Math/CS 467/667 Take Home Midterm Answer Key

1. Let B be the matrix given by

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	2.2635	2.2056	1.2430
1.6521	1.5253	1.6423	1.2430	1.5505

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

The following Matlab script

```
clear all;
getB;
v=ones(5,1);
disp(sprintf("%10s %15s","iteration","lambda"));
for i=1:8
    w=B*v;
    lambda=(v'*w)/(v'*v);
    disp(sprintf("%10d %15.10f",i,lambda));
    [y,i]=max(abs(w));
    alpha=w(i);
    v=w/alpha;
end
```

along with the file `getB.m` given by

```
B=[
  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  2.2635  2.2056  1.2430;
  1.6521  1.5253  1.6423  1.2430  1.5505];
```

implements the power method and produces the output

```
 iteration          lambda
         1    8.8244200000
         2    8.9143037789
         3    8.9148122076
         4    8.9148171504
         5    8.9148171993
         6    8.9148171997
         7    8.9148171998
         8    8.9148171998
```

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

```
clear all
```

```
getB;
v=ones(5,1);
disp(sprintf("%10s %15s","iteration","lambda"));
for i=1:8
    w=B\v;
    lambda=(v'*v)/(v'*w);
    disp(sprintf("%10d %15.10f",i,lambda));
    [y,i]=max(abs(w));
    alpha=w(i);
    v=w/alpha;
end
```

with output

```
 iteration          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
```

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

(iii) [$*$] Use the shifted inverse power method to find one more eigenvalue.

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

```
clear all
getB;
v=ones(5,1);
disp(sprintf("%10s %15s","iteration","lambda"));
s=2;
BS=B-s*eye(5);
for i=1:15
    w=BS\v;
    lambda=(v'*v)/(v'*w)+s;
    disp(sprintf("%10d %15.10f",i,lambda));
    [y,i]=max(abs(w));
    alpha=w(i);
    v=w/alpha;
end
```

with output

```
 iteration          lambda
         1    9.4040886684
```

```
 2  -10.2345718845
 3    0.7178667250
 4    0.8711806295
 5    0.8831941425
 6    0.8864238859
 7    0.8876008577
 8    0.8880553351
 9    0.8882374136
10    0.8883127823
11    0.8883448616
12    0.8883588253
13    0.8883650093
14    0.8883677831
15    0.8883690388
```

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

```
>> eig(B)
ans =

  0.0116508697060692
  0.0299229096932658
  0.3651389250153996
  0.8883700958342445
  8.9148171997510222
```

2. Consider the two-point linear boundary problem

$$\begin{cases} y'' + \cos(x)y' - (x^2+1)y = 2 \\ y(-2) = y(2) = -1. \end{cases}$$

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

The Matlab code for this consisted of the script

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

dataout=fopen("data2.dat","w");
for N=2.^[2:8]
    fprintf(dataout,"# n=%d\n",N);
    [xn,yn]=solvebvp(p,q,r,-2,2,N,-1,-1);
    for j=1:length(xn)
        fprintf(dataout,"%22.15e %22.15e\n",xn(j),yn(j));
    end
    fprintf(dataout,"\n\n");
end
fclose(dataout);
```

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

