1. Let $B$ be the matrix given by
   \[
   B = \begin{bmatrix}
   1.8635 & 1.7135 & 1.9593 & 1.5685 & 1.6521 \\
   1.7135 & 1.8984 & 1.7439 & 1.6447 & 1.5253 \\
   1.9593 & 1.7439 & 2.6919 & 2.2635 & 1.6423 \\
   1.5685 & 1.6447 & 2.6919 & 2.2056 & 1.2430 \\
   1.6521 & 1.5253 & 1.6423 & 1.2430 & 1.5505
   \end{bmatrix}
   \]

   (i) Use the power method to find the largest eigenvalue.

   The following Matlab script along with the file `getB.m` given by
   \[
   B = \begin{bmatrix}
   1.8635 & 1.7135 & 1.9593 & 1.5685 & 1.6521 \\
   1.7135 & 1.8984 & 1.7439 & 1.6447 & 1.5253 \\
   1.9593 & 1.7439 & 2.6919 & 2.2635 & 1.6423 \\
   1.5685 & 1.6447 & 2.6919 & 2.2056 & 1.2430 \\
   1.6521 & 1.5253 & 1.6423 & 1.2430 & 1.5505
   \end{bmatrix}
   \]

   implements the power method and produces the output

   \[
   \begin{array}{c|c}
   \text{iteration} & \lambda \\
   \hline
   1 & 8.8244200000 \\
   2 & 8.914307789 \\
   3 & 8.9148122076 \\
   4 & 8.9148171504 \\
   5 & 8.9148171993 \\
   6 & 8.9148171997 \\
   7 & 8.9148171998 \\
   8 & 8.9148171998
   \end{array}
   \]

   From the output we see that the largest eigenvalue of $B$ is about 8.9148.

   (ii) Use the inverse power method to find the smallest eigenvalue.

   A slight modification of the above code to implement the inverse power method gives

   \[
   \text{clear all}
   \]
2 \texttt{getB;}
3 \texttt{v=ones(5,1);}
4 \texttt{disp(sprintf("\%10s \%15s","iteration","lambda");}
5 \texttt{for \texttt{i=1:8}}
6 \hspace{1em}w=B\backslash v;
7 \hspace{1em}\texttt{lambda=(v'*v)/(v'*w);}
8 \hspace{1em}\texttt{disp(sprintf(\%10d \%15.10f\",i,lambda));}
9 \hspace{1em}[y,i]=\texttt{max(abs(w));}
10 \hspace{1em}\texttt{alpha=w(i);}
11 \hspace{1em}v=w/alpha;
12 \texttt{end}

with output

\begin{tabular}{ll}
\textit{iteration} & \texttt{lambda} \\
1 & 2.4124813641 \\
2 & 0.0118139793 \\
3 & 0.0116738143 \\
4 & 0.0116543513 \\
5 & 0.0116513976 \\
6 & 0.0116509497 \\
7 & 0.0116508818 \\
8 & 0.0116508715 \\
\end{tabular}

From the output we see that the smallest eigenvalue of $B$ is about 0.011651.

\begin{enumerate}
\item[(iii)] Use the shifted inverse power method to find one more eigenvalue.
\end{enumerate}

We decide to shift by $s = 2$ to find another eigenvalue. A modification of the above code to implement the shifted inverse power method gives

1 \texttt{clear all}
2 \texttt{getB;}
3 \texttt{v=ones(5,1);}
4 \texttt{disp(sprintf("\%10s \%15s","iteration","lambda");}
5 \texttt{s=2;}
6 \texttt{BS=B-s*eye(5);}
7 \texttt{for \texttt{i=1:15}}
8 \hspace{1em}w=BS\backslash v;
9 \hspace{1em}\texttt{lambda=(v'*v)/(v'*w)+s;}
10 \hspace{1em}\texttt{disp(sprintf(\%10d \%15.10f\",i,lambda));}
11 \hspace{1em}[y,i]=\texttt{max(abs(w));}
12 \hspace{1em}\texttt{alpha=w(i);}
13 \hspace{1em}v=w/alpha;
14 \texttt{end}

with output

\begin{tabular}{ll}
\textit{iteration} & \texttt{lambda} \\
1 & 9.4040886684 \\
\end{tabular}
From the output we see that another eigenvalue of $B$ is about 0.88837.

