -0.62899      0.0000     0.64289
      13 -0.14612    -0.99821      0.0000      1.0089

 MAXIMUM ABSOLUTE VALUES
 NODE         13          13           0          13
 VALUE  -0.14612    -0.99821      0.0000      1.0089

 /OUTPUT FILE= ansys_stress_solution_30.txt
```

The following figure shows the deformation found

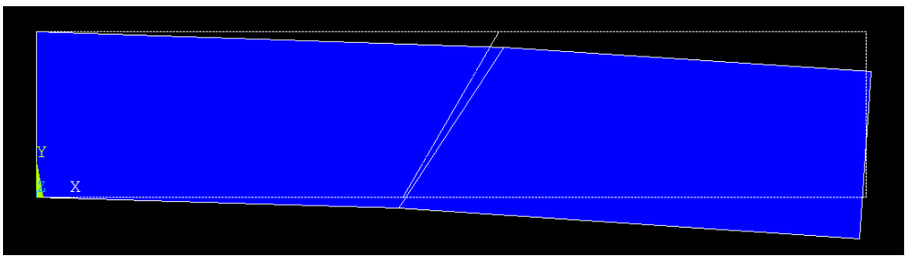


Figure 4.20: deflection found using 2 elements using ANSYS, 30 degrees

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```

 PRINT S    ELEMENT SOLUTION PER ELEMENT

 ***** POST1 ELEMENT NODAL STRESS LISTING *****

  LOAD STEP=     1  SUBSTEP=     1
   TIME=    1.0000      LOAD CASE=   0

  THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


  ELEMENT=       1        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
       9  -297.61      12.886      0.0000     -52.150      0.0000      0.0000
      11  -138.87     -13.160      0.0000     -13.331      0.0000      0.0000
       3   128.86     -25.087      0.0000     -15.021      0.0000      0.0000
       1   302.17      15.057      0.0000      33.142      0.0000      0.0000

  ELEMENT=       2        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
      11  -129.96     -3.2502      0.0000      17.425      0.0000      0.0000
      13  -6.2065    -0.12448      0.0000     -22.705      0.0000      0.0000
       5   9.7219     -5.3409      0.0000      1.4979      0.0000      0.0000
       3   123.50      21.262      0.0000     -32.751      0.0000      0.0000
```

The following shows the direct stress contour generated in ANSYS

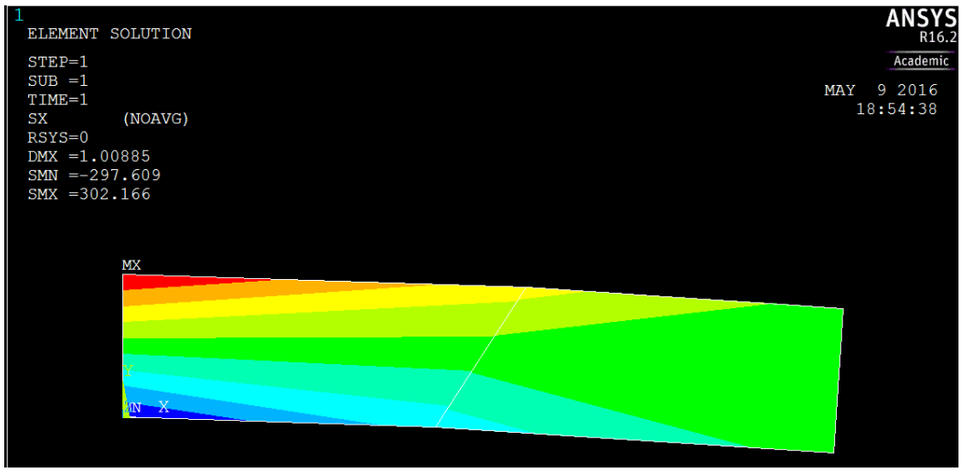


Figure 4.21: Contour of direct stress found using 2 elements using ANSYS, 30 degrees

4.3.4 45 degrees distortion

4.3.4.1 summary of result

	x (meter)	y (meter)
Matlab	−0.135417	−0.934503
ANSYS	−0.13542	−0.93450

Table 4.16: Short summary of test case 45 degrees distortion

4.3.4.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.017363
2	0.064453	−0.096794
3	0.110558	−0.415091
4	0.119874	−0.603430
5	0.124609	−0.901330
6	0.000000	0.000000
7	0.001251	−0.315104
8	0.002702	−0.916990
9	0.000000	0.008169
10	−0.061186	−0.093436
11	−0.093955	−0.220369
12	−0.120788	−0.592443
13	−0.135417	−0.934503

Table 4.17: Matlab result. nodal solutions, angle [45] degree

The following figure shows the deformation found

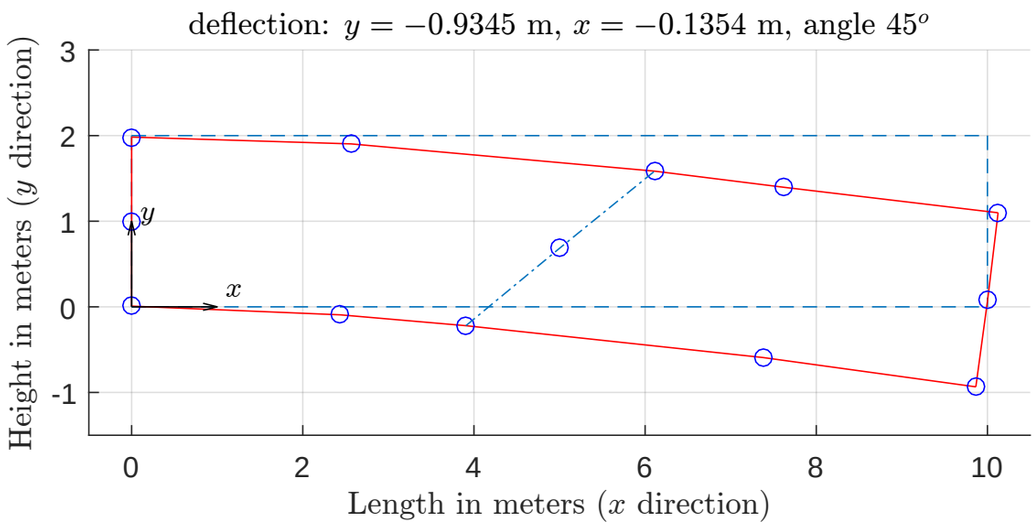


Figure 4.22: deflection found using 2 elements using Matlab, 45 degrees

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	−294.696
11	4.0000	0.0000	−120.055
3	6.0000	2.0000	96.964
1	0.0000	2.0000	304.372
10	2.5000	0.0000	−207.376
7	5.0000	1.0000	−11.546
2	2.5000	2.0000	200.668
6	0.0000	1.0000	4.838
center	2.5000	1.0000	−3.354
Gauss point 1	1.0566	0.4226	−148.254
Gauss point 2	3.9434	0.4226	−94.038
Gauss point 3	3.9434	1.5774	77.871
Gauss point 4	1.0566	1.5774	151.005

Table 4.18: Matlab result. direct stress σ_x at each node, First element, angle [45] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	4.0000	0.0000	−98.498
13	10.0000	0.0000	−17.052
5	10.0000	2.0000	22.022
3	6.0000	2.0000	91.497
12	7.5000	0.0000	−57.775
8	10.0000	1.0000	2.485
4	7.5000	2.0000	56.759
7	5.0000	1.0000	−3.501
center	7.5000	1.0000	−0.508
Gauss point 1	6.0566	0.4226	−47.876
Gauss point 2	8.9434	0.4226	−19.267
Gauss point 3	8.9434	1.5774	21.706
Gauss point 4	6.0566	1.5774	43.404

Table 4.19: Matlab result. direct stress at each node, Second element, angle [45] degree

The following shows the direct stress contour generated in Matlab

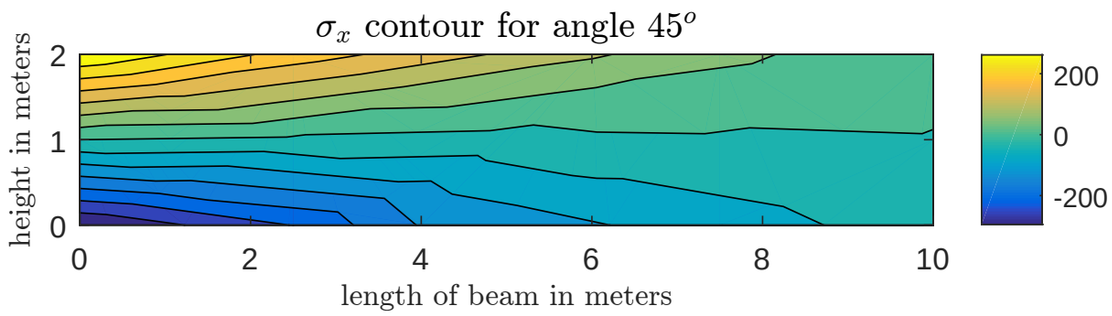


Figure 4.23: Contour of direct stress found using 2 elements using Matlab, 45 degrees

4.3.4.3 ANSYS result

```
1
2  PRINT U    NODAL SOLUTION PER NODE
3
4   ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
5
6   LOAD STEP=     1  SUBSTEP=     1
7    TIME=    1.0000      LOAD CASE=   0
8
9   THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
10
11    NODE      UX           UY           UZ          USUM
12       1   0.0000    -0.17363E-01  0.0000     0.17363E-01
13       2  0.64453E-01-0.96794E-01  0.0000     0.11629
14       3  0.11056    -0.41509      0.0000     0.42956
15       4  0.11987    -0.60343      0.0000     0.61522
16       5  0.12461    -0.90133      0.0000     0.90990
17       6   0.0000      0.0000      0.0000      0.0000
18       7  0.12507E-02-0.31510      0.0000     0.31511
19       8  0.27021E-02-0.91699      0.0000     0.91699
20       9   0.0000     0.81687E-02  0.0000     0.81687E-02
21      10 -0.61186E-01-0.93436E-01  0.0000     0.11169
22      11 -0.93955E-01-0.22037      0.0000     0.23956
23      12 -0.12079    -0.59244      0.0000     0.60463
24      13 -0.13542    -0.93450      0.0000     0.94426
25
26  MAXIMUM ABSOLUTE VALUES
27  NODE         13          13           0          13
28  VALUE  -0.13542    -0.93450      0.0000     0.94426
29
30  /OUTPUT FILE= ansys_stress_solution_45.txt
```

