Instructor: P. Choboter
Name: Joseph McGuire

Class: Math 502 (W2022)

Homework # 1

Exercise 0

In a short (less than half a page) paragraph, tell me about yourself and what you are hoping to get out of this course. For example, is there a numerical component to your thesis research? Is there a certain topic you were hoping to cover this quarter?

Answer. Hi my name is Joseph, I'm a grad student in math. I'm doing my thesis on community structure detection in networks, I'm hoping to learn some numerical methods applicable to machine learning algorithms and general optimization problems. □

Exercise A.1

Refer to Appendix A.3. Let $e = (2, -5, 3, 4)$. Calculate the following quantities:

(a) $\|e\|_\infty$

(b) $\|e\|_1$

(c) $\|e\|_2$

Solution. (a)

$$\|e\|_\infty = \|(2, -5, 3, 4)\|_\infty = \max\{|2|, |-5|, |3|, |4|\} = 5$$

(b)

$$\|e\|_1 = \|(2, -5, 3, 4)\|_1 = |2| + |-5| + |3| + |4| = 14$$

(c)

$$\|e\|_2 = \|(2, -5, 3, 4\|_2 = \sqrt{2^2 + (-5)^2 + 3^2 + 4^2} = \sqrt{4 + 25 + 9 + 16} = \sqrt{54}$$

□

Exercise A.2

Refer to Appendix A.3. Let $A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$. Use the definitions stated in Appendix $A.3$ to calculate the following quantities:

(a) $\|A\|_\infty$

(b) $\|A\|_1$

Homework # 1

(c) $\|A\|_2$

By all means, use MATLAB to help with calculations. But do not use the norm command. For part (c), you may find the commands eig and sqrt useful.

Solution. 1.

$$\|A\|_\infty = \max_{1\le i\le 4} \sum_{j=1}^{4} |a_{ij}| = |13| + |14| + |15| + |16| = 58$$

2.

$$\|A\|_1 = \max_{1\le j\le 4} \sum_{i=1}^{4} |a_{ij}| = |4| + |8| + |12| + |16| = 40$$

3.

$$\|A\|_2 = \sqrt{\rho(A^T A)} = 38.6227$$

Matlab code attached.

□

Exercise A.3

Refer to Appendix A.4. Let $u(x) = x^2 - x$ be a function defined on $0 \le x \le 1$. Calculate

(a) $\|u\|_\infty$
(b) $\|u\|_1$
(c) $\|u\|_2$

Solution. (a)

$$\|u\|_\infty = \max_{x\in[0,1]} |u(x)|$$

we know that $x^2 \le x$ for $x \in [0, 1]$ hence:

$$\max_{x\in[0,1]} |u(x)| = \max_{[0,1]} x - x^2,$$

taking the first derivative and setting it equal to zero will give us the maximum and the maximum of this will occur at:

$$1 - 2x = 0 \iff \frac{1}{2} = x$$

and $|u(1/2)| = |1/4 - 1/2| = |-1/4| = 1/4$.

Homework # 1

(b) $\|u\|_1 = \int_0^1 x - x^2 \, dx = \frac{x^2}{2} - \frac{x^3}{3}\Big|_{x=0}^{x=1} = \frac{1}{2} - \frac{1}{3} = \frac{1}{6}$

(c) $\|u\|_2 = \sqrt{\int_0^1 (x-x^2)^2 \, dx} = \sqrt{\int_0^1 x^4 - 2x^3 + x^2 \, dx} = \sqrt{\frac{1}{5} - \frac{2}{4} + \frac{1}{3}} = \frac{1}{\sqrt{30}}$

□

Exercise A.4

Refer to Appendix A.5.

Let U be the grid function defined by $U_i = x_i^2 - x_i$, where $x_1 = 0.2, x_2 = 0.4, x_3 = 0.6, x_4 = 0.8, x_5 = 1.0$. Calculate the following quantities:

1. $\|U\|_\infty$
2. $\|U\|_1$
3. $\|U\|_2$

Solution. Matlab code attached.

(a)
$$\|U\|_\infty = \max_{i=1,2,3,4,5} |U_i| = 0.24$$

(b)
$$\|U\|_1 = 0.2 \sum_{i=1}^{4} |U_i| = 0.6$$

(c)
$$\|U\|_2 = \sqrt{0.2 \sum_{i=1}^{4} |U_i|^2} = 0.6633$$

□

Exercise 1.2

(a) Use the method of undetermined coefficients to set up the 5×5 matrix ("Vandermonde") system that would determine a fourth-order accurate finite difference approximation to $u''(x)$ based on 5 equally spaced points,

$u''(x) = c_{-2}u(x-2h) + c_{-1}u(x-h) + c_0 u(x) + c_1 u(x+h) + c_2 u(x+2h) + O(h^4)$.

Homework # 1

(b) Compute the coefficients using the MATLAB code fdstencil.m (available either for from the textbook's website or from our Canvas site), and check that they satisfy the system you determined in part (a).

(c) Test this finite difference formula to approximate $u''(1)$ for $u(x) = \sin(2x)$ with values of h from the array hvals = logspace(-1, -4, 13). Make a table of the error vs. h for several values of h and compare against the predicted error from the leading term of the expression printed by fdstencil. You may want to look at the m-file chap1example1.m for guidance on how to make such a table. Also produce a log-log plot of the absolute value of the error vs. h. You should observe the predicted accuracy for larger values of h. For smaller values numerical cancellation in computing the linear combination of u values impact the accuracy observed.

(a) *Solution.* Via a Taylor expansion in h of $u(x+h)$ we have that:

$$u(x+h) = u(x) + hu'(x) + \frac{1}{2}h^2u''(x) + \frac{1}{6}h^3u'''(x) + \frac{1}{24}h^4u^{(4)}(x) + O(h^5)$$

$$u(x+2h) = u(x) + 2hu'(x) + 2h^2u''(x) + \frac{4}{3}h^3u'''(x) + \frac{2}{3}h^4u^{(4)}(x) + O(h^5)$$

$$u(x-h) = u(x) - hu'(x) + \frac{1}{2}h^2u''(x) - \frac{1}{6}h^3u'''(x) + \frac{1}{24}h^4u^{(4)}(x) + O(h^5)$$

$$u(x-2h) = u(x) - 2hu'(x) + 2h^2u''(x) - \frac{4}{3}h^3u'''(x) + \frac{2}{3}h^4u^{(4)}(x) + O(h^5)$$

So that substituting these into the scheme above and collecting coefficients:

$$\begin{aligned}
u''(x) &= u(x)(c_{-2} + c_{-1} + c_0 + c_1 + c_2) \\
&\quad + u'(x)(-hc_{-1} - 2hc_{-2} + hc_1 + 2hc_2) \\
&\quad + u''(x)(2h^2c_{-2} + \frac{1}{2}h^2c_{-1} + \frac{1}{2}h^2c_1 + 2h^2c_2) \\
&\quad + u'''(x)\left(-\frac{4}{3}h^3c_{-2} - \frac{1}{6}h^3c_{-1} + \frac{1}{6}h^3c_1 + \frac{4}{3}h^3c_2\right) + \\
&\quad + u^{(4)}(x)\left(\frac{2}{3}h^4c_{-2} + \frac{1}{24}h^4c_{-1} + \frac{1}{24}h^4c_1 + \frac{2}{3}h^4c_2\right) + O(h^5)
\end{aligned}$$

Homework # 1

In matrix form this is the equation:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 2 & \frac{1}{2} & 0 & \frac{1}{2} & 2 \\ -\frac{4}{3} & -\frac{1}{6} & 0 & \frac{1}{6} & \frac{4}{3} \\ \frac{2}{3} & \frac{1}{24} & 0 & \frac{1}{24} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} c_{-2} \\ c_{-1} \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1/h^2 \\ 0 \\ 0 \end{bmatrix}$$

Let A be the coefficient matrix above then this is:

$$\begin{bmatrix} c_{-2} \\ c_{-1} \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = A^{-1}[0, 0, 1/h^2, 0, 0]^T$$

Solving for A^{-1} using Matlab code attached, we have that:

$$\begin{bmatrix} c_{-2} \\ c_{-1} \\ c_0 \\ c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} -0.0833 \\ 1.3333 \\ -2.5 \\ 1.3333 \\ -0.0833 \end{bmatrix}$$

□

(b) Running fdstencil for a second derivative 5 step scheme, yields us the same results! Matlab code attached

(c) See the code attached. We see that with predicted error and absolute error follow very closely as $h \to 0$, however diverge when absolute error nears $O(10^{-12})$ where because of rounding error the two diverge and absolute error begins to increase at this point.

Table of Contents

Problem A.2

```
% Establish the Matrix
A = [1,2,3,4;
    5,6,7,8;
    9,10,11,12;
    13,14,15,16];
% Set it's transpose
AT = A.';
% Find the product of A^T A
product = AT * A;
% Find the maximum eigenvalue of A
e = max(eig(product));
% Take the square root
A_2 = sqrt(e)
```

```
A_2 =

   38.6227
```

Problem A.4

```
x = 0.2:0.2:1;
% Compute the Grid Norms
U_infinity = max(abs(x.^2 - x))
U_1 = 0.2 * sum(abs(x))
U_2 = sqrt(0.2 * sum(abs(x.^2 - x).^2))
```

```
U_infinity =

    0.2400
```

```
U_1 =

    0.6000
```

```
U_2 =
```

```
    0.1824
```

Problem 1.2.a

```
B = [1,1,1,1,1;
    -2,-1, 0, 1, 2;
    2, 1/2, 0, 1/2, 2;
    -4/3, -1/6, 0, 1/6, 4/3;
    2/3, 1/24, 0, 1/24, 2/3]

inv(B)
```

```
B =

    1.0000    1.0000    1.0000    1.0000    1.0000
   -2.0000   -1.0000         0    1.0000    2.0000
    2.0000    0.5000         0    0.5000    2.0000
   -1.3333   -0.1667         0    0.1667    1.3333
    0.6667    0.0417         0    0.0417    0.6667


ans =

    0.0000    0.0833   -0.0833   -0.5000    1.0000
   -0.0000   -0.6667    1.3333    1.0000   -4.0000
    1.0000   -0.0000   -2.5000   -0.0000    6.0000
   -0.0000    0.6667    1.3333   -1.0000   -4.0000
    0.0000   -0.0833   -0.0833    0.5000    1.0000
```

Problem 1.2.b

```
fdstencil(2,-2:2)
```

```
The derivative u^(2) of u at x0 is approximated by

          1/h^2 * [
                     -8.333333333333333e-02 * u(x0-2*h) +
                      1.333333333333333e+00 * u(x0-1*h) +
                     -2.500000000000000e+00 * u(x0) +
                      1.333333333333333e+00 * u(x0+1*h) +
                     -8.333333333333333e-02 * u(x0+2*h) ]

For smooth u,
       Error = 0 * h^3*u^(5) + -0.0111111 * h^4*u^(6) + ...
```

