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

List of Tables

List of Figures

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 $0 \le x \le 1$. 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 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 $Ay = \alpha By$ and then Matlab $eig(A, B)$ is used to find the eigenvalues. Using second order centered difference gives

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

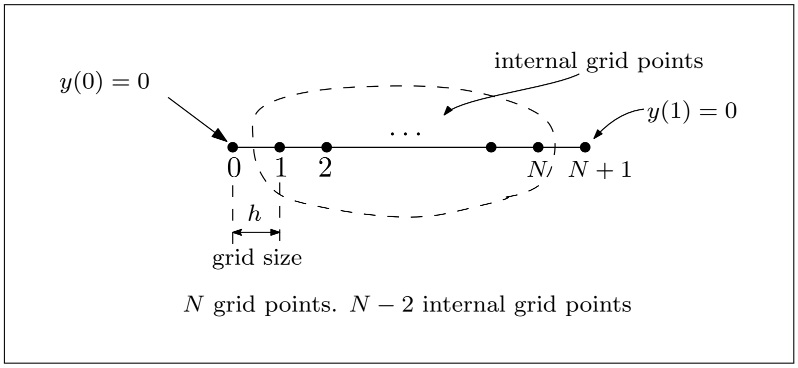

Figure 2: grid numbering used in problem 1

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

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

Hence

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

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 $i = 2$

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

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

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

At $i = N-1$

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

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.