The following figure shows the deformation found

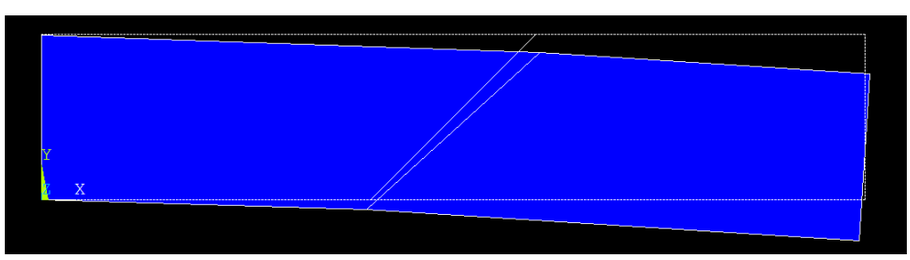


Figure 4.24: deflection found using 2 elements using ANSYS, 45 degrees

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```
1
2  PRINT S    ELEMENT SOLUTION PER ELEMENT
```

```
 ***** POST1 ELEMENT NODAL STRESS LISTING *****

  LOAD STEP=     1  SUBSTEP=     1
   TIME=    1.0000      LOAD CASE=   0

  THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


  ELEMENT=       1        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
       9  -294.70      13.722      0.0000     -72.599      0.0000      0.0000
      11  -120.06     -12.923      0.0000     -16.680      0.0000      0.0000
       3   96.964     -41.242      0.0000     -23.566      0.0000      0.0000
       1   304.37      27.298      0.0000      54.789      0.0000      0.0000

  ELEMENT=       2        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
      11  -98.498     -15.367      0.0000      25.908      0.0000      0.0000
      13  -17.052      6.6768      0.0000     -26.033      0.0000      0.0000
       5   22.022     -14.911      0.0000      2.3133      0.0000      0.0000
       3   91.497      45.921      0.0000     -32.004      0.0000      0.0000
```

The following shows the direct stress contour generated in ANSYS

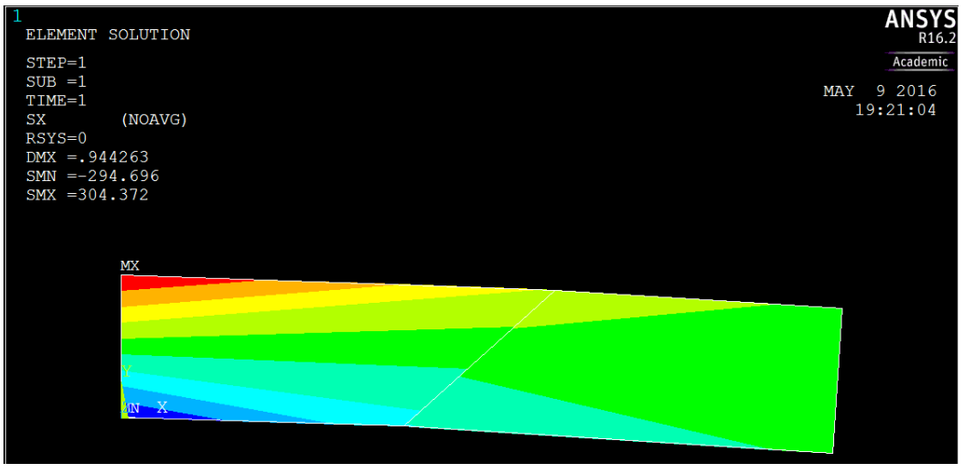


Figure 4.25: Contour of direct stress found using 2 elements using ANSYS, 45 degrees

4.3.5 50 degrees distortion

4.3.5.1 summary of result

	x (meter)	y (meter)
Matlab	−0.129803	−0.901557
ANSYS	−0.12980	−0.90156

Table 4.20: Short summary of test case 50 degrees distortion

4.3.5.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.018726
2	0.063489	−0.097037
3	0.107149	−0.426310
4	0.113960	−0.585035
5	0.118879	−0.868383
6	0.000000	0.000000
7	0.001133	−0.311076
8	0.002731	−0.883753
9	0.000000	0.009386
10	−0.060301	−0.092294
11	−0.090400	−0.200709
12	−0.114927	−0.574883
13	−0.129803	−0.901557

Table 4.21: Matlab result. nodal solutions, angle [50] degree

The following figure shows the deformation found

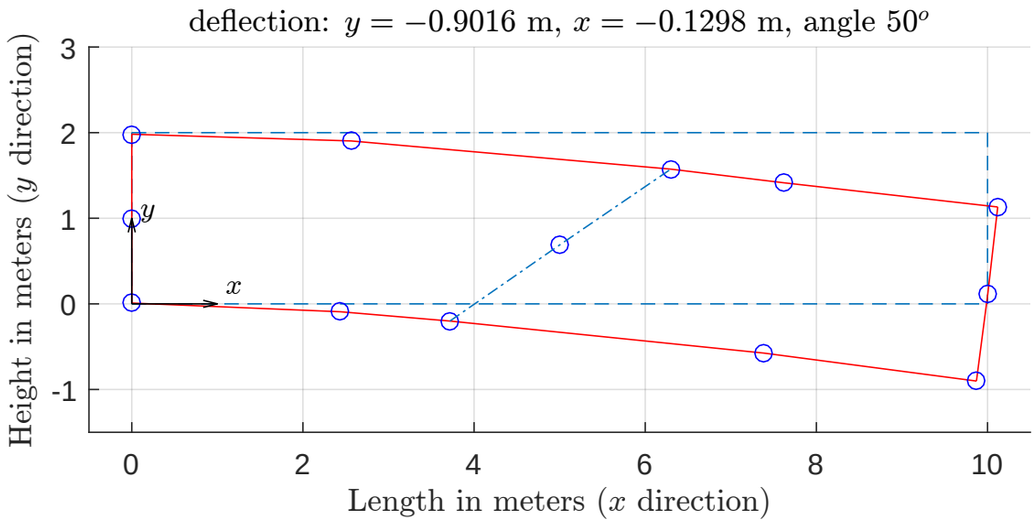


Figure 4.26: deflection found using 2 elements using Matlab, 50 degrees

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	−292.998
11	3.8082	0.0000	−111.352
3	6.1918	2.0000	80.771
1	0.0000	2.0000	305.494
10	2.5000	0.0000	−202.175
7	5.0000	1.0000	−15.290
2	2.5000	2.0000	193.133
6	0.0000	1.0000	6.248
center	2.5000	1.0000	−4.521
Gauss point 1	1.0566	0.4226	−146.283
Gauss point 2	3.9434	0.4226	−90.990
Gauss point 3	3.9434	1.5774	69.513
Gauss point 4	1.0566	1.5774	149.676

Table 4.22: Matlab result. direct stress σ_x at each node, First element, angle [50] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	3.8082	0.0000	−84.728
13	10.0000	0.0000	−22.141
5	10.0000	2.0000	27.249
3	6.1918	2.0000	79.463
12	7.5000	0.0000	−53.435
8	10.0000	1.0000	2.554
4	7.5000	2.0000	53.356
7	5.0000	1.0000	−2.633
center	7.5000	1.0000	−0.039
Gauss point 1	6.0566	0.4226	−41.931
Gauss point 2	8.9434	0.4226	−19.803
Gauss point 3	8.9434	1.5774	22.719
Gauss point 4	6.0566	1.5774	38.858

Table 4.23: Matlab result. direct stress at each node, Second element, angle [50] degree

The following shows the direct stress contour generated in Matlab

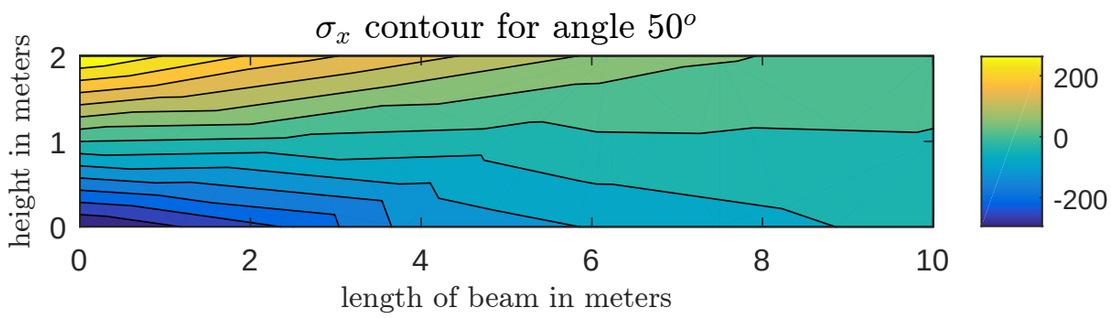


Figure 4.27: Contour of direct stress found using 2 elements using Matlab, 50 degrees

4.3.5.3 ANSYS result

```
1
2  PRINT U    NODAL SOLUTION PER NODE
3
4   ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****
5
6   LOAD STEP=     1  SUBSTEP=     1
7    TIME=    1.0000       LOAD CASE=   0
8
9   THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM
10
11    NODE       UX           UY           UZ           USUM
12       1    0.0000     -0.18726E-01   0.0000      0.18726E-01
13       2   0.63489E-01-0.97037E-01   0.0000      0.11596
14       3   0.10715     -0.42631       0.0000      0.43957
15       4   0.11396     -0.58503       0.0000      0.59603
16       5   0.11888     -0.86838       0.0000      0.87648
17       6    0.0000       0.0000       0.0000       0.0000
18       7   0.11328E-02-0.31108       0.0000      0.31108
19       8   0.27311E-02-0.88375       0.0000      0.88376
20       9    0.0000      0.93858E-02   0.0000      0.93858E-02
```

```
      10 -0.60301E-01-0.92294E-01  0.0000      0.11025
      11 -0.90400E-01-0.20071      0.0000      0.22013
      12 -0.11493    -0.57488      0.0000      0.58626
      13 -0.12980    -0.90156      0.0000      0.91085

 MAXIMUM ABSOLUTE VALUES
 NODE         13          13           0          13
 VALUE  -0.12980    -0.90156      0.0000     0.91085

