Math232 - Numerical Analysis II

Dr. Chrysoula Tsogka

HW4

Matteo Polimeno, Alex Ho

Department of Applied Mathematics, UC Merced

(Dated: February 21, 2020)

Table of Error for different step sizes			
h	error	ratio	observed order
0.1	9.23789e-03	NaN	NaN
0.05	2.39009e-03	3.86508	1.95050
0.025	6.07866e-04	3.93193	1.97524
0.01250	1.53277e-04	3.96581	1.98762

TABLE I. Table of the error and order of accuracy for the method implemented to numerically solve Eq. (1). We readily see it is indeed a second order method (see fourth column).

PROBLEM 1

Part (a)

Here we solve the boundary value problem

$$u'' = e^x,\ x\ \in\ (0,1),\ u(0) = 3,\ u'(1) = -5, \tag{1}$$

analytically. The general solution is easily found by integrating twice

$$u(x) = e^x + C_0 x + C_1, \tag{2}$$

where $C_0,\ C_1 \in \mathbb{R}$. To find the two constants we apply the boundary conditions, which yield

$$C_0 = -5 - e, \tag{3}$$

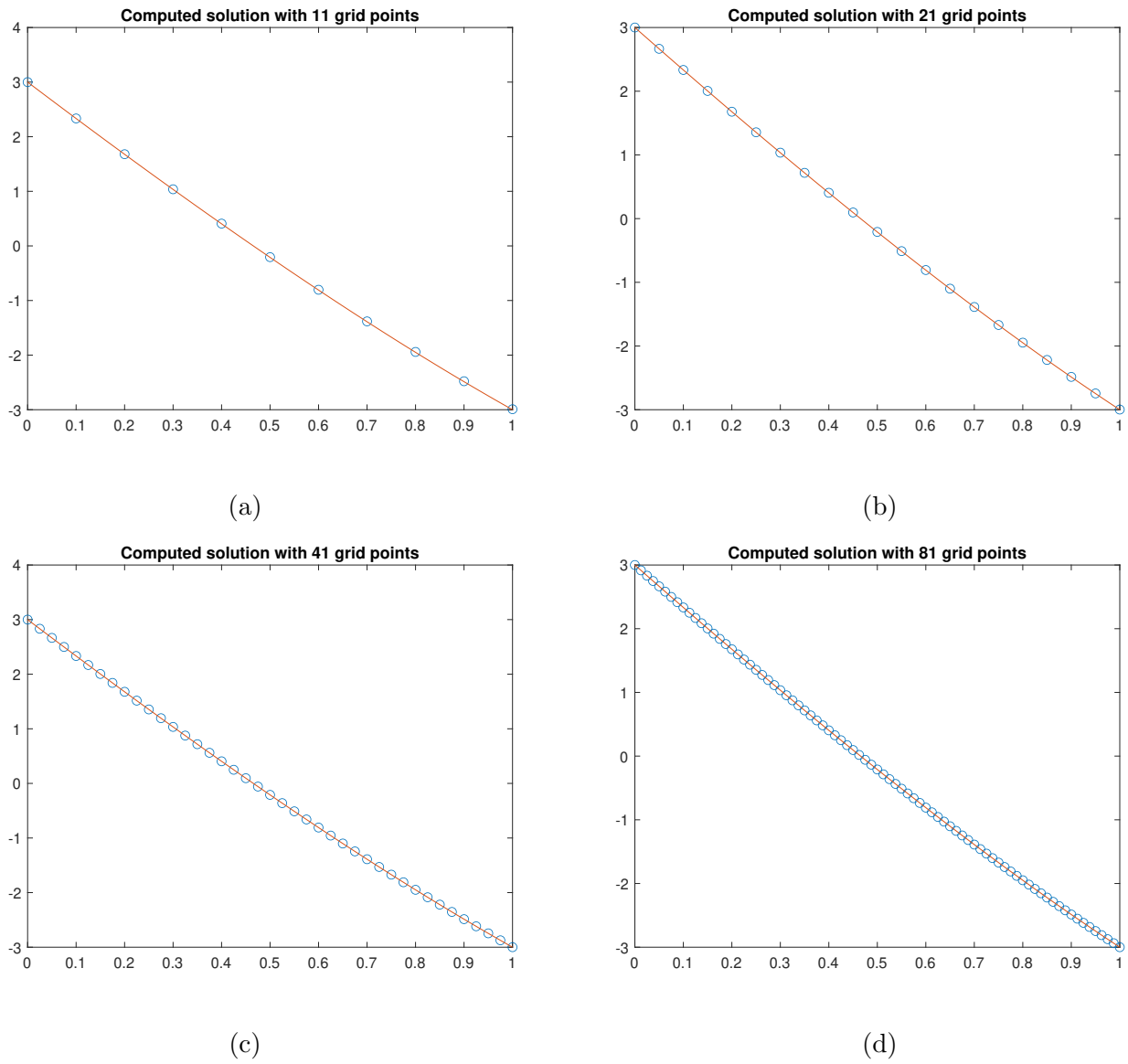
$$C_1 = 3. \tag{4}$$

Therefore, the solution to Eq. (1) is

$$u(x) = e^x - (5 + e)x + 3.$$

Part (b)

Here we solve Eq. (1) using a second order finite different scheme. We display our results in Figs.(1),(2) and Table I. We can readily see that the error is second order accurate.