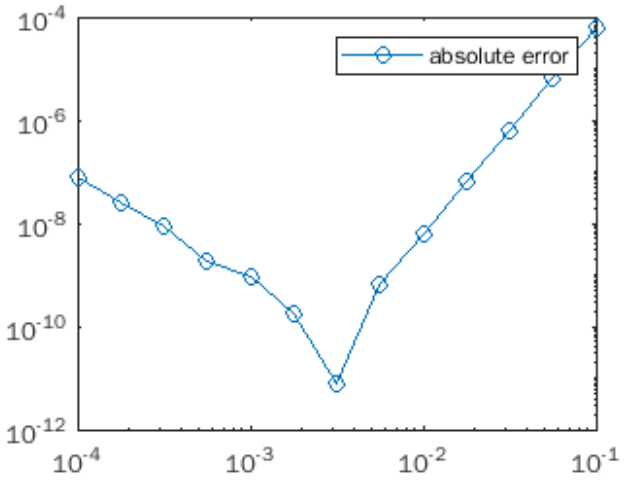
Problem 1.2.c

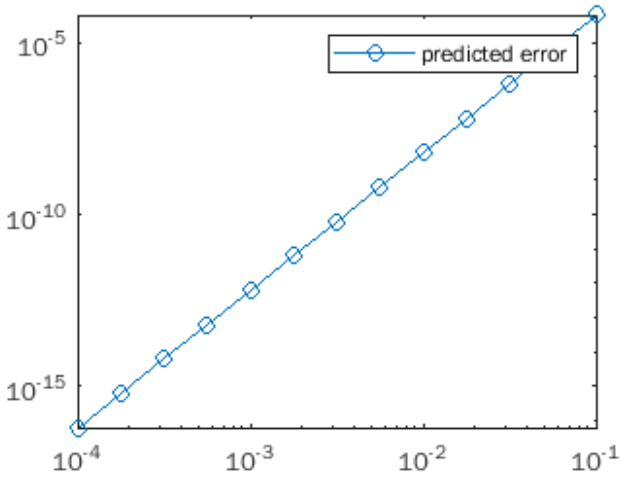
```
% Establish the steps
```

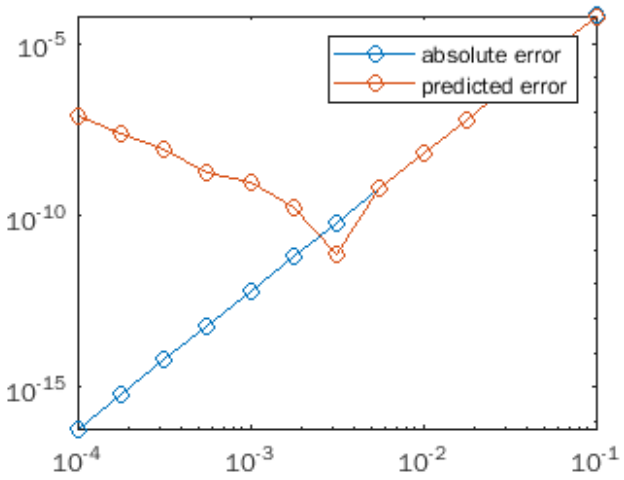
```
h = logspace(-1,-4,13);
% Initialize Arrays
absolute_error = double(1:length(h));
predicted_error = double(1:length(h));
disp('h     , absolute error,      predicted_error')
for i = 1:length(h)
    % setups up the stencil around 1
    x_stencil = [1 - 2*h(i), 1 - h(i), 1, 1 + h(i), 1 + 2*h(i)];
    % Formula given from fdstencil
    u_double_prime = 1/h(i)^2 * [-8.333333333333333e-02 * sin(2 *
 x_stencil(1)) +...
                      1.333333333333333e+00 * sin(2 * x_stencil(2)) + ...
                     -2.500000000000000e+00 * sin(2 * x_stencil(3)) + ...
                      1.333333333333333e+00 * sin(2 * x_stencil(4)) + ...
                     -8.333333333333333e-02 * sin(2 * x_stencil(5)) ];
    absolute_error(i) = abs(-4 * sin(2) - u_double_prime);
    predicted_error(i) = (-64 * sin(2) * -0.0111111 * h(i)^4);
    disp(fprintf('  %e   %e      %e', h(i) , absolute_error(i),
 predicted_error(i)))
end
% Make pretty graphs
clf
figure(1)
loglog(h,absolute_error, '-O')
legend('absolute error')
figure(2)
loglog(h,predicted_error, '-o')
legend('predicted error')
figure(3)
loglog(h,predicted_error, '-o',h,absolute_error, '-O')
legend('absolute error', 'predicted error')
```

```
h     , absolute error,      predicted_error
  1.000000e-01   6.443065e-05      6.466109e-05    47

  5.623413e-02   6.458816e-06      6.466109e-06    47

  3.162278e-02   6.463798e-07      6.466109e-07    47

  1.778279e-02   6.465148e-08      6.466109e-08    47

  1.000000e-02   6.456361e-09      6.466109e-09    47

  5.623413e-03   6.234884e-10      6.466109e-10    47

  3.162278e-03   7.630785e-12      6.466109e-11    47

  1.778279e-03   1.796154e-10      6.466109e-12    47

  1.000000e-03   9.471615e-10      6.466109e-13    47

  5.623413e-04   1.886757e-09      6.466109e-14    47

  3.162278e-04   8.774234e-09      6.466109e-15    47
```

```
1.778279e-04    2.396112e-08        6.466109e-16     47

1.000000e-04    7.816317e-08        6.466109e-17     47
```







Published with MATLAB® R2021b

Instructor: P. Choboter
Name: Joseph McGuire

Class: Math 502 (W2022)

Homework #2

Exercise 2.2 Green's function with Neumann boundary conditions.

(a) Determine the Green's functions for the two-point boundary value problem $u''(x) = f(x)$ on $0 < x < 1$ with Neumann boundary conditions at $x = 0$ and a Dirichlet condition at $x = 1$, i.e. find the function $G(x; \bar{x})$ solving

$$G''(x; \bar{x}) = \delta(x - \bar{x}), \qquad G'(0) = 0, \qquad G(1) = 0$$

the function $G_0(x)$ solving

$$G_0''(x) = 0, \qquad G_0'(0) = 1, \qquad G_0(1) = 0$$

and the function $G_1(x)$ solving

$$G_1''(x) = 0, \qquad G_1'(0) = 0, \qquad G_1(1) = 1.$$

Solution. First, we'll tackle $G''(x; \bar{x}) = \delta(x - \bar{x})$ on $x \neq \bar{x}$ this gives solutions of:

$$G(x; \bar{x}) = \begin{cases} Ax + B & 0 < x < \bar{x} \\ Cx + D & \bar{x} < x < 1 \end{cases}.$$

Imposing $G'(0) = 0$ and $G(1) = 0$ gives us:

$$A = 0 \qquad D = -C.$$

Imposing continuity at $x \to \bar{x}$ gives:

$$B = C\bar{x} - C.$$

Then imposing the jump condition at $x \to \bar{x}$:

$$C = 1$$

so that:

$$G(x; \bar{x}) = \begin{cases} \bar{x} - 1 & 0 \leq x \leq \bar{x} \\ x - 1 & \bar{x} \leq x \leq 1 \end{cases}.$$

For G_0, we'll similarly get:

$$G_0(x) = Ax + B,$$

imposing $G_0'(0) = 1$ gives:

$$A = 1$$

Homework #2

and then $G_0(1) = 0$:

$$A + B = 0.$$

So that:

$$G_0(x) = x - 1$$

Then for $G_1(x) = Cx + D$ we impose $G_1'(0) = 0$:

$$C = 0$$

and then $G_1(1) = 1$:

$$C + D = 1$$

So that we have:

$$G_1(x) = 1$$

□

(b) Using this as guidance, find the general formulas for the elements of the inverse of the matrix in equation (2.54). Write out the 5×5 matrices A and A^{-1} for the case $h = 0.25$

Solution. For a general mixed (Dirichlet-Neumann) ODE, we can use (a) as guidance and should find that, the columns of A^{-1} will be given by:

$$(A^{-1})_{i0} = G_0(x_i)$$

$$(A^{-1})_{i(m+1)} = G_1(x_i)$$

$$(A^{-1})_{ij} = hG(x_i; x_j) = \begin{cases} hG_L(x_i; x_j) & i = 1, \ldots, j \\ hG_R(x_i; x_j) & i = j+1, \ldots, m \end{cases},$$

where G_L is definition of G on $[a, \bar{x}]$ and G_R is the definition of G on $[\bar{x}, b]$, assuming we have a similar BVP on the interval $[a, b]$.

So in our particular case we'll find that:

$$A^{-1} = \begin{bmatrix} -1 & -0.1875 & -0.1250 & -0.0625 & 1 \\ -0.75 & -0.1875 & -0.1250 & -0.0625 & 1 \\ -0.5 & -0.1250 & -0.1250 & -0.0625 & 1 \\ -0.25 & -0.0625 & -0.0625 & -0.0625 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Homework #2

$$A = \begin{bmatrix} -4 & 4 & 0 & 0 & 0 \\ 16 & -32 & 16 & 0 & 0 \\ 0 & 16 & -32 & 16 & 0 \\ 0 & 0 & 16 & -32 & 16 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

□

Exercise 2.3 (*solvability condition for Neumann problem*)

Determine the null space of the matrix A^T, where A is given in equation (2.58), and verify that the condition (2.62) must hold for the linear system to have solutions.

Solution. With A given in equation 2.58 (Levesque), we have that

$$A^T = \frac{1}{h^2}\begin{bmatrix} -h & 1 & & & & & \\ h & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & h \\ & & & & & 1 & -h \end{bmatrix},$$

note that row-reduction preserves the null space, so we will use Gaussian elimination to determine the null space of A^T.

$$R_2 \to R_1 + R_2 : \begin{bmatrix} -h & 1 & & & & & \\ & -1 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & h \\ & & & & & 1 & -h \end{bmatrix},$$

then for $i = 2, \ldots, m$,

$$R_{i+1} \to R_i + R_{i+1} : \begin{bmatrix} -h & 1 & & & & & \\ & -1 & 1 & & & & \\ & & -1 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & & & -1 & h \\ & & & & & & \end{bmatrix}.$$

The nullspace of this then must satisfy:

$$\text{Null}(A^T)U = 0$$

this is equivalently:

$$\begin{aligned} hU_0 &= U_1 \\ U_1 &= U_2 \\ \vdots &\quad \vdots \\ U_{m-1} &= U_m \\ U_m &= hU_{m+1}. \end{aligned}$$

So that $U = C[1, h, \ldots, h, 1]^T$, where C is any scalar. For this system to be consistent (have at least one solution), we can use the fact that the kernel of A^T is orthogonal to the image of A to determine what conditions does F need to satisfy to actually be in the range of A. Thus showing there does in fact exist a U such that $AU = F$. So suppose $\langle U, F\rangle = 0$, this is equivalently:

$$\sigma_0+\frac{h}{2}f(x_0)+h\sum_{i=1}^{m} f(x_i)-\sigma_1+\frac{h}{2}f(x_{m+1}) = 0 \iff \frac{h}{2}f(x_0)+h\sum_{i=1}^{m} f(x_i)+\frac{h}{2}f(x_{m+1}) = \sigma_1-\sigma_0 \checkmark$$

This is exactly the condition that needs to be met for $AU = F$ to be consistent. □

Exercise 2.4.prelim (*preliminary to exercise 2.4*) Find the exact solution to the problem

$$\begin{aligned} u''(x) &= e^x, \qquad 0 < x < 3, \\ u(0) &= \alpha, \qquad u'(3) = \sigma \end{aligned}$$

by simply integrating both sides of the differential equation as many times as necessary. Use this exact solution to test the code in Exercise 2.4. Try a couple of different values of α and σ for yourself, but choose only one set of values to submit as part of your homework solution.

Solution. Through direct integration, we have that the general solution to $u'' = e^x$ on $0 < x < 3$ is:

$$u(x) = e^x + Cx + D$$

imposing $u(0) = \alpha$:

$$\alpha = 1 + D \iff D = \alpha - 1$$

imposing $u'(3) = \sigma$:

$$\sigma = e^3 + C \iff C = \sigma - e^3.$$

Homework #2

Giving us:

$$u(x) = e^x + (\sigma - e^3)x + (\alpha - 1).$$

□

Exercise 2.4 (*boundary conditions in* `bvp` *codes*)

(a) Modify the m-file so that it implements a Dirichlet boundary condition at $x = a$ and a Neumann condition at $x = b$ and test the modified program.

Solution. Changes made:

- Changed $\beta \to \alpha$
- True solution is now:

$$e^x + (\sigma - e^b)(x - a) + (\alpha - e^a)$$

- Change:
  `fdcoeffF(1,x(1), x(1:3)` $\to$ `fdcoeffF(0,x(1), x(1:3)` for the fist row of A
- Change the last row of A: `fdcoeffF(0,x(m2),x(m:m2)` $\to$ `fdcoeffF(1,x(m2),x(m:m2)`

Code attached □

(b) Make the same modification to m-file `bvp_4.m` , implements a fourth order accurate method. Again test the modified program.

Solution. Same changes to the file as in part(a), code attached. □

Exercise 2.6 (*ill-posed boundary value problem*)

Consider the following linear boundary value problem with Dirichlet boundary conditions:

$$u''(x) + u(x) = 0 \qquad \text{for } a < x < b$$
$$u(a) = \alpha, \qquad u(b) = \beta.$$

Note that this equation arises from a linearized pendulum, for example.

(a) Modify the m-file `bvp_2.m` to solve this problem. Test your modified routine on the problem with

$$a = 0, \qquad b = 1, \qquad \alpha = 2, \qquad \beta = 3.$$

Determine the exact solution for comparison.

Solution. Solving the BVP in the general case of $u(a) = \alpha$ and $u(b) = \beta$ on $[a, b]$ gives us:

$$u(x) = A\cos(x) + B\sin(x)$$

where

$$A = \frac{\alpha - B\sin(a)}{\cos(a)} \qquad B = \frac{\beta\cos(a) - \alpha\cos(b)}{\sin(a-b)}$$

so that when $a = 0, b = 1, \alpha = 2, \beta = 3$:

$$B = \frac{3 - 2\cos(1)}{\sin(-1)} \approx -2.28 \qquad A = 2$$

so that:

$$u(x) = 2\cos(x) - 2.28\sin(x)$$

So that the numerical solution converges as seen in Figure 1: □

(b) Let $a = 0$ and $b = \pi$. For what values of α and β does this boundary value problem have solution? Sketch the family of solutions in a case where there are infinitely many solutions.