For reference note that the complete set of eigenvalue of $B$ may be found using the built-in Matlab command `eig` and are given by

```matlab
>> eig(B)
ans =
    0.0116508697060692
    0.0299229096932658
    0.3651389250153996
    0.8883700958342445
    8.9148171997510222
```
2. Consider the two-point linear boundary problem

\[
\begin{align*}
\begin{cases}
y'' + \cos(x)y' - (x^2 + 1)y &= 2 \\
y(-2) &= y(2) = -1.
\end{cases}
\end{align*}
\]

(i) Solve the above boundary problem for grid sizes of \( h = 4/n \) where \( n = 4, 8, 16, 32, \ldots, 256 \). Plot your solutions.

The Matlab code for this consisted of the script

```matlab
1 clear all
2 p=@(x) -cos(x);
3 q=@(x) x.*x+1;
4 r=@(x) 2+0*x;
5
6 dataout=fopen("data2.dat","w");
7 for N=2.^[2:8]
8 fprintf(dataout,"# n=%d\n",N);
9 [xn,yn]=solvebvp(p,q,r,-2,2,N,-1,-1);
10 for j=1:length(xn)
11 fprintf(dataout,"%22.15e %22.15e\n",xn(j),yn(j));
12 end
13 fprintf(dataout,"\n\n");
14 end
15 fclose(dataout);
```

along with `solvebvp.m` for solving for the two-point boundary value problem

```matlab
1 \%solvebvp.m -- Solve boundary value problem
2 \% y''=py'+qy+r
3 \% y(a)=g1, y(b)=g2
4 function [xn,yn]=solvebvp(p,q,r,a,b,N,g1,g2)
5 h=(b-a)/N;
6 xn=[a:h:b]';
7 Aa=-(1+h/2*p(xn(3:N)));
8 Ab=2+h^2*q(xn(2:N));
9 Ac=(-1+h/2*p(xn(2:N-1)));
10 f=-h^2*r(xn(2:N));
11 f(1)=f(1)+(1+h/2*p(xn(2)))*g1;
12 f(N-1)=f(N-1)+(1-h/2*p(xn(N)))*g2;
13 ym=tridiag(Aa,Ab,Ac,f);
14 yn=[g1; ym; g2];
```

and the tridiagonal solver `tridiag.m` developed in class

```matlab
1 function f=tridiag(a,b,c,f)
2 n=length(b);
3 for i=1:n-1
```

4
Math/CS 467/667 Take Home Midterm Answer Key

```plaintext
4     a(i)=a(i)/b(i);
5     b(i+1)=b(i+1)-a(i)*c(i);
6     end
7     for i=1:n-1
8     f(i+1)=f(i+1)-a(i)*f(i);
9     end
10    f(n)=f(n)/b(n);
11    for i=n-1:-1:1
12    f(i)=(1/b(i))*(f(i)-c(i)*f(i+1));
13    end
14    end
```

This code creates a data file `data2.dat` consisting of the solution for each resolution separated by two blank lines. We plot the data file using the plotting program `gnuplot` along with the script `data2.gnu` which contains the commands:

```plaintext
1 set terminal pstex
2 set style data lines
3 set size 0.8,0.8
4 set key spacing 1.2
5 set key bottom
6 set output "data2-0.tex"
7 plot [:] [-1.35:-0.85] "data2.dat" index 0 ti "$n=4$"
8 set output "data2-1.tex"
9 plot [:] [-1.35:-0.85] "data2.dat" index 1 ti "$n=8$"
10 set output "data2-2.tex"
11 plot [:] [-1.35:-0.85] "data2.dat" index 2 ti "$n=16$"
12 set output "data2-3.tex"
13 plot [:] [-1.35:-0.85] "data2.dat" index 3 ti "$n=32$"
14 set output "data2-4.tex"
15 plot [:] [-1.35:-0.85] "data2.dat" index 4 ti "$n=64$"
16 set output "data2-5.tex"
17 plot [:] [-1.35:-0.85] "data2.dat" index 5 ti "$n=128$"
18 set output "data2-6.tex"
19 plot [:] [-1.35:-0.85] "data2.dat" index 6 ti "$n=256$"
```

