
HW3 EMA 471 Intermediate Problem Solving for Engineers

SPRING 2016
ENGINEERING MECHANICS DEPARTMENT
UNIVERSITY OF WISCONSIN, MADISON

INSTRUCTOR: PROFESSOR ROBERT J. WITT

BY

NASSER M. ABBASI

DECEMBER 30, 2019

Contents

0.1	Problem 1	3
0.1.1	Results	6
0.1.2	Source code	8
0.2	Problem 2	13
0.2.1	Power method	19
0.2.2	Results	20
0.2.3	Source code	24
0.3	Problem 3	32
0.3.1	Power method	35
0.3.2	Results	36
0.3.3	Source code	39

List of Tables

1	First eigenvalue	6
2	second eigenvalue	7
3	Third eigenvalue	7
4	First (smallest) eigenvalue λ_1	20
5	second eigenvalue	21
6	Third eigenvalue	22

List of Figures

1	problem 1 description	3
2	grid numbering used in problem 1	4
3	First mode shape	6
4	Second mode shape	7
5	Third mode shape	8
6	problem 2 description	14
7	problem 2 geometry	15
8	Grid used for problem 2	16
9	First mode shape, each solver on separate plot	20
10	First mode shape, combined plot	21
11	Second mode shape	22
12	Third mode shape	23
13	problem 3 description	32
14	problem 3 geometry	33
15	Grid used for problem 3	34
16	mode shape result from the three numerical method on one plot	37
17	zoom in showing the result of the three methods	38
18	mode shape result from the three numerical method	39

0.1 Problem 1

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

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

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

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

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

Figure 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

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -2x_2 - \lambda^2 x_1 \end{aligned}$$

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

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

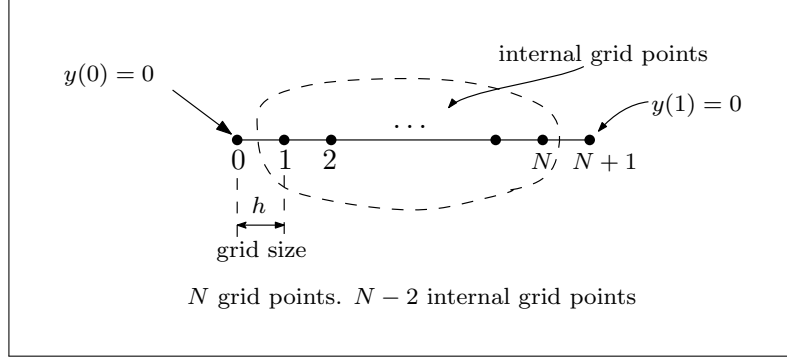


Figure 2: grid numbering used in problem 1

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

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

Hence

$$\begin{aligned} \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + 2\frac{y_{i+1} - y_{i-1}}{2h} + \lambda^2 y_i &= 0 \\ y_{i+1} - 2y_i + y_{i-1} + h(y_{i+1} - y_{i-1}) + h^2 \lambda^2 y_i &= 0 \\ y_{i-1}(1-h) + y_i(h^2 \lambda^2 - 2) + y_{i+1}(1+h) &= 0 \end{aligned}$$

At node $i = 1$,

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

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

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

At node $i = 2$

$$\begin{aligned} y_1(1-h) + y_2(h^2 \lambda^2 - 2) + y_3(1+h) &= 0 \\ y_1(1-h) - 2y_2 + y_3(1+h) &= -y_2(h^2 \lambda^2) \end{aligned}$$

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

$$\begin{aligned} y_{N-1}(1-h) + y_N(h^2 \lambda^2 - 2) + y_{N+1}(1+h) &= 0 \\ y_{N-1}(1-h) - 2y_N &= -y_N(h^2 \lambda^2) - y_{N+1}(1+h) \end{aligned}$$

At $i = N - 1$

$$\begin{aligned} y_{N-2}(1-h) + y_{N-1}(h^2 \lambda^2 - 2) + y_N(1+h) &= 0 \\ y_{N-2}(1-h) - 2y_{N-1} + y_N(1+h) &= -y_{N-1}(h^2 \lambda^2) \end{aligned}$$

Hence the structure is

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

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

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

$Ay = \alpha By$

Where $\alpha = \lambda^2$ and $B = -h^2I$. 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.