 /OUTPUT FILE= ansys_stress_solution_50.txt
```

The following figure shows the deformation found

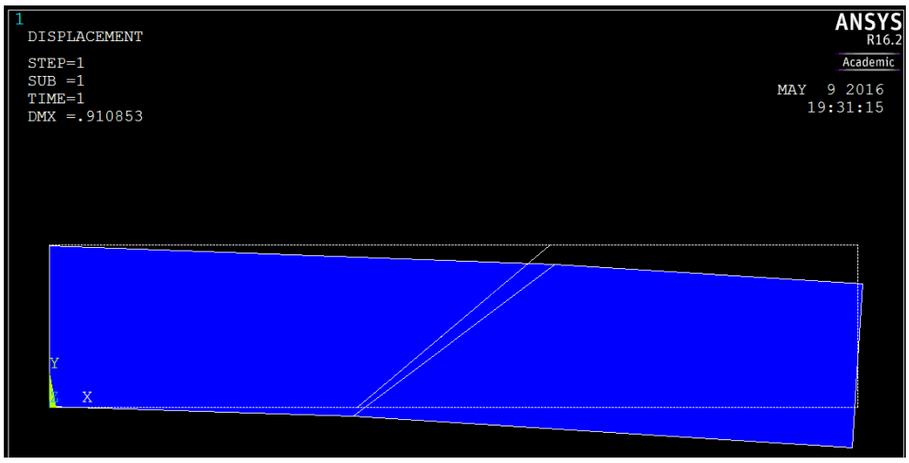


Figure 4.28: deflection found using 2 elements using ANSYS, 50 degrees

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```

 PRINT S    ELEMENT SOLUTION PER ELEMENT

 ***** POST1 ELEMENT NODAL STRESS LISTING *****

  LOAD STEP=     1  SUBSTEP=     1
   TIME=    1.0000      LOAD CASE=   0

  THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


  ELEMENT=       1        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
       9  -293.00      13.200      0.0000     -78.471      0.0000      0.0000
      11  -111.35     -11.561      0.0000     -17.821      0.0000      0.0000
       3   80.771     -46.122      0.0000     -27.702      0.0000      0.0000
       1   305.49      31.354      0.0000      61.078      0.0000      0.0000

  ELEMENT=       2        PLANE183
    NODE    SX          SY          SZ          SXY         SYZ         SXZ
      11  -84.728     -21.076      0.0000      26.004      0.0000      0.0000
      13  -22.141      9.8112      0.0000     -25.739      0.0000      0.0000
       5   27.249     -18.956      0.0000     0.41536      0.0000      0.0000
       3   79.463      57.025      0.0000     -26.399      0.0000      0.0000
```

The following shows the direct stress contour generated in ANSYS

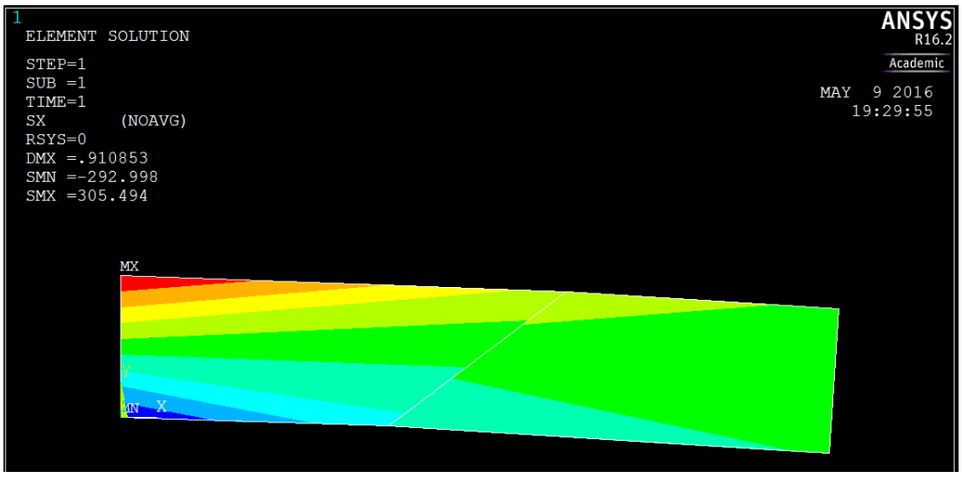


Figure 4.29: Contour of direct stress found using 2 elements using ANSYS, 50 degrees

4.3.6 55 degrees distortion

4.3.6.1 summary of result

	x (meter)	y (meter)
Matlab	−0.123001	−0.86157
ANSYS	−0.123	−0.86157

Table 4.24: Short summary of test case 55 degrees distortion

4.3.6.2 Matlab result

global node #	x (meter)	y (meter)
1	0.000000	−0.019943
2	0.062204	−0.097514
3	0.102304	−0.438312
4	0.107168	−0.562073
5	0.112704	−0.830774
6	0.000000	0.000000
7	0.000874	−0.305999
8	0.002574	−0.844631
9	0.000000	0.010255
10	−0.059355	−0.090697
11	−0.086450	−0.177473
12	−0.108133	−0.554224
13	−0.123001	−0.861570

Table 4.25: Matlab result. nodal solutions, angle [55] degree

The following figure shows the deformation found

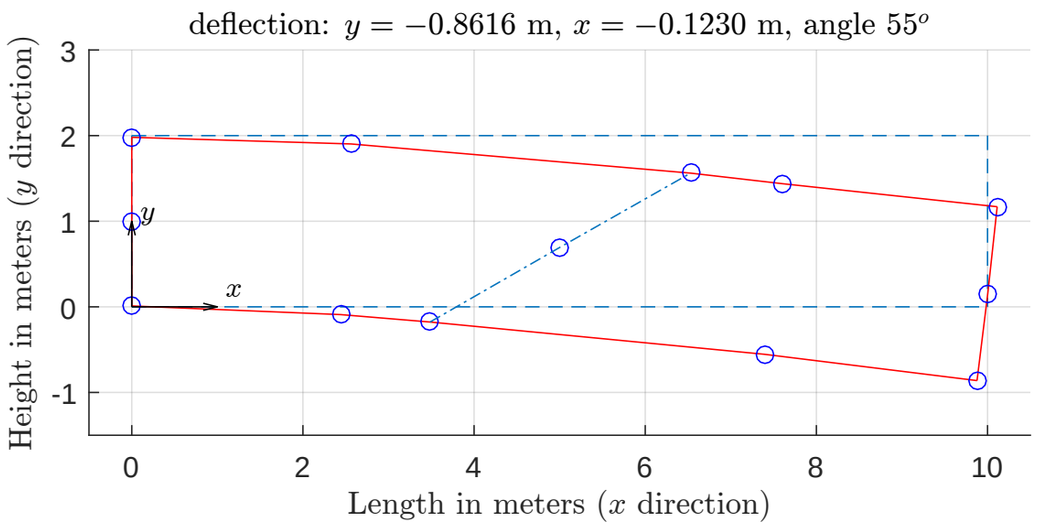


Figure 4.30: deflection found using 2 elements using Matlab, 55 degrees

The following table shows Matlab result for the direct stress σ_x found at each node for each element.

global node #	x	y	σ_x N/m$\hat{2}$
9	0.0000	0.0000	−290.606
11	3.5719	0.0000	−101.874
3	6.4281	2.0000	61.164
1	0.0000	2.0000	306.879
10	2.5000	0.0000	−196.240
7	5.0000	1.0000	−20.355
2	2.5000	2.0000	184.022
6	0.0000	1.0000	8.137
center	2.5000	1.0000	−6.109
Gauss point 1	1.0566	0.4226	−143.860
Gauss point 2	3.9434	0.4226	−87.902
Gauss point 3	3.9434	1.5774	59.234
Gauss point 4	1.0566	1.5774	148.092

Table 4.26: Matlab result. direct stress σ_x at each node, First element, angle [55] degree

global node #	x	y	σ_x N/m$\hat{2}$
11	3.5719	0.0000	−70.386
13	10.0000	0.0000	−27.809
5	10.0000	2.0000	32.546
3	6.4281	2.0000	69.339
12	7.5000	0.0000	−49.097
8	10.0000	1.0000	2.369
4	7.5000	2.0000	50.943
7	5.0000	1.0000	−0.523
center	7.5000	1.0000	0.923
Gauss point 1	6.0566	0.4226	−35.405
Gauss point 2	8.9434	0.4226	−20.508
Gauss point 3	8.9434	1.5774	24.022
Gauss point 4	6.0566	1.5774	35.581

Table 4.27: Matlab result. direct stress at each node, Second element, angle [55] degree

The following shows the direct stress contour generated in Matlab

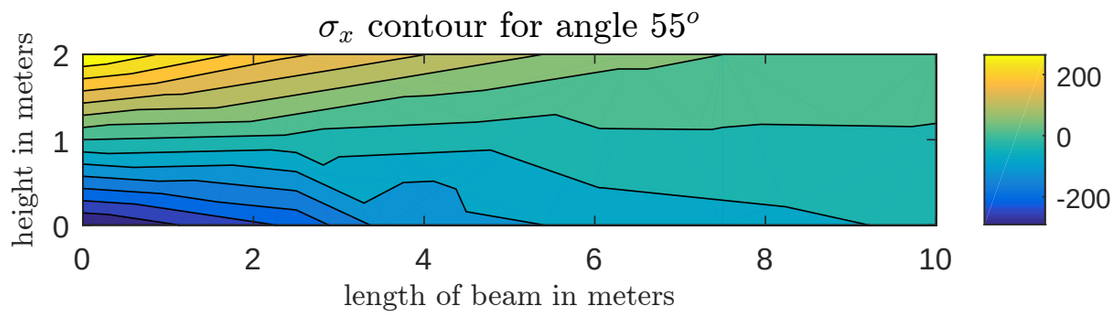


Figure 4.31: Contour of direct stress found using 2 elements using Matlab, 55 degrees

4.3.6.3 ANSYS result

```
PRINT U    NODAL SOLUTION PER NODE

 ***** POST1 NODAL DEGREE OF FREEDOM LISTING *****

 LOAD STEP=     1  SUBSTEP=     1
  TIME=    1.0000      LOAD CASE=   0

 THE FOLLOWING DEGREE OF FREEDOM RESULTS ARE IN THE GLOBAL COORDINATE SYSTEM

   NODE      UX           UY           UZ           USUM
      1   0.0000    -0.19943E-01  0.0000      0.19943E-01
      2  0.62204E-01-0.97514E-01  0.0000      0.11566
      3  0.10230    -0.43831      0.0000      0.45009
      4  0.10717    -0.56207      0.0000      0.57220
      5  0.11270    -0.83077      0.0000      0.83838
      6   0.0000      0.0000      0.0000       0.0000
      7  0.87390E-03-0.30600      0.0000      0.30600
      8  0.25744E-02-0.84463      0.0000      0.84463
      9   0.0000     0.10255E-01  0.0000      0.10255E-01
     10 -0.59355E-01-0.90697E-01  0.0000      0.10839
     11 -0.86450E-01-0.17747      0.0000      0.19741
     12 -0.10813    -0.55422      0.0000      0.56467
     13 -0.12300    -0.86157      0.0000      0.87031

MAXIMUM ABSOLUTE VALUES
NODE         13          13           0          13
VALUE  -0.12300    -0.86157      0.0000     0.87031

/OUTPUT FILE= ansys_stress_solution_55.txt
```