Solution. When $a = 0$ and $b = \pi$, the above expression for B and A doesn't work. So starting from our general solution, we'll impose boundary conditions:

$$u(0) = \alpha \qquad u(\pi) = \beta$$

This will then give us:

$$A = \alpha \qquad \beta = -\alpha.$$

So there is no unique solution for B, moreover for this system to be consistent we need $\beta = -\alpha$ otherwise there will be no solution. In the case of $\alpha = -\beta$, we then have an infinite number of solutions as seen in (Figure 2), with $\alpha = 2$ and $\beta = -2$. □

(c) Solve the problem with

$$a = 0, \qquad b = \pi, \qquad \alpha = 1, \qquad \beta = -1.$$

using your modified `bvp_2.m`. Which solution to the boundary value problem does this appear to converge to as $h \to 0$? Change the boundary value at $b = \pi$ to $\beta = 1$. Now how does the numerical solution behave as $h \to 0$?

Homework #2

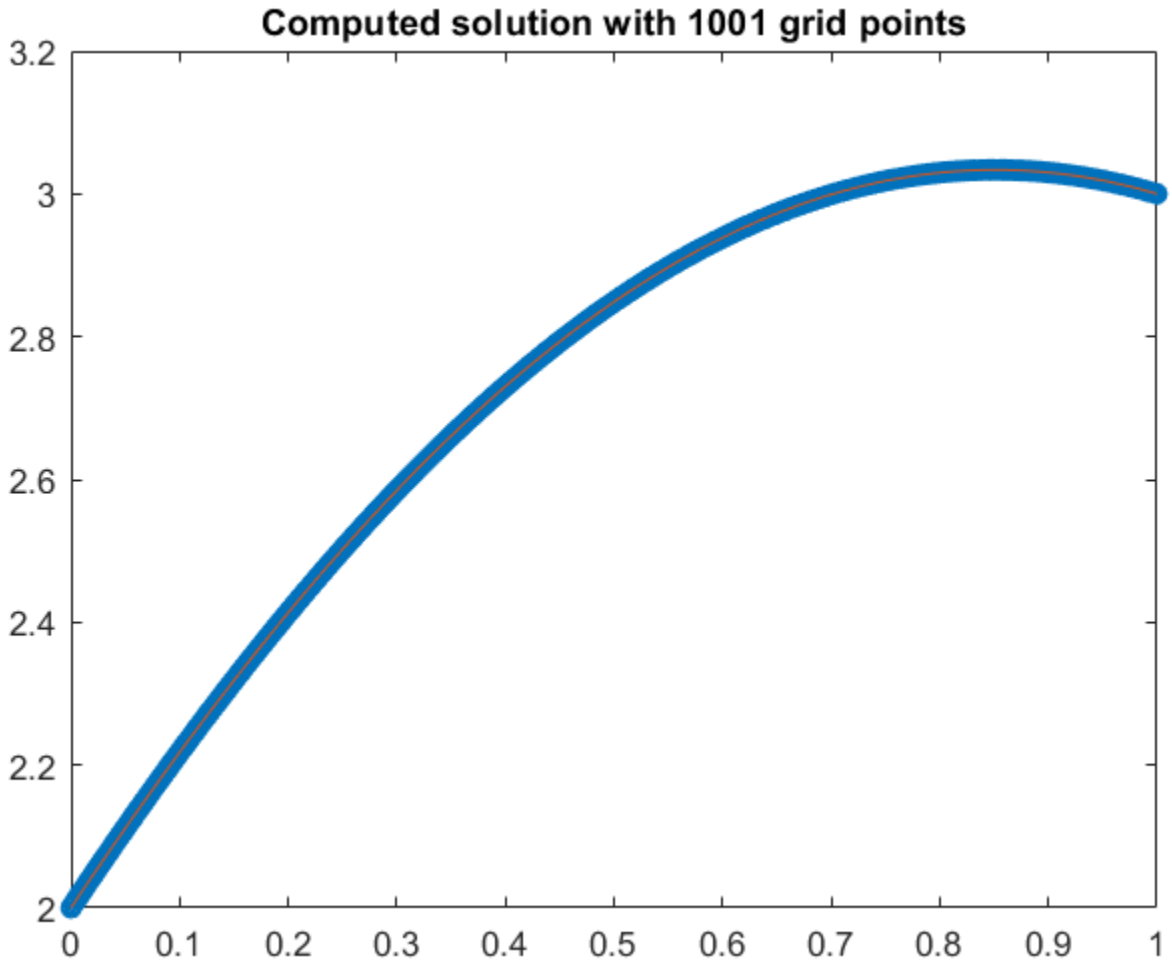


Figure 1: The numerical solution to the BVP when $a = 0, b = 1, \alpha = 2, \beta = 3$

Homework #2

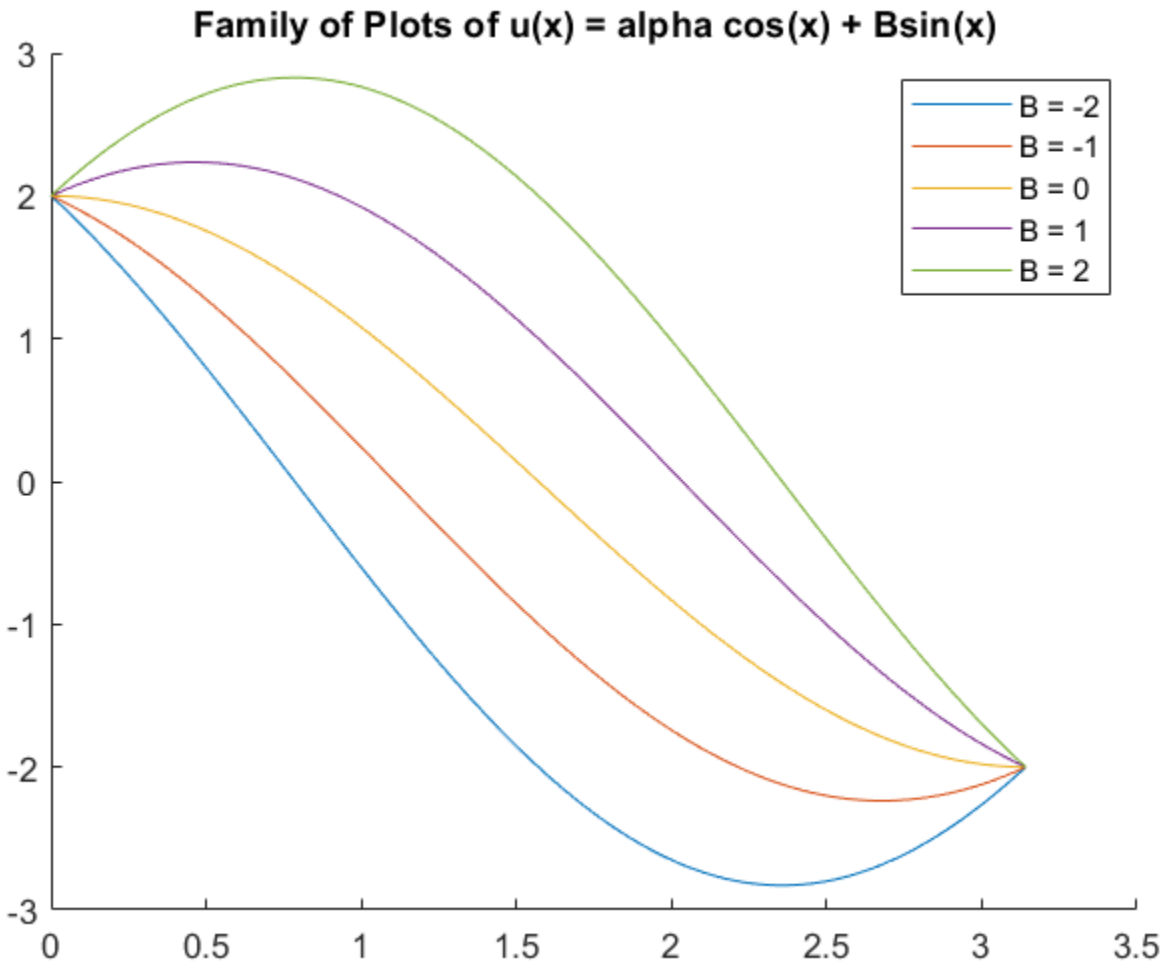


Figure 2: The family of solutions to our BVP when $\alpha = -\beta$, $\alpha = 2$

. Setting $a = 0, b = \pi, \alpha = 1 = \beta$ gives us the numerical solution is converging to the particular solution of $B = -\beta$, even though the coefficient B in the code is undefined when $b = \pi$ and $a = 0$, as seen in (Figure 3). If we try this with $\beta = 1 = \alpha$, then we end up with a nonsensical exact solution (the result of round off error) and a trivial numerical solution, as seen in (Figure 4).

The code for this is given below:

□

(d) You might expect the linear system in part (c) to be singular since the boundary

Homework #2

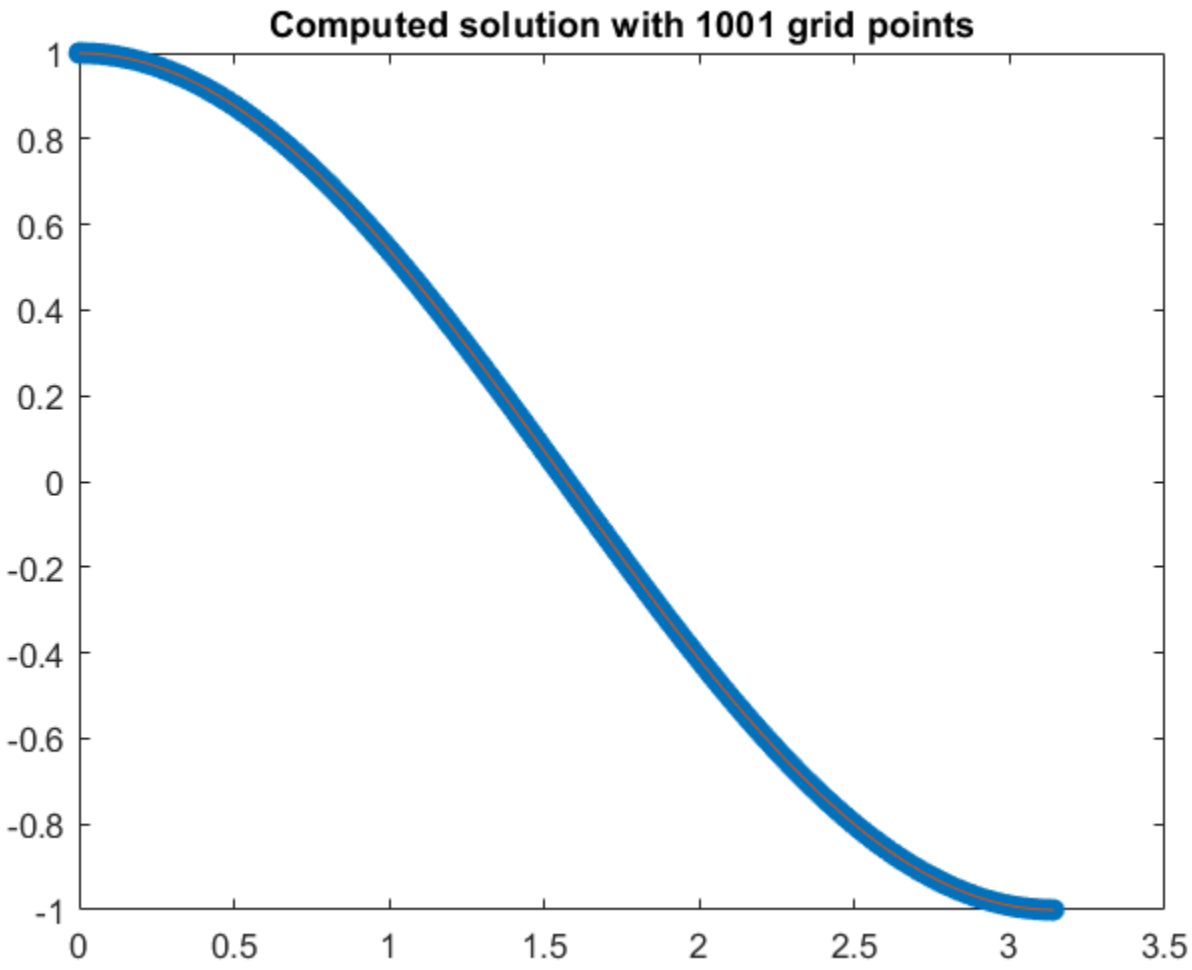


Figure 3: The numerical solution and exact solution obtained if $b = \pi$, $a = 0$ and $\alpha = -\beta$ with $\alpha = 1$, note the exact solution is not correct in that there is no unique solution in this case although due to round-off error Matlab approximates this as $B = -\beta$