```
%solvebvp.m -- Solve boundary value problem
% y''=py'+qy+r
% y(a)=g1, y(b)=g2
function [xn,yn]=solvebvp(p,q,r,a,b,N,g1,g2)
    h=(b-a)/N;
    xn=[a:h:b]';
    Aa=-(1+h/2*p(xn(3:N)));
    Ab=2+h^2*q(xn(2:N));
    Ac=(-1+h/2*p(xn(2:N-1)));
    f=-h^2*r(xn(2:N));
    f(1)=f(1)+(1+h/2*p(xn(2)))*g1;
    f(N-1)=f(N-1)+(1-h/2*p(xn(N)))*g2;
    ym=tridiag(Aa,Ab,Ac,f);
    yn=[g1; ym; g2];
```

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

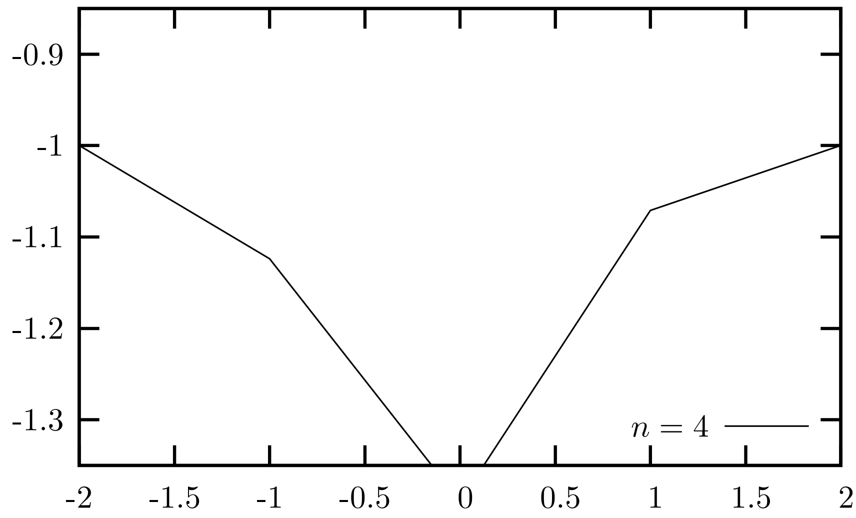
```
function f=tridiag(a,b,c,f)
    n=length(b);
    for i=1:n-1
```

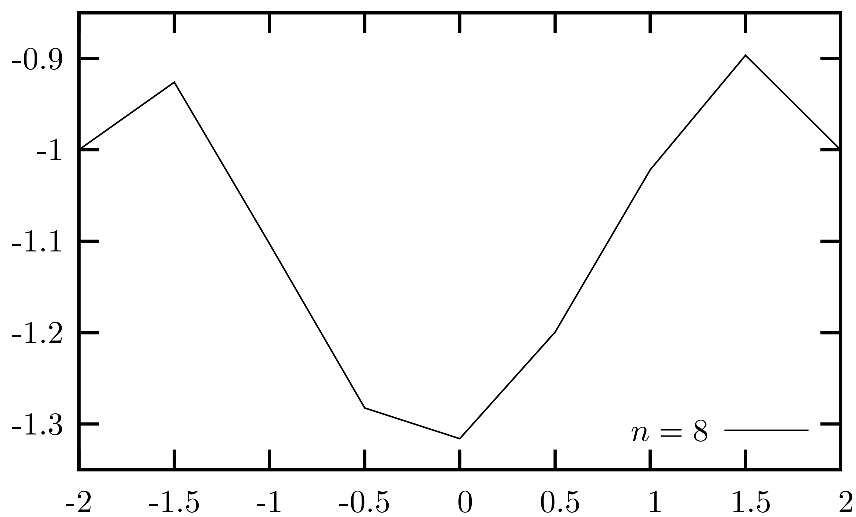
```
        a(i)=a(i)/b(i);
        b(i+1)=b(i+1)-a(i)*c(i);
    end
    for i=1:n-1
        f(i+1)=f(i+1)-a(i)*f(i);
    end
    f(n)=f(n)/b(n);
    for i=n-1:-1:1
        f(i)=(1/b(i))*(f(i)-c(i)*f(i+1));
    end
end
```

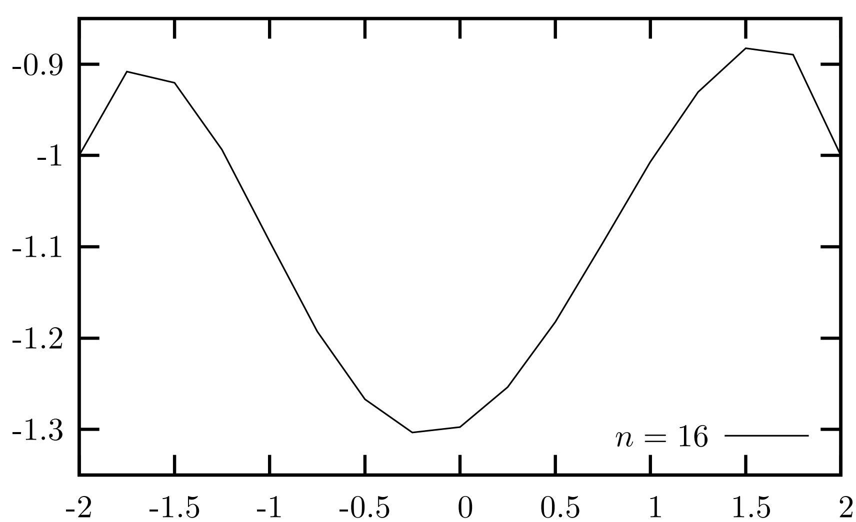