The following figures show the deformation found

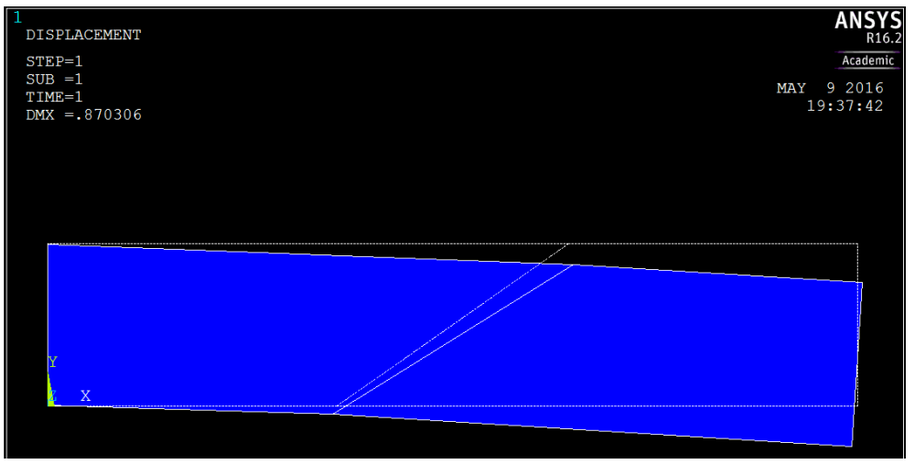


Figure 4.32: deflection found using 2 elements using ANSYS, 55 degrees

The following table shows ANSYS result for the direct stress σ_x found at each node for each element.

```
 PRINT S    ELEMENT SOLUTION PER ELEMENT

***** POST1 ELEMENT NODAL STRESS LISTING *****

 LOAD STEP=     1  SUBSTEP=     1
  TIME=    1.0000      LOAD CASE=   0

 THE FOLLOWING X,Y,Z VALUES ARE IN GLOBAL COORDINATES


 ELEMENT=       1        PLANE183
   NODE    SX          SY          SZ          SXY         SYZ         SXZ
      9  -290.61      12.453      0.0000     -82.985      0.0000      0.0000
     11  -101.87     -10.078      0.0000     -19.122      0.0000      0.0000
      3   61.164     -49.512      0.0000     -32.211      0.0000      0.0000
      1   306.88      34.681      0.0000      65.950      0.0000      0.0000

 ELEMENT=       2        PLANE183
   NODE    SX          SY          SZ          SXY         SYZ         SXZ
     11  -70.386     -27.088      0.0000      23.673      0.0000      0.0000
     13  -27.809      13.053      0.0000     -24.279      0.0000      0.0000
      5   32.546     -23.115      0.0000     -3.3748      0.0000      0.0000
      3   69.339      69.307      0.0000     -15.921      0.0000      0.0000
```

The following shows the direct stress contour generated in ANSYS

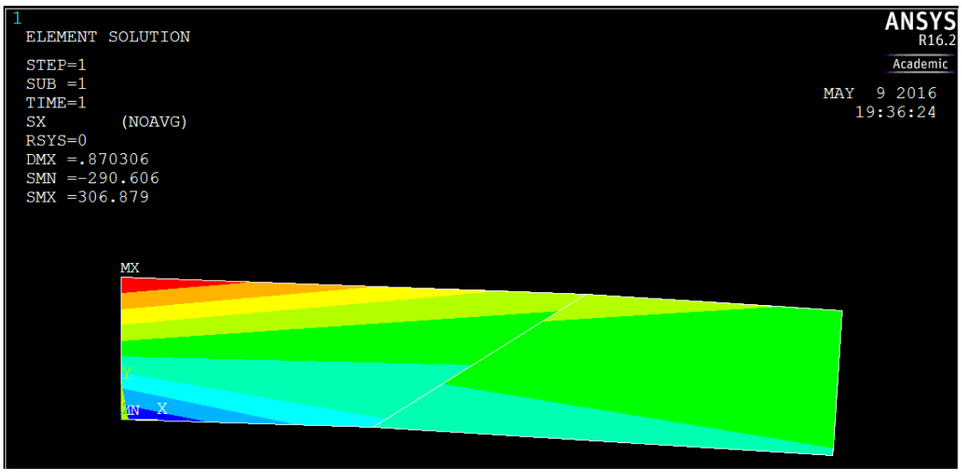


Figure 4.33: Contour of direct stress found using 2 elements using ANSYS, 55 degrees

4.4 Observations, discussion and conclusions

It is clear from the above result that the 8 node element used did not behave well at all when it became distorted.

The element used is a serendipity element[1]. These elements are known not to give good results when they distorted[2]. There are a number of distortions that an element could have, such as an edge distortion or angular distortion and others. In this report we only looked at angular distortion.

As the angular distortion increased, the result became less accurate. The inaccuracy also accelerated when the angle was above 35^0 based on the diagram generated below. At angle 55^0, the vertical deflection reported by the finite element Matlab program (and ANSYS as well) was −0.86157 meters where the theoretical result should be close to −1.03 meters. This is over 16% error. A signification error in accuracy. The following graph shows how the error in the vertical deflection changed as a function of the distortion angle.

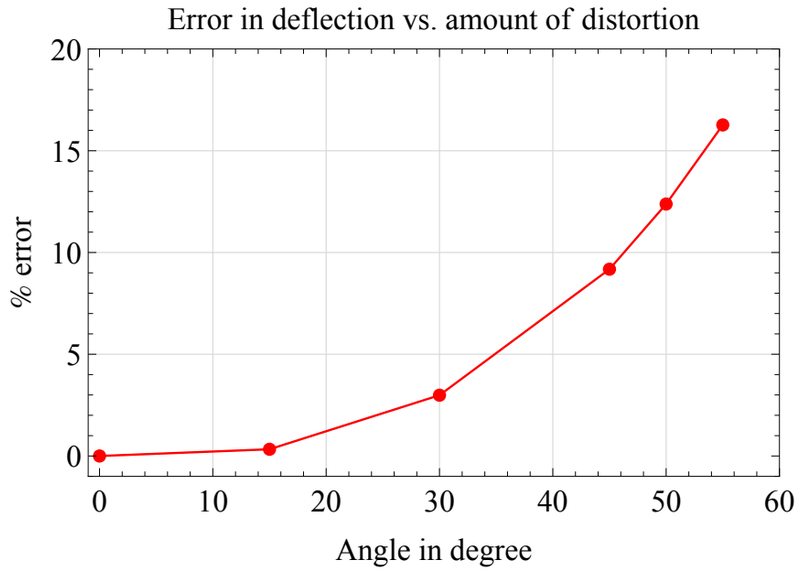


Figure 4.34: Error as function of amount of of angular element distortion

From the above one can see that to keep the error below 5%, the angular distortion should not be more than about 35^0 as the element behaves very badly after that. ANSYS will not even accept the analysis when trying angle of 60^0 and gave number of errors relating to element shape. This means this element is not suitable for modeling physical regions which are highly irregular. This element is suitable for fitting in physical region which has straight edges and mostly rectangular regions. Curved regions and other such regions would need to be modeled using other elements.

When it comes to post processing and stress recovery, which the term used for calculating stresses in the post processing stage, there are three methods that can be used. These are

1. direct method. In this method the stress is found at arbitrary points by directly evaluating $\sigma = EBu$ at the point. Where in this, the u is the nodal solution. This requires finding the Jacobian at point in question. This method is simple and works very well as long as there is no distortion.

   Once distortion is added, it produced bad result in stress values when compared to ANSYS results.

2. Method of extrapolation. This method is described in reference [1], pages 230-232. In this method, the stress at the nodes of the element (or at any other arbitrary point) is found by extrapolating the stress values found at the four Gaussian points. This works well because the stress at the Gaussian points is the most accurate since these

[1]Element has no node in the middle
[2]Reference [2]

are also the integration points used. This was the method used in this project and worked very well. Results from the Matlab implementation all agreed with ANSYS result for the direct stress at the nodes.

3. Patch Recovery. This is based on using a polynomial of same order as the shape functions and then using such polynomial on a small patch around the point on interest to find the stress. It would be interesting to compare this method in the future with the extrapolation method to see which is more accurate or easier to implement.

When making the contour plots for σ_x, one can see that the stress along the same line between the two elements is no longer smooth as the angle increases. ANSYS shows this to be smooth transition in the stress contour, but this must be because ANSYS did averaging across element boundaries.

In the Matlab implementation, No stress averaging was made between nodes across elements, hence the distortion (contour lines) is more clear in the stress contour along the line between the two elements as the following plot shows when the angle is 55^0. This shows clearly that stress across elements is not smooth and changes abruptly now.

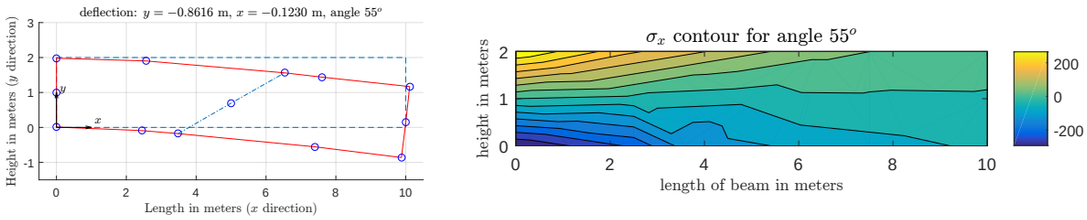


Figure 4.35: Stress contour, at 55 degrees

The more angular distortion there is, the sharper this distortion in the stress contour between the two element became. This is due to the element becoming less accurate as it distorts. Comparing the above plot to the one when the angle was zero (no distortion) one can see that in the no distortion case the stress across the elements is smooth and has same values at the nodes connecting the two elements.