0.1.1 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 1: First eigenvalue

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

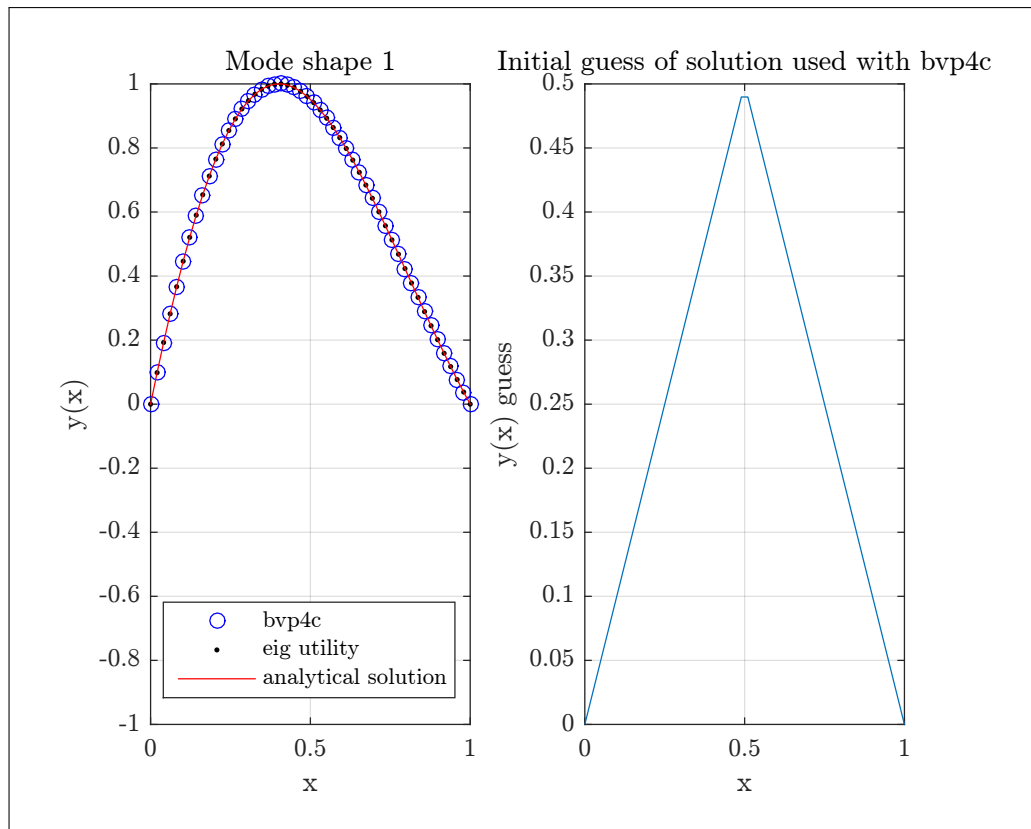


Figure 3: First mode shape

2. Second mode shape

Table 2: second eigenvalue

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

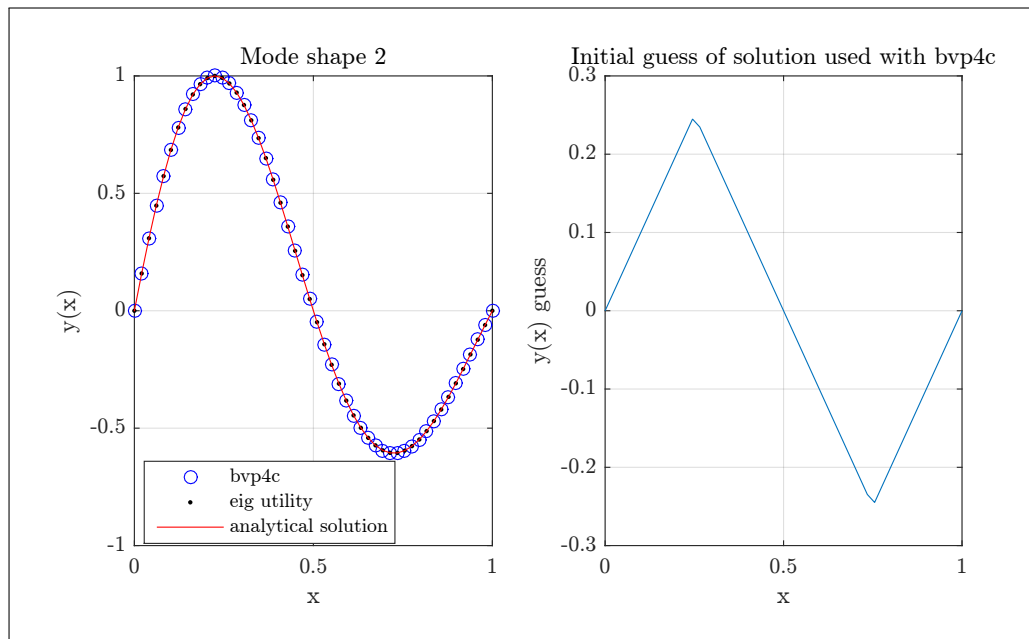


Figure 4: Second mode shape

3. Third mode shape

Table 3: Third eigenvalue

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

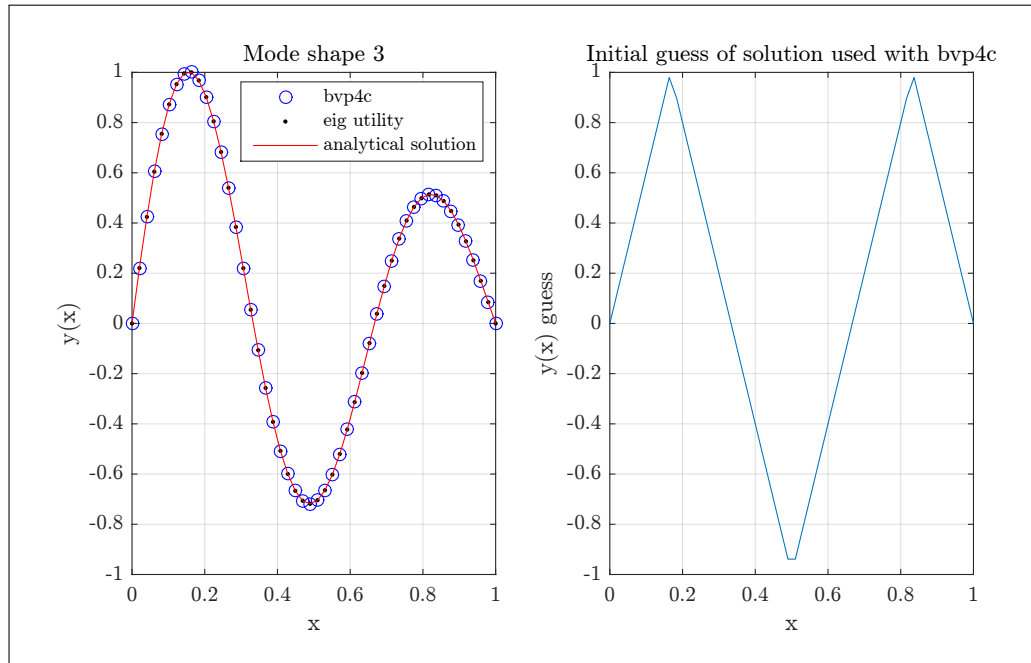


Figure 5: Third mode shape

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
```

0.1.2 Source code

```
1 function nma_HW3_EMA_471_problem_1()
2 % Solves  $y''+2 y' + \lambda^2 y = 0$ 
3 %
4 % see HW3, EMA 471
5 % by Nasser M. Abbasi
6 %
7 clc; close all;
8 initialize(); %GUI
9 N = 50; %number of grid points. Smaller will also work.
10 x = linspace(0,1,N); %all grid used is based on this one same grid.
11
12 guess_lambda_for_bvp4c = [3,6,9];%guess eigenvalue for bvp4c only
```



```

13
14 %look at first 3 mode shapes (one more than asked for, to verify)
15 for mode_shape = 1:3
16     make_test(mode_shape, x, guess_lambda_for_bvp4c(mode_shape), N);
17 end
18
19 end
20 %=====
21 function make_test(mode_shape_number, x, guess_lambda, N)
22
23 y_bvp4c    = get_y_bvp4c(x, guess_lambda, mode_shape_number);
24 y_eig      = get_eigenvector_matlab_eig(x, N, mode_shape_number);
25 y_analytic = get_y_analytic(x, mode_shape_number);
26
27 %done. Plot all mode shapes
28 plot_result(x, y_bvp4c, y_eig, y_analytic, mode_shape_number);
29 end
30 %=====
31 %This is the bvp4c solver only
32 function y_bvp4c = get_y_bvp4c(x, guess_lambda, mode_shape_number)
33
34 initial_solution = bvpinit(x,@set_initial_mode_shape,guess_lambda);
35 y_bvp4c          = bvp4c(@rhs, @bc, initial_solution);
36 value            = y_bvp4c.parameters;
37
38 fprintf('\n*****\n');
39 fprintf(['running mode %d. Eigenvalue, obtained', ...
40         'with bvp4c, is %9.7f.\n'],...
41         mode_shape_number, value)
42
43 y_bvp4c = deval(y_bvp4c,x);    %interpolate on our own grid
44 y_bvp4c = y_bvp4c(1,:);
45 y_bvp4c = y_bvp4c/max(y_bvp4c); %normalize
46
47 %-----
48 % internal function
49 % This defines the initial guess for the eigenvector
50 % the fundamental mode shape is a sawtooth
51 function solinit = set_initial_mode_shape(x)
52     switch mode_shape_number
53     case 1
54         if x <= 0.5
55             f = x;
56             fp = 1;
57         else
58             f = 1 - x;
59             fp = -1;

```

```

60         end
61     case 2
62         if x <= 0.25
63             f = x;
64             fp = 1;
65         elseif x > 0.25 && x <= 0.75
66             f = 0.5 - x;
67             fp = -1;
68         else
69             f = x - 1;
70             fp = 1;
71         end
72     case 3
73         h = 1/6;
74         if x<=h
75             f=1/h*x;
76             fp=1/h;
77         elseif x>h&&x<=3*h
78             f=2-x/h;
79             fp=-1/h;
80         elseif x>3*h&&x<(5*h)
81             f=(-4+1/h*x);
82             fp=1/h;
83         elseif x>5*h
84             f=(6-x/h);
85             fp=-1/h;
86         end
87     end
88     solinit = [ f ; fp ];
89 end
90 %-----
91 % internal function. sets up the RHS of the
92 %state space for bvp4c
93 % similar to ode45 RHS
94 function f = rhs(~,x,lam)
95
96     x1 = x(2);
97     x2 = -2*x(2)-lam^2*x(1);
98     f = [ x1
99           x2];
100 end
101 %-----
102 % Internal function. sets up the boundary
103 % conditions vector. Must have ~
104 % above in third arg
105 function res = bc(ya,yb,~)
106     res = [ ya(1)

```