The resulting output was
(ii) Calculate the value of \( y(1) \) to 5 significant digits.

The following Matlab code uses the relationship that \( y_j \approx y(1) \) for \( j = 1 + 3n/4 \). Note that 0.05 is added in the code and the integer part is taken to ensure correct rounding.

```matlab
clear all
p=@(x) -cos(x);
q=@(x) x.*x+1;
r=@(x) 2+0*x;

for N=2.^[5:12]
    [xn,yn]=solvebvp(p,q,r,-2,2,N,-1,-1);
    j=floor(3/4*N+1.05);
    disp(sprintf("%5d %5d %8.4g %15.10g",N,j,xn(j),yn(j)));
end
```

The output is:

<table>
<thead>
<tr>
<th>n</th>
<th>j</th>
<th>xj</th>
<th>yj</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>25</td>
<td>1</td>
<td>-1.002991565</td>
</tr>
<tr>
<td>64</td>
<td>49</td>
<td>1</td>
<td>-1.001970132</td>
</tr>
<tr>
<td>128</td>
<td>97</td>
<td>1</td>
<td>-1.001713523</td>
</tr>
<tr>
<td>256</td>
<td>193</td>
<td>1</td>
<td>-1.001649292</td>
</tr>
<tr>
<td>512</td>
<td>385</td>
<td>1</td>
<td>-1.001633229</td>
</tr>
<tr>
<td>1024</td>
<td>769</td>
<td>1</td>
<td>-1.001629213</td>
</tr>
<tr>
<td>2048</td>
<td>1537</td>
<td>1</td>
<td>-1.001628209</td>
</tr>
<tr>
<td>4096</td>
<td>3073</td>
<td>1</td>
<td>-1.001627958</td>
</tr>
</tbody>
</table>

It is clear that \( y(1) \approx -1.0016 \) to 5 significant digits.

(iii) [∗] Numerically verify the order of convergence of your solution with theoretical expectations.
Since this ordinary differential equation two-point boundary value problem solver employs a 2nd order method, theoretical expectations are that the error decreases by a factor of 4 every time \( n \) is doubled. The following Matlab program computes an approximation for the exact answer using \( n = 2^{15} \) and then checks from \( n = 16, 32, \ldots, 1024 \) that the error decreases by a factor of 4 each time.

```matlab
1 clear all
2 p=@(x) -cos(x);
3 q=@(x) x.*x+1;
4 r=@(x) 2+0*x;
5
6 N1=[2.^[4:10],2^15];
7 M=length(N1);
8 for i=1:M
9    [xn,yn]=solvebvp(p,q,r,-2,2,N1(i),-1,-1);
10       j=floor(3/4*N1(i)+1.05);
11       y1(i)=yn(j);
12 end
13 e1=y1-y1(M);
14
15 disp(sprintf("%5s %16s %20s %16s","n","y","error","ratio"));
16 for i=2:M-1
17    disp(sprintf("%5d %16.11f %20.12e %16.12f",...
18        N1(i),y1(i),e1(i),e1(i-1)/e1(i)));
19 end;
```