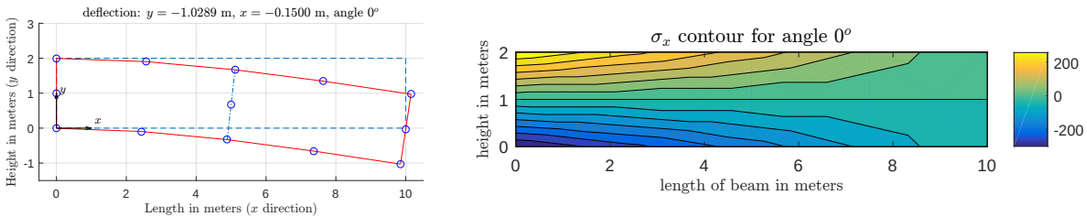


Figure 4.36: Stress contour, at zero degrees

In conclusion, it is recommended that this element be used only when there is no distortion in the geometry.

4.4.1 References

1 Concepts And Applications Of Finite Element Analysis. 4th edition. Robert D. Cook, David S. Malkus, Michael E. Plesha, Robert J. Witt. John Wiley & Sons. Inc.

2 Effect Of Element Distortions On The Performance Of Isoparametric Elements. Nam-Sua Lee, Klaus-Jurgen Bathe. Dept of Mechanical Engineering. MIT. International Journal For Numerical Methods In Engineering. Vol 36, 3553-3676 (1993)

3 ANSYS help manuals for APDL and general ANSYS use.

4.5 Appendix

4.5.1 APDL used for ANSYS

The following is the APDL script used for ANSYS analysis. ANSYS version 16.02, student version was used.

```
!APDL script to generate solution for EMA 471 final project
!to use to compare result with my Matlab Finite element program
!Nasser M. Abbasi
!ANSYS 16.02

!To read this from the APDL mechanical, simply use the
!  FILE->read input from ...
!that is all. This will process eveything and will generate
!table of stresses and deformation into text files in the default
!directory and will plot the deformed shape on the screen

/CWD,'X:\data\public_html\my_courses\univ_wisconsin_madison\spring_2016\EMA_471
   \project\my_project\ansys'

/FILNAM,EMA_471_ansys_APDL
/title, EMA 471 final project, EMA option
/prep7

!KEYOPT(1)=0 (8 node), KEYOPT(3)=0 (plane stress), KEYOPT(6)=0 (pure
   displacement)
ET,1,PLANE183,0,,0,,,0        ! QUAD 8, plain stress plain strain element
MP, ex,   1, 1.0E4   !Elastic moduli
MP, prxy, 1, 0.3     !Major Poisson's ratios
MP, nuxy, 1, 0.3     !minor Poisson's ratios, same for isotropic

!define distortion angle. Change as needed. See report.
PI=ACOS(-1)
*SET,L,10.0  ! length of beam
*SET,H,2.0   ! depth of beam
*SET,angle,0.0*PI/180.0 ! angle, set it to any value needed
shift = 0.5*H*TAN(angle)!


! DEFINE ALL NODES, 13 of them.
N, 1, 0.0             , H   ,0      ! node 1, top left corner
N, 2, 0.25*L          , H   ,0      ! node 2, etc...
N, 3, 0.5*L+shift     , H   ,0      !
N, 4, 0.75*L          , H   ,0      !
N, 5, L               , H   ,0      ! last node in top of beam, node 5
N, 6, 0.0             , 0.5*H ,0    ! node 6, middle of left edge of beam.
N, 7, 0.5*L           , 0.5*H ,0
N, 8, L               , 0.5*H ,0
N, 9, 0.0             , 0.0   ,0     ! node 9, at origin, bottom left corner
N, 10, 0.25*L         , 0.0   ,0
N, 11, 0.5*L-shift    , 0.0   ,0
N, 12, 0.75*L         , 0.0   ,0
N, 13, L              , 0.0   ,0


MAT, 1
!real, 1,2,1
EN, 1, 9,11,3,1,10,7,2,6

MAT, 1
!real, 1,2,1
EN, 2, 11,13,5,3,12,8,4,7

```

```
!set degree of freedom.
D, 9, UX, 0.0
D, 6, UX, 0.0
D, 6, UY, 0.0
D, 1, UX, 0.0

F, 8, fy, -20.0

!ERESX,NO !do this to see GAUSSIAN points stress
finish

/solu
antyp, static
solve
SAVE EMA_471_ansys_APDL
finish

/post1

/OUTPUT,ansys_nodel_solution_0,txt
PRNSOL,U,COMP


/OUTPUT,ansys_stress_solution_0,txt
PRESOL,S,COMP
/OUTPUT


!I need to find out how to send image to a file to include it
!in report, plot deformation
/REPLOT
GPLOT
PLDISP,1

!Plot stress contour
PLESOL, S,X, 0,1.0

/output

!makes  SAVE EMA_471_ansys_APDL.dbb
SAVE EMA_471_ansys_APDL
```

4.5.2 Matlab source code

The following is the listing of the m file for Matlab implementation. For making the contour plot, an external m file was used from Mathworks file exchange called `tricontf` and is included in the zip file. This function is not listed here.