This code creates a data file `data2.dat` consisting of the solution for each resolution separated by two blank lines. We plot the data file using the plotting program `gnuplot` along with the script `data2.gnu` which contains the commands

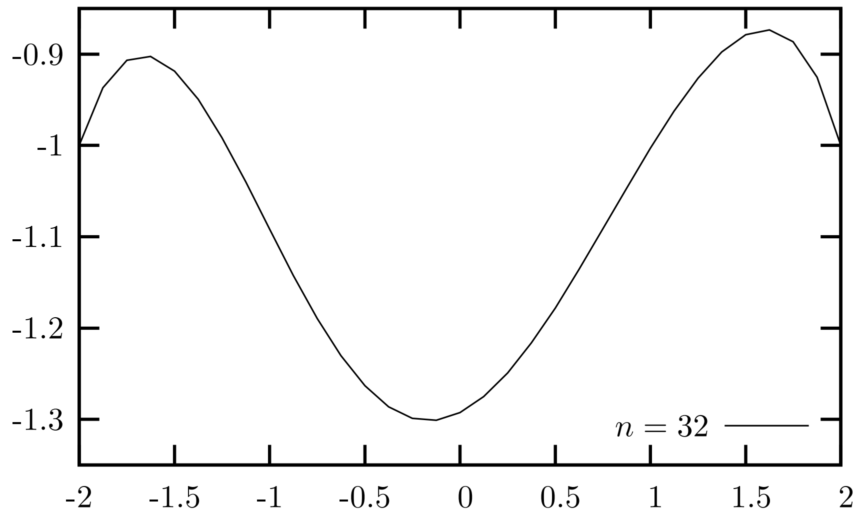
```
set terminal pstex
set style data lines
set size 0.8,0.8
set key spacing 1.2
set key bottom
set output "data2-0.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 0 ti "$n=4$"
set output "data2-1.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 1 ti "$n=8$"
set output "data2-2.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 2 ti "$n=16$"
set output "data2-3.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 3 ti "$n=32$"
set output "data2-4.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 4 ti "$n=64$"
set output "data2-5.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 5 ti "$n=128$"
set output "data2-6.tex"
plot [:] [-1.35:-0.85] "data2.dat" index 6 ti "$n=256$"