```

107         yb(1)
108         ya(2)-1
109         ];
110     end
111 end
112 %=====
113 %This is the solver using Matlab eig
114 function y_eig = get_eigenvector_matlab_eig(x, N, mode_shape)
115
116 h = x(2)-x(1); % find grid spacing to set up A for eig() use
117 A = setup_A_matrix(h,N-2);
118 B = -eye(N-2)*h^2;
119
120 [eig_vec,eig_values] = eig(A,B); %eigenvalue/vector from matlab
121 eig_values           = diag(eig_values);
122 sorted_eig_values   = sort(eig_values); %sort them
123
124 %now match the original position of the eigenvalue with its
125 %corresponding eigenvector. Hence find the index of
126 %correct eigenvalue so use to index to eigenvector
127 found_eig_vector = eig_vec(:,eig_values == ...
128                         sorted_eig_values(mode_shape));
129
130 %Set the sign correctly
131 if found_eig_vector(1) > 0
132     y_eig = [0 ; found_eig_vector ; 0];
133 else
134     y_eig = [0 ; -found_eig_vector ; 0];
135 end
136
137 y_eig = y_eig/max(y_eig); %normalize
138
139 fprintf('eigenvalue from eig is %9.7f\n',...
140         sqrt(sorted_eig_values(mode_shape)));
141 %-----
142 %Internal function, sets up the A matrix for use
143 %for the eig() method
144 function A = setup_A_matrix(h,N)
145     A = zeros(N);
146     A(1,1) = -2;
147     A(1,2) = 1+h;
148     for i = 2:N-1
149         A(i,i-1:i+1) = [1-h,-2,1+h];
150     end
151     A(N,N) = -2;
152     A(N,N-1) = 1-h;
153 end

```

```

154 end
155 %=====
156 function y_analytic = get_y_analytic(x, mode_shape)
157 % This is the known analytical solution. From problem statement
158
159 y_analytic = exp(-x) .* sin(mode_shape*pi*x);
160 y_analytic = y_analytic / max(y_analytic); %normalize
161 end
162
163 %=====
164 %This function just plots the eigenshapes found from all solvers
165 function plot_result(x, y_bvp4c, y_eig, y_analytic,mode_shape)
166 figure();
167 subplot(1,2,1);
168 plot(x,y_bvp4c,'bo',...
169      x,y_eig,'k.',...
170      x,y_analytic,'r')
171 axis([0 1 -1 1]);
172 title(sprintf('Mode shape %d',mode_shape));
173 xlabel('x')
174 ylabel('y(x)')
175 legend('bvp4c','eig utility','analytical solution',...
176        'Location','southwest')
177 grid;
178 %set(gca,'TickLabelInterpreter','Latex','fontsize',8);
179
180 subplot(1,2,2);
181 initial_mode_shape = set_initial_mode_shape_plot(x, mode_shape);
182 plot(x,initial_mode_shape);
183 grid;
184 title('Initial guess of solution used with bvp4c');
185 xlabel('x'); ylabel('y(x) guess');
186 %set(gca,'TickLabelInterpreter','Latex','fontsize',8);
187
188 %-----
189 %Internal function. To display guess mode shape for plotting
190 function f = set_initial_mode_shape_plot(x, mode_shape)
191     % Internal function.
192     % plots the initial mode shape guess used.
193     %
194     switch mode_shape
195     case 1
196         f = x.*(x<=0.5)+(1-x).*(x>0.5);
197     case 2
198         f = x.*(x<=0.25)+(0.5-x).*(x>0.25&x<=0.75)+...
199             (x-1).*(x>0.75);
200     case 3

```