To run the Matlab program, the command is `nma_project_EMA_471`

```
function nma_project_EMA_471()
%Final project Finite Elements option, EMA option.
%By Nasser M. Abbasi
%EMA 471, spring 2016, Univ. Of Wisconsin, Madison
%This files uses a Mathworks file exchange function
%which is included in the zip file and is in the same folder
%as this m file. Needed for making the contour plot.

close all; clc;

data      = PRE_PROCESSOR();    %set up data structure
solution  = SOLVE(data);        %assemble and solve KD=F

%stress calculations and print results
```

```
POST_PROCESSOR(solution,data);

end
%==========================================
function data = PRE_PROCESSOR()
%In this function we allocate the mapping tables and
%set all the problem parameters. All is saved in a struct data

%get folder name we are running from. Needed to save results
if(~isdeployed)
    data.baseFolder = fileparts(which(mfilename));
    cd(data.baseFolder);
end

%we are using 2 by 2 Gaussian rule for integration.
wt(1) = 1;           wt(2) = 1;
gs(1) = -0.57735027; gs(2)  = 0.57735027;

%allocate data parameters
data.wt = wt;
data.gs = gs;
data.num_elements = 2;
data.angle_degree = 0;  %change as needed
data.angle = data.angle_degree*pi/180;

%  global node numbering:
%
%  1     2      3       4         5
%  o-----o------o-------o---------o
%  |            |                 |
%  o 6          o7                o8
%  |            |                 |
%  o-----o------o-------o---------o
%  9     10     11      12        13
%
%
data.elem_map_nodes = [9,11,3,1,10,7,2,6;   %element (1)
                       11,13,5,3,12,8,4,7];  %element (2)

L          = 10;
H          = 2;
data.L     = L;  %meter
data.H     = H;   %meter
data.shift = (H/2)*tan(data.angle); %meter

data.global_coord_tbl = [ ...  %global coordinates of the 13 nodes
    0,          H;          %node 1, top left corner
    L/4,        H;          %node 2, etc...
    L/2+data.shift,  H;     %if angle is 90 degrees, then shift=0
    (3/4)*L,    H;
    L,          H;          %last node in top of beam, node 5
    0,          H/2;        %node 6, middle of left edge of beam.
    L/2,        H/2;
    L,          H/2;
    0,          0;          %node 9, at origin, bottom left corner
    L/4,        0;
    L/2-data.shift,  0;
    (3/4)*L,    0;
    L,          0];

data.young_module  = 10^4;
data.mu            = 0.3;
data.E0            = data.young_module/(1-data.mu^2)*...
```

```
                    [1,data.mu,0;data.mu,1,0;0,0,(1-data.mu)/2];

data.elem_map_dofs = zeros(data.num_elements ,16);
display_diagram_of_dof(data.L,data.H);

%elem_map_dofs is used to merge the element k to the global K
for i=1:data.num_elements
    for j=1:8
        data.elem_map_dofs(i,2*j-1)= 2*data.elem_map_nodes(i,j)-1;
        data.elem_map_dofs(i,2*j)  = 2*data.elem_map_nodes(i,j);
    end
end

end
%=====================================================
function solution = SOLVE(data)
%all the problem data description is now in struct data. This
%function assembles the stiffness matrix and solves KD=F
%and returns the solution

N        = size(data.global_coord_tbl,1);
k_global = zeros(2*N,2*N);

%do initial verification that shape functions adds to one.

%throws error if not verified
verify_shape_functions_sum_to_one(data.gs);
%fprintf('verified shape functions ok....\n');

for k = 1:data.num_elements
    %obtain global coordinates for the node of this element
    [x_coord,y_coord] = find_XY_coordinates(k,...
                      data.elem_map_nodes,data.global_coord_tbl);
    k_elem = zeros(16,16); %allocate local element k

    for i = 1:2  %we are using 2 by 2 Gaussian integration rule
        xi = data.gs(i);
        for j = 1:2
            eta   = data.gs(j);
            J     = get_J(xi,eta,x_coord,y_coord);
            detJ  = det(J);
            if detJ<0
                error('Internal error. negative |J| detected.');
            end
            B = get_B(xi,eta,J);  %find the strain rate matrix
            v = B'*data.E0*B;

            for ii = 1:16  %numerical integration
                for jj = 1:16
                    if(ii<=jj) %only do upper diagonal, symmetry
                        k_elem(ii,jj) = k_elem(ii,jj)+...
                            data.wt(i)*data.wt(j)*v(ii,jj)*detJ;
                    end
                end
            end

        end
    end

    %First copy the upper part of k to the lower part,  symmetric
    k_elem  = k_elem + triu(k_elem,1)';

    %Merge it to global stiffness matrix
```

```
141     k_global = merge_element_to_global(k, k_elem, ...
142                                        k_global,data.elem_map_dofs);
143 end
144 
145 %we need now to zero out rows/cols {1,11,12,17} from the
146 %above, since these correspond to the fixed boundary conditions.
147 %Make sure to keep the diagonal element. Since the boundary
148 %conditions are zero at these, we do not have to patch the F
149 %vector on the RHS as normally we would. put a 1 in the diagonal
150 
151 fix   = [1,11,12,17];  %these are the fixed DOF to zero out
152 
153 %----- COMMENTED OUT -----------------------
154 %below is one method to fix. It gives same as the next method.
155 %But the next method below this is simpler since it keeps the
156 %same sizes of data kept for reference
157 
158 %k_global(fix,:)=[];
159 %k_global(:,fix)=[];
160 %r_global       = zeros(22,1);
161 %r_global(end)  = -20;  % Newton
162 %d              = k_global\r_global;  %solve for displacement
163 
164 %----- END COMMENTED --------------------
165 
166 %second method to fix K
167 
168 for i = 1:length(fix)
169     tmp                      =  k_global(fix(i),fix(i));
170     k_global(fix(i),:)       = 0;
171     k_global(:,fix(i))       = 0;
172     k_global(fix(i),fix(i)) = tmp; %same effect as setting to 1.
173 end
174 
175 r_global      = zeros(26,1);
176 r_global(16)  = -20;  % Newton
177 solution      = k_global\r_global;  %solve for displacement
178 
179 %finished. Now write the solution to file to include it in report
180 write_nodal_solution_to_file(solution,data.baseFolder,...
181                                                 data.angle_degree);
182 
183 figure(); %show the global stiffness matrix structure using spy
184 spy(k_global);
185 title('global stiffness matrix after fixing for B.C.',...
186     'interpreter','Latex','Fontsize',11);
187 %print(gcf, '-dpdf', '-r600','../images/spy.pdf');
188 
189 end
190 %=================================================
191 function POST_PROCESSOR(solution,data)
192 %This is final stage. We find stress and make contour plots
193 %and draw the deflection shape of the beam
194 
195 draw_deflection(solution,data);
196 generate_stress_diagram(solution,data);
197 end
198 %=======================================
199 function write_nodal_solution_to_file(d,baseFolder,angle)
200 %d is the nodal solution vector. 26 by 1.
201 %write the X,Y solution to text file to use in report
202 
203 d              = reshape(d,2,13)';
```

```
fileName       = [baseFolder sprintf(...
                       '/data/deformation_matlab_%d.tex',angle)];
fileName       = strrep(fileName, '/', filesep);
[fileID0,errMsg] = fopen(fileName,'w');

if fileID0<0
    fprintf('Error opening %s\n, the message is [%s\n]',...
                                                fileName,errMsg);
    error(errMsg);
end

fprintf(fileID0,'\\begin{table}[!htbp]\n');
fprintf(fileID0,'\\centering\n');
fprintf(fileID0,'\\captionsetup{width=.8\\textwidth}\n');
fprintf(fileID0,'\\begin{tabular}{|l|l|l|l|}\\hline\n');
fprintf(fileID0,'global node \\#& $x$ (meter)& $y$ (meter)\\\\\\hline\n');
for i=1:size(d,1)
    fprintf(fileID0,'$%d$ & $%7.6f$& $%7.6f$\\\\ \n',i,d(i,1),d(i,2));
end
fprintf(fileID0,'\\hline\n\\end{tabular}\n');
fprintf(fileID0,...
  '\\caption{Matlab result. nodal solutions, angle [$%d$] degree}\n',angle);
fprintf(fileID0,'\\end{table}\n');
fprintf(fileID0,'\\FloatBarrier\n');

fclose(fileID0);

end
%=========================================
function generate_stress_diagram(solution,data)

%find the stress at node of each element
element_stress_1 = stress_calculation(solution,data,1);
element_stress_2 = stress_calculation(solution,data,2);

%make one diagram of the overall stress contour across the beam
stress_diagram(data,element_stress_1,element_stress_2);
end
%=============================================
function element_stress = stress_calculation(solution,data,...
                                                element_number)

%first calculate stress at the 4 Gaussian points for
%element in order to use for extrapolation. These are ordered
%anticlock wise.

%used to store stress at the 4 Gaussian points
gauss_stress = zeros(4,1);

%these are the r,s coordinates of the Gaussian element inside
%the element itself used for extrapolation. See report for
%more details. These are ordered anticlock wise, with one in
%the center. So there are 9 of them. Also, the Gaussian stress
%is added as well. So we end up with 9+4=13 total stress points
%for each element. This should be enough to make nice contuor with

z  = sqrt(3);
r  = [-z,z,z,-z,0,z,0,-z];
s  = [-z,-z,z,z,-z,0,z,0];

%these are the natural coordinates in xi,eta space used
%to calculate the stress at Gaussian points, using the full 8
%shape functions
```

```
z   =  1/sqrt(3);
xi  = [-z,z,z,-z];
eta = [-z,-z,z,z];

L  = data.L;
H  = data.H;

%this is used to find global coordinates of all stress points, for
%contour plot this is center of element in global space
if element_number == 1
    X0=L/4;
    Y0=H/2;
else
    X0=(3/4)*L;
    Y0=H/2;
end

%find element nodal coordinates in global space
%this only has nodes. We will add the center and the gaussian
%points also later to make contour plot
[x_coord,y_coord] = find_XY_coordinates(element_number,...
                     data.elem_map_nodes,data.global_coord_tbl);

%Now make matrix to store all stress result in for element 1.
%We are finding stress at 13 points. 8 for nodes, one for center,
%and the 4 Gaussian points. we need 4 columns. First two are
%the x,y in global space of the point, and the stress. The
%first column is just the point ID, for tracking. Not used for
%plotting.

element_stress = zeros(13,4);

global_node_numbers = data.elem_map_nodes(element_number,:);
U   = solution(data.elem_map_dofs(element_number,:));
for i = 1:length(xi)
    J    = get_J( xi(i), eta(i), x_coord, y_coord); %jacobian
    B    = get_B( xi(i), eta(i), J); %strain rate matrix
    %find actual strain from displacements at nodes
    strain  = B  * U;
    stress  = data.E0 * strain;
    gauss_stress(i) = stress(1); %use direct stress only
end

%Now do the extrapolation. See report
for i=1:length(r)
    element_stress(i,4)=0;
    element_stress(i,1)=global_node_numbers(i);
    element_stress(i,2)=x_coord(i);
    element_stress(i,3)=y_coord(i);
    for j=1:4 %extrapolation
        switch j
            case 1,
                f = (1/4)*(1-r(i))*(1-s(i));
            case 2,
                f = (1/4)*(1+r(i))*(1-s(i));
            case 3,
                f = (1/4)*(1+r(i))*(1+s(i));
            case 4,
                f = (1/4)*(1-r(i))*(1+s(i));
        end
        element_stress(i,4) = element_stress(i,4)+ f * ...
                                              gauss_stress(j);
    end
```

```
330 end
331
332 %now add the center and the 4 gaussian points we found before.
333 %These come after the nodes. This is in order to improve the
334 %contour plot by having more points.
335 element_stress(9,4) = 0;
336 element_stress(9,1) = -1;  %we do not have a global node number
337                            %for this. this is just a place holder
338 element_stress(9,2)=X0;
339 element_stress(9,3)=Y0;
340 for j=1:4 %extrapolation
341   element_stress(9,4)= element_stress(9,4)+ 1/4 * gauss_stress(j);
342 end
343
344 %now add the acutal Gaussian stress found. This make it up to
345 %13 points first find the global coordinates of the element
346 %gaussian points
347 z = 1/sqrt(3);
348 gauss_global_coordinates=[X0-z*(1/4)*L,Y0-z*(1/2)*H;...
349     X0+z*(1/4)*L,Y0-z*(1/2)*H;...
350     X0+z*(1/4)*L,Y0+z*(1/2)*H;...
351     X0-z*(1/4)*L,Y0+z*(1/2)*H];
352
353 for i=1:length(gauss_global_coordinates)
354     element_stress(9+i,4)=gauss_stress(i);