(a) (b) (c) (d)

FIG. 1. Plot of the analytical solution (line) and the numerical solutions (empty dots) for different number of grid points. The solutions overlap.

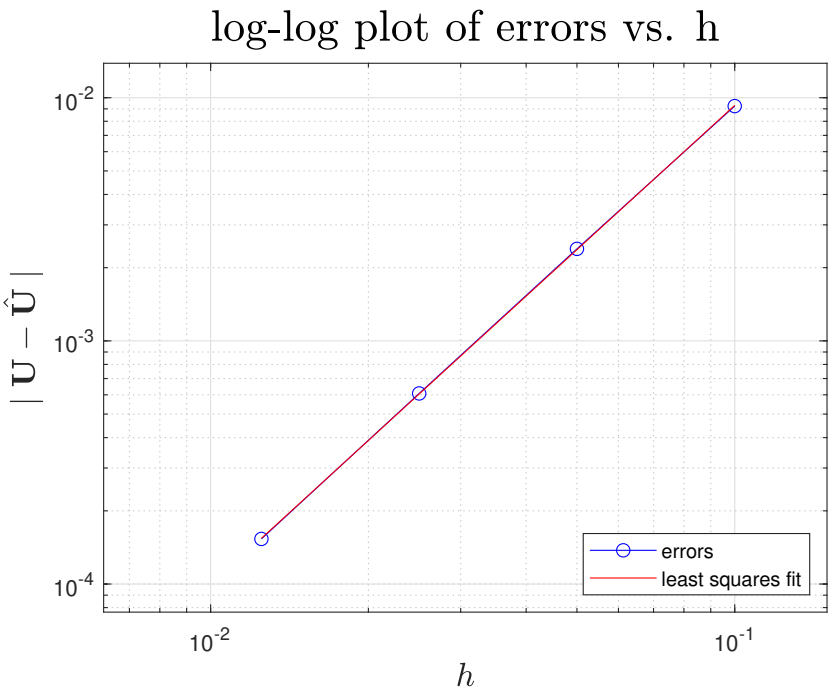
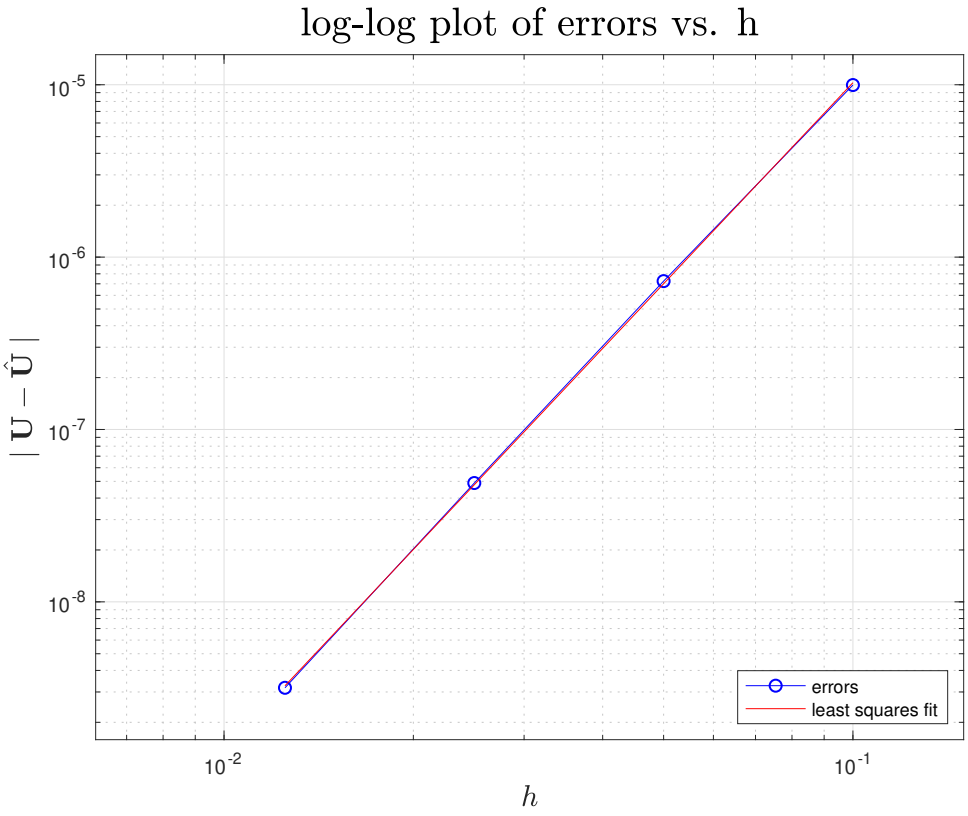


FIG. 2. Plot of the error for the method vs the step-size h in a log-log scale. This plot along with Table I show that the method is second-order accurate.