The output is

<table>
<thead>
<tr>
<th>( n )</th>
<th>( y )</th>
<th>( \text{error} )</th>
<th>( \text{ratio} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>-1.00299156465</td>
<td>-1.363686014051e-03</td>
<td>3.940237053496</td>
</tr>
<tr>
<td>64</td>
<td>-1.00197013225</td>
<td>-3.422536077700e-04</td>
<td>3.984431378054</td>
</tr>
<tr>
<td>128</td>
<td>-1.00171352333</td>
<td>-8.564469614281e-05</td>
<td>3.996203188103</td>
</tr>
<tr>
<td>256</td>
<td>-1.00164929190</td>
<td>-2.141326550475e-05</td>
<td>3.999609313385</td>
</tr>
<tr>
<td>512</td>
<td>-1.00163322908</td>
<td>-5.350441215413e-06</td>
<td>4.002149475649</td>
</tr>
<tr>
<td>1024</td>
<td>-1.00162921306</td>
<td>-1.334424415633e-06</td>
<td>4.009549849906</td>
</tr>
</tbody>
</table>

The fact that the list of ratios given in the last column of the above table is close to 4 verifies the method is converging with second order.
3. Consider the initial value problem

\[
\begin{align*}
y' &= \cos(y) + \cos(x) \\
y(0) &= 0.
\end{align*}
\]

(i) Construct a 3rd-order Taylor method for solving this problem.

In general, a third order Taylor method has the update rule

\[
y_{n+1} = y_n + h f_0(x_n, y_n) + \frac{h^2}{2} f_1(x_n, y_n) + \frac{h^3}{3!} f_2(x_n, y_n)
\]

where

\[f_i(x, y) = \frac{d^i}{dx^i} f(x, y)\]

The following Maple script creates C and Matlab code for a Taylor method with order N and forcing function f.

```maple
1 #Build a Taylor method for y'=f of order N
2 restart;
3 with(codegen):
4 N:=3;
5 f:=cos(y(x))+cos(x);
6 s:=[seq(diff(y(x),x$(N-i+1))=F||(N-i),i=1..N),y(x)=yn,x=xn];
7 r[1]:=f;
8 for i from 2 to N do
9     r[i]:=diff(r[i-1],x);
10 end;
11 jet:=seq(F||(i-1)=subs(s,r[i]),i=1..N);
12 t:=yn;
13 for i from 1 to N do
14     t:=t+h^i/i!*F||(i-1);
15 end;
16 t2:=horner(t,h);
17 jeto:=optimize([[jet,yn=t2]]);
18 C([jeto]);
```

The output of this script is

\[
\begin{align*}
t1 &= \cos(yn) \\
t3 &= \cos(xn) \\
F0 &= t1+t3 \\
t4 &= \sin(yn) \\
t6 &= \sin(xn) \\
F1 &= -t4*F0-t6 \\
t7 &= F0*F0 \\
F2 &= -t1*t7-t4*F1-t3
\end{align*}
\]
yn = yn+(F0+(F1/2.0+F2*h/6.0)*h)*h;

which we embed into Matlab to make the Taylor integrator

```matlab
function yn=taylor3a(x0,y0,xn,N)
    h=(xn-x0)/N;
    xn=x0;
    yn=y0;
    for n=1:N
        % The following Taylor time step was generated by Maple
        t1 = cos(yn);
        t3 = cos(xn);
        F0 = t1+t3;
        t4 = sin(yn);
        t6 = sin(xn);
        F1 = -t4*F0-t6;
        t7 = F0*F0;
        F2 = -t1*t7-t4*F1-t3;
        yn = yn+(F0+(F1/2.0+F2*h/6.0)*h)*h;
        xn=x0+h*n;
    end;
```

(ii) Compute $y(10)$ to 5 significant digits using your 3rd-order Taylor method.