355     element_stress(9+i,1)=-1; %place holder
356     element_stress(9+i,2)=gauss_global_coordinates(i,1);
357     element_stress(9+i,3)=gauss_global_coordinates(i,2);
358 end
359
360 end
361 %=======================================
362 function stress_diagram(data,element_stress_1,element_stress_2)
363
364 %we are done!  Now we can make contour of first element stress
365 %This uses tricontf, which is a mathworks file exchange file
366 %since Matlab does not have such a function build in
367 figure;
368
369 % I commented out the stress average last minute. I think it is
370 %better NOT to do stress averging across elements, in order
371 %to more clearly see the difference. I left the code here for
372 %reference in case need to use it later
373
374 %---------- COMMENTED OUT --------------------
375 %now do stress avergaing on the nodes that are between
376 %element one and two. These nodes have global node
377 %numbers of 2,7,11 which correspond to local nodes
378 % 2,6,3 for first element and nodes 1,8,4 for second element.
379
380 %element_stress_1(2,4) = (element_stress_1(2,4)+element_stress_2(1,4))/2;
381 %element_stress_1(6,4) = (element_stress_1(6,4)+element_stress_2(8,4))/2;
382 %element_stress_1(3,4) = (element_stress_1(3,4)+element_stress_2(4,4))/2;
383
384 %now that we averaged the stress, remove these entry from the second
385 %element before merging, since it is duplicate
386 %element_stress_2 = element_stress_2([2:3,5:7,9:end],:);
387
388 %---------- END COMMENTED OUT ----------------
389
390 x = [element_stress_1(:,2);element_stress_2(:,2)];
391 y = [element_stress_1(:,3);element_stress_2(:,3)];
392 z = [element_stress_1(:,4);element_stress_2(:,4)];
```

```
M = delaunay(x,y);
max_stress = max(z);
min_stress = min(z);
range_of_stress = linspace(min_stress,max_stress,15);

%this below uses mathworks file exchange function.
%It is in the same folder
[~,h]=tricontf(x,y,M,z,range_of_stress,'-k');
%set(h,'edgecolor','none');
axis equal tight;
%hold on;
%[~,h]=tricont(x,y,M,z,range_of_stress,'-k');
colorbar;

title(sprintf('$\\sigma_x$ contour for angle $%d^o$',...
                                        data.angle_degree),...
                            'Fontsize',12,'interpreter','Latex');

xlabel('length of beam in meters','Fontsize',10,'interpreter','Latex');
ylabel('height in meters','interpreter','Latex','Fontsize',10);

%uncomment to write the plot
%print(gcf, '-dpdf', '-r600',...
%    sprintf('../images/stress_matlab_%d.pdf',data.angle_degree));

write_the_stress_table(data,element_stress_1,element_stress_2);

end
%=========================================
function write_the_stress_table(data,element_stress_1,...
                                                element_stress_2)

fileName         = [data.baseFolder sprintf(...
                                '/data/stress_matlab_%d.tex',...
                   data.angle_degree)];
fileName         = strrep(fileName, '/', filesep);
[fileID0,errMsg] = fopen(fileName,'w');

if fileID0<0
    fprintf('Error opening %s\n, the message is [%s\n]',...
                                                fileName,errMsg);
    error(errMsg);
end

fprintf(fileID0,'\\begin{table}[!htbp]\n');
fprintf(fileID0,'\\centering\n');
fprintf(fileID0,'\\begin{minipage}{0.49\\textwidth}\n');
fprintf(fileID0,'\\centering\n');
fprintf(fileID0,'\\captionsetup{width=.95\\textwidth}\n');
fprintf(fileID0,'\\begin{tabular}{|l|l|l|l|}\\hline\n');
fprintf(fileID0,['global node \\# & $x$ & $y$ & $\\sigma_x$',...
                 '{\\footnotesize N/m^2} \\\\\\hline\n']);
for i=1:size(element_stress_1,1)
    if i==9
        fprintf(fileID0,...
               'center & $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n',...
               element_stress_1(i,2),element_stress_1(i,3),...
               element_stress_1(i,4));
    elseif i==10
        fprintf(fileID0,['{\\footnotesize Gauss point 1}&',...
                         '$%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_1(i,2),element_stress_1(i,3),...
                                        element_stress_1(i,4));
```

```
    elseif i==11
        fprintf(fileID0,['{\\footnotesize Gauss point 2}&',...
                      '$%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_1(i,2),element_stress_1(i,3),...
                                              element_stress_1(i,4));
    elseif i==12
        fprintf(fileID0,['{\\footnotesize Gauss point 3}&',...
                       '$%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_1(i,2),element_stress_1(i,3),...
            element_stress_1(i,4));
    elseif i==13
        fprintf(fileID0,['{\\footnotesize Gauss point 4}&',...
                       '$%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_1(i,2),element_stress_1(i,3),...
            element_stress_1(i,4));
    else
        fprintf(fileID0,'$%d$ & $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n',...
            element_stress_1(i,1),element_stress_1(i,2),...
            element_stress_1(i,3),...
            element_stress_1(i,4));
    end
end
fprintf(fileID0,'\\hline\n\\end{tabular}\n');
fprintf(fileID0,...
    ['\\caption{Matlab result. direct stress $\\sigma_x$ at',...
     ' each node, First element, angle [$%d$] degree}\n'],...
    data.angle_degree);
fprintf(fileID0,'\\end{minipage}\n');
fprintf(fileID0,'\\hfill\n');

fprintf(fileID0,'\\begin{minipage}{0.49\\textwidth}\n');
fprintf(fileID0,'\\centering\n');
fprintf(fileID0,'\\captionsetup{width=.95\\textwidth}\n');

fprintf(fileID0,'\\begin{tabular}{|l|l|l|l|}\\hline\n');
fprintf(fileID0,['global node \\# & $x$ & $y$ & $\\sigma_x$',...
                      ' {\\footnotesize N/m^2} \\\\\\hline\n']);
for i=1:size(element_stress_2,1)
    if i==9
        fprintf(fileID0,'center & $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n',...
           element_stress_2(i,2),element_stress_2(i,3),...
           element_stress_2(i,4));
    elseif i==10
        fprintf(fileID0,['{\\footnotesize Gauss point 1}&',...
                      ' $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_2(i,2),element_stress_2(i,3),...
            element_stress_2(i,4));
    elseif i==11
        fprintf(fileID0,['{\\footnotesize Gauss point 2}&',...
                      ' $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_2(i,2),element_stress_2(i,3),...
            element_stress_2(i,4));
    elseif i==12
        fprintf(fileID0,['{\\footnotesize Gauss point 3}&',...
                       '$%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_2(i,2),element_stress_2(i,3),...
            element_stress_2(i,4));
    elseif i==13
        fprintf(fileID0,['{\\footnotesize Gauss point 4}&',...
                      ' $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n'],...
            element_stress_2(i,2),element_stress_2(i,3),...
            element_stress_2(i,4));
    else
```

```
519         fprintf(fileID0,'$%d$ & $%4.4f$ & $%4.4f$ & $%5.3f$ \\\\ \n',...
520             element_stress_2(i,1),element_stress_2(i,2),...
521             element_stress_2(i,3),...
522             element_stress_2(i,4));
523     end
524 end
525 fprintf(fileID0,'\\hline\n\\end{tabular}\n');
526 
527 fprintf(fileID0,...
528     ['\\caption{Matlab result. direct stress at each node,',...
529      ' Second element, angle [$%d$] degree}\n'],...
530       data.angle_degree);
531 fprintf(fileID0,'\\end{minipage}\n');
532 fprintf(fileID0,'\\end{table}\n');
533 fprintf(fileID0,'\\FloatBarrier\n');
534 
535 fclose(fileID0);
536 
537 end
538 %=====================================
539 function B = get_B(xi,eta,J)
540 %calculate the B matrix
541 
542 B1 = [1,0,0,0;
543       0,0,0,1;
544       0,1,1,0];
545 
546 gamma = 1/det(J) * [J(2,2) ,-J(1,2);
547                     -J(2,1) ,J(1,1)];
548 
549 B2    = [gamma, zeros(2,2);
550          zeros(2,2),gamma];
551 
552 Z  = zeros(2,1);
553 
554 N1 = [dfdxi(1,xi,eta);
555       dfdeta(1,xi,eta)];
556 
557 N2 = [dfdxi(2,xi,eta);
558       dfdeta(2,xi,eta)];
559 
560 N3 = [dfdxi(3,xi,eta);
561       dfdeta(3,xi,eta)];
562 
563 N4 = [dfdxi(4,xi,eta);
564       dfdeta(4,xi,eta)];
565 
566 N5 = [dfdxi(5,xi,eta);
567       dfdeta(5,xi,eta)];
568 
569 N6 = [dfdxi(6,xi,eta);
570       dfdeta(6,xi,eta)];
571 
572 N7 = [dfdxi(7,xi,eta);
573       dfdeta(7,xi,eta)];
574 
575 N8 = [dfdxi(8,xi,eta);
576       dfdeta(8,xi,eta)];
577 
578 B3 = [N1,Z,  N2,Z,  N3,Z, N4,Z, N5,Z, N6,Z, N7,Z, N8,Z;
579       Z,N1,  Z,N2,  Z,N3, Z,N4, Z,N5, Z,N6, Z,N7, Z,N8];
580 
581 B  = B1*B2*B3;
```

```
end
%=======================================
function [x_coord,y_coord] = find_XY_coordinates(k,...
                                                 elem_map_node,...
                                                 global_coord_tbl)
%This function returns the x,y global coordinates of
%specific element nodes

N       = size(elem_map_node,2);  %number of nodes in element
x_coord = zeros(N,1);  %x for this element
y_coord = zeros(N,1);  %y for this element

%collect this element node coordinates, go over each node
%of this element and find its global x,y coordinates
for i = 1:N
  global_node_of_this_element_node = elem_map_node(k,i);
  x_coord(i) = global_coord_tbl(global_node_of_this_element_node,1);
  y_coord(i) = global_coord_tbl(global_node_of_this_element_node,2);
end
end
%====================================
function J = get_J(xi,eta,x_coord,y_coord)

J = [ ddxi(xi,eta,x_coord),  ddxi(xi,eta,y_coord);
      ddeta(xi,eta,x_coord), ddeta(xi,eta,y_coord)];

end
%==================================
function v = ddxi(xi,eta,c)
%find dx/d(xi) or dy/d(xi)
v = c(1)*(1/4*(1-eta^2)+(1/2)*(1-eta)*xi+(eta-1)/4)...
    +c(2)*(1/4*(eta^2-1)+(1/2)*(1-eta)*xi+(1-eta)/4)...
    +c(3)*(1/4*(eta^2-1)+(1/2)*(eta+1)*xi+(eta+1)/4)...
    +c(4)*(1/4*(1-eta^2)+(1/2)*(eta+1)*xi+(-eta-1)/4)...
    -c(5)*(1-eta)*xi...
    +c(6)*(1/2)*(1-eta^2)...
    -c(7)*xi*(eta+1)...
    -c(8)*(1/2)*(1-eta^2);
end
%==================================
function v = ddeta(xi,eta,c)
%find dx/d(eta) or dy/d(eta)
v = c(1)*(1/2*eta*(1-xi)+(1/4)*(1-xi^2)+(xi-1)/4)...
    +c(2)*((1/2)*(1+xi)*eta+1/4*(1-xi^2)+(-xi-1)/4)...
    +c(3)*((1/2)*eta*(1+xi)+1/4*(xi^2-1)+(xi+1)/4)...
    +c(4)*((1/2)*(1-xi)*eta+1/4*(xi^2-1)+(1-xi)/4)...
    -c(5)*(1/2)*(1-xi^2)...
    -c(6)*eta*(xi+1)...
    +c(7)*(1/2)*(1-xi^2)...
    -c(8)*eta*(1-xi);
end
%==================================
function v = dfdxi(shape_function_number,xi,eta)
%evaluate shape function at some x,y point
switch shape_function_number
    case 1
        v=(1/4)*(1-eta^2)+(1/2)*(1-eta)*xi+(eta-1)/4;
    case 2
        v=(1/4)*(eta^2-1)+(1/2)*(1-eta)*xi+(1-eta)/4;
    case 3
        v=(1/4)*(eta^2-1)+(1/2)*(eta+1)*xi+(eta+1)/4;
    case 4
        v=(1/4)*(1-eta^2)+(1/2)*(eta+1)*xi+(-eta-1)/4;
```

```
    case 5
        v=-(1-eta)*xi;
    case 6
        v=(1/2)*(1-eta^2);
    case 7
        v=-(eta+1)*xi;
    case 8
        v=(1/2)*(eta^2-1);
end

end
%==================================
function v = dfdeta(shape_function_number,xi,eta)
switch shape_function_number
    case 1
        v=(1/2)*eta*(1-xi)+(1/4)*(1-xi^2)+(xi-1)/4;
    case 2
        v=(1/2)*eta*(xi+1)+(1/4)*(1-xi^2)+(-xi-1)/4;
    case 3
        v=(1/2)*eta*(xi+1)+(1/4)*(xi^2-1)+(xi+1)/4;
    case 4
        v=(1/2)*eta*(1-xi)+(1/4)*(xi^2-1)+(-xi+1)/4;
    case 5
        v=(1/2)*(xi^2-1);
    case 6
        v=-eta*(xi+1);
    case 7
        v=(1/2)*(1-xi^2);
    case 8
        v=-eta*(1-xi);
end

end
%============================
function k_global = merge_element_to_global(k,k_elem,...
                                        k_global,elem_map_dofs)

%assemble local element k to global K

for i=1:16
    for j =1:16
       global_i =  elem_map_dofs(k,i);
       global_j =  elem_map_dofs(k,j);
       k_global(global_i,global_j) = k_global(global_i,global_j)...
                                                    + k_elem(i,j);
    end
end

end
%=====================
function draw_deflection(solution,data)
%This function is called at the end, after we have solved
%the problem using finite elements and have nodal (x,y)
%deformations. It plots the deformed shape against the undeformed
%original shape.

u = reshape(solution,2,13)';
L = data.L;
H = data.H;

figure();
%This is before demformation shape