```
201         h = 1/6;
202         f = 1/h*x.*(x<=h)+(2-x/h).*(x>h&x<=3*h)+...
203             (-4+1/h*x).*(x>3*h&x<(5*h))+(6-x/h).*(x>5*h);
204     end
205 end
206 end
207 %=====
208 function initialize()
209     reset(0);
210     set(groot, 'defaulttextinterpreter', 'Latex');
211     set(groot, 'defaultAxesTickLabelInterpreter', 'Latex');
212     set(groot, 'defaultLegendInterpreter', 'Latex');
213 end
```

0.2 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:

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

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:

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

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:

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

Figure 6: problem 2 description

The geometry of the problem is as follows

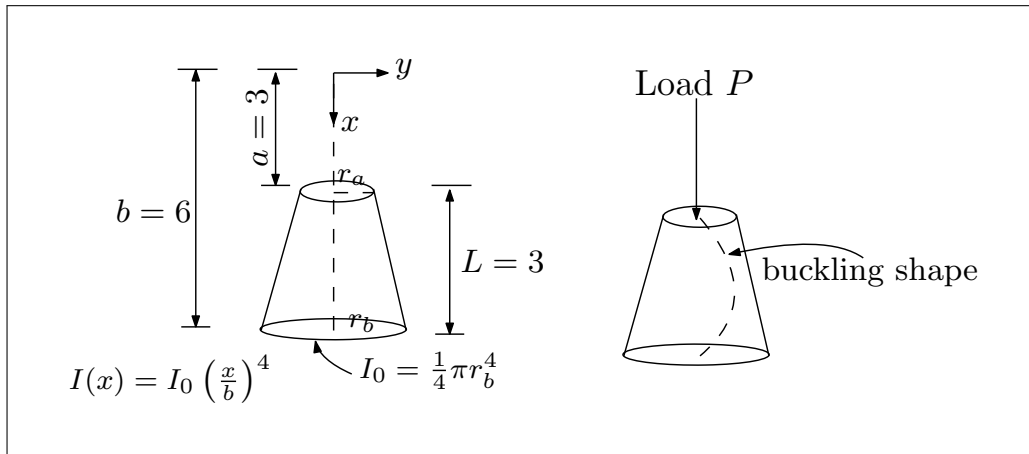


Figure 7: problem 2 geometry

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 $1 \leq z \leq 2$. 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 be discretized first. The following shows the grid used

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

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

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

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

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

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

$$Ay = \alpha By$$

Where $\alpha = \lambda^2$ and $B = -h^2 I$. 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

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

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

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

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 $E = 10^9 \text{ Pa}$, $I_0 = \frac{1}{4}\pi (r_b)^4$ where $r_b = 0.2$ meter are

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

Hence

$$P_1 = 3.4451 \times 10^5 \text{ N}$$

$$P_2 = 4 (3.4451 \times 10^5) = 1.3781 \times 10^6 \text{ N}$$

$$P_3 = 9 (3.4451 \times 10^5) = 3.1006 \times 10^6 \text{ N}$$

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.

0.2.1 Power method

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

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

Hence

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

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

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

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

$$Ay = \lambda^2 y$$

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

0.2.2 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 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

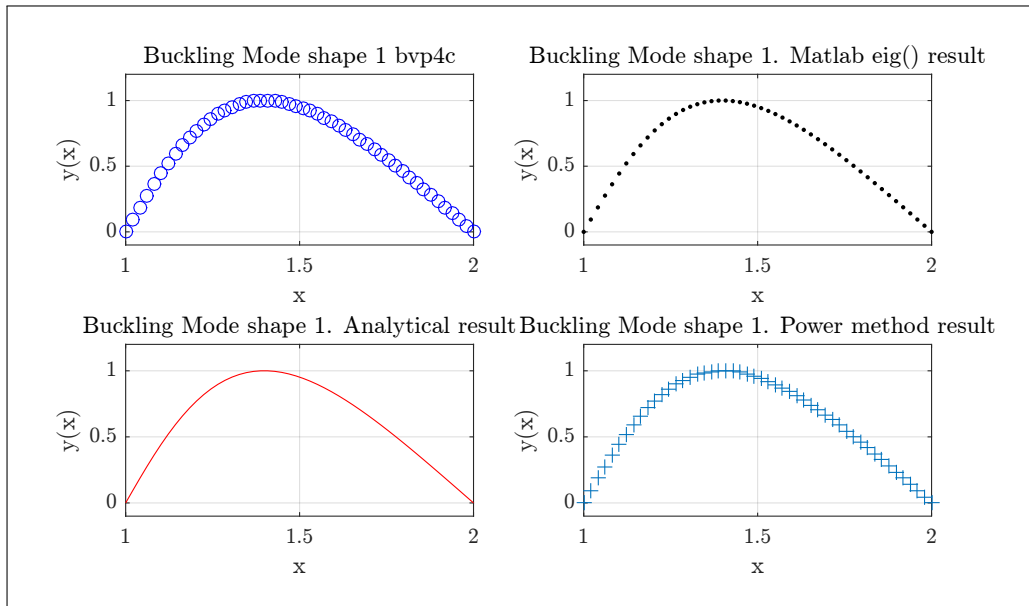


Figure 9: First mode shape, each solver on separate plot

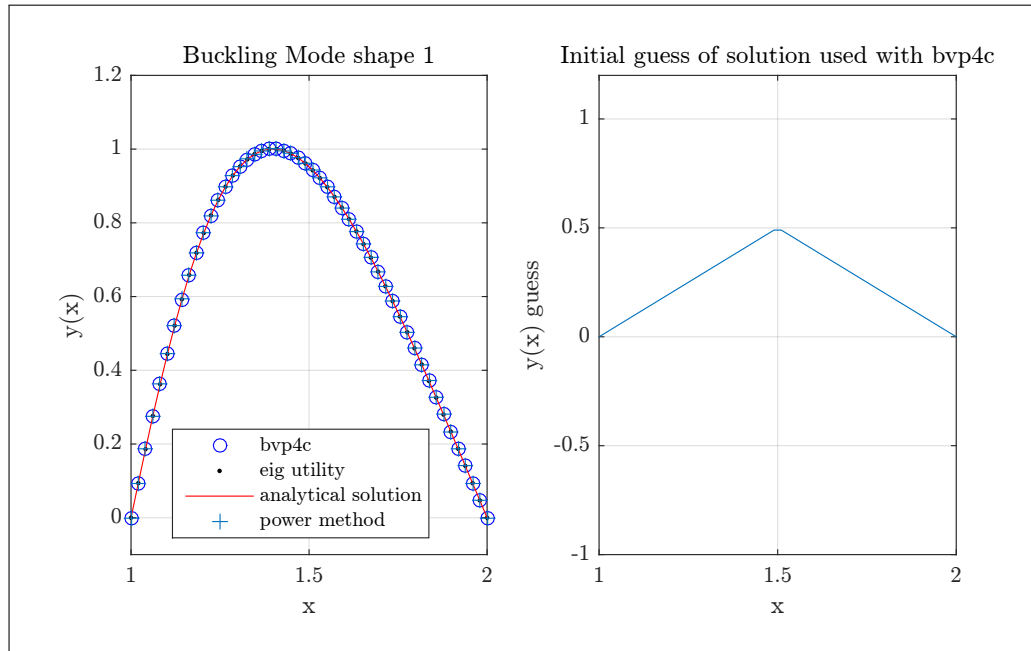


Figure 10: First mode shape, combined plot

2. Second mode shape

Table 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

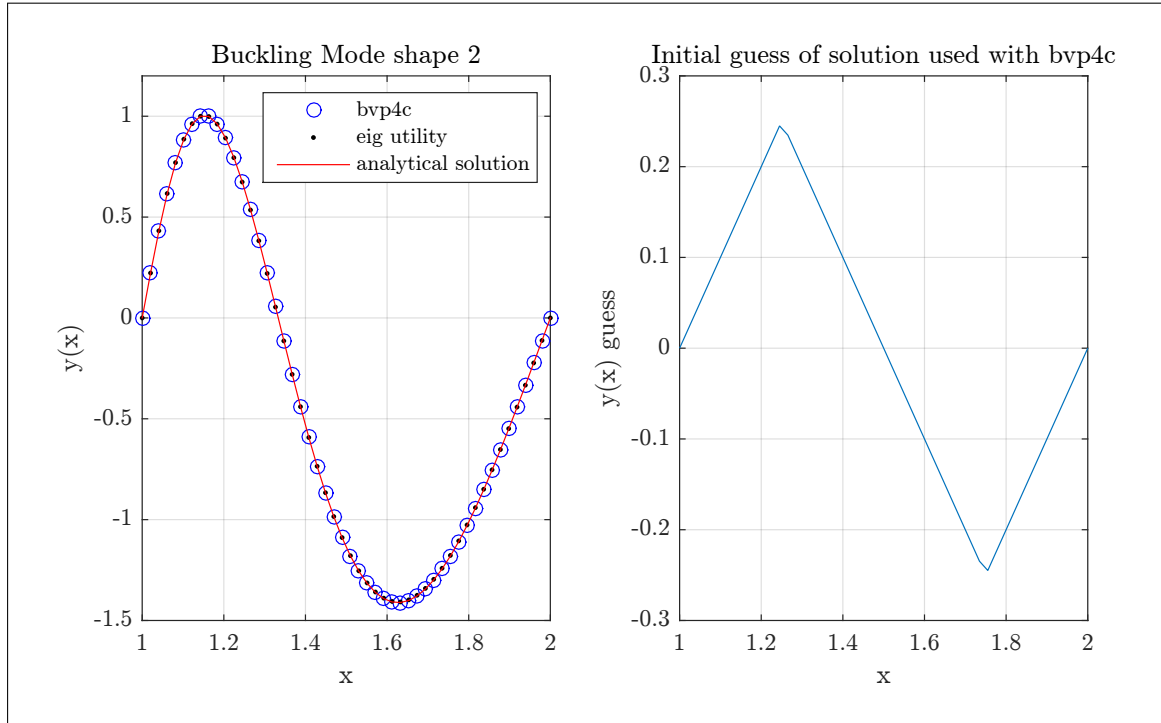


Figure 11: Second mode shape

3. Third mode shape

Table 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

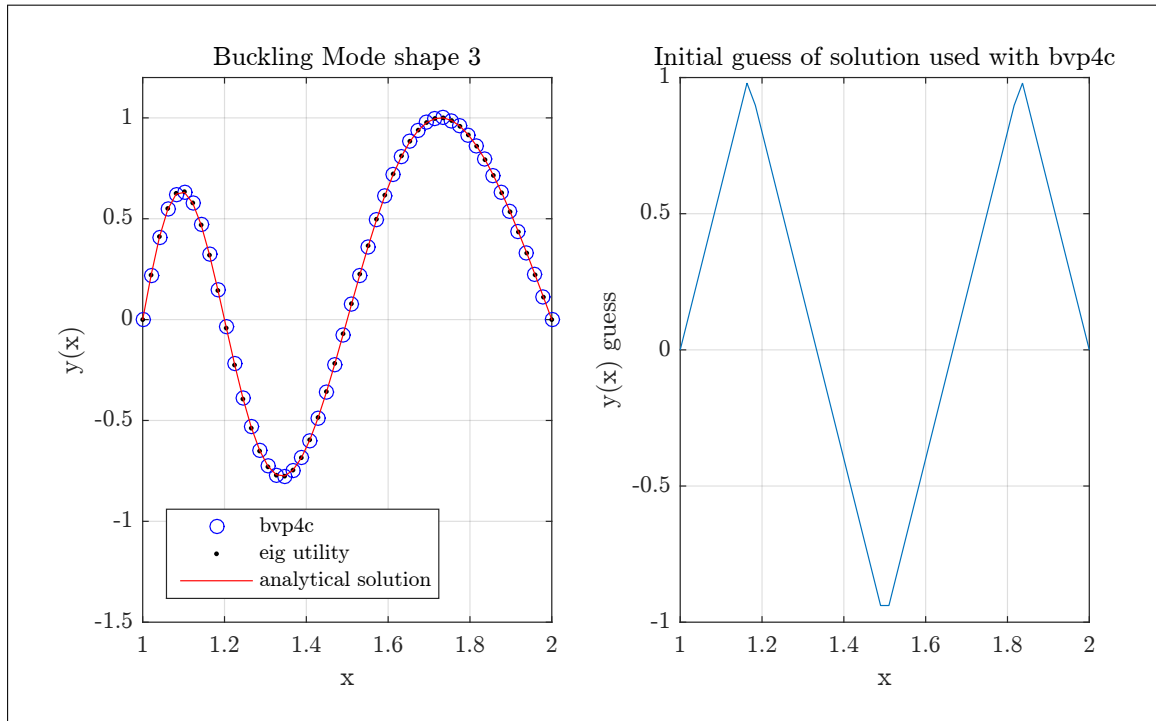


Figure 12: Third mode shape

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
```