The script for computing $y(10)$ is given by

```matlab
clear all
disp(sprintf("%5s %16s","n","y"));
for N=2.^[3:8];
y=taylor3a(0,0,10,N);
disp(sprintf("%5d %16.11f",N,y));
end
```

with output

<table>
<thead>
<tr>
<th>n</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.86457548562</td>
</tr>
<tr>
<td>16</td>
<td>0.86482318490</td>
</tr>
<tr>
<td>32</td>
<td>0.86411075690</td>
</tr>
<tr>
<td>64</td>
<td>0.86398264165</td>
</tr>
<tr>
<td>128</td>
<td>0.8639574966</td>
</tr>
<tr>
<td>256</td>
<td>0.86396241613</td>
</tr>
</tbody>
</table>

It is clear that $y(10) \approx 0.86396$ to 5 significant digits.

(iii) [*] Numerically verify the order of convergence of your code to be 3rd order.
Since this is a 3rd order method, theoretical expectations are that the error decreases by a factor of $2^3 = 8$ every time $n$ is doubled. The following Matlab program computes an approximation for the exact answer using $n = 2^{15}$ and then checks from $n = 16, 32, \ldots, 1024$ that the error decreases by a factor of 8 each time.

```matlab
N1=[2.^[3:10],2^15];
M=length(N1);
for i=1:M
    y1(i)=taylor3a(0,0,10,N1(i));
end
e1=y1-y1(M);

for i=2:M-1
    disp(sprintf("%5s %16s %20s %16s","n","y","error","ratio"));
    disp(sprintf("%5d %16.11f %20.12e %16.12f",N1(i),y1(i),e1(i),e1(i-1)/e1(i)));
end;
```

The output is

<table>
<thead>
<tr>
<th>n</th>
<th>y</th>
<th>error</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>0.86482318490</td>
<td>8.611087197790e-04</td>
<td>0.712348420285</td>
</tr>
<tr>
<td>32</td>
<td>0.86411078690</td>
<td>1.487107152024e-04</td>
<td>5.790495450224</td>
</tr>
<tr>
<td>64</td>
<td>0.86398264165</td>
<td>2.056547146689e-05</td>
<td>7.231087088949</td>
</tr>
<tr>
<td>128</td>
<td>0.86396474966</td>
<td>2.673474348347e-06</td>
<td>7.692413985423</td>
</tr>
<tr>
<td>256</td>
<td>0.86396241613</td>
<td>3.39949226560e-07</td>
<td>7.86443824924</td>
</tr>
<tr>
<td>512</td>
<td>0.86396211901</td>
<td>4.283216037404e-08</td>
<td>7.936674678264</td>
</tr>
<tr>
<td>1024</td>
<td>0.86396208156</td>
<td>5.374421463422e-09</td>
<td>7.969631832105</td>
</tr>
</tbody>
</table>

The fact that the list of ratios given in the last column of the above table is close to 8 verifies the method is converging with third order.
Math/CS 467/667 Take Home Midterm Answer Key

4. Consider the general initial value problem

\[
\begin{align*}
\begin{align*}
    y' &= f(x, y) \\
    y(x_0) &= y_0
\end{align*}
\end{align*}
\]

(i) Starting with Euler’s explicit method, use Richardson extrapolation with step sizes \(2h\) and \(3h\) to create a method that is 2nd-order convergent.

Since Euler’s explicit method is first order, it has an asymptotic error formula of the form

\[
y(x) - y_h(x) = hD(x) + h^2 E(x) + \mathcal{O}(h^3)
\]

where \(y(x)\) is the exact solution and \(y_h\) is the approximation obtained by Euler’s method. It may be assumed that \(D(x)\) and \(E(x)\) do not depend greatly on \(h\). Therefore

\[
y(x) - y_{2h}(x) = 2hD(x) + 4h^2 E(x) + \mathcal{O}(h^3)
\]

and

\[
y(x) - y_{3h}(x) = 3hD(x) + 9h^2 E(x) + \mathcal{O}(h^3).
\]

Multiplying the first equation by 3, the second by 2 and subtracting eliminates \(D(x)\) and obtains

\[
y(x) - 3y_{2h}(x) + 2y_{3h}(x) = -6h^2 E(x) + \mathcal{O}(h^3).
\]

Therefore, the method given by \(y(x) \approx 3y_{2h}(x) - 2y_{3h}(x)\) is second order.