```

```
top_line    = [0,L/4,L/2+data.shift,(3/4)*L,L;H,H,H,H,H];
left_line   = [0,0,0;H,H/2,0];
right_line  = [L,L,L;H,H/2,0];
bottom_line = [0,L/4,L/2-data.shift,(3/4)*L,L;0,0,0,0,0];

line(top_line(1,:),top_line(2,:),'LineStyle','--'); hold on;
line(left_line(1,:),left_line(2,:),'LineStyle','--');
line(right_line(1,:),right_line(2,:),'LineStyle','--');
line(bottom_line(1,:),bottom_line(2,:),'LineStyle','--');

%this is after adding deformation
line(top_line(1,:)+u(1:5,1)',top_line(2,:)+u(1:5,2)',...
                                                'Color','red');
line(left_line(1,:)+u([1 6 9],1)',left_line(2,:)+u([1 6 9],2)',...
                                                  'Color','red');
line(right_line(1,:)+u([5 8 13],1)',right_line(2,:)+...
                                  u([5 8 13],2)','Color','red');
line(bottom_line(1,:)+u(9:13,1)',bottom_line(2,:)+...
                                      u(9:13,2)','Color','red');

%There are the nodes. Draw nodes on top line
for i=1:size(top_line,2)
    plot(top_line(1,i)+u(i,1),top_line(2,i)+u(i,2),'bo');
end

%draw nodes on left
idx=5;
plot(left_line(1,2)+u(1+idx,1),left_line(2,2)+u(1+idx,2),'bo');

%draw nodes on right
idx=7;
plot(right_line(1,2)+u(1+idx,1),right_line(2,2)+u(1+idx,2),'bo');

%draw nodes in middle
idx=6;
plot(L/2+u(1+idx,1),H/2+u(1+idx,2),'bo');

%Draw nodes on bottom line
idx=8;
for i=1:size(bottom_line,2)
    plot(bottom_line(1,i)+u(i+idx,1),bottom_line(2,i)+...
                                                u(i+idx,2),'bo');
end

%draw dashed line between elements
line([bottom_line(1,3)+u(11,1),...
      L/2+u(7,1),...
      L/2+data.shift+u(3,1)],...
      [bottom_line(2,3)+u(11,2),...
      H/2+u(7,2),...
      H+u(3,2)],...
      'LineStyle','-.');

%put title, x,y arrows at (0,0) and save the image to include in
%document/report at end

title(sprintf('deflection: $y=%3.4f$ m, $x=%3.4f$ m, angle $%d^o$',...
    u(end,2),u(end,1),data.angle_degree),...
    'interpreter','Latex','Fontsize',11);

xlabel('Length in meters ($x$ direction)','interpreter',...
                                          'Latex','Fontsize',11);
ylabel('Height in meters ($y$ direction)','interpreter',...
```

```
                                        'Latex','Fontsize',11);

quiver(0,0,1,0,1,'MaxHeadSize',0.5,'Color','black');
text(1.1,.2,'$x$','interpreter','Latex');
quiver(0,0,0,1,1,'MaxHeadSize',0.5,'Color','black');
text(0.1,1.1,'$y$','interpreter','Latex');
axis equal;
xlim([-0.5,L+0.5]);
ylim([-1.5,H+1]);
grid;

%print(gcf, '-dpdf', '-r600',...
%sprintf('../images/deflection_matlab_%d.pdf',data.angle_degree));

end

%================================
function v = get_shape_function(shape_function_number,xi,eta)

switch shape_function_number
    case 1
        v=-(1/4)*(1-eta^2)*(1-xi)-(1/4)*(1-eta)*(1-xi^2)+...
                                            (1/4)*(1-eta)*(1-xi);
    case 2
        v=-(1/4)*(1-eta^2)*(1+xi)-(1/4)*(1-eta)*...
                                    (1-xi^2)+(1/4)*(1-eta)*(1+xi);
    case 3
        v=-(1/4)*(1-eta^2)*(1+xi)-(1/4)*(1+eta)*...
                                    (1-xi^2)+(1/4)*(1+eta)*(1+xi);
    case 4
        v=-(1/4)*(1-eta^2)*(1-xi)-(1/4)*(1+eta)*...
                                    (1-xi^2)+(1/4)*(1+eta)*(1-xi);
    case 5
        v=(1/2)*(1-eta)*(1-xi^2);
    case 6
        v=(1/2)*(1-eta^2)*(1+xi);
    case 7
        v=(1/2)*(1+eta)*(1-xi^2);
    case 8
        v=(1/2)*(1-eta^2)*(1-xi);
end

end
%=======================================
function verify_shape_functions_sum_to_one(gs)
for i=1:2
    xi = gs(i);
    for j=1:2
        eta   = gs(j);
        chk_1 = 0;
        for k=1:8 %sum all shape functions at this Gaussian point
            chk_1 = chk_1 + get_shape_function(k,xi,eta);
        end
        if chk_1 ~= 1
            error(['Internal error. sum of shape functions',...
                   ' not 1 at $\xi=%3.3f,\eta=%3.3f'],...
                xi,eta);
        end
    end
end
end
%================================
function display_diagram_of_dof(L,H)
```

```
834
835 figure();
836 top_line    = [0,L/4,L/2,(3/4)*L,L;
837     H,H,H,H,H];
838 left_line   = [0,0,0;
839     H,H/2,0];
840 right_line  = [L,L,L;
841     H,H/2,0];
842 bottom_line = [0,L/4,L/2,(3/4)*L,L;
843     0,0,0,0,0];
844 middle_line = [L/2;H/2];
845
846 line(top_line(1,:),top_line(2,:)); hold on;
847 %axis equal;
848 xlim([-2,L+2]);
849 ylim([-.75,H+1]);
850 line(left_line(1,:),left_line(2,:));
851 line(right_line(1,:),right_line(2,:));
852 line(bottom_line(1,:),bottom_line(2,:));
853
854 k=0;
855 node_number=0;
856 for i=1:size(top_line,2)
857     x=top_line(1,i); y=top_line(2,i);
858     plot(x,y,'ro');
859     quiver(x,y,0.5,0,1,'MaxHeadSize',2,'Color','black');
860     k=k+1;
861     node_number=node_number+1;
862     text(x+.1,y-.1,sprintf('$%d$',node_number),...
863                                         'interpreter','Latex',...
864         'Fontsize',11,'Color','red');
865     text(x+.3,y+.1,sprintf('$%d$',k),'interpreter',...
866                                           'Latex','Fontsize',11);
867     quiver(x,y,0,0.5,1,'MaxHeadSize',2,'Color','black');
868     k=k+1;
869     text(x,y+.6,sprintf('$%d$',k),'interpreter',...
870                                           'Latex','Fontsize',11);
871 end
872 for i=1:size(left_line,2)
873     x=left_line(1,i); y=left_line(2,i);
874     plot(x,y,'ro');
875     quiver(x,y,0.5,0,1,'MaxHeadSize',2,'Color','black');
876     quiver(x,y,0,0.5,1,'MaxHeadSize',2,'Color','black');
877 end
878 k=k+1;
879 node_number=node_number+1;
880 text(.1,H/2-.1,sprintf('$%d$',node_number),'interpreter','Latex',...
881     'Fontsize',11,'Color','red');
882 text(.5,H/2,sprintf('%d',k),'interpreter','Latex','Fontsize',11);
883 k=k+1;
884 text(x,H/2+.5,sprintf('%d',k),'interpreter','Latex','Fontsize',11);
885
886 for i=1:size(middle_line,2)
887     x=middle_line(1,i); y=middle_line(2,i);
888     plot(x,y,'ro');
889     quiver(x,y,0.5,0,1,'MaxHeadSize',2,'Color','black');
890     k=k+1;
891     node_number=node_number+1;
892     text(x+.1,H/2-.15,sprintf('$%d$',node_number),...
893                                         'interpreter','Latex',...
894         'Fontsize',11,'Color','red');
895     text(x+.5,H/2,sprintf('%d',k),'interpreter',...
896                                           'Latex','Fontsize',11);
```

```
    quiver(x,y,0,0.5,1,'MaxHeadSize',2,'Color','black');
    k=k+1;
    text(x+.1,H/2+.4,sprintf('%d',k),'interpreter',...
                                          'Latex','Fontsize',11);
end

for i=1:size(right_line,2)
    x=right_line(1,i); y=right_line(2,i);
    plot(x,y,'ro');
    quiver(x,y,0.5,0,1,'MaxHeadSize',2,'Color','black');
    quiver(x,y,0,0.5,1,'MaxHeadSize',2,'Color','black');
end
k=k+1;
node_number=node_number+1;
text(L+.1,H/2-.1,sprintf('$%d$',node_number),...
          'interpreter','Latex','Fontsize',11,'Color','red');
text(L+.5,H/2,sprintf('%d',k),'interpreter','Latex','Fontsize',11);
k=k+1;
text(L,H/2+.5,sprintf('%d',k),'interpreter','Latex','Fontsize',11);


for i=1:size(bottom_line,2)
    x=bottom_line(1,i); y=bottom_line(2,i);
    plot(x,y,'ro');
    quiver(x,y,0.5,0,1,'MaxHeadSize',2,'Color','black');
    k=k+1;
    node_number=node_number+1;
    text(x-.1,-.2,sprintf('$%d$',node_number),...
                                       'interpreter','Latex',...
        'Fontsize',11,'Color','red');
    text(x+.4,.1,sprintf('%d',k));
    quiver(x,y,0,0.5,1,'MaxHeadSize',2,'Color','black');
    k=k+1;
    text(x+.1,.4,sprintf('%d',k));
end

title({'global D.O.F. numbering used (black letters).',...
    'with associated global node numering (in red letters)'},...
    'interpreter','Latex','Fontsize',11);

%
%print(gcf, '-dpdf', '-r600', sprintf('../images/dof.pdf'));
%
end
```