0.2.3 Source code

```
1 function nma_HW3_EMA_471_problem_2()
2 % Solves  $z^4 y'' + \lambda^2 y = 0$ 
3 %
4 % see HW3, EMA 471, Spring 2016
5 % by Nasser M. Abbasi
6 %
7 clc; close all; initialize();
8
9 %look at first 3 mode shapes (one more than asked for,
10 %in order to verify)
11 N = 50; %number of grid points.
12
13 %domain of problem, in normalized z-space.
14 x = linspace(1,2,N);
15
16 %these are guess values for lambda for bvp4c only
17 guess_lambda = [6,12,18];
18
19 % try three mode shapes
20 for k = 1:3
21     process(k, x, guess_lambda(k), N);
22 end
23
24 end
25 %=====
26 %Main process function. Calls all solvers and call
27 %the main plot function
28 function process(mode_shape_number, x, guess_lambda, N)
29
30     y_bvp4c = get_y_bvp4c(x, guess_lambda, mode_shape_number);
31     y_eig = get_eigenvector_matlab_eig(x,N-2,mode_shape_number);
32     y_analytic = get_y_analytic(x, mode_shape_number);
33
34     %power method only for lowest eigenvalue
35     if mode_shape_number==1
36         y_power = get_y_power(x,N-2);
37         plot_result_1(x, y_bvp4c, y_eig, y_analytic, \ ...
38             y_power, mode_shape_number);
39     else
40         plot_result_2(x, y_bvp4c, y_eig, \ ...
41             y_analytic, mode_shape_number);
42     end
43
```



```

44 end
45 %=====
46 %This function finds the eigenvalue and eigenvector
47 %using Matlab eig()
48 function y_eig = get_eigenvector_matlab_eig(x,N,mode_shape_number)
49
50 h           = x(2)-x(1); % find grid spacing
51 A           = setup_A_matrix(h,N);
52 B           = -eye(N)*h^2;
53 [eig_vector,eig_values] = eig(A,B);
54 eig_values  = diag(eig_values); %they are on diagonal
55 sorted_eig_values      = sort(eig_values); %sort, small->large
56
57 %now need to match the original position of the
58 %eigenvalue with its corresponding eigenvector. Hence find the
59 %index of correct eigenvalue to use as index to eigenvector
60 found_eig_vector = eig_vector(:,...
61     eig_values == sorted_eig_values(mode_shape_number));
62
63 %Set is sign correctly
64 if found_eig_vector(1) > 0
65     y_eig = [0 ; found_eig_vector ; 0];
66 else
67     y_eig = [0 ; -found_eig_vector ; 0];
68 end
69
70 y_eig      = y_eig/max(y_eig); %normalize
71
72 %normalize eigenvalues
73 sorted_eig_values = sqrt(sorted_eig_values)/pi;
74
75 fprintf('eigenvalue from eig is %9.7f\n',...
76     sorted_eig_values(mode_shape_number)*pi);
77
78 calculate_critial_load(sorted_eig_values(mode_shape_number)*pi);
79
80 %-----%
81 function A = setup_A_matrix(h,N)
82     A      = zeros(N);
83     A(1,1) = -2*(1+h)^4;
84     A(1,2) = (1+h)^4;
85     for i = 2:N-1
86         A(i,i-1:i+1) = [(1+i*h)^4,-2*(1+i*h)^4,(1+i*h)^4];
87     end
88     A(N,N)  = -2*(1+N*h)^4;
89     A(N,N-1) = (1+N*h)^4;
90 end

```

```

91 end
92 %=====
93 function y = get_y_power(x,N)
94
95 h             = x(2)-x(1); % find grid spacing
96 A             = setup_A_matrix_for_power(h,N);
97 A_inv        = setup_A_inv_matrix_for_power(N);
98
99 % Starting guess for the eigenvector. Use unit vector
100 y = ones(N,1);
101
102 % This below from EX 11, applied it here:
103 % set tolerance; "while" loop will run until there is
104 %no difference between old and new estimates for eigenvalues
105 %to within the tolerance
106
107 tol           = 1e-6;
108 eigenvalue_1_old = 0;
109 eigenvalue_1_new = 1;
110
111 while abs(eigenvalue_1_new - eigenvalue_1_old)/abs(eigenvalue_1_new) > tol
112     y_new = A_inv*y; % generate updated value for eigenvector
113     eigenvalue_1_old = eigenvalue_1_new; % update old eigenvalue
114     eigenvalue_1_new = max(y_new); % update new eigenvalue
115     y = y_new/eigenvalue_1_new; % renormalize eigenvector estimate
116 end
117
118 y = [0;y;0];
119 y = y/max(y); %normalize
120
121 % Taken Per EX 11:
122 % add boundary conditions to complete eigenvector; also
123 % note that we have found the largest value of the inverse
124 % of what we're looking for, so...
125
126 % the lambda we're seeking is actually the
127 % inverse of the square root of what we've found
128 lam = 1/sqrt(eigenvalue_1_new);
129
130 fprintf('eigenvalue obtained with the power iteration method %9.7f\n',...
131     lam);
132 calculate_critial_load(lam);
133
134 %-----%
135 function A = setup_A_matrix_for_power(h,N)
136     A = zeros(N);
137     A(1,1) = 2/h^2*(1+h)^4;

```

```

138     A(1,2) = -1/h^2*(1+h)^4;
139     for i = 2:N-1
140         A(i,i-1:i+1) = [-1/h^2*(1+i*h)^4,2/h^2*(1+i*h)^4,...
141                         -1/h^2*(1+i*h)^4];
142     end
143     A(N,N) = 2/h^2*(1+N*h)^4;
144     A(N,N-1) = -1/h^2*(1+N*h)^4;
145 end
146 %-----%
147 function A_inv = setup_A_inv_matrix_for_power(N)
148     %We are looking for smallest eigenvalue. Use inverse.
149     A_inv = zeros(N);
150     for i = 1:N
151         b_rhs = zeros(N,1);
152         b_rhs(i,1) = 1;
153         A_inv(:,i) = A\b_rhs;
154     end
155 end
156 end
157
158 %=====
159 function y_analytic = get_y_analytic(z,n)
160 b = 6; %meter
161 a = 3; %meter
162 L = b-a; %meter length of column
163
164 %from question statement
165 y_analytic = z.*sin(n*pi*(b/L).*(1-a./(L*z)));
166
167 y_analytic = y_analytic/max(y_analytic); %normalize
168
169 E = 10^9;
170 rb = 0.2; %meter, radius of lower section
171 I0 = (1/4)*pi*(rb)^4;
172
173 critical_load = n^2*pi^2*(a/b)^2* E*I0/L^2;
174 lam = sqrt(critical_load*b^4 / (E*L^2*I0) );
175
176 fprintf('eigenvalue from analytical is %9.7f\n',lam);
177 fprintf('critical load from analytical is %9.3f\n\n',...
178         critical_load);
179
180 end
181
182 %=====
183 function plot_result_1(x, y_bvp4c_normalized, y_eig, ...
184                       y_analytic, ...