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

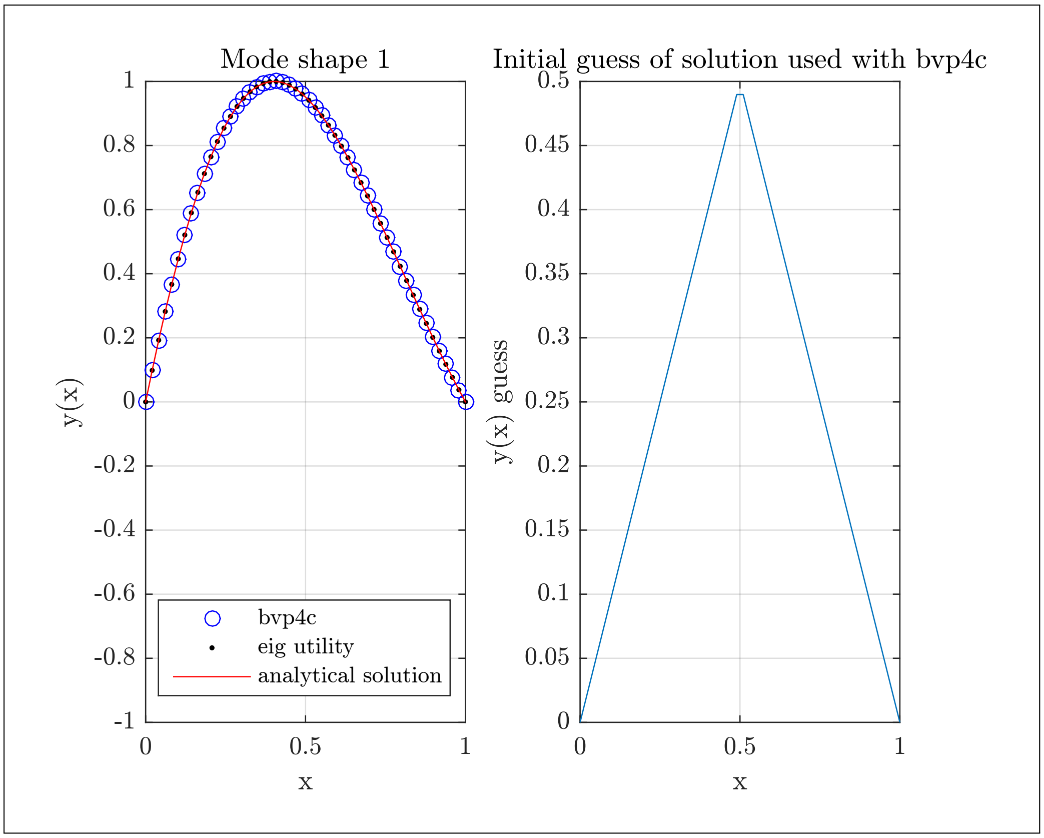

Figure 3: First mode shape

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

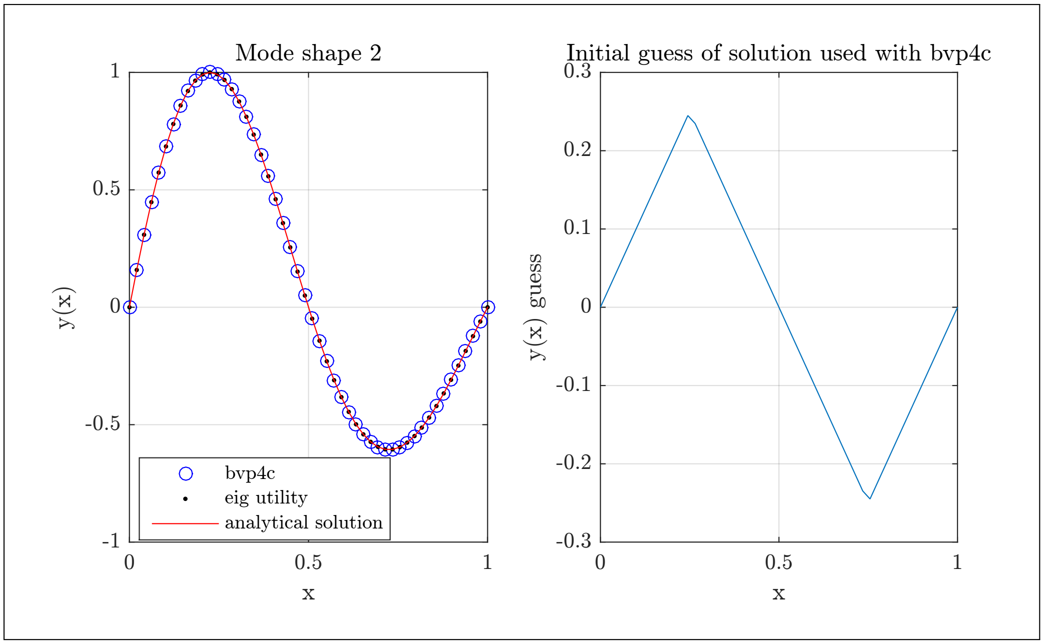

Figure 4: Second mode shape

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

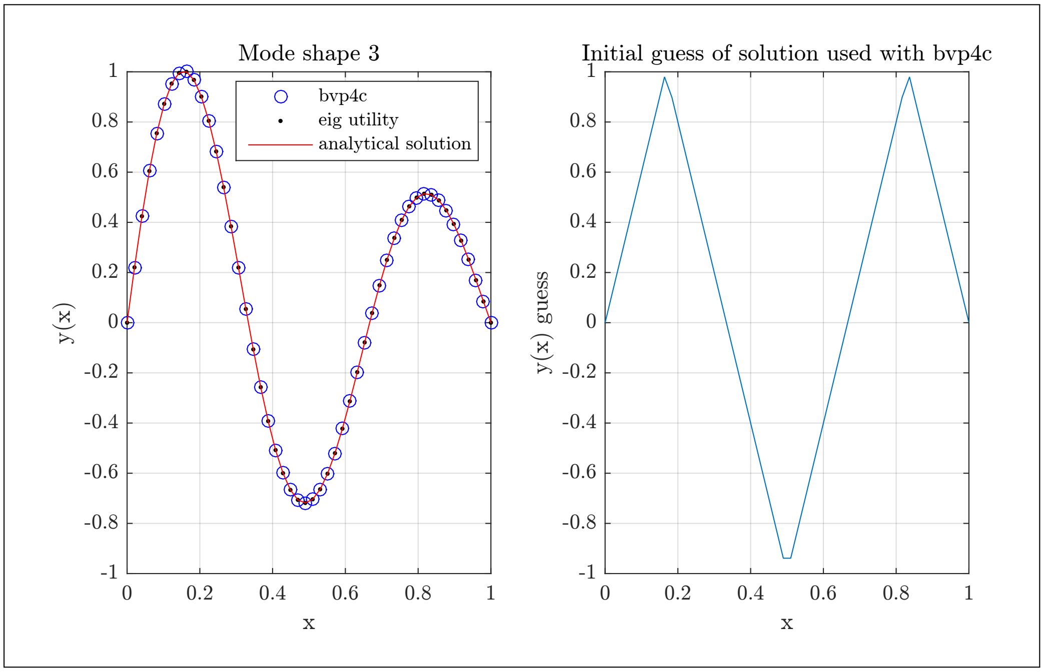

Figure 5: Third mode shape

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

```
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
```