Homework #2

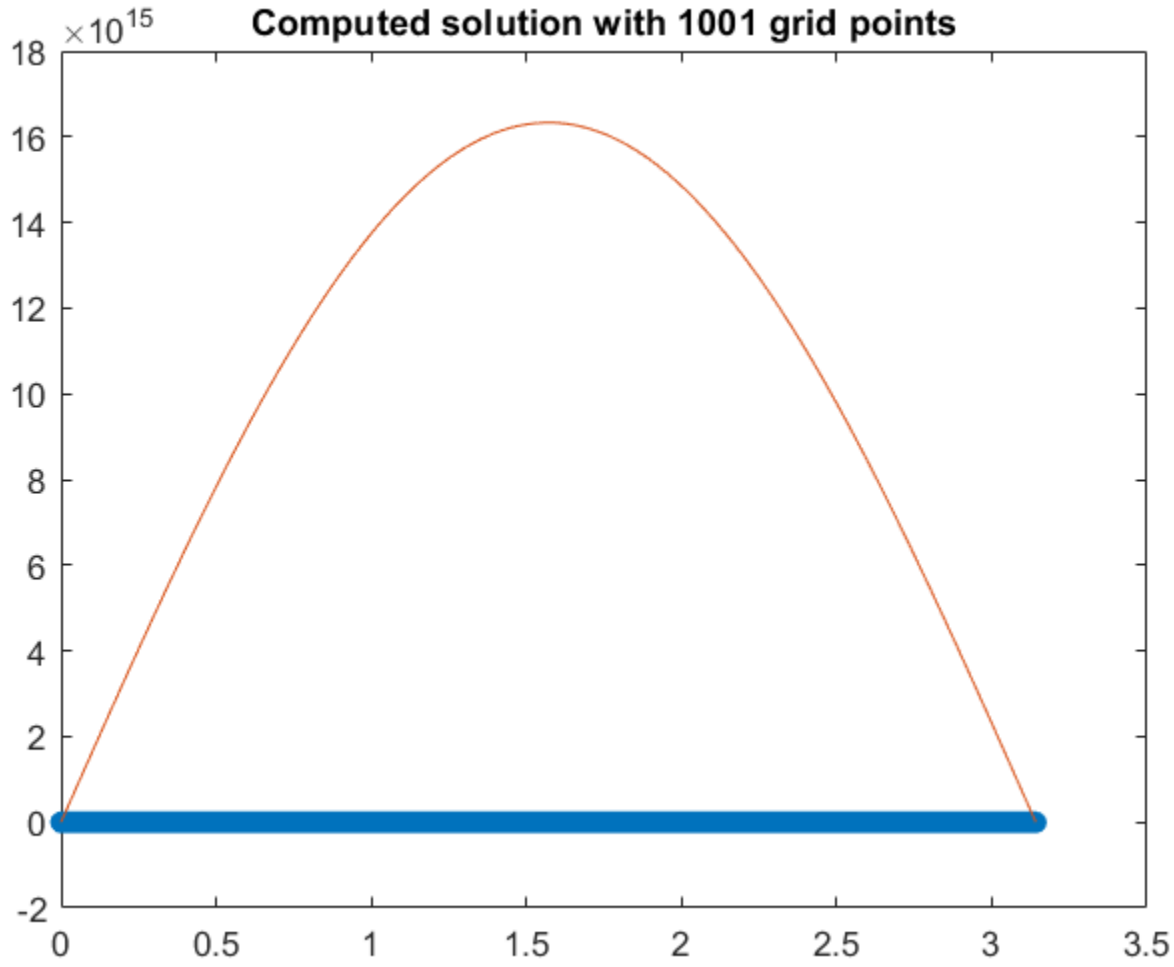


Figure 4: The numerical "solution" in blue, and the exact "solution" in red when $\beta = \alpha = 1$ and $a = 0$ and $b = \pi$. Although, note that neither make sense as there is no solution to this BVP in this case.

value problem is not well posed. It is not well posed. It is not, because of discretization error. Compute the eigenvalues of the matrix A for this problem and show that an eigenvalue approaches 0 as $h \to 0$. Also show that $\|A^{-1}\|_2$ blows up as $h \to 0$ so that the discretization is unstable.

Solution. The discretization matrix A will in this problem will be:

$$A = \frac{1}{h^2}\begin{bmatrix} h^2-2 & 1 & & & \\ 1 & h^2-2 & & & \\ \ddots & \ddots & \ddots & & \\ & 1 & h^2-a & 1 & \\ & & 1 & h^2-2 & \end{bmatrix}.$$

This matrix is tridiagonal, so that the eigenvalues must satisfy:

$$\lambda_p = 1 - \frac{2}{h^2} + \frac{1}{h^2}\cos(2ph) \qquad h = \frac{b-a}{m+1} \qquad p = 1, \ldots, m.$$

This is equivalenly:

$$\lambda_p = 1 - \frac{2}{h^2} + \frac{2}{h^2}(1 - \frac{1}{2}(2ph)^2 + \frac{(2ph)^4}{4!} + O(h^6))$$

$$\lambda_1 = 1 - \frac{2}{h^2} + \frac{2}{h^2}(1 - \frac{h^2}{2} + \frac{16h^4}{4!} + O(h^6))$$

$$\lambda_1 = 1 - \frac{2}{h^2} + \frac{2}{h^2} - 1 + \frac{16h^2}{12} + O(h^4)$$

$$\lambda_1 = \frac{4h^2}{3} + O(h^4)$$

and so $\lim_{h\to 0} \lambda_1 = 0$.
So that A is singular as $h \to 0$, and hence $\|A^{-1}\|$ is unbounded as $h \to 0$ and so our scheme is not stable as $h \to 0$. □

Exercise 2.7 (*nonlinear pendulum*)

(a) Write a program to solve the boundary value problem for the nonlinear pendulum as discussed in the text. See if you can find yet another solution for the boundary conditions illustrated in Figures 2.4 and 2.5.

Solution. With an initial guess of $\vec{\theta} = \vec{1}$, we end up with the solution converging in Figure 5.

□

Homework #2

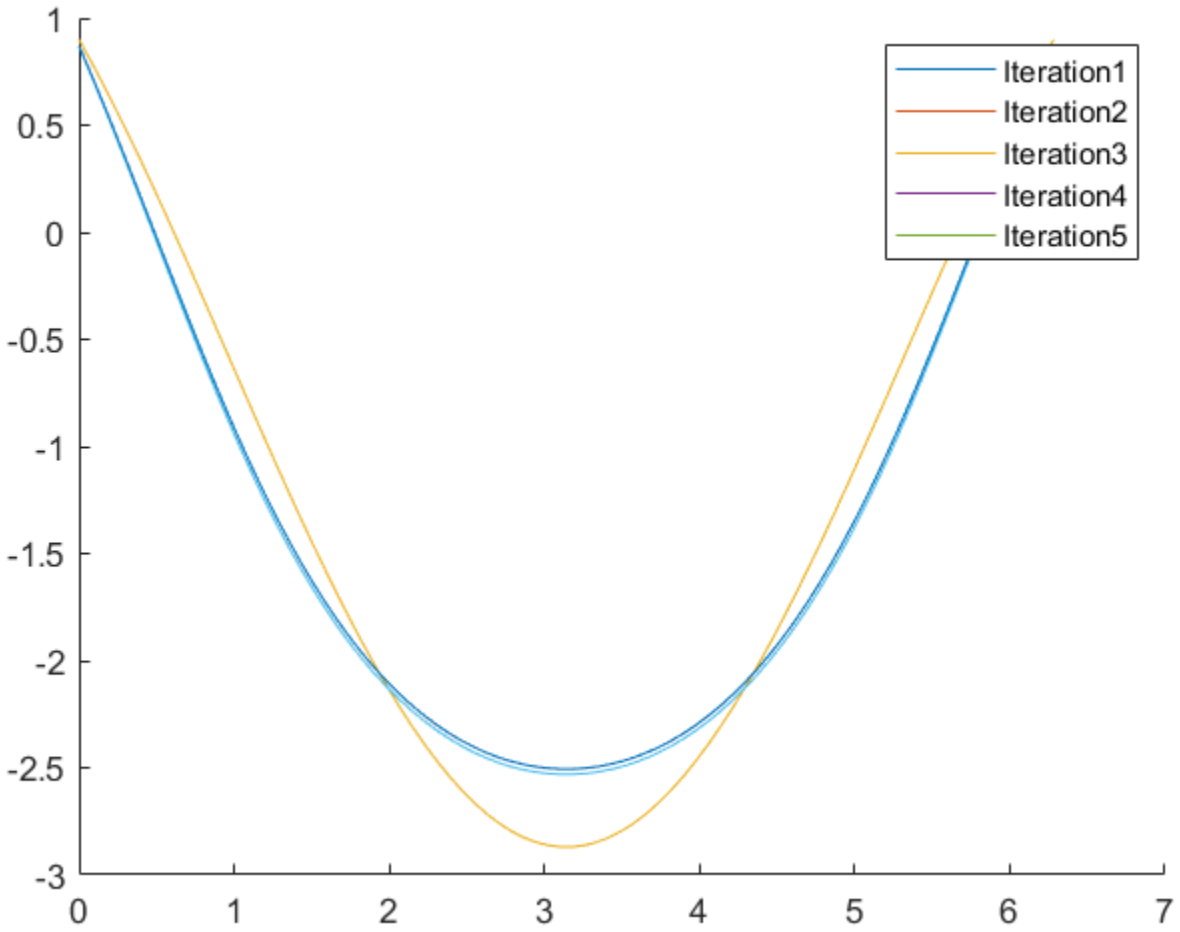


Figure 5: With an initial guess of $\theta = 1$ and $T = 2\pi$

Homework #2

(b) Find a numerical solution to this BVP with same general behavior as seen in Figure 2.5 for the case of a longer time interval, say $T = 20$, again with $\alpha = \beta = 0.7$. Try larger values of T. What does $\max_i \theta_i$ approach as T is increased? Note that for large T this solution exhibits 'boundary layers'.

Solution. My solution for $T = 20$, after 5 iterations is shown in Figure 6.
My solution for $T = 50$, after 5 iterations is shown in Figure 7.
My solution for $T = 100$, after 5 iterations is shown in Figure 8
My Solution for $T = 500$, after 5 iterations is shown in Figure 9.
After an initial spike in the values of $\max_i \theta_i \to 0$. □

Code

2.2(b)
```
%% Problem 2.2(b)
% Using our formulas we found for A^{-1}:

% Define the domain and mesh
clear
h = 0.25;
a = 0;
b = 1;
m = round((b-a)/h)-1;
x_mesh = linspace(a,b,m+2);
x_bar_mesh = linspace(a,b,m+2);

A_inverse = ones(m+2, m+2);
for i = 1:(m+2)
    for j = 1:(m+2)
        if j == 1
            A_inverse(i,j) = x_mesh(i) - 1;

        elseif j == (m+2)
            A_inverse(i,j) = 1;

        else
            if i <= j
                A_inverse(i,j) = h*(x_bar_mesh(j) - 1);
            elseif i > j
```