(ii) Starting with Euler’s explicit method, use Richardson extrapolation with step sizes \(h\), \(2h\) and \(3h\) to create a method that is 3rd-order convergent.

Starting with

\[
y(x) - y_h(x) = hD(x) + h^2 E(x) + \mathcal{O}(h^3)
\]

and

\[
y(x) - y_{2h}(x) = 2hD(x) + 4h^2 E(x) + \mathcal{O}(h^3).
\]

multiply the first equation by 2 and subtract to obtain

\[
y(x) - 2y_h(x) + y_{2h}(x) = -2h^2 E(x) + \mathcal{O}(h^3).
\]

Now, multiply this result by 3 and subtract from the answer to the previous part to obtain

\[
-2y(x) + 6y_h(x) - 6y_{2h}(x) + 2y_{3h}(x) = \mathcal{O}(h^3).
\]

Therefore, the method given by \(y(x) \approx 3y_h(x) - 3y_{2h}(x) + y_{3h}(x)\) is third order.

(iii) Use the 3rd-order method created above to compute \(y(2)\) to 5 significant digits when \(f(x, y) = -xy + (4x/y)\), \(x_0 = 0\) and \(y_0 = 1\).
The extrapolation method may be coded as

```matlab
function yn=richard4c(x0,y0,xn,f,N)
    h=(xn-x0)/N;
    xn=x0;
    yn=y0;
    for n=1:N
        y1=euler4c(xn,yn,xn+h,f,6);
        y2=euler4c(xn,yn,xn+h,f,3);
        y3=euler4c(xn,yn,xn+h,f,2);
        yn=3*(y1-y2)+y3;
        xn=x0+h*n;
    end;
end;
```

where the Euler method is given by

```matlab
function yn=euler4c(x0,y0,xn,f,N)
    h=(xn-x0)/N;
    xn=x0;
    yn=y0;
    for n=1:N
        yn=yn+h*f(xn,yn);
        xn=x0+h*n;
    end;
end;
```

Test this method using the script

```matlab
clear all
f=@(x,y) -x*y+(4*x/y);
N1=[2.^[3:10],2^15];
M=length(N1);
for i=1:M
    y1(i)=richard4c(0,1,2,f,N1(i));
end
e1=y1-y1(M);
disp(sprintf("%5s %16s %20s %16s","n","y","error","ratio"));
for i=2:M-1
disp(sprintf("%5d %16.11f %20.12e %16.12f","n","e1","error","ratio"));
    N1(i),y1(i),e1(i),e1(i-1)/e1(i));
end;
```
which produces the output

<table>
<thead>
<tr>
<th>n</th>
<th>y</th>
<th>error</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>1.98622917476</td>
<td>1.340517712323e-05</td>
<td>7.069641462451</td>
</tr>
<tr>
<td>32</td>
<td>1.98621746967</td>
<td>1.700086410983e-06</td>
<td>7.884997513436</td>
</tr>
<tr>
<td>64</td>
<td>1.98621598185</td>
<td>2.122761348688e-07</td>
<td>7.884997513436</td>
</tr>
<tr>
<td>128</td>
<td>1.98621579605</td>
<td>2.647186536642e-08</td>
<td>7.884997513436</td>
</tr>
<tr>
<td>256</td>
<td>1.98621577289</td>
<td>3.303734619209e-09</td>
<td>7.884997513436</td>
</tr>
<tr>
<td>512</td>
<td>1.98621576999</td>
<td>4.126134989235e-10</td>
<td>7.884997513436</td>
</tr>
<tr>
<td>1024</td>
<td>1.98621576963</td>
<td>5.156652882476e-11</td>
<td>7.884997513436</td>
</tr>
</tbody>
</table>

Therefore $y(2) \approx 1.98622$ to 5 significant digits. Moreover, the fact that the list of ratios given in the last column of the above table is close to 8 verifies the method is converging with third order.