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:

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

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:

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

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 6: problem 2 description

The geometry of the problem is as follows

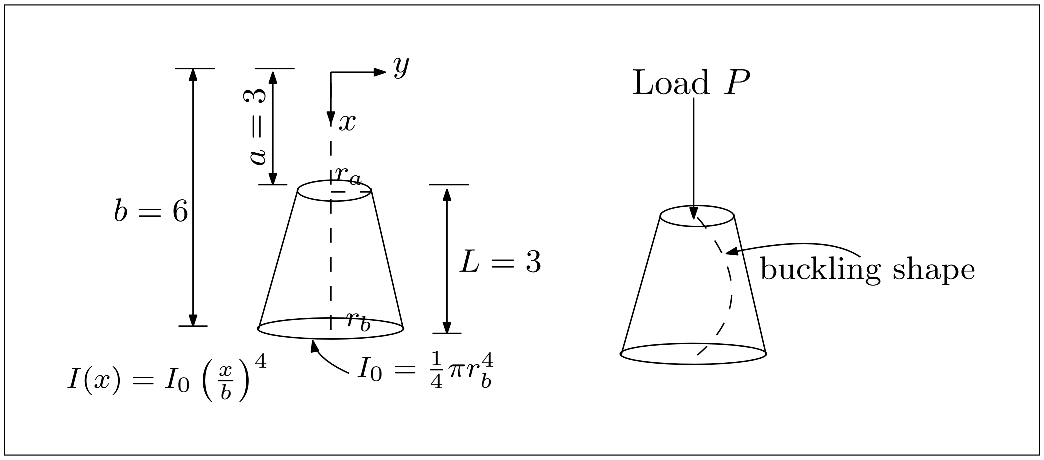

Figure 7: problem 2 geometry

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 $1 \le z \le 2$. 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

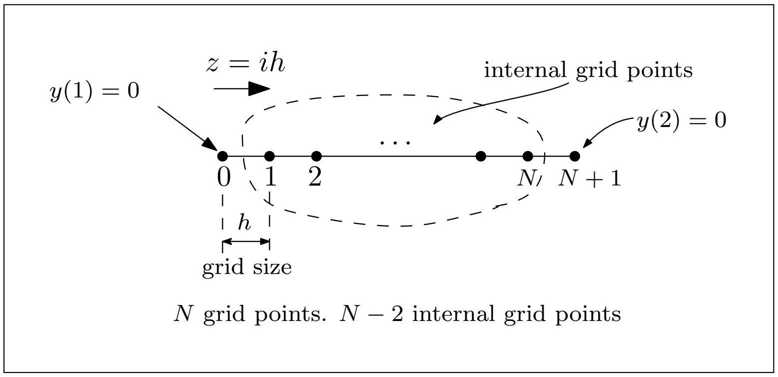

Figure 8: Grid used for problem 2

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.

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

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

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

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

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

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

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

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.