Part (c)



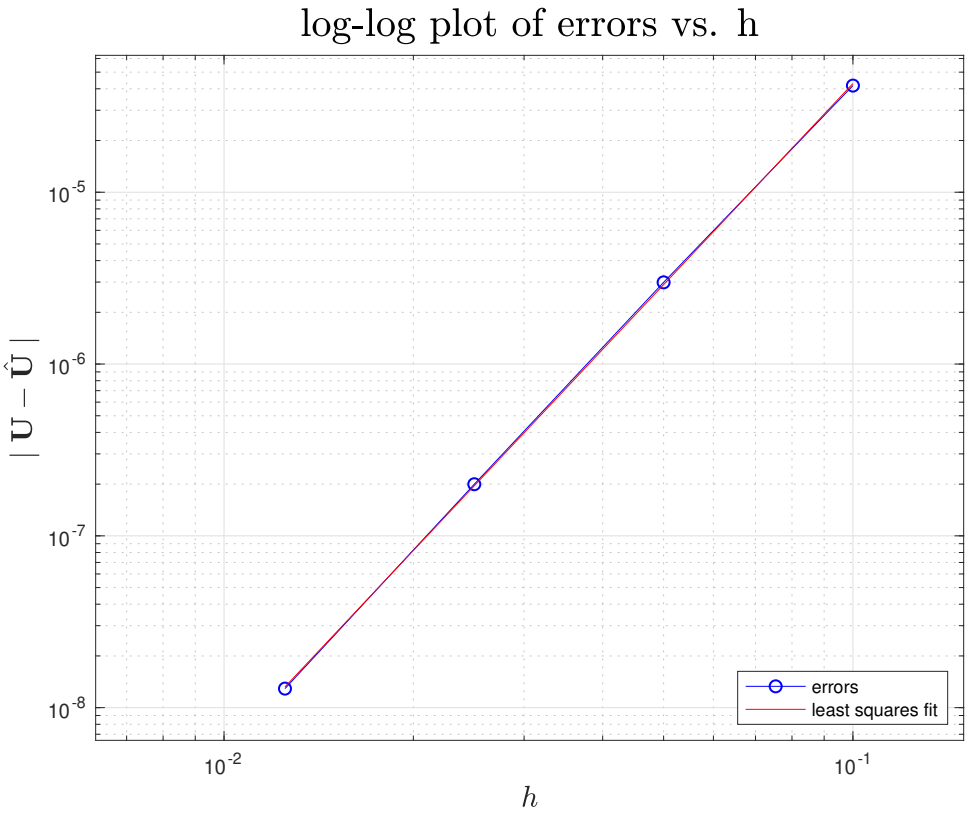
h	error	ratio	observed order
0.10000	9.96133e-06	NaN	NaN
0.05000	7.25623e-07	13.72798	3.77905
0.02500	4.88995e-08	14.83907	3.89133
0.01250	3.17217e-09	15.41515	3.94628

FIG. 3. We are using extrapolation method for this part. Above we have the graph is the log-log plot of the error at different h and a table displaying the h and it's corresponding error, ratio and order of convergence.

To obtain a fourth order using extrapolation, we have to define a fine grid and a coarse grid using our second order accurate order method. The fine grid will have grid width, $\frac{h}{2}$, which is half of the coarse grid. Then we'll obtain a set of solution U and V where U correspond to coarse grid and V correspond to the fine grid. Then the extrapolation tells us that if we match up the error coefficient of the two grids and find the difference, we'll be able to cancel out the first error term and obtain the second error term. For a central finite different, the second error term is fourth order; therefore, we obtain a fourth order method

in the interior points. For our Neumann Boundary condition at $x = 1$, I used a fourth order backward difference to ensure all of my errors are fourth order. The result is verified in Fig. 3., were we can see that using $\frac{1}{3}(4V - U)$ for the interior points indeed give us a fourth order convergence.