Homework #2

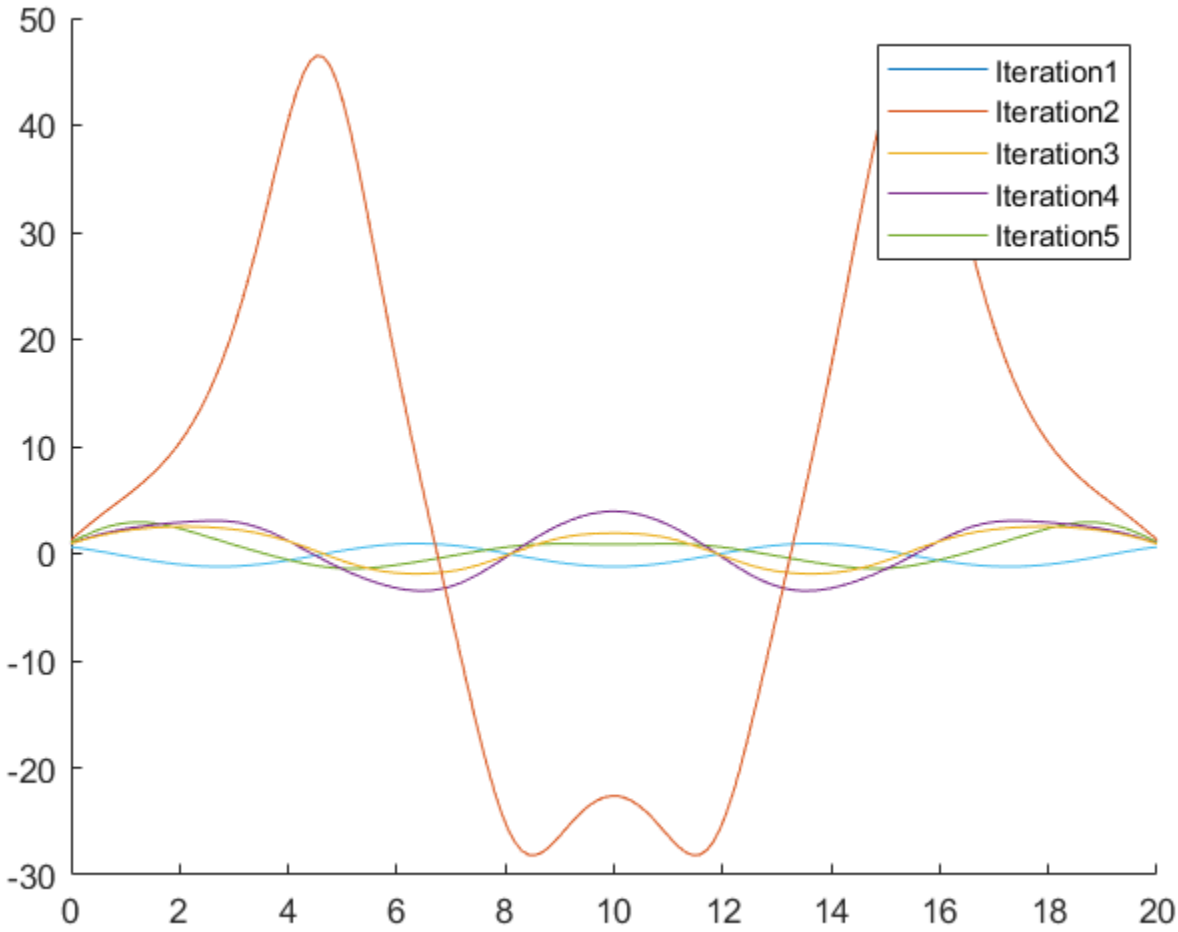


Figure 6: $\alpha = \beta = 0.7$, $\theta^{|0|} = 0.7$, $T = 20$

Homework #2

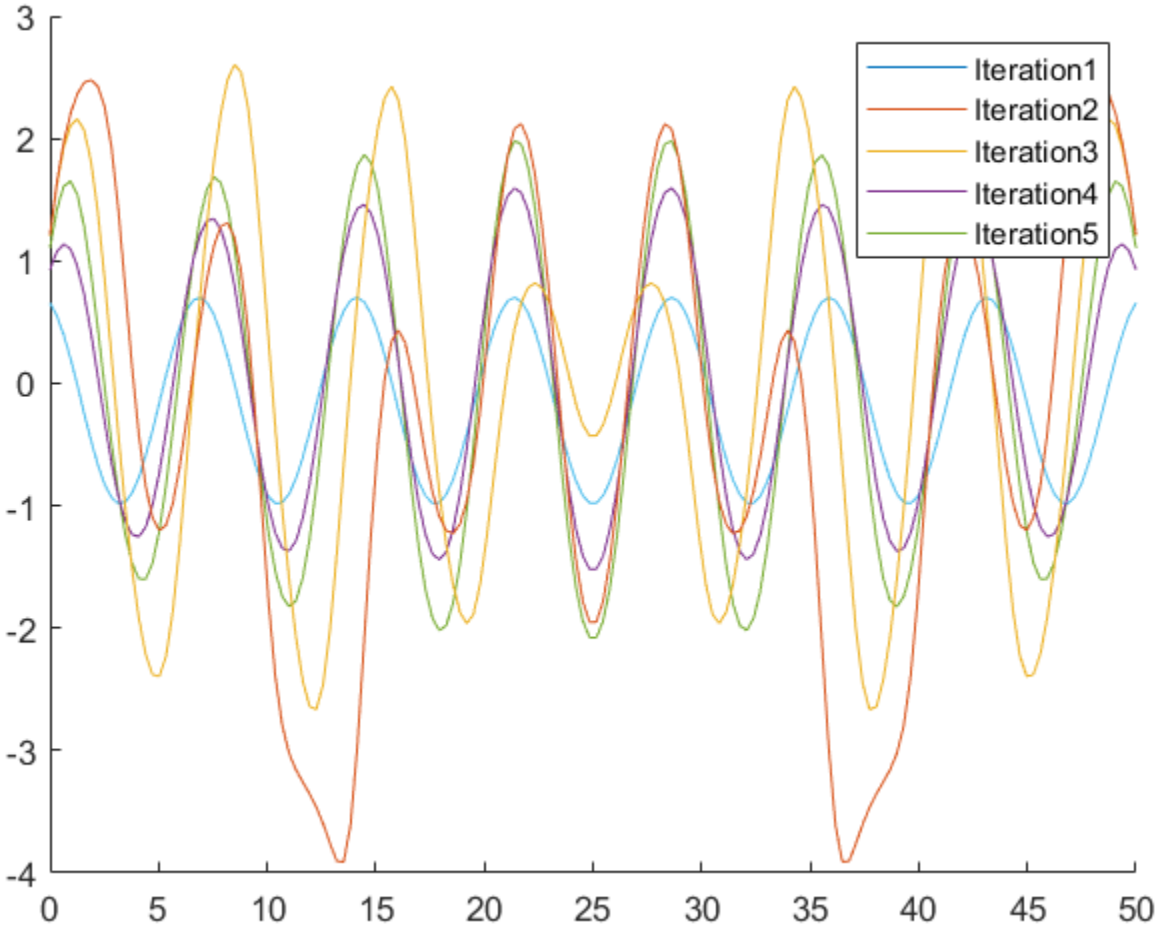


Figure 7: $\alpha = \beta = 0.7$, $\theta^{|0|} = 0.7$, $T = 50$

Homework #2

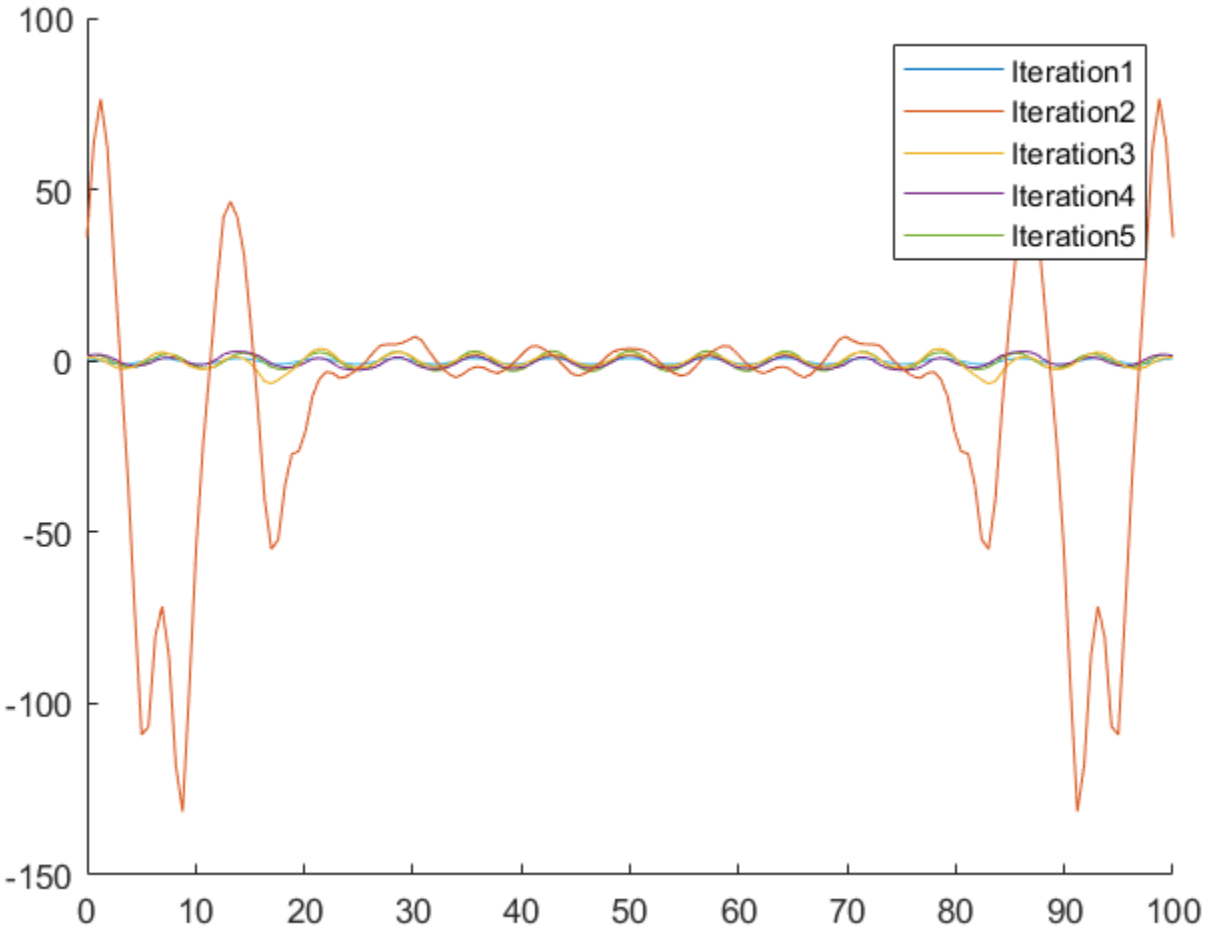


Figure 8: $\alpha = \beta = 0.7$, $\theta^{|0|} = 0.7$, $T = 100$

Homework #2

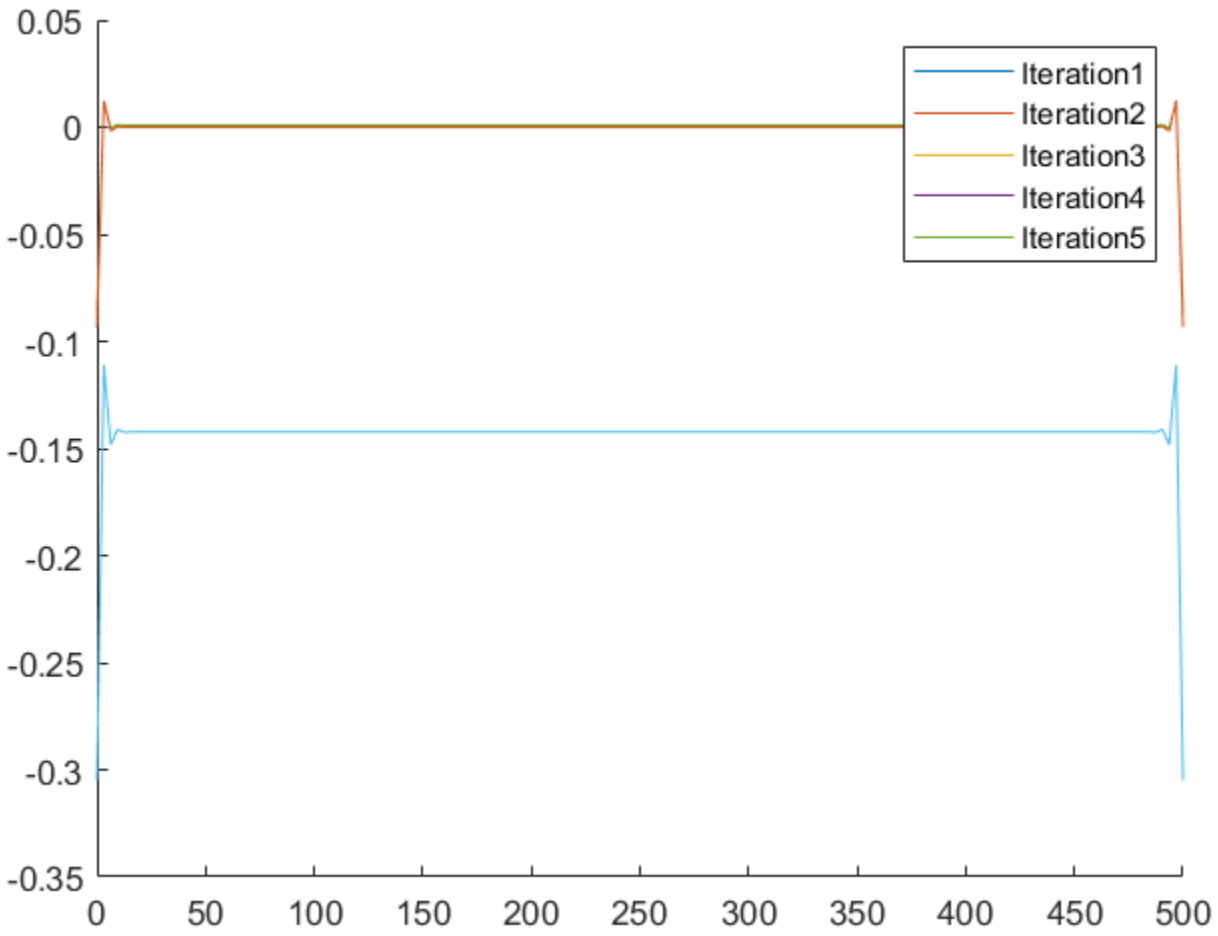


Figure 9: $\alpha = \beta = 0.7$, $\theta^{|0|} = 0.7$, $T = 500$

Homework #2

```
                    A_inverse(i,j) = h*(x_mesh(i) - 1);
                end
            end
        end
    end

    A_inverse
    A = inv(A_inverse)
```