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

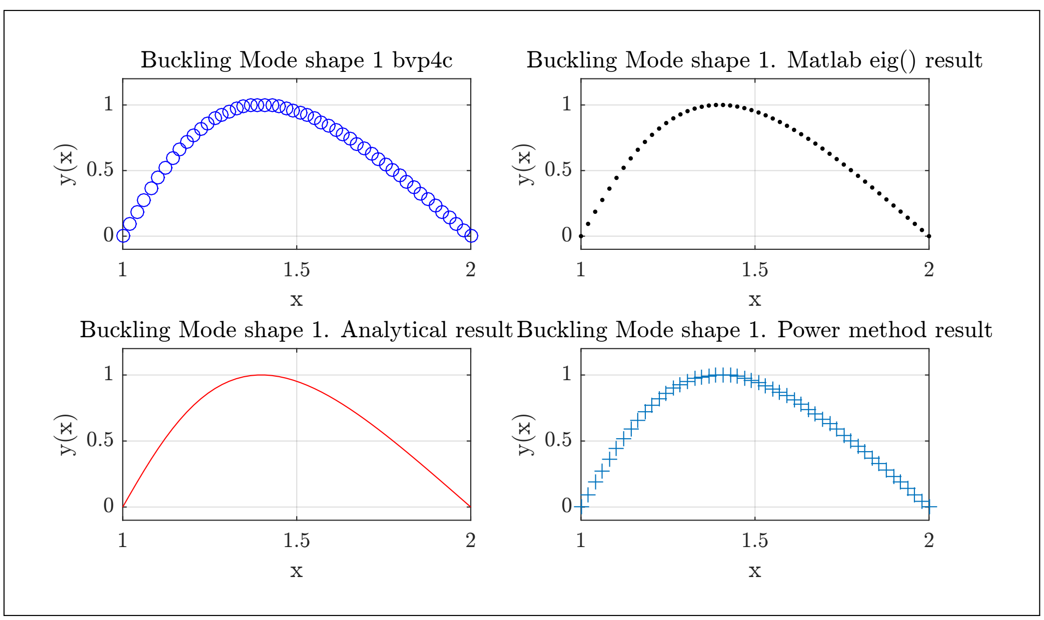

Figure 9: First mode shape, each solver on separate plot

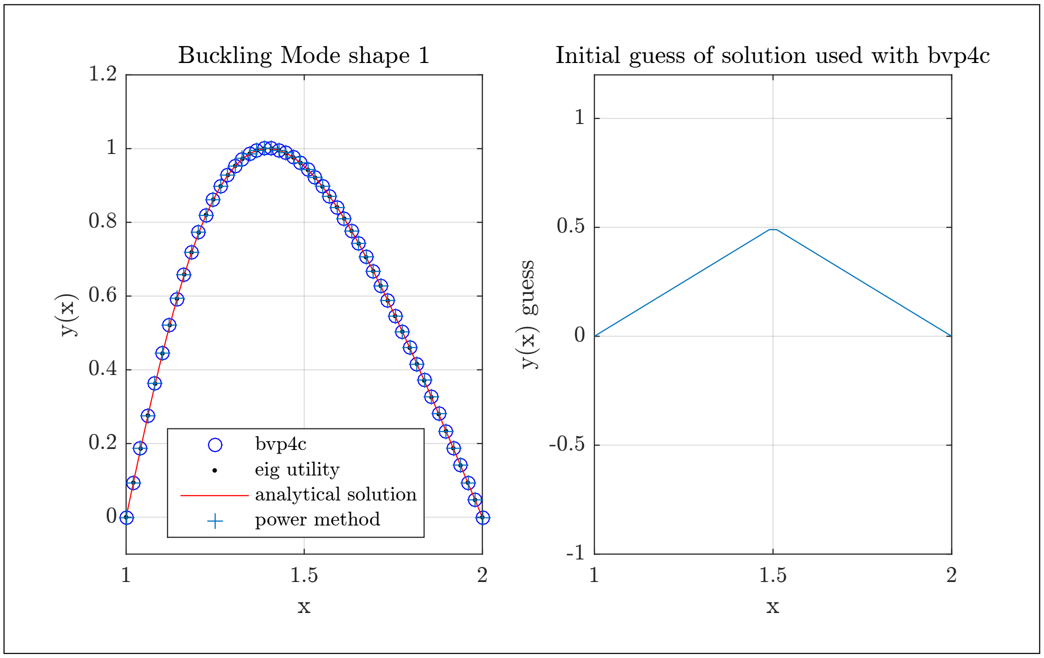

Figure 10: First mode shape, combined plot

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

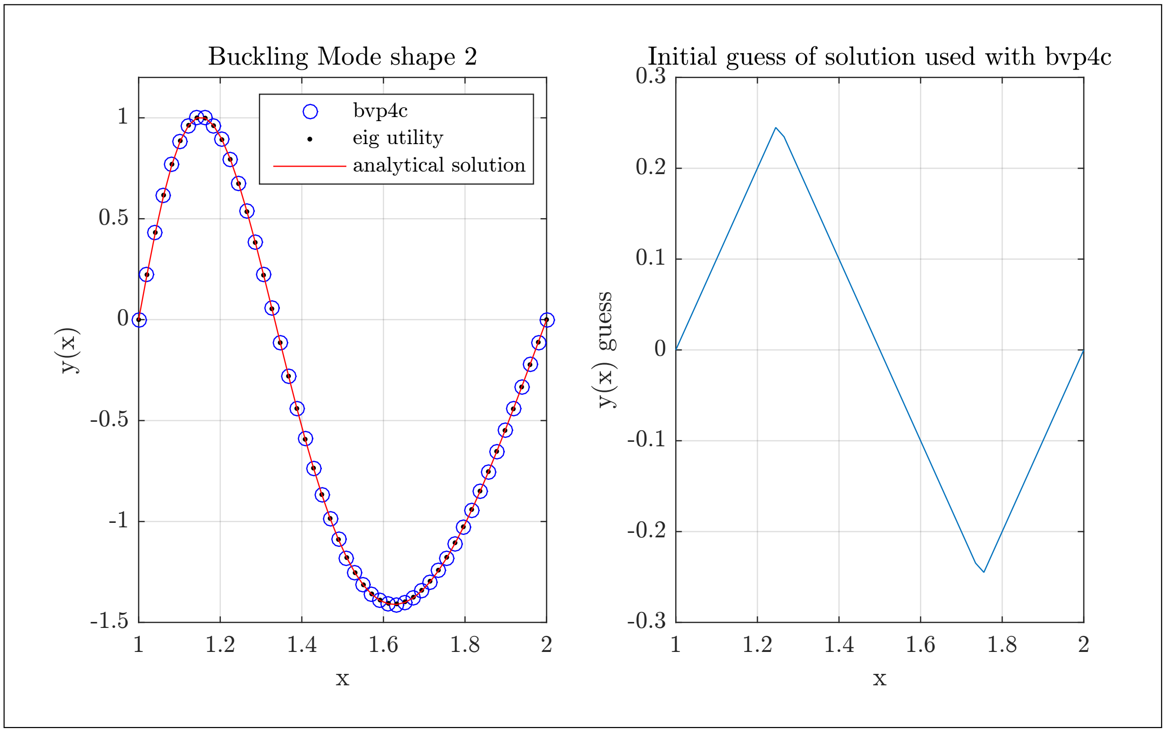

Figure 11: Second mode shape

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

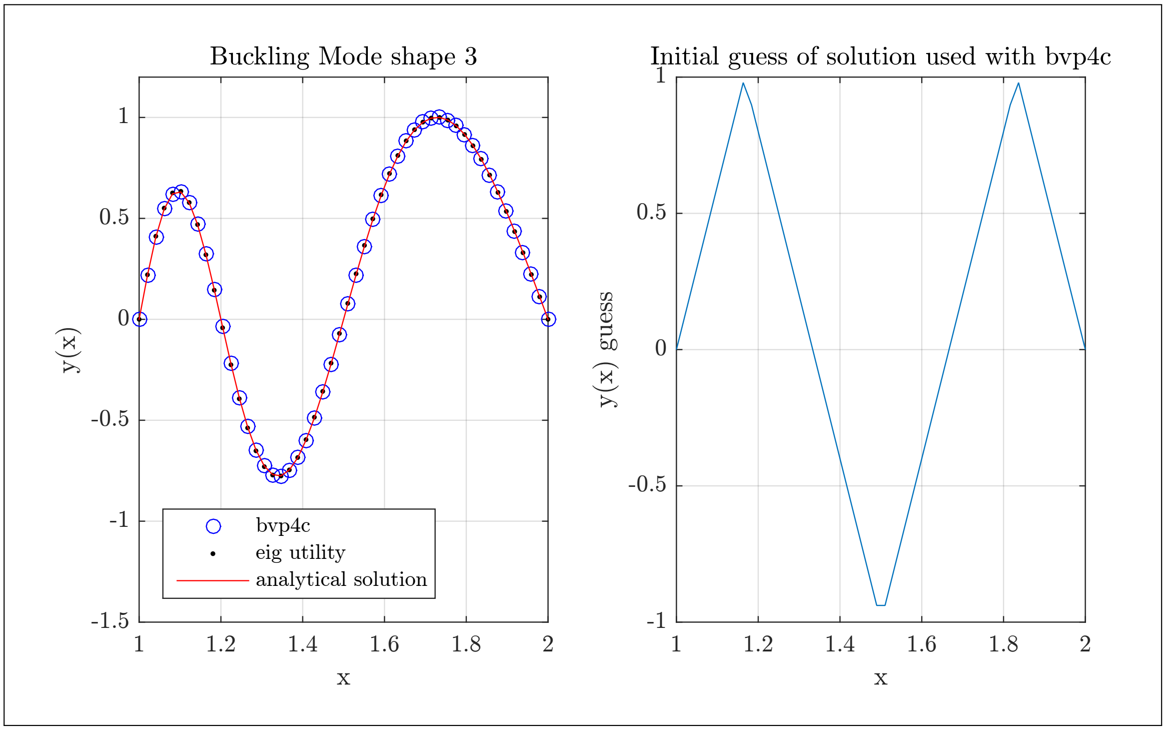

Figure 12: Third mode shape

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

```
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;
                    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
    %-----------------------------%
    function f = rhs(t,x,lam)