```

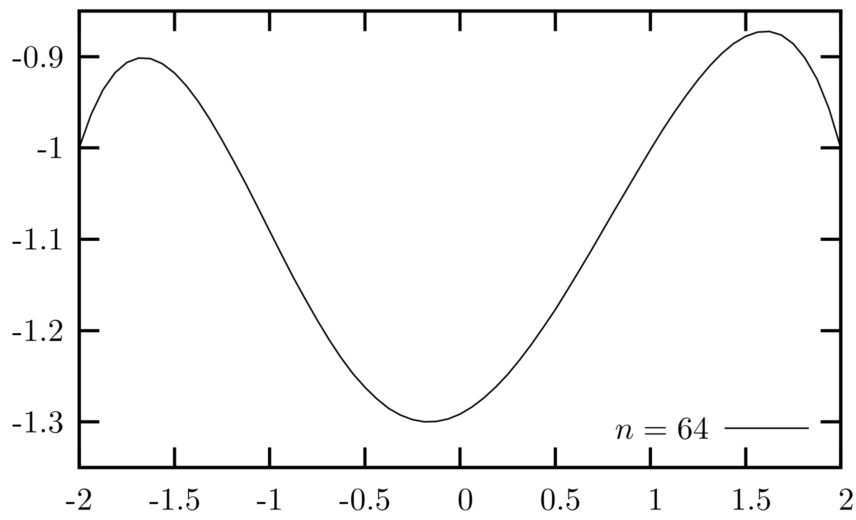
The resulting output was

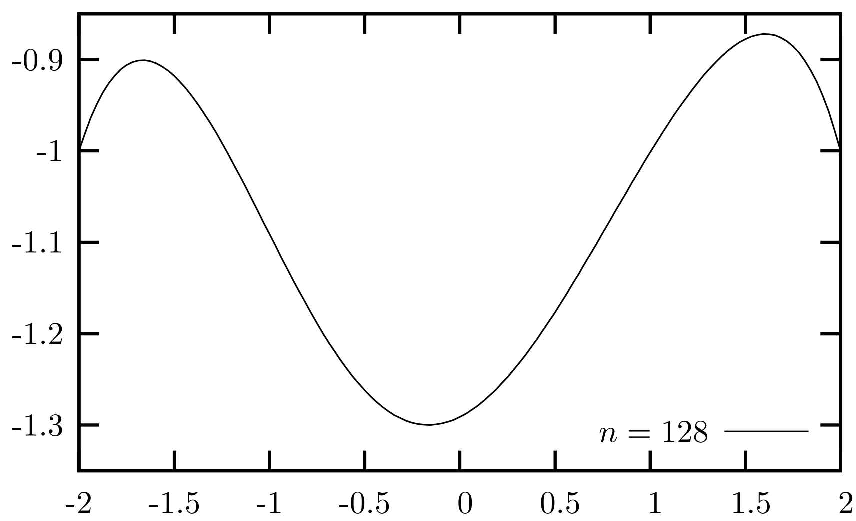

-0.9
-1
-1.1
-1.2
-1.3
$n = 4$
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

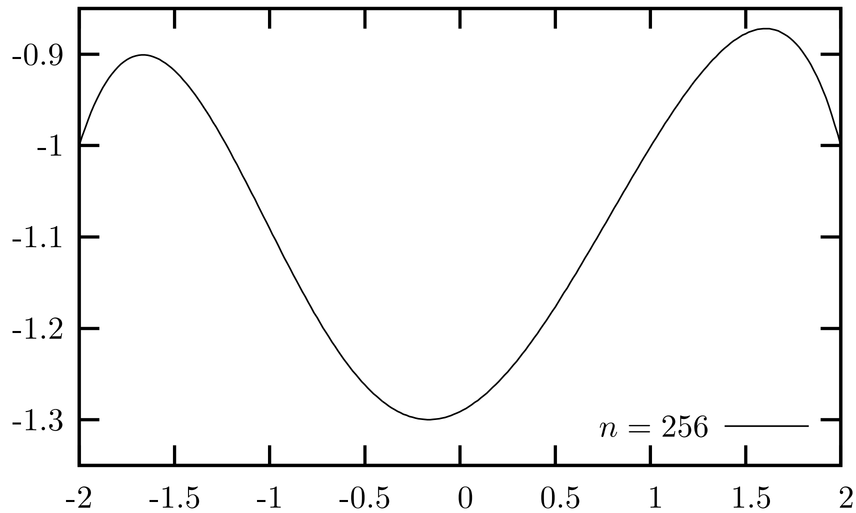

-0.9
-1
-1.1
-1.2
-1.3
$n = 8$
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


-0.9
-1
-1.1
-1.2
-1.3
$n = 16$
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


-0.9
-1
-1.1
-1.2
-1.3
$n = 32$
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2


-0.9
-1
-1.1
-1.2
-1.3
$n = 64$
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2


-0.9
-1
-1.1
-1.2
-1.3
$n = 128$
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2



(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.

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

disp(sprintf("%5s %5s %8s %15s","n","j","xj","yj"));
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