Part (d)



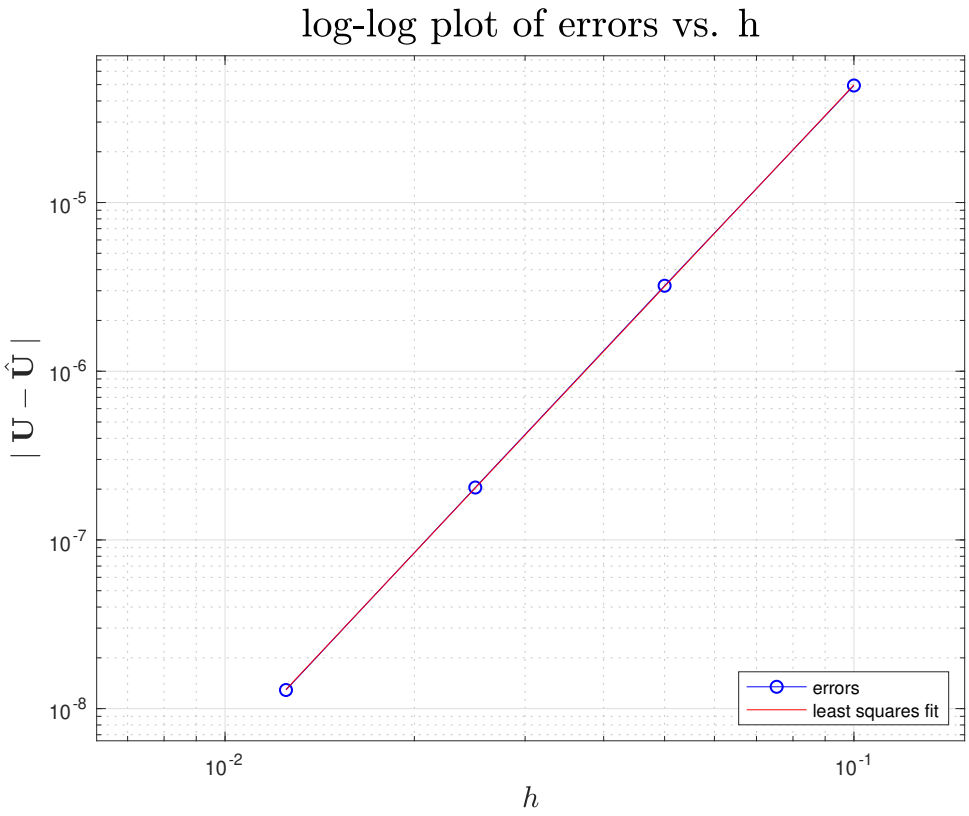
h	error	ratio	observed order
0.10000	4.17571e-05	NaN	NaN
0.05000	2.98834e-06	13.97332	3.80460
0.02500	1.99710e-07	14.96339	3.90336
0.01250	1.29049e-08	15.47550	3.95191

FIG. 4. We are using deferred correction solution for this part. Above we have the graph is the log-log plot of the error at different h and a table displaying the h and it's corresponding error, ratio and order of convergence.

Since our function is relatively easy to derive and therefore, we can obtain $f^{(4)}(x)$ by

simply deriving our function. After doing so, we can simply subtract our second order error term off from our function and obtain a fourth order error function. So, our $AU = F$ becomes $AU = F + \frac{1}{12}h^2F''(x)$ (note that we only add the second derivative to our interior points). In Fig. 4, we display our result from our code we use to verify this result. We indeed obtain a fourth order accuracy.

Part (e)



h	error	ratio	observed order
0.10000	4.94037e-05	NaN	NaN
0.05000	3.21196e-06	15.38120	3.94310
0.02500	2.04472e-07	15.70856	3.97348
0.01250	1.28923e-08	15.86004	3.98732

FIG. 5. We are using fourth order method to estimate function at each grid points. Graph display log-log plot of error versus hour grid width, h. The table display the size of each grid width and its error, ratio, and order of convergence.

For this question, we will be using fourth order accuracy methods to estimate our $u(x)$ at each grid point. Central finite difference for the interior point but since fourth order finite difference requires two grid point to the left and two to the right, the first and last interior point will have one short from the one of the sides. The fix this, we use the only fixed on one side and four more grid point from the other side, a total of 6 grid point, to achieve a fourth order accuracy. The last boundary grid point corresponds to Neumann boundary condition and there fore requires a fourth order accuracy method to approximate the first derivative. This method requires 5 grid points and is a form of backward difference method.

Part (f)

Here we solve again Eq. (1), but we implement a spectral collocation method. We show our results in Figs.(6),(7). The solutions overlap already even with just 9 grid points (Fig.(6)(a)), and from Fig. (7) we readily see how the method reaches machine precision very quickly and then keeps on oscillating around values of the order of 10^{-12}. This is consistent with the theory and shows the spectral accuracy of the method.