        %This function sets up the RHS of the state space
        %setup for this problem.
        %similar to ode45 RHS

        x1 = x(2);
        x2 = -lam^2*x(1)/t^4;
        f = [ x1
            x2];
    end
    %----------------------------%
    function res = bc(ya,yb,~)
        %This sets up the boundary conditions vector.
        %Must have ~ above in third agrs!
        res = [ ya(1)
            yb(1)
            ya(2)-1
            ];
    end
end
%======================================================%
```

```
function calculate_critial_load(lam)

E  = 10^9;
b  = 6; %meter
a  = 3; %meter
L  = b-a; %meter length of column
rb = 0.2; %meter, radius of lower section
I0 = (1/4)*pi*(rb)^4;

P = lam^2 * E * L^2 * I0/ b^4;
fprintf('Critical load is %9.3f.\n\n',P);
end
%==================================================================%
function initialize()
reset(0);
set(groot,'defaulttextinterpreter','Latex');
set(groot, 'defaultAxesTickLabelInterpreter','Latex');
set(groot, 'defaultLegendInterpreter','Latex');
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:

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

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 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.

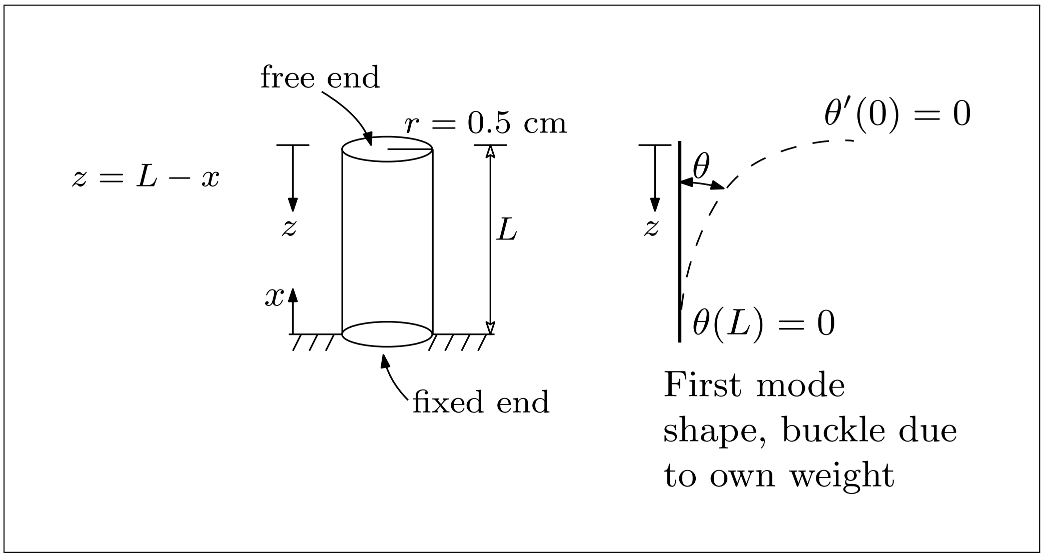

Figure 14: problem 3 geometry

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

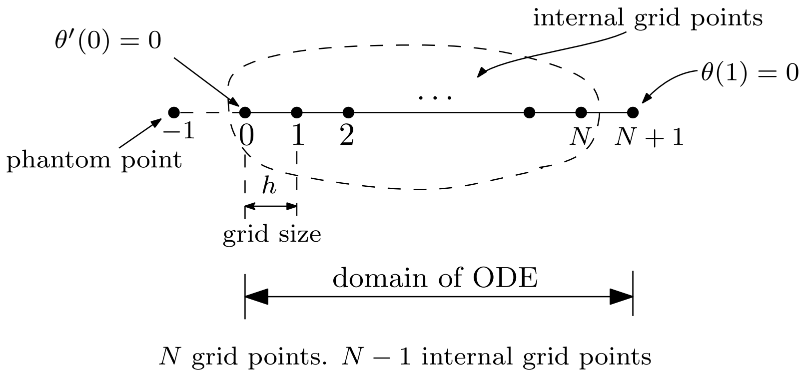

Figure 15: Grid used for problem 3

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.

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