```
    n     j       xj              yj
   32    25        1    -1.002991565
   64    49        1    -1.001970132
  128    97        1    -1.001713523
  256   193        1    -1.001649292
  512   385        1    -1.001633229
 1024   769        1    -1.001629213
 2048  1537        1    -1.001628209
 4096  3073        1    -1.001627958
```

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.

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

N1=[2.^[4:10],2^15];
M=length(N1);
for i=1:M
    [xn,yn]=solvebvp(p,q,r,-2,2,N1(i),-1,-1);
    j=floor(3/4*N1(i)+1.05);
    y1(i)=yn(j);
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",...
        N1(i),y1(i),e1(i),e1(i-1)/e1(i)));
end;
```

The output is

n	y	error	ratio
32	-1.00299156465	-1.363686014051e-03	3.940237053496
64	-1.00197013225	-3.422536077700e-04	3.984431378054
128	-1.00171352333	-8.564469614281e-05	3.996203188103
256	-1.00164929190	-2.141326550475e-05	3.999609313385
512	-1.00163322908	-5.350441215413e-06	4.002149475649
1024	-1.00162921306	-1.334424415633e-06	4.009549849906

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

$$\begin{cases} y' = \cos(y) + \cos(x) \\ y(0) = 0. \end{cases}$$

(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.

```
#Build a Taylor method for y'=f of order N
restart;
with(codegen):
N:=3;
f:=cos(y(x))+cos(x);
s:=[seq(diff(y(x),x$(N-i+1))=F||(N-i),i=1..N),y(x)=yn,x=xn];
r[1]:=f;
for i from 2 to N
do
    r[i]:=diff(r[i-1],x);
end;
jet:=seq(F||(i-1)=subs(s,r[i]),i=1..N);
t:=yn;
for i from 1 to N
do
    t:=t+h^i/i!*F||(i-1);
end;
t2:=horner(t,h);
jeto:=optimize([jet,yn=t2]);
C([jeto]);
```

The output of this script is

```
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;
```

which we embed into Matlab to make the Taylor integrator

```
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

```
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

```
    n                y
    8    0.86457548562
   16    0.86482318490
   32    0.86411078690
   64    0.86398264165
  128    0.86396474966
  256    0.86396241613
```

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 3nd 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.

```
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);

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

The output is

n	y	error	ratio
16	0.86482318490	8.611087197790e-04	0.712348420285
32	0.86411078690	1.487107152024e-04	5.790495450224
64	0.86398264165	2.056547146689e-05	7.231087088949
128	0.86396474966	2.673474348347e-06	7.692413985423
256	0.86396241613	3.399449226560e-07	7.864433824924
512	0.86396211901	4.283216037404e-08	7.936674678264
1024	0.86396208156	5.374421463422e-09	7.969631832105

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.

4. Consider the general initial value problem

$$\begin{cases} y' = f(x,y) \\ y(x_0) = y_0. \end{cases}$$

(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

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

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

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

and

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

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

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

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

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

and

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

multiply the first equation by 2 and subtract to obtain

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

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

```
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;
```

where the Euler method is given by

```
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;
```

Test this method using the script

```
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",...
        N1(i),y1(i),e1(i),e1(i-1)/e1(i)));
end;
```

which produces the output

n	y	error	ratio
16	1.98622917476	1.340517712323e-05	7.069641462451
32	1.98621746967	1.700086410983e-06	7.884997513436
64	1.98621598185	2.122726134868e-07	8.008976678891
128	1.98621579605	2.647186536642e-08	8.018800736123
256	1.98621577289	3.303734619209e-09	8.012709378200
512	1.98621576999	4.126134989235e-10	8.006850546160
1024	1.98621576963	5.156652882476e-11	8.001575989838

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.