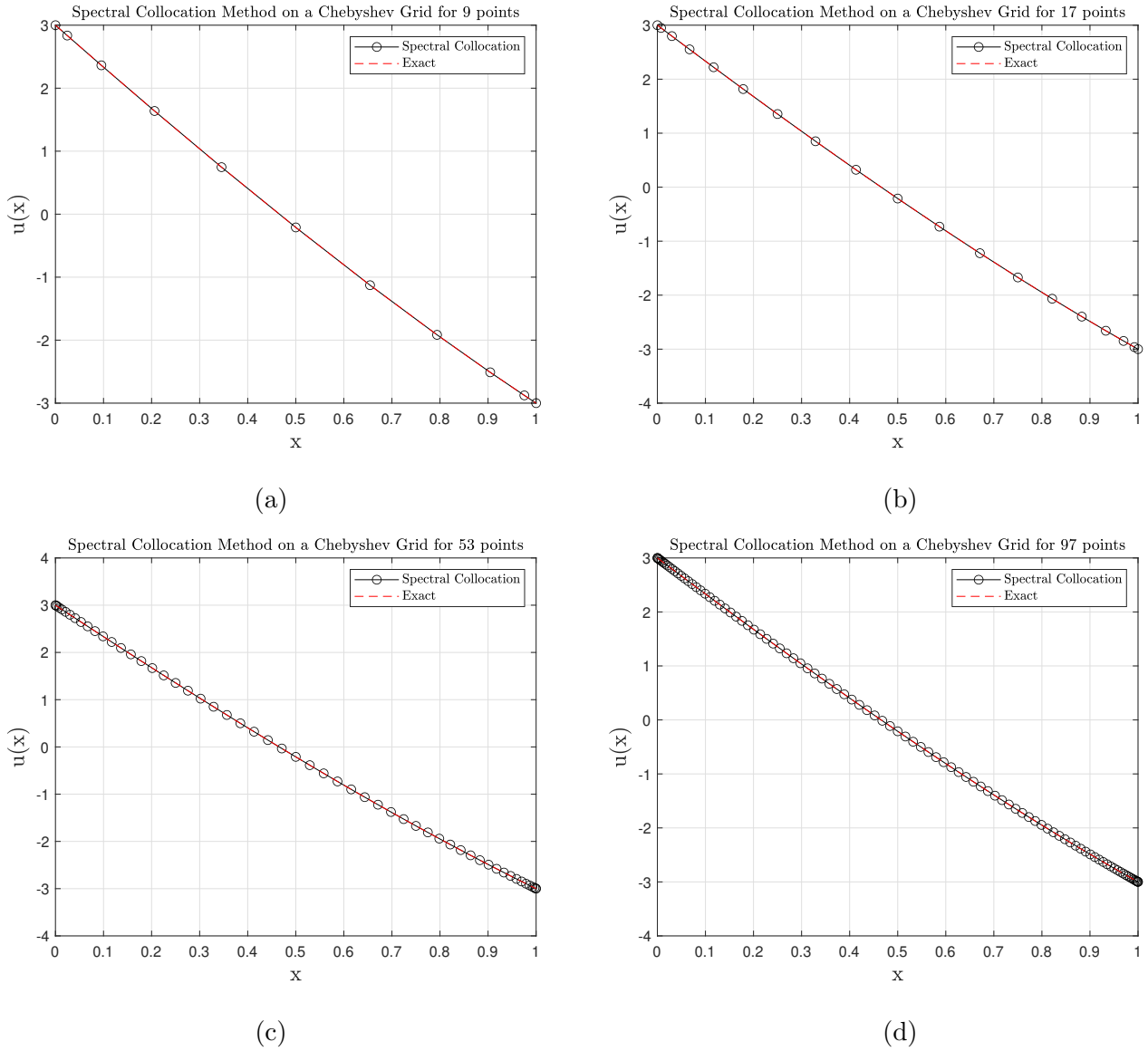


FIG. 6. Plot of the analytical solution (line) and the numerical solutions (empty dots) for different number of grid points. The solutions overlap.

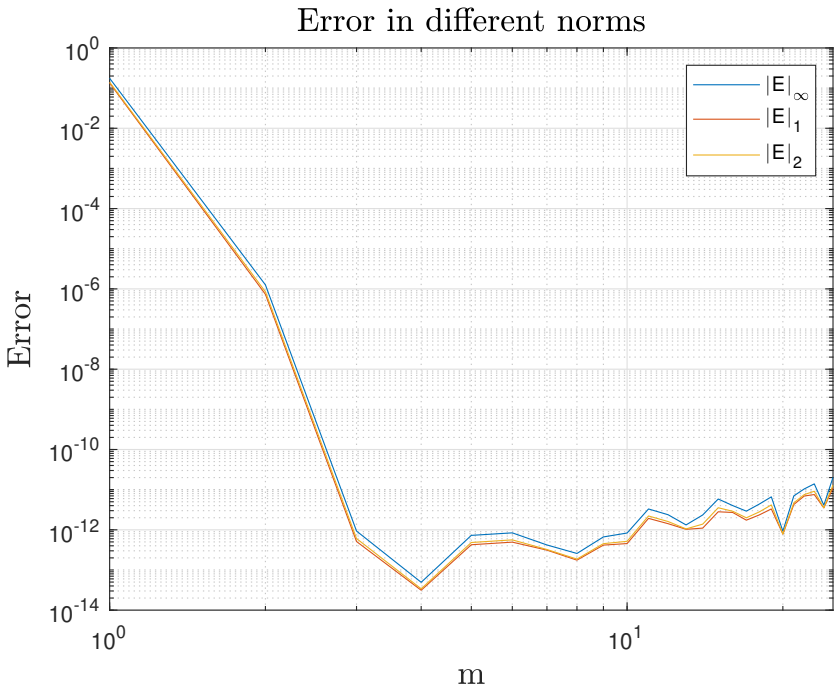


FIG. 7. Plot of the error of the method in the infinity norm (blue line), the 2-norm (yellow line) and the 1-norm (orange line), vs. the total number of grid points m, in a log-log scale. We see that there the overall behavior of the errors is the same and we reach spectral accuracy quite quickly in each case.