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

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

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 $\lambda^2 = \frac{\rho g A L^3}{EI}$.

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.

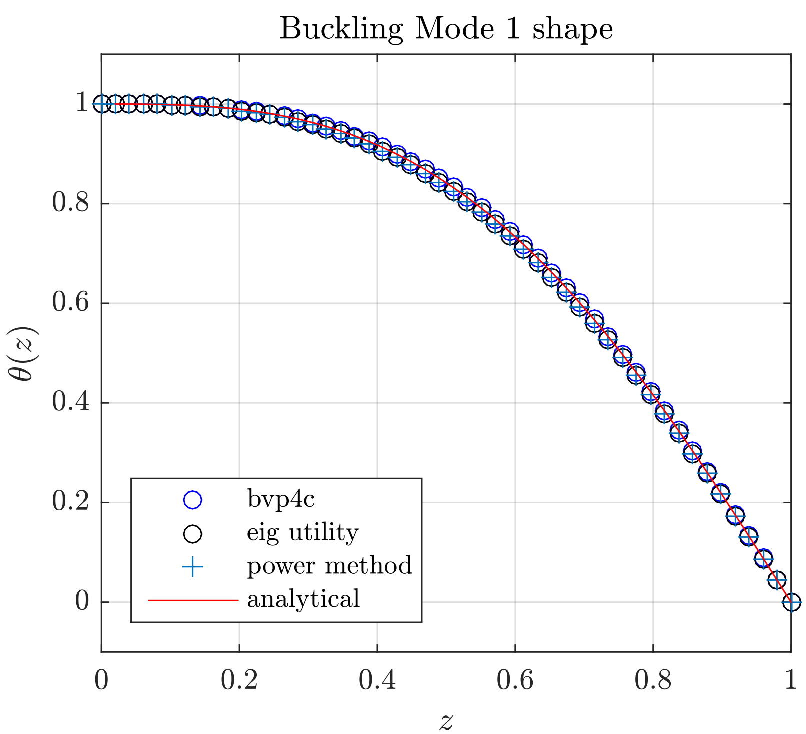

Figure 16: mode shape result from the three numerical method on one plot

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.

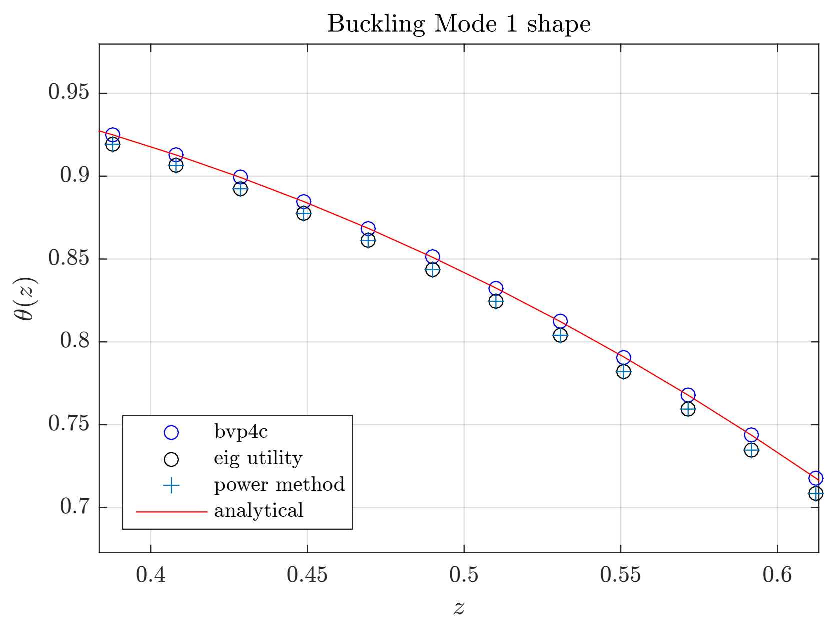

Figure 17: zoom in showing the result of the three methods

The following shows the result in separate plots

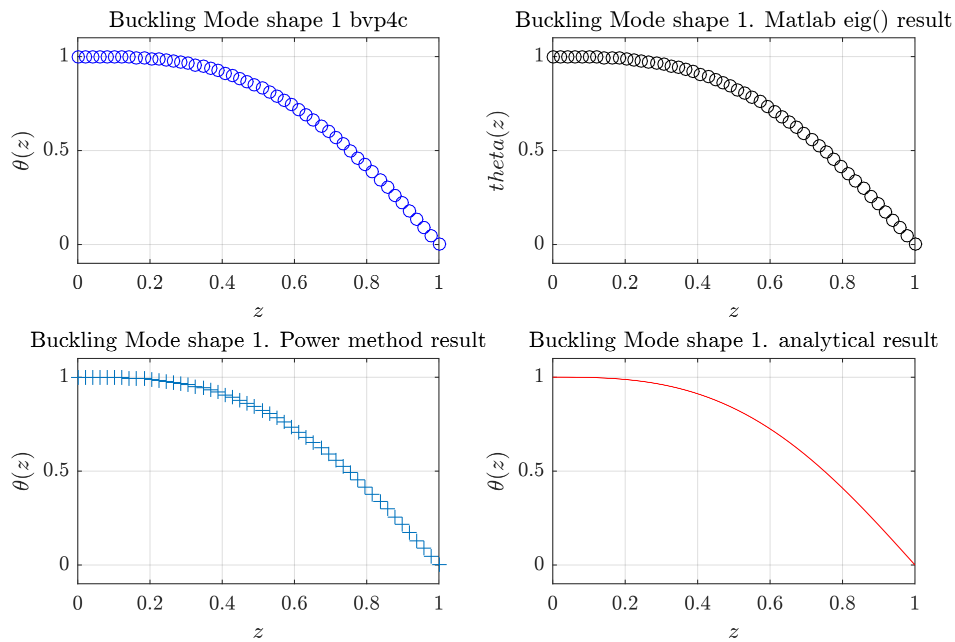

Figure 18: mode shape result from the three numerical method

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