```

```

185         y_power, mode_shape_number)
186 figure();
187 subplot(1,2,1);
188 plot(x,y_bvp4c_normalized(1,:), 'bo', ...
189      x,y_eig, 'k.', ...
190      x,y_analytic, 'r', ...
191      x,y_power, '+');
192 axis([1 2 -.1 1.2])
193 title(sprintf('Buckling Mode shape %d', mode_shape_number));
194 xlabel('x')
195 ylabel('y(x)')
196 legend('bvp4c', 'eig utility', 'analytical solution', ...
197        'power method', 'Location', 'southwest')
198 grid;
199 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
200
201 subplot(1,2,2);
202 initial_mode_shape = set_initial_mode_shape_plot(x-1, ...
203                                                  mode_shape_number);
204 plot(x, initial_mode_shape); axis([1 2 -1 1.2]);
205 grid;
206 title('Initial guess of solution used with bvp4c');
207 xlabel('x'); ylabel('y(x) guess');
208 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
209
210 figure();
211 subplot(2,2,1);
212 plot(x,y_bvp4c_normalized(1,:), 'bo');
213 title(sprintf('Buckling Mode shape %d bvp4c', mode_shape_number));
214 xlabel('x'); axis([1 2 -.1 1.2]);
215 ylabel('y(x)'); grid;
216 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
217
218 subplot(2,2,2);
219 plot(x,y_eig, 'k. ');
220 title(sprintf('Buckling Mode shape %d. Matlab eig() result', ...
221             mode_shape_number));
222 xlabel('x'); axis([1 2 -.1 1.2]);
223 ylabel('y(x)'); grid;
224 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
225
226 subplot(2,2,3);
227 plot(x,y_analytic, 'r');
228 title(sprintf('Buckling Mode shape %d. Analytical result', ...
229             mode_shape_number));
230 xlabel('x'); axis([1 2 -.1 1.2]);
231 ylabel('y(x)'); grid;

```

```

232 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
233
234 subplot(2,2,4);
235 plot(x,y_power,'+');
236 title(sprintf('Buckling Mode shape %d. Power method result',...
237             mode_shape_number));
238 xlabel('x'); axis([1 2 -.1 1.2]);
239 ylabel('y(x)'); grid;
240 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
241
242 end
243
244 %=====
245 function plot_result_2(x, y_bvp4c_normalized, y_eig, ...
246                       y_analytic, mode_shape_number)
247
248 figure();
249 subplot(1,2,1);
250 plot(x,y_bvp4c_normalized(1,:), 'bo', ...
251      x,y_eig, 'k.', ...
252      x,y_analytic, 'r')
253
254 axis([1 2 -1.5 1.2]);
255 title(sprintf('Buckling Mode shape %d', mode_shape_number));
256 xlabel('x')
257 ylabel('y(x)')
258 legend('bvp4c', 'eig utility', 'analytical solution', ...
259       'Location', 'southwest')
260 grid;
261 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
262
263 subplot(1,2,2);
264 initial_mode_shape = set_initial_mode_shape_plot(x-1, ...
265             mode_shape_number);
266 plot(x, initial_mode_shape);
267 grid;
268 title('Initial guess of solution used with bvp4c');
269 xlabel('x'); ylabel('y(x) guess');
270 %set(gca,'TickLabelInterpreter', 'Latex','fontsize',8);
271
272 end
273
274 %=====
275 function f = set_initial_mode_shape_plot(x, mode_shape_number)
276     % Internal function.
277     % plots the initial mode shape guess.
278     %

```

```

279     switch mode_shape_number
280         case 1
281             f = x.*(x<=0.5)+(1-x).*(x>0.5);
282         case 2
283             f = x.*(x<=0.25)+(0.5-x).*(x>0.25&x<=0.75)+...
284                 (x-1).*(x>0.75);
285         case 3
286             h = 1/6;
287             f = 1/h*x.*(x<=h)+(2-x/h).*(x>h&x<=3*h)+...
288                 (-4+1/h*x).*(x>3*h&x<(5*h))+(6-x/h).*(x>5*h);
289     end
290 end
291 %=====
292 function y_bvp4c_normalized = ...
293         get_y_bvp4c(x,guess_lambda,mode_shape_number)
294
295 initial_solution = bvpinit(x,@set_initial_mode_shape,guess_lambda);
296 y_bvp4c          = bvp4c(@rhs,@bc,initial_solution);
297 value           = y_bvp4c.parameters;
298 fprintf('\n*****\n');
299 fprintf('running mode %d\nEigenvalue obtained with bvp4c, is %9.7f\n',...
300         mode_shape_number,value);
301 calculate_critial_load(value);
302
303 y_bvp4c          = deval(y_bvp4c,x);    %interpolate
304 y_bvp4c_normalized = y_bvp4c/max(y_bvp4c(1,:)); %normalize
305
306 %-----
307 function solinit = set_initial_mode_shape(x)
308     % internal function
309     % This defines the initial guess for the eigenvector;
310     % the first guess of
311     % the fundamental mode shape is a sawtooth
312     %
313     switch mode_shape_number
314         case 1
315             if x <= 0.5
316                 f = x;
317                 fp = 1;
318             else
319                 f = 1 - x;
320                 fp = -1;
321             end
322         case 2
323             if x <= 0.25
324                 f = x;
325                 fp = 1;

```

```

326         elseif x > 0.25 && x <= 0.75
327             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

```

0.3 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:

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

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/m³ 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 13: problem 3 description

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

For domain $0 \leq z \leq 1$. 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, bvp4c 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.