MATLAB CODE

Here is the code used to generate the plots for Problem.

```
%
% bvp_2.m
% second order finite difference method for the bvp
%   u''(x) = f(x),   u'(ax)=sigma,   u(bx)=beta
% Using 3-pt differences on an arbitrary nonuniform grid.
% Should be 2nd order accurate if grid points vary smoothly, but may
% degenerate to "first order" on random or nonsmooth grids.
%
% Different BCs can be specified by changing the first and/or last rows of
% A and F.
%
% From  http://www.amath.washington.edu/~rjl/fdmbook/  (2007)

ax = 0;
bx = 1;
sigma = 3;    % Dirichlet boundary condition at ax
beta = -5;      %Neumann boundary condtion at bx

f = @(x) exp(x);  % right hand side function
utrue = @(x) exp(x) - (5+exp(1))*x + 2;%exp(x) + (sigma-exp(ax))*(x - bx) + beta - exp(bx);  % true soln

% true solution on fine grid for plotting:
xfine = linspace(ax,bx,101);
ufine = utrue(xfine);

```

```
% Solve the problem for ntest different grid sizes to test
    convergence:
m1vals = [10 20 40 80];
ntest = length(m1vals);
hvals = zeros(ntest,1);  % to hold h values
E = zeros(ntest,1);      % to hold errors

for jtest=1:ntest
  m1 = m1vals(jtest);
  m2 = m1 + 1;
  m = m1 - 1;                    % number of interior grid points
  hvals(jtest) = (bx-ax)/m1;  % average grid spacing, for
      convergence tests

  % set grid points:
  gridchoice = 'uniform';          % see xgrid.m for other choices
  x = xgrid(ax,bx,m,gridchoice);

  % set up matrix A (using sparse matrix storage):
  A = spalloc(m2,m2,3*m2);   % initialize to zero matrix

  % first row for Dirichlet BC at ax:
  A(1,1:3) = fdcoeffF(0, x(1), x(1:3));

  % interior rows:
  for i=2:m1
      A(i,i-1:i+1) = fdcoeffF(2, x(i), x((i-1):(i+1)));
  end

  % last row for Neumann BC at bx:
  A(m2,m:m2) = fdcoeffF(1,x(m2),x(m:m2));
```

```
55
56    % Right hand side:
57    F = f(x);
58    F(1) = sigma;
59    F(m2) = beta;
60
61    % solve linear system:
62    U = A\F;
63
64
65    % compute error at grid points:
66    uhat = utrue(x);
67    err = U - uhat;
68    E(jtest) = max(abs(err));
69    disp(' ')
70    fprintf('Error with %i points is %9.5e\n',m2,E(jtest))
71
72 %    clf
73    plot(x,U,'o')  % plot computed solution
74    title(sprintf('Computed solution with %i grid points',m2));
75    hold on
76    plot(xfine,ufine)  % plot true solution
77    hold off
78
79    %pause to see this plot:
80    drawnow
81    input('Hit <return> to continue ');
82
83    end
84
85 error_table(hvals, E);   % print tables of errors and ratios
```

```
86 error_loglog(hvals, E); % produce log-log plot of errors and
       least squares fit
```

```
clc; close all; clear all;
%
% bvp_2.m
% second order finite difference method for the bvp
%   u''(x) = f(x),   u'(ax)=sigma,   u(bx)=beta
% Using 3-pt differences on an arbitrary nonuniform grid.
% Should be 2nd order accurate if grid points vary smoothly, but
    may
% degenerate to "first order" on random or nonsmooth grids.
%
% Different BCs can be specified by changing the first and/or last
     rows of
% A and F.
%
% From  http://www.amath.washington.edu/~rjl/fdmbook/  (2007)

ax = 0;
bx = 1;
sigma = 3;   % Dirichlet boundary condition at ax
beta = -5;     %Neumann boundary condtion at bx

f = @(x) exp(x);  % right hand side function
utrue = @(x) exp(x) - (5+exp(1))*x + 2;%exp(x) + (sigma-exp(ax))*(
    x - bx) + beta - exp(bx);  % true soln

% true solution on fine grid for plotting:
xfine = linspace(ax,bx,101);
ufine = utrue(xfine);

% Solve the problem for ntest different grid sizes to test
    convergence:
```

```
28 m1vals = [10 20 40 80];
29 ntest = length(m1vals);
30 hvals = zeros(ntest,1);  % to hold h values
31 hvals2 = zeros(ntest,1);
32 E = zeros(ntest,1);       % to hold errors
33
34 for jtest=1:ntest
35   m1 = m1vals(jtest);
36   m2 = m1 + 1;
37   m = m1 - 1;              % number of interior grid points
38   hvals(jtest) = (bx-ax)/m1;  % average grid spacing, for
       convergence tests
39
40   % Finer Grids (finding V)
41   n1 = 2*m1vals(jtest);
42   n2 = n1 + 1;
43   n = n1 - 1;
44   hvals2(jtest) = (bx-ax)/n1;
45
46   % set grid points:
47   gridchoice = 'uniform';          % see xgrid.m for other choices
48   x = xgrid(ax,bx,m,gridchoice);
49   x2 = xgrid(ax,bx,n,gridchoice); %% FIner grid
50
51   % set up matrix A (using sparse matrix storage):
52   A = spalloc(m2,m2,3*m2);   % initialize to zero matrix
53   A2 = spalloc(n2,n2,3*n2);
54
55   % first row for Dirichlet BC at ax:
56   A(1,1:3) = fdcoeffF(0, x(1), x(1:3));
57   A2(1,1:3) = fdcoeffF(0, x(1), x(1:3));
```

```
58
59  % interior rows:
60  for i=2:m1
61      A(i,i-1:i+1) = fdcoeffF(2, x(i), x((i-1):(i+1)));
62  end
63
64  for i=2:n1
65      A2(i,i-1:i+1) = fdcoeffF(2, x2(i), x2((i-1):(i+1)));
66  end
67
68  % last row for Neumann BC at bx:
69  A(m2,m-2:m2) = fdcoeffF(1,x(m2),x(m-2:m2));
70  A2(n2,n-2:n2) = fdcoeffF(1,x2(n2),x2(n-2:n2));
71
72  % Right hand side:
73  F = f(x);
74  F(1) = sigma;
75  F(m2) = beta;
76
77  F2 = f(x2);
78  F2(1) = sigma;
79  F2(n2) = beta;
80
81  % solve linear system:
82  U = A\F;
83  V = A2\F2;
84  U(2:end-1) = (4*V(3:2:end-2)-U(2:end-1))/3;
85
86  % compute error at grid points:
87  uhat = utrue(x);
88  err = U - uhat;
```

```
89      err(end) = 0;
90      E(jtest) = max(abs(err));
91      disp(' ')
92      fprintf('Error with %i points is %9.5e\n',m2,E(jtest))
93
94  %     clf
95      plot(x,U,'o')  % plot computed solution
96      title(sprintf('Computed solution with %i grid points',m2));
97      hold on
98      plot(xfine,ufine)  % plot true solution
99      hold off
100
101     %pause to see this plot:
102     drawnow
103     input('Hit <return> to continue ');
104
105     end
106
107 error_table(hvals, E);   % print tables of errors and ratios
108 error_loglog(hvals, E);  % produce log-log plot of errors and
        least squares fit
```

```
clc; close all; clear all;
%
% bvp_2.m
% second order finite difference method for the bvp
%   u''(x) = f(x),   u'(ax)=sigma,   u(bx)=beta
% Using 3-pt differences on an arbitrary nonuniform grid.
% Should be 2nd order accurate if grid points vary smoothly, but
    may
% degenerate to "first order" on random or nonsmooth grids.
%
% Different BCs can be specified by changing the first and/or last
     rows of
% A and F.
%
% From  http://www.amath.washington.edu/~rjl/fdmbook/  (2007)

ax = 0;
bx = 1;
sigma = 3;   % Dirichlet boundary condition at ax
beta = -5;     %Neumann boundary condtion at bx

f = @(x) exp(x);  % right hand side function
utrue = @(x) exp(x) - (5+exp(1))*x + 2;%exp(x) + (sigma-exp(ax))*(
    x - bx) + beta - exp(bx);  % true soln

% true solution on fine grid for plotting:
xfine = linspace(ax,bx,101);
ufine = utrue(xfine);

% Solve the problem for ntest different grid sizes to test
    convergence:
```

```
28 m1vals = [10 20 40 80];
29 ntest = length(m1vals);
30 hvals = zeros(ntest,1);  % to hold h values
31 hvals2 = zeros(ntest,1);
32 E = zeros(ntest,1);       % to hold errors
33
34 for jtest=1:ntest
35   m1 = m1vals(jtest);
36   m2 = m1 + 1;
37   m = m1 - 1;              % number of interior grid points
38   hvals(jtest) = (bx-ax)/m1;  % average grid spacing, for
       convergence tests
39
40   % set grid points:
41   gridchoice = 'uniform';          % see xgrid.m for other choices
42   x = xgrid(ax,bx,m,gridchoice);  %% FIner grid
43
44   % set up matrix A (using sparse matrix storage):
45   A = spalloc(m2,m2,3*m2);   % initialize to zero matrix
46
47   % first row for Dirichlet BC at ax:
48   A(1,1:3) = fdcoeffF(0, x(1), x(1:3));
49
50   % interior rows:
51   for i=2:m1
52       A(i,i-1:i+1) = fdcoeffF(2, x(i), x((i-1):(i+1)));
53   end
54
55   % last row for Neumann BC at bx:
56   A(m2,m-2:m2) = fdcoeffF(1,x(m2),x(m-2:m2));
57
```

```
58    % Right hand side:
59    F = f(x)+hvals(jtest)^2*f(x)/12;
60    F(1) = sigma;
61    F(m2) = beta;
62
63
64    % solve linear system:
65    U = A\F;
66
67    % compute error at grid points:
68    uhat = utrue(x);
69    err = U - uhat;
70    err(end) = 0;
71    E(jtest) = max(abs(err));
72    disp(' ')
73    fprintf('Error with %i points is %9.5e\n',m2,E(jtest))
74
75 %    clf
76    plot(x,U,'o')  % plot computed solution
77    title(sprintf('Computed solution with %i grid points',m2));
78    hold on
79    plot(xfine,ufine)  % plot true solution
80    hold off
81
82    %pause to see this plot:
83    drawnow
84    input('Hit <return> to continue ');
85
86    end
87
88 error_table(hvals, E);   % print tables of errors and ratios
```

```
89 error_loglog(hvals, E); % produce log-log plot of errors and
       least squares fit
```

```
%
% ------------------------------
% MATLAB: SpectralMethod1.m
% ------------------------------
%
% This matlab code computes the solution to the Dirichlet, two-
    point
% boundary value problem of the form
%
% u'' = exp(x) on [0,1], u(0) = 3, u(1) = -5
%
% using a spectral collocation method on a uniform grid.
%

clear all;

% set the values of the Dirichlet and Neumann boundary conditions

alpha = 3.0;
beta  =  -5.0;

ax = 0;
bx = 1;

% set the number of interior grid points

mvals = 1:4:100;
for ii = 1:length(mvals)
 m = mvals(ii);
% compute the mesh width