```
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

    end
end

%=================================================================%
function  [y_bvp4c_normalized, eigen_value] = ...
                                    get_y_bvp4c(x,guess_lambda)

initial_solution  = bvpinit(x,@set_initial_mode_shape,...
                            guess_lambda);
y_bvp4c           = bvp4c(@rhs,@bc,initial_solution);
eigen_value       = y_bvp4c.parameters;
fprintf('\n********************\n');
fprintf('Eigenvalue obtained with bvp4c is\n');
disp(eigen_value);

calculate_critial_length(eigen_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
        %
        f  = 1-x;
        fp = -1;
        solinit = [ f ; fp ];
    end
    %-----------------------------%
    function f = rhs(t,x,lam)
        %This function sets up the RHS of the state space
        %setup for this problem.
        %similar to ode45 RHS
        x1 = x(2);
        x2 = -t*lam^2*x(1);
        f = [ x1
              x2];
    end
    %----------------------------%
    function res = bc(ya,yb,~)
        %This sets up the boundary conditions vector.
        %Must have ~ above in third agrs!
```

```
        res = [ ya(2)
                yb(1)
                yb(2)+1
              ];
    end
end

%======================================================%
function y_analytic = get_y_analytic(z,eigen_value)

  y_analytic = sqrt(z) .* besselj(-1/3,(2/3)*eigen_value*z.^(3/2));
  y_analytic = y_analytic/max(y_analytic);  %normalize


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