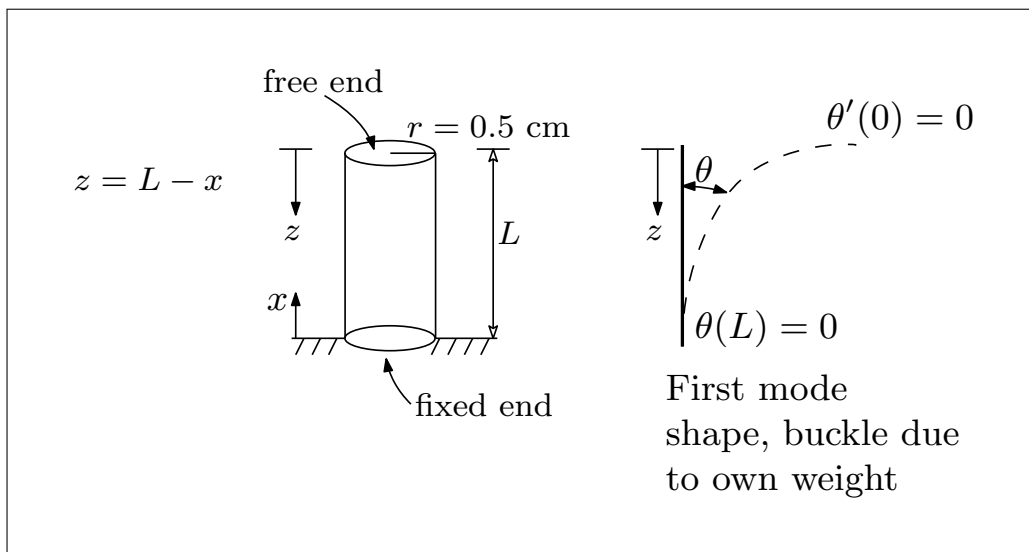


Figure 14: problem 3 geometry

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

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

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

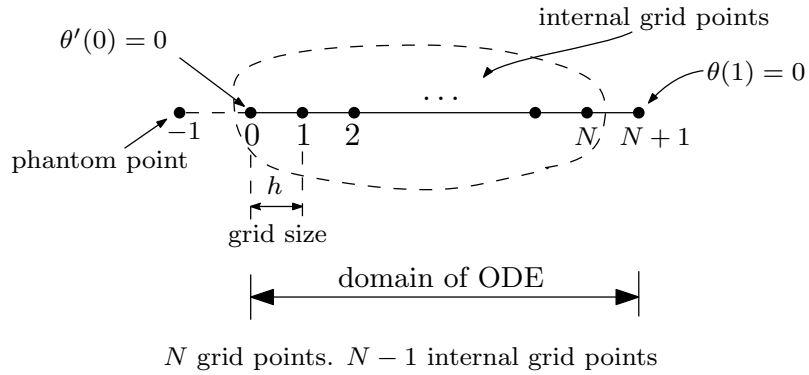


Figure 15: Grid used for problem 3

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.

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

Hence

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

At node $i = 0$

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

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

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

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

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

0.3.1 Power method

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

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

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

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

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

0.3.2 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.

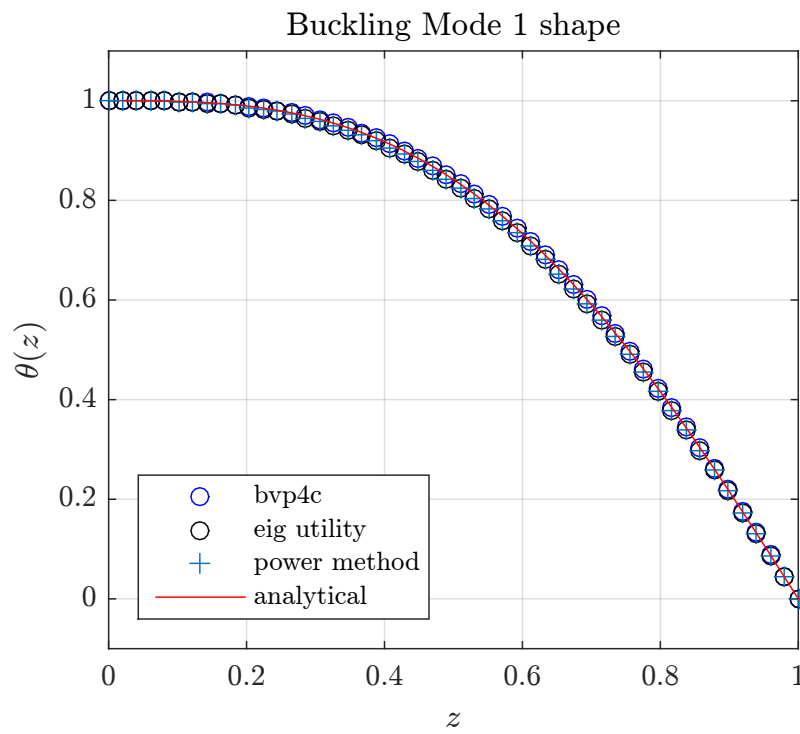


Figure 16: mode shape result from the three numerical method on one plot

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.

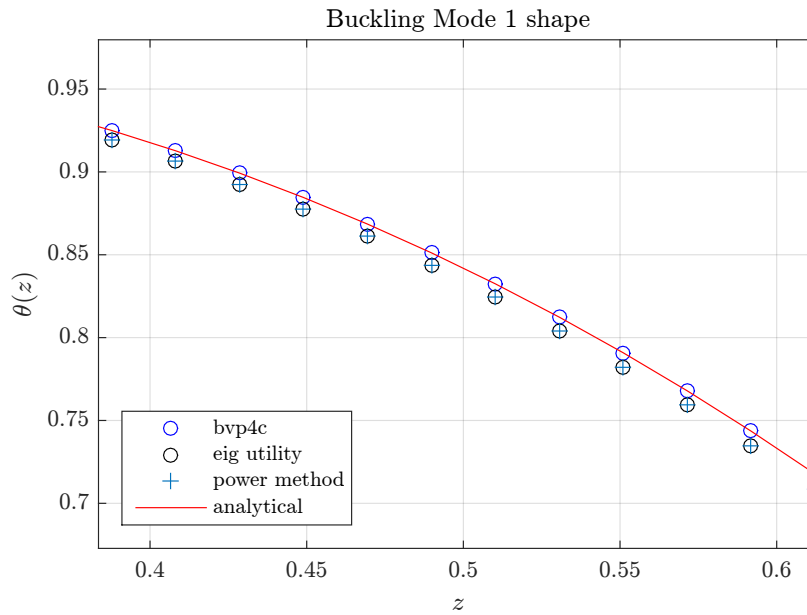


Figure 17: zoom in showing the result of the three methods

The following shows the result in separate plots

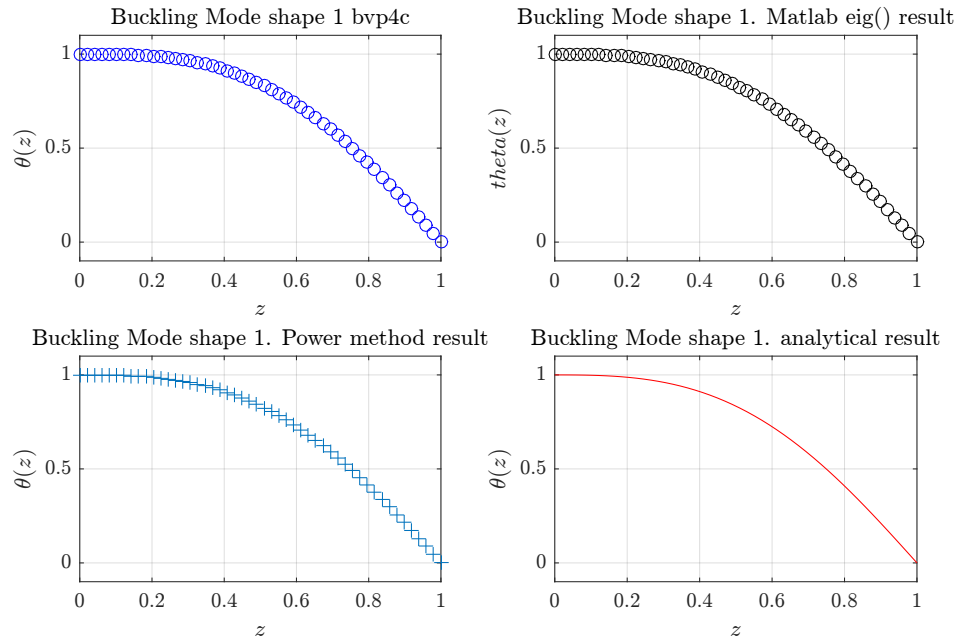


Figure 18: mode shape result from the three numerical method

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
```

0.3.3 Source code

```
1 function nma_HW3_EMA_471_problem_3()
2 % Solves  $z^4 y'' + \lambda^2 y = 0$ 
3 %
4 % see HW3, EMA 471, Spring 2016
```