```

```
31 h = (bx-ax) / ( m + 1.0 );
32
33 % compute the grid
34
35 %x = 0 : h : 3.0;
36
37 gridchoice = 'chebyshev';          % see xgrid.m for other choices
38 x = xgrid(ax,bx,m,gridchoice);
39
40 % compute the spectral collocation matrix
41
42 A = zeros( m + 2 );
43
44 % set the Dirichlet BC at x = 0
45
46 A(1,1) = 1.0;
47
48 % compute the interior rows
49
50 for j = 2 : m + 1
51
52     A(j,:) = fdcoeffF( 2, x(j), x );
53
54 end;
55
56 % set the Neumann BC at x = 3
57
58 A(m+2,1:m+2) = fdcoeffF(1,x(m+2),x);
59
60 % compute the right-hand side
61
```

```
62 f       = zeros( m + 2, 1 );
63 f       = exp( x );
64 f(1)    = alpha;
65 f(m+2) = beta;
66
67 % solve the linear system of equations
68
69 u = A \ f;
70
71 % compute the exact solution
72
73 %C0 = alpha - 1.0;
74 %C1 = ( beta - alpha + 1.0 - exp( 3.0 ) ) / 3.0;
75
76 %uexact = C0 + C1 * x + exp( x );
77 uexact = exp(x) - (5+exp(1))*x + 2;
78
79 error_inf(ii) = max( abs( uexact - u ) );
80 error_one(ii) = h * sum( abs( uexact - u ) );
81 error_two(ii) = sqrt( h * sum( abs( uexact - u ).^2 ) );
82 %plot the solution
83
84 figure(ii)
85 plot( x, u, 'ko-', x, uexact, 'r--' );
86 xlabel('x','interpreter','latex','fontsize',15 );
87 ylabel('u(x)','interpreter','latex','fontsize',15 );
88 grid on
89 title( sprintf('Spectral Collocation Method on a Chebyshev Grid
      for %i points',m),'interpreter','latex');
90 legend( 'Spectral Collocation','Exact','interpreter','latex')
91 end
```

```
92
93  figure(ii+1)
94  errgrid = 1:length(mvals);
95  loglog(errgrid,error_inf)
96  hold on
97  loglog(errgrid,error_one)
98  hold on
99  loglog(errgrid,error_two)
100 grid on
101 xlabel('m','interpreter','latex','fontsize',15)
102 ylabel('Error','interpreter','latex','fontsize',15)
103 title('Error in different norms','interpreter','latex','fontsize'
      ,15)
104 legend('\mid{E}\mid_{\infty}','\mid{E}\mid_{1}','\mid{E}\mid_{2}')
```