```

5 % by Nasser M. Abbasi
6 %
7 clc; close all; initialize();
8
9 N          = 50;          %number of grid points.
10
11 %domain of problem, in normalized z-space.
12 x          = linspace(0,1,N);
13 guess_lambda = 2.8;
14
15 [y_bvp4c,eig_bvp4c] = get_y_bvp4c(x, guess_lambda);
16 y_eig           = get_eigenvector_matlab_eig(x, N-1);
17 y_power        = get_y_power(x,N-1);
18
19 %use bvp4c found eigenvalue to find analytical solution
20 %by using expression given in problem statement
21 y_analytic     = get_y_analytic(x,eig_bvp4c);
22
23 plot_result(x, y_bvp4c, y_eig, y_power, y_analytic);
24 end
25 %=====
26 %This function find the eigenvalue and eigenvector
27 %using Matlab eig()
28 function y_eig = get_eigenvector_matlab_eig(x,N)
29
30 h          = x(2)-x(1); % find grid spacing
31 A          = setup_A_matrix(N,h);
32 B          = setup_B_matrix(N,h);
33 [eig_vector,eig_values] = eig(A,B);
34 eig_values = diag(eig_values); %they are on diagonal
35 sorted_eig_values = sort(eig_values); %sort , small to large
36
37 %now need to match the original position of the eigenvalue
38 %with its corresponding eigenvector. Hence find the index of
39 %correct eigenvalue so use to index to eigenvector
40 found_eig_vector = eig_vector(:,eig_values == sorted_eig_values(1));
41
42 %Set its sign correctly
43 if found_eig_vector(1) > 0
44     y_eig = [found_eig_vector ; 0];
45 else
46     y_eig = [-found_eig_vector ; 0];
47 end
48
49 y_eig = y_eig/max(y_eig); %normalize
50
51 %normalize eigenvalues

```



```

52 sorted_eig_values = sqrt(sorted_eig_values)/pi;
53 fprintf('\n*****\n');
54 fprintf('eigenvalue from eig is\n');
55 disp(sorted_eig_values(1)*pi);
56
57 calculate_critial_length(sorted_eig_values(1)*pi);
58
59 %-----%
60 function A = setup_A_matrix(N,h)
61     A      = zeros(N);
62     eps    = 1e-6;
63     A(1,1) = -2/eps;
64     A(1,2) = 2/eps;
65     for i = 2:N-1
66         A(i,i-1:i+1) = [1,-2,1]/((i-1)*h);
67     end
68     A(N,N)  = -2/(N*h);
69     A(N,N-1) = 1/(N*h);
70 end
71 %-----%
72 function B = setup_B_matrix(N,h)
73     B = -h^2 * eye(N);
74 end
75
76 end
77 %=====
78 function y = get_y_power(x,N)
79
80 h      = x(2)-x(1); % find grid spacing
81 A      = setup_A_matrix_for_power(h,N);
82 A_inv = setup_A_inv_matrix_for_power(N);
83
84 % Starting guess for the eigenvector. Use unit vector
85 y = ones(N,1);
86
87 % This below from EX 11, apply it here:
88 % set tolerance; "while" loop will run until there is no
89 %difference between old and new estimates for eigenvalues to
90 %within the tolerance
91
92 tol = 1e-6;
93 eigenvalue_1_old = 0;
94 eigenvalue_1_new = 1;
95
96 while abs(eigenvalue_1_new - eigenvalue_1_old)/abs(eigenvalue_1_new) > tol
97
98     % generate updated value for eigenvector

```

```

99     y_new = A_inv*y;
100
101
102     eigenvalue_1_old = eigenvalue_1_new; % update old eigenvalue
103     eigenvalue_1_new = max(y_new);      % update new eigenvalue
104     y = y_new/eigenvalue_1_new; %renormalize eigenvector estimate
105 end
106
107 y = [y;0];
108 y = y/max(y); %normalize
109
110 % Taken Per EX 11:
111 % add boundary conditions to complete eigenvector; also
112 %note that we have found the largest value of the inverse of
113 %what we're looking for, so...
114
115 % the lambda we're seeking is actually the
116 % inverse of the square root of what we've found
117 lam = 1/sqrt(eigenvalue_1_new);
118
119 fprintf('\n*****\n');
120 fprintf('eigenvalue obtained with the power iteration method\n');
121 disp(lam);
122
123 calculate_critial_length(lam);
124
125 %-----%
126 function A = setup_A_matrix_for_power(h,N)
127     A = zeros(N);
128     eps = 1e-6;
129     A(1,1) = -2/eps;
130     A(1,2) = 2/eps;
131     for i = 2:N-1
132         A(i,i-1:i+1) = [1,-2,1]/((i-1)*h);
133     end
134     A(N,N) = -2/(N*h);
135     A(N,N-1) = 1/(N*h);
136     A = -A/h^2;
137 end
138 %-----%
139 function A_inv = setup_A_inv_matrix_for_power(N)
140     %We are looking for smallest eigenvalue. Use inverse.
141     A_inv = zeros(N);
142     for i = 1:N
143         b_rhs = zeros(N,1);
144         b_rhs(i,1) = 1;
145         A_inv(:,i) = A\b_rhs;

```

```

146         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,...
210                     y_power, y_analytic)
211
212 figure();
213 plot(x,y_bvp4c_normalized(1,:), 'bo', ...
214      x,y_eig, 'ko', ...
215      x,y_power, '+', ...
216      x,y_analytic, 'r');
217
218 axis([0 1 -0.1 1.1])
219 title('Buckling Mode 1 shape');
220 xlabel('$z$')
221 ylabel('$\theta(z)$')
222 legend('bvp4c', 'eig utility', 'power method', ...
223        'analytical', 'Location', 'southwest')
224 grid;
225 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
226
227
228 figure();
229 subplot(2,2,1);
230 plot(x,y_bvp4c_normalized(1,:), 'bo');
231 title(sprintf('Buckling Mode shape %d bvp4c', 1));
232 xlabel('$z$'); axis([0 1 -0.1 1.1])
233 ylabel('$\theta(z)$'); grid;
234 %set(gca, 'TickLabelInterpreter', 'Latex', 'fontsize', 8);
235
236 subplot(2,2,2);
237 plot(x,y_eig, 'ko');
238 title(sprintf('Buckling Mode shape %d. Matlab eig() result', 1));
239 xlabel('$z$'); axis([0 1 -0.1 1.1])

```

```

240 ylabel('$\theta(z)$'); grid;
241 %set(gca,'TickLabelInterpreter','Latex','fontsize',8);
242
243 subplot(2,2,3);
244 plot(x,y_power,'+');
245 title(sprintf('Buckling Mode shape %d. Power method result',1));
246 xlabel('$z$'); axis([0 1 -0.1 1.1])
247 ylabel('$\theta(z)$'); grid;
248 %set(gca,'TickLabelInterpreter','Latex','fontsize',8);
249
250 subplot(2,2,4);
251 plot(x,y_power,'r');
252 title(sprintf('Buckling Mode shape %d. analytical result',1));
253 xlabel('$z$'); axis([0 1 -0.1 1.1])
254 ylabel('$\theta(z)$'); grid;
255 %set(gca,'TickLabelInterpreter','Latex','fontsize',8);
256
257 end
258 %=====
259 function calculate_critical_length(lam)
260
261     r      = 0.05; %meter radius
262     g      = 9.81; %acc. due to gravity
263     density = 7500; % kg/m^3
264     E      = 100*10^9; %Pa
265     I0     = (1/4)*pi*(r)^4;
266     L      = (lam^2*E*I0/(density*g*pi*r^2))^(1/3);
267     fprintf('Critical length is\n');
268     disp(L);
269 end
270 %=====
271 function initialize()
272     reset(0);
273     set(groot,'defaulttextinterpreter','Latex');
274     set(groot,'defaultAxesTickLabelInterpreter','Latex');
275     set(groot,'defaultLegendInterpreter','Latex');
276
277     format long g
278 end

```