2.4(a)

```
    %% 2.4(a), Dirichlet at x = a and Neumann at x = b

    %
    % bvp_2.m
    % second order finite difference method for the bvp
    %   u''(x) = f(x),   u(ax)=alpha,   u'(bx)=sigma
    % 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)

    clear
    ax = 0;
    bx = 3;
    sigma = 1;   % Neumann boundary condition at bx
    alpha = 3;     % Dirichlet boundary condtion at ax

    f = @(x) exp(x);  % right hand side function
    utrue = @(x) exp(x) + (sigma-exp(bx))*(x - ax) + (alpha - exp(ax));  % 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 80 160 1000];
```

Homework #2

```
ntest = length(m1vals);
hvals = zeros(ntest,1);  % to hold h values
E = zeros(ntest,1);      % to hold errors

figure(1)
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(1,1:2) = fdcoeffF(0, x(1), x(1:2));

  % 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));

  % Right hand side:
  F = f(x);
  F(1) = alpha;
  F(m2) = sigma;

  % solve linear system:
  U = A\F;


  % compute error at grid points:
  uhat = utrue(x);
```

Homework #2

```
  err = U - uhat;
  E(jtest) = max(abs(err));
  disp(' ')
  disp(sprintf('Error with %i points is %9.5e',m2,E(jtest)))

  clf
  figure(1)
  plot(x,U,'o')  % plot computed solution
  title(sprintf('Computed solution with %i grid points',m2));
  hold on
  plot(xfine,ufine)  % plot true solution
  % pause to see this plot:
  %drawnow
  %input('Hit <return> to continue ');

  end
  hold off
  figure(2)
error_table(hvals, E);   % print tables of errors and ratios
error_loglog(hvals, E);  % produce log-log plot of errors and least squares fit
```

2.4(b)

```
%% 2.4(b)

%
% bvp4.m
% second order finite difference method for the bvp
%   u''(x) = f(x),   u'(ax)=sigma,   u(bx)=beta
% fourth order finite difference method for the bvp
%   u'' = f,   u'(ax)=sigma,   u(bx)=beta
% Using 5-pt differences on an arbitrary grid.
% Should be 4th order accurate if grid points vary smoothly.
%
% 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/chapter2  (2007)

clear
```

Homework #2

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

f = @(x) exp(x);  % right hand side function
utrue = @(x) exp(x) + (sigma-exp(bx))*(x - ax) + (alpha - exp(ax));  % 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

figure(3)
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';
  x = xgrid(ax,bx,m,gridchoice);

  % set up matrix A (using sparse matrix storage):
  A = spalloc(m2,m2,5*m2);   % initialize to zero matrix

  % first row for Dirichlet BC on u'(x(1))
  A(1,1:5) = fdcoeffF(0, x(1), x(1:5));
  % second row for u''(x(2))
  A(2,1:6) = fdcoeffF(2, x(2), x(1:6));

  % interior rows:
  for i=3:m
     A(i,i-2:i+2) = fdcoeffF(2, x(i), x((i-2):(i+2)));
```

Homework #2

```
    end

  % next to last row for u''(x(m+1))
  A(m1,m-3:m2) = fdcoeffF(2,x(m1),x(m-3:m2));
  % last row for Neumann BC on u(x(m+2))
  A(m2,m-2:m2) = fdcoeffF(1,x(m2),x(m-2:m2));

  % Right hand side:
  F = f(x);
  F(1) = alpha;
  F(m2) = sigma;

  % solve linear system:
  U = A\F;


  % compute error at grid points:
  uhat = utrue(x);
  err = U - uhat;
  E(jtest) = max(abs(err));
  disp(' ')
  disp(sprintf('Error with %i points is %9.5e',m2,E(jtest)))

  clf
  plot(x,U,'o')  % plot computed solution
  title(sprintf('Computed solution with %i grid points',m2));
  hold on
  plot(xfine,ufine)  % plot true solution

  % pause to see this plot:
  %drawnow
  %input('Hit <return> for next plot ');

  end
  hold off
figure(4)
error_table(hvals, E);   % print tables of errors and ratios
error_loglog(hvals, E);  % produce log-log plot of errors and least squares fit
```

Homework #2

2.6

```
%% Exercise 2.6(d)
clear

ax = 0;
bx = pi;
beta = 1;   % Dirichlet boundary condition at bx
alpha = 1;     % Dirichlet boundary condtion at ax

f = @(x) zeros(size(x));  % right hand side function
B_coeff = (beta - alpha*cos(bx))/(sin(bx));
A_coeff = alpha;
utrue = @(x) A_coeff*cos(x) + B_coeff*sin(x);

% 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 80 160 1000];
ntest = length(m1vals);
hvals = zeros(ntest,1);  % to hold h values
E = zeros(ntest,1);      % to hold errors

figure(5)
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
  h = hvals(jtest);
  % 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(1,1:2) = fdcoeffF(0, x(1), x(1:2));
```

Homework #2

```
    % interior rows:
    for i=2:m1
       A(i,i-1:i+1) = fdcoeffF(2, x(i), x((i-1):(i+1)));
       A(i,i) = A(i,i) - (hvals(jtest)^2 / 2) * A(i,i);
    end

    % last row for Neumann BC at bx:
    A(m2,m:m2) = fdcoeffF(0,x(m2),x(m:m2));


    % Right hand side:
    F = f(x);
    F(1) = alpha;
    F(m2) = beta;

    % solve linear system:
    U = A\F;


    % compute error at grid points:
    uhat = utrue(x);
    err = U - uhat;
    E(jtest) = max(abs(err));
    disp(' ')
    disp(sprintf('Error with %i points is %9.5e',m2,E(jtest)))

    clf
    plot(x,U,'o')  % plot computed solution
    title(sprintf('Computed solution with %i grid points',m2));
    hold on
    plot(xfine,ufine)  % plot true solution

    % pause to see this plot:
    %drawnow
    %input('Hit <return> to continue ');

    end
    hold off
figure(6)
error_table(hvals, E);   % print tables of errors and ratios
```

Homework #2

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

2.7

```
clear
clf
% Initialize number of iterations
iter = 5;
% Initialize the total time
T = 10000;
m = 160;

t = linspace(0,T,m);
% Set Initial Guess
theta_0 = 0.7;
theta = 0.7*ones(m);
theta_m1 = 0.7;
% Initialize step
h = T/(m+1);
% Initialize G-vector and Jacobian
G = zeros(m,1);
J = zeros(m,m);
figure(7)
for i=1:iter
    % Update the first row of Jacobian and G-vector
    G(1,1) = (theta_0 - 2*theta(1) + theta(2))/h^2 + sin(theta(1));
    J(1,1) = -2/h^2 + cos(theta(1));
    J(1,2) = 1/h^2;
    % Update the interior rows
    for i=2:(m-1)
        G(i,1) = (theta(i-1) - 2*theta(i) + theta(i+1))/h^2 + sin(theta
        J(i,i-1) = 1/h^2;
        J(i,i) = (-2/h^2) + cos(theta(i));
        J(i,i+1) = 1/h^2;
    end
    % Update the last row
    G(m,1) = (theta(m-1) - 2*theta(m) + theta_m1)/h^2 + sin(theta(m));
    J(m,m-1) = 1/h^2;
    J(m,m) = (-2/h^2) + cos(theta(m));
    % Solve for delta
    delta = -J\G;
```

Homework #3

```
    % Update theta
    theta = theta + delta;

    hold on
    plot(t,theta)
end
% Plot
labels = strings(iter);
for i=1:iter
    labels(i) = "Iteration" + int2str(i);
end
legend(labels)

hold off
```

Exercise 3.1 (*code for Poisson problem*)

The Matlab script `possion.m` (avaliable from the textbook's site or from our course's Canvas page) solves the Poisson problem on a square $m \times m$ grid with $\Delta x = \Delta y = h$, using the 5-point Laplacian. It is set up to solve a test problem for which the exact solution is $u(x, y) = \exp(x + y/2)$, using Dirichlet boundary conditions and the right hand side $f(x, y) = 1.25 \exp(x + y/2)$.

(a) Test this scrip by performing a grid refinement study to verify that it is second order accurate.

(b) Modify the script so that it works on a rectangular domain $[a_x, b_x] \times [a_y, b_y]$, but sill with $\Delta x = \Delta y = h$. Test your modified script on a non-square domain.

(a) *Solution.* Adding a `for` loop over the `m_vals = [99, 149, 199, 299, 499]`, I get the following results:

□

(b) *Solution.* To change the code for a rectangular domain, first we introduce new variables:

```
a_x = 0;
a_y = 3;
b_x = 2;
b_y = 4;

h_vals = [1/10, 1/20, 1/50, 1/100, 1/500];
```

Homework #3

h	error	ratio	observed order
0.01000	1.44721e-06	NaN	NaN
0.00667	6.43254e-07	2.24983	1.99981
0.00500	3.61832e-07	1.77777	1.99998
0.00333	1.60816e-07	2.24998	1.99998
0.00200	5.79009e-08	2.77743	1.99975

Figure 10: Error table for $h = 0.01, 0.00667, 0.005, 0.00333, 0.002$, confirms the order of the scheme is $O(h^2)$

```
m_vals = (b_x - a_x)./h_vals - 1;
n_vals = (b_y - a_y)./h_vals - 1;
error = 0*h_vals;
```

as part of initialization. Notice, this differs from the square domain case, as we define our values of h and then define m and n based on these values.

Then introducing dummy variables inside our `for` loop:

```
m_x = m_vals(i);
h = h_vals(i)
m_y = n_vals(i);
x = linspace(a_x,b_x,m_x+2);    % grid points x includi
y = linspace(a_y,b_y,m_y+2);    % grid points y includi
```

We also change the number of interior nodes to match our rectangular domain:

```
Iint = 2:(m_x+1);                % indices of interior p
Jint = 2:(m_y+1);                % indices of interior p
Xint = X(Iint,Jint);        % interior points
Yint = Y(Iint,Jint);
```

This also necessitates changing the defining of the boundary conditions, as well as adjusting F to be the appropriate size:

```
    % adjust the rhs to include boundary terms:
rhs(:,1) = rhs(:,1) - usoln(Iint,1)/h^2;
rhs(:,m_y) = rhs(:,m_y) - usoln(Iint,m_y+2)/h^2;
rhs(1,:) = rhs(1,:) - usoln(1,Jint)/h^2;
rhs(m_x,:) = rhs(m_x,:) - usoln(m_x+2,Jint)/h^2;

% convert the 2d grid function rhs into a column vector for rh
```

Homework #3

Figure 11: Plotting the error on a log-log plot

```
F = reshape(rhs,m_x*m_y,1);
```

Finally, we adjust the size of the matrix A by adjusting the size of the pattern matrices that are used in the `kron(*,*)` function in Matlab:

```
    % form matrix A:
I_x = speye(m_x);
I_y = speye(m_y);
e_x = ones(m_x,1);
e_y = ones(m_y,1);
```

Homework #3

```
T = spdiags([e_x -4*e_x e_x],[-1 0 1],m_x,m_x);
S = spdiags([e_y e_y],[-1 1],m_y,m_y);
%  kron(where to copy to make big matrix, the little matrix to
A = (kron(I_y,T) + kron(S,I_x)) / h^2;
```

Then truly finally, we adjust the real solutions shape:

```
usoln(Iint,Jint) = reshape(uvec,m_x,m_y);
```

Finally plotting the table and log-log plot of the error, we confirm the adjustments have kept the scheme $O(h^2)$ in truncation error:

```
Least squares fit gives E(h) = 0.205241 * h^1.99741


    h        error       ratio       observed order
 0.10000   2.05570e-03       NaN             NaN
 0.05000   5.18387e-04   3.96557         1.98753
 0.02000   8.31691e-05   6.23293         1.99702
 0.01000   2.08013e-05   3.99827         1.99937
 0.00200   8.32187e-07  24.99594         1.99990
```

Figure 12: Error table of the rectangular schema

□

Homework #3

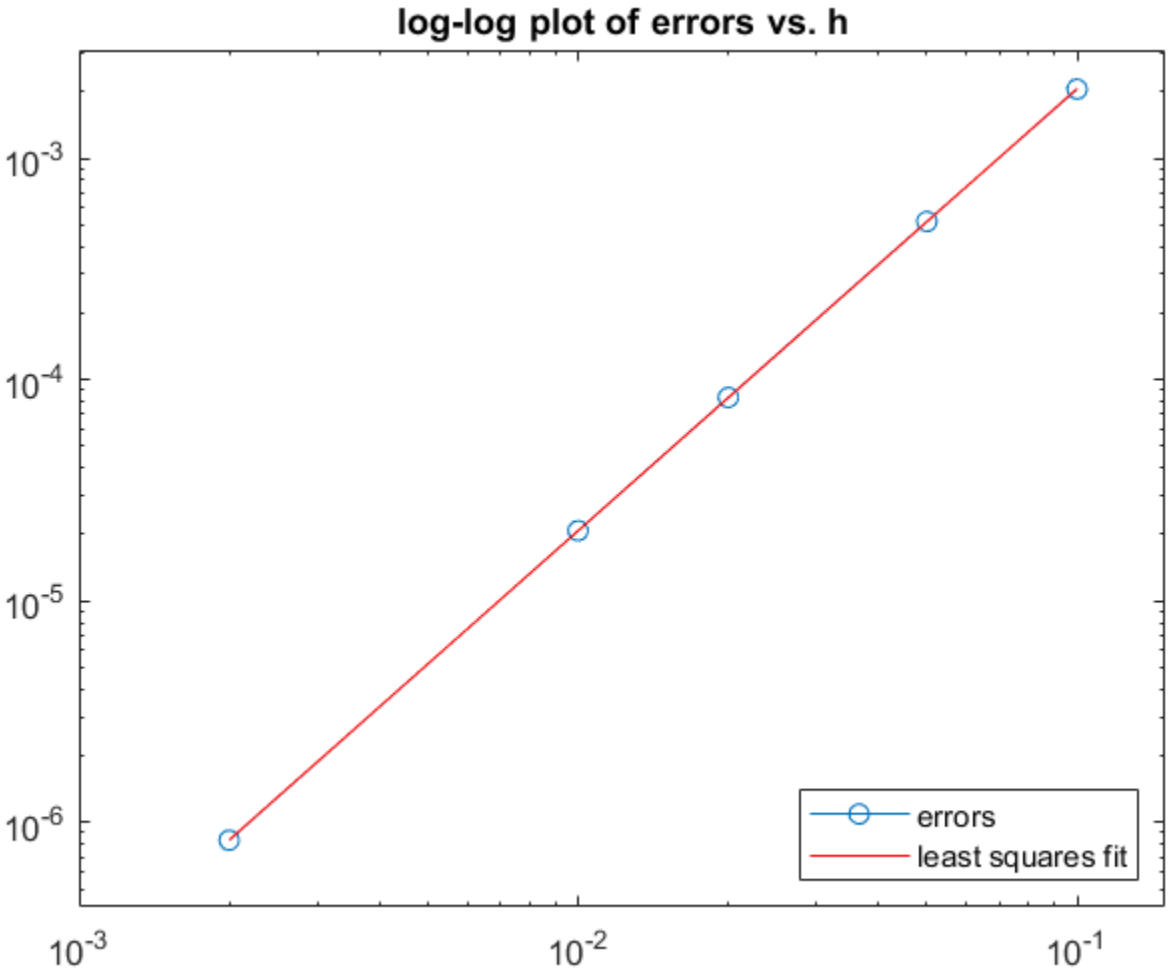


Figure 13: Log-log plot of the error from table 12

Exercise 4.0 (steepest descent prerequisite mathematics)

Show the following three facts, for $\vec{f}$ a constant vector in $\mathbb{R}^n$, $\vec{u}$ a vector of variables in $\mathbb{R}^n$, and A an $n \times n$ symmetric matrix:

(a) $\nabla(\vec{u}^T\vec{f}) = \vec{f}$,

(b) $\nabla(\vec{f}^T\vec{u}) = \vec{f}$,

(c) $\nabla(\vec{u}^T A\vec{u}) = 2A\vec{u}$.

Instructor: P. Choboter
Name: Joseph McGuire

Class: Math 502 (W2022)

Homework #3

In your proof, feel free to demonstrate the details for the $n = 3$ case, so that:

$$\vec{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \vec{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}, \qquad A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \qquad \text{and } \nabla g(\vec{u}) = \nabla g(x, y, z) = \begin{bmatrix} \dfrac{\partial g}{\partial x} \\ \dfrac{\partial g}{\partial y} \\ \dfrac{\partial g}{\partial z} \end{bmatrix}.$$

(a) and (b) *Solution.*

$$\begin{aligned} \nabla(u^T f) &= \nabla\left(\sum_{i=1}^{n} u_i f_i\right) \\ \nabla\left(\sum_{i=1}^{n} u_i f_i\right)_i &= \frac{\partial}{\partial u_i} u_i f_i \\ &= f_i \\ \nabla(u^T f) &= f \end{aligned}$$

Since $u^T f = f^T u = \sum_{i=1}^{n} u_i f_i$, where u_i and f_i, are the i^{th}-components of their respective vectors, we also have that $\nabla(f^T u)_i = f_i$. Thus

$$\nabla(f^T u) = f$$

□

Homework #3

(c) *Solution.*

$$\begin{aligned}
\nabla(u^T A u) &= \nabla\left(u^T \left(\sum_{i=1}^n A_{i,j}u_i\right)_j\right) \\
&= \nabla\left(\sum_{j=1}^n\sum_{i=1}^n A_{i,j}u_i u_j\right) \\
\nabla\left(\sum_{j=1}^n\sum_{i=1}^n A_{i,j}u_i u_j\right)_k &= \frac{\partial}{\partial u_k}\left(\sum_{j=1}^n\sum_{i=1}^n A_{i,j}u_i u_j\right) \\
&= \frac{\partial}{\partial u_k}\left(\sum_{j=1}^n u_j\sum_{i=1}^n A_{i,j}u_i\right) \\
&= \frac{\partial}{\partial u_k}u_k\sum_{i=1}^n A_{i,j}u_i + \sum_{j=1,i\neq k}^n u_j\frac{\partial}{\partial u_k}\sum_{i=1}^n A_{i,j}u_i \\
&= \sum_{i=1,i\neq k}^n A_{i,k}u_i + 2A_{k,k}u_k + \sum_{i=1,j\neq k}^n A_{i,k}u_i \\
&= 2\sum_{i=1}^n A_{i,k}u_i.
\end{aligned}$$

Thus $\nabla(u^T A u)_k = (2Au)_k$ for $k = 1, \ldots, n$ and so $\nabla(u^T A u) = 2Au$. □

Exercise 4.3 (fixing conjugate gradient code)

The file `conjugate_gradient_broken.m` is a modified version of an m-file that implements the conjugate gradient algorithm (without preconditioning) to solve $Au = f$. The modification is that two lines of code were removed. Download the file from our Canvas site, and repair the file so that it successfully implements the conjugate gradient algorithm once again. Rename the file `conjugate_gradient.m`, and test it by running the command `>> u = conjugate_gradient(A,F,1.0e-5)`; where A is some large, symmetric, positive-definite matrix (defined as a sparse matrix in MATLAB), and F a column vector of the correct length. To help you test your code, note that afer you run `poisson.m` from Exercise 3.1 above, MATLAB has a matrix A and a vector F that would work nicely.

Note: The **Conjugate Gradient algorithm** is used to solve $A\vec{u} = \vec{f}$ (or minimize $\phi(\vec{u}) = \frac{1}{2}\vec{u}^T A\vec{u} - \vec{u}^T\vec{f} + c$), where A is an $n \times n$ symmetric, positive-definite matrix, $\vec{u}$ and $\vec{f}$ are vectors in $\mathbb{R}^n$, and c is a scalar. The CG algorithm can be expressed as follows (see p.87 of LeVeque):

Choose an initial guess $\vec{u}_0$ (possibly the zero vector)

Homework # 4

$\vec{r_0} = \vec{f} - A\vec{u_0}$

$\vec{p_0} = \vec{r_0}$

for $k = 1, 2, \ldots$

$w_{\vec{k-1}} = Ap_{\vec{k-1}}$

$\alpha_{k-1} = \dfrac{r_{\vec{k-1}}^{\ T} r_{\vec{k-1}}}{p_{\vec{k-1}}^{\ T} w_{\vec{k-1}}}$

$\vec{u_k} = u_{\vec{k-1}} + \alpha_{k-1} p_{\vec{k-1}}$

$\vec{r_k} = r_{\vec{k-1}} - \alpha_{k-1} w_{\vec{k-1}}$

If $\|\vec{r_k}\|$ is less than some tolerance, then stop.

$\beta_{k-1} = \dfrac{\vec{r_k}^T \vec{r_k}}{r_{\vec{k-1}}^{\ T} r_{\vec{k-1}}}$

$\vec{p_k} = \vec{r_k} + \beta_{k-1} p_{\vec{k-1}}$

end

Solution. The three lines of code changed were, although admittly the line `u = unew;` is not entirely necessary, I just added it for clarity:

```
MAXITS = length(f);
u = 0*f;
r = f - A*u;
p = r;
for k = 1:MAXITS
    w = A*p;
    alpha = (r'*r)/(p'*w);
    unew = u + alpha * p;
    rnew = r - alpha*w;
    if( norm(rnew) < tol )
        fprintf('Converged! its= %7.0f, tol=%10.3e\n', [k tol]);
        return;
end
beta = (rnew'*rnew)/(r'*r);
p = rnew + beta*p;
r = rnew;
u = unew;
```

These are the initialization of r was missing and the update rule for u was missing. Testing this on A and F with error tolerance 10^{-4} gives us a reasonable answer: □

Homework # 4

```
>> u = conjugate_gradient(A,F,1.0e-5)
Converged! its=      64, tol= 1.000e-05
```

Figure 14: Convergence message after adjustment

Exercise 5.2 (Lipshitz constant for an ODE) Let $f(u) = \log(u)$.

(a) Determine the best possible Lipshitz constant for this function over $2 \leq u < \infty$.

Solution. To determine this, we'll find L such that:

$$L = \max_{u \in [2,\infty)} |f_u(u,t)|.$$

Note that since $f(u,t) = \log(u)$ is just a function of u, we only maximize over $u \in [2,\infty)$. Since $f_u(u,t) = \dfrac{\partial}{\partial u}\log(u) = \dfrac{1}{u}$. So that:

$$\max_{u \in [2,\infty)} \left|\frac{1}{u}\right| = \frac{1}{2} = L.$$

So here, the optimal L is $L = \dfrac{1}{2}$. □

(b) Is $f(u)$ Lipshitz continuous over $0 < u < \infty$?

Solution. No since:

$$|\log(u) - \log(\epsilon)| \neq O(u - \epsilon) \qquad \text{as } \epsilon \to 0^+,$$

since $\lim\limits_{\epsilon \to 0^+} \log(\epsilon) = -\infty$.

□

(c) Consider the initial value problem

$$\begin{aligned} u'(t) &= \log(u(t)), \\ u(0) &= 2. \end{aligned}$$

Explain why we know that this problem has a unique solution for all $t \geq 0$ based on the existence and uniqueness theory described in Section 5.2.1. (Hint: Argue that f is Lipshitz continuous in a domain that the solution never leaves, though the domain is not symmetric about $\eta = 2$ as assumed in the theorem quoted in the book.)

Solution. In (a) we showed that $f(u) = \log(u)$ is Lipshitz on $u \in [2, \infty)$.
So we need to show that $u \in [2, \infty)$ in this IVP, and then the result follows from (a). Note that $u(0) = 2$, and that $u' = \log(u)$, that is the derivative is positive on $t \in [0, \infty)$. Since $u(0) = 2$, we have that $u'(0) = \log(2)$. Furthermore, since $u' = \log(u)$ we know the derivative is monotonically increasing, since $\log(\cdot)$ is monotonically increasing. Suppose that there's some $t \in [0, \infty)$ where $0 < u(t) < 2$. This would imply that $u' < 0$ for some interval on $[0, \infty)$, a contradiction since $u'(0) = 2$. Thus $u \in [2, \infty)$ for $t \in [0, \infty)$ and hence this IVP has a unique solution. □

Exercise 5.8 (Use of `ode113` and `ode45`) This problem can be solved by modifying the m-files `odesample.m` and `odesampletest.m` avaliable from either the textbook's webpage or from our course Canvas page. Consider the third order initial value problem

$$v'''(t) + v''(t) + 4v'(t) + 4v(t) = 4t^2 + 8t - 10,$$
$$v(0) = -3, \qquad v'(0) = -2, \qquad v''(0) = 2.$$

(a) Verify that the function

$$v(t) = -\sin(2t) + t^2 - 3$$

is a solution to this problem. How do we know it is the unique solution?

Solution. If $v(t) = -\sin(2t) + t^2 - 3$, then:

$$\begin{aligned} v'(t) &= -2\cos(2t) + 2t \\ v''(t) &= +4\sin(2t) + 2 \\ v'''(t) &= 8\cos(2t), \end{aligned}$$

so that:

$$\begin{aligned} v'''(t) + v''(t) + 4v'(t) + 4v(t) &= (8\cos(2t)) + (4\sin(2t) + 2) \\ &\quad (-8\cos(2t) + 8t) + (-4\sin(2t) + 4t^2 - 12) \\ &= 4t^2 + 8t - 10\checkmark. \end{aligned}$$

And:

$$\begin{aligned} v(0) &= 0 + 0 - 3\checkmark \\ v'(0) &= -2\checkmark \\ v''(0) &= 2\checkmark \end{aligned}$$

Moreover, because this DE is linear in v''', v'', v', v, we know by Duhamel's principle that this IVP has a unique solution. □

Homework # 4

(b) Rewrite this problem a first order system of the form $u'(t) = f(u(t), t)$ where $u(t) \in \mathbb{R}^3$. Make sure you also specify the initial condition $u(0) = \eta$ as a 3-vector.

Solution. Let $u_1 = v, u_2 = v', u_3 = v''$, so that:

$$u_1' = u_2 \qquad u_2' = u_3 \qquad u_3' = -u_3 - 4u_2 - 4u_1 + 4t^2 + 8t - 10.$$

Then our initial condition:

$$u(0) = [-3, -2, 2]^T$$

□

(c) Test the MATLAB solver by specifying different tolerances spanning several orders of magnitude. Create a table showing the maximum error in the computed solution for each tolerance and the number of function evaluations required to achieve this accuracy.

Solution. Here our $f(u, t)$ is:

$$f(u, t) = [u_2, u_3, -u_3 - 4u_2 - 4u_1 + 4t^2 + 8t - 10]^T,$$

changing this in the `odesample.m`, we get:

```
>> odesampletest

      tol          max error    f evaluations

  1.000e-01      6.271e-04         27
  1.000e-02      4.875e-04         29
  1.000e-03      6.338e-04         33
  1.000e-04      1.196e-04         41
  1.000e-05      1.996e-05         47
  1.000e-06      7.727e-07         63
  1.000e-07      2.087e-07         73
  1.000e-08      1.283e-08         87
  1.000e-09      4.231e-10        115
  1.000e-10      6.669e-11        131
  1.000e-11      6.143e-12        147
  1.000e-12      1.364e-12        157
  1.000e-13      5.418e-14        177
```

Figure 15: Table of Maximum error for 5.8(c)

□

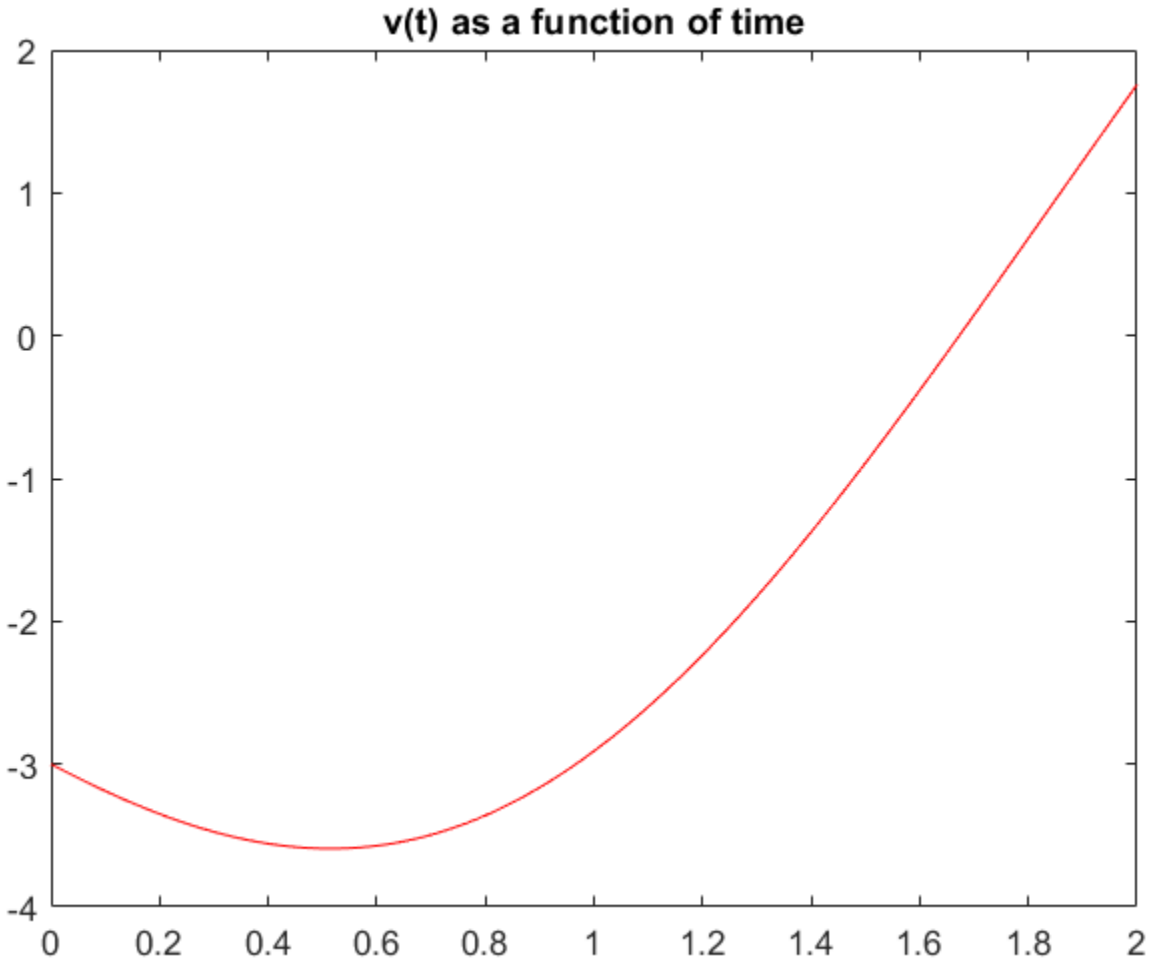


Figure 16: Plot of the numerical solution versus true solution for 5.8

(d) Repeat step (d) using the MATLAB function `ode45` , which uses an embedded pair of Runge-Kutta methods instead of Adams-Bashforth-Moulton methods.

Solution. Using `ode45` solver in (Table 17) and (Figure 18).

□

Exercise 5.9 (truncation errors)

Compute the leading term in the local truncation error of the following methods:

(a) the trapezoidal method (5.22),

Homework # 4

```
>> odesampletest

      tol          max error    f evaluations

  1.000e-01      9.882e-06         67
  1.000e-02      1.024e-05         67
  1.000e-03      1.044e-05         67
  1.000e-04      9.925e-06         67
  1.000e-05      5.394e-06         85
  1.000e-06      5.069e-07        127
  1.000e-07      4.763e-08        199
  1.000e-08      4.573e-09        313
  1.000e-09      4.398e-10        493
  1.000e-10      4.359e-11        781
  1.000e-11      4.381e-12       1237
  1.000e-12      4.312e-13       1951
  1.000e-13      4.086e-14       3091
```

Figure 17: Max Error for 5.8.d, now with ode45 solver

Solution. The trapezoidal method is given by:

$$\frac{U^{n+1}-U^n}{k} = \frac{1}{2}(f(U^n)+f(U^{n+1})),$$

so the local truncation error at a grid point $t = t_n$:

$$\tau^n = \frac{U^{n+1}-U^{n-1}}{k} - \frac{1}{2}f(U^n) - \frac{1}{2}f(U^{n+1}),$$

where k is our time-step. Taylor expanding:

$$U^{n+1} = U^n + k(U^n)' + \frac{k^2}{2}(U^n)'' + \frac{k^3}{6}(U^n)''' + O(k^4)$$

then we can use the fact that:

$$f(U^n) = (U^n)'.$$

So introducing these into the local truncation error:

$$\tau(t_n) = \frac{U^n + k(U^n)' + \frac{k^2}{2}(U^n)'' + \frac{k^3}{6}(U^n)''' - U^n}{k} - \frac{1}{2}(U^n)' - \frac{1}{2}(U^{n+1})' + O(k^3)$$

Homework # 4

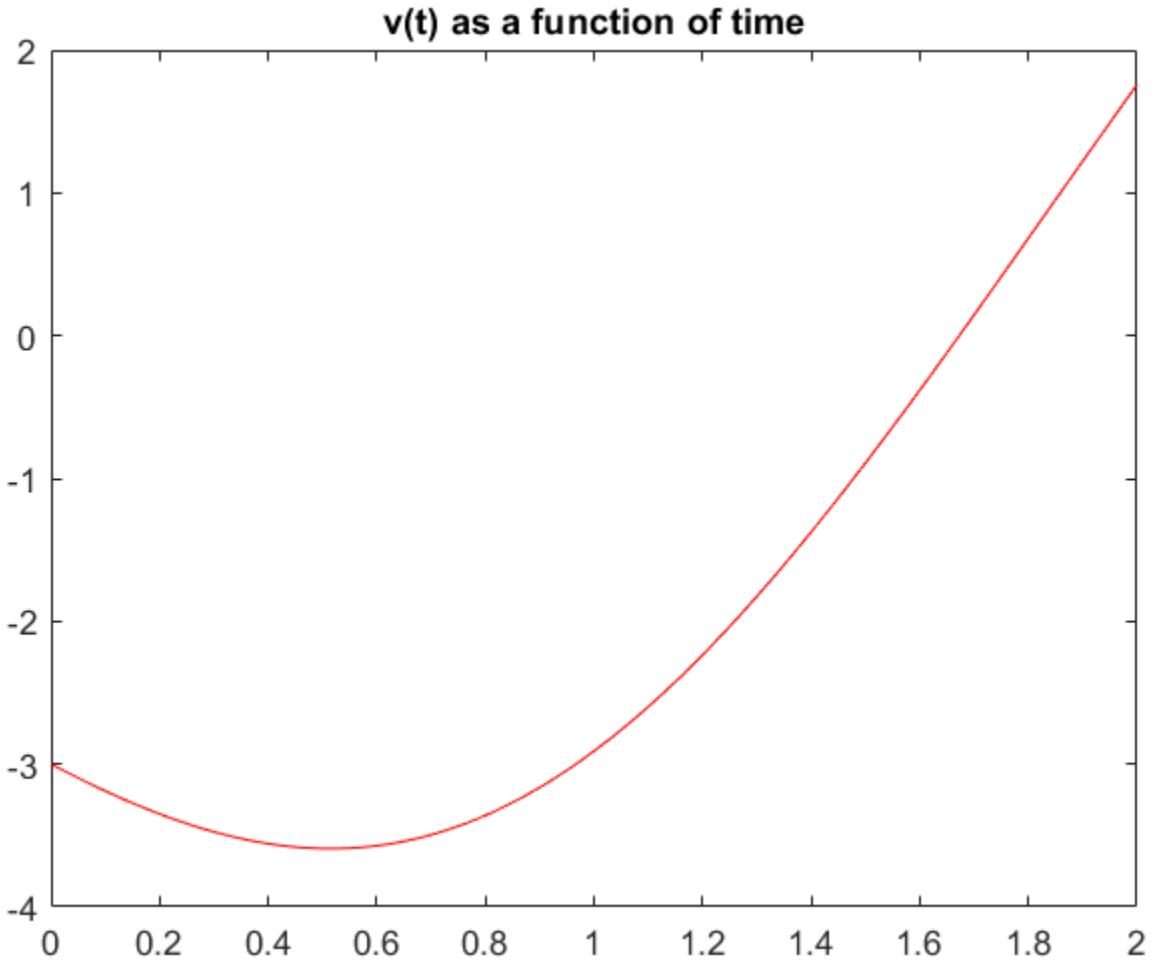


Figure 18: Plot of numerical solution for 5.8 with ode45 solver's solution

$$= k(U^n)' + \frac{k}{2}(U^n)'' + \frac{k^2}{6}(U^n)''' - \frac{1}{2}(U^n)' - \frac{k}{2}(U^n)'' - \frac{k^2}{4}(U^n)''' + O(k^3)$$

$$= -\frac{k^2}{12}(U^n)''' + O(k^3).$$

□

(b) the 2-step BDF method (5.25),

Solution. 2-step Backwards Difference Formula gives a local truncation error of:

$$\tau^n = \frac{1}{2k}(3U^n+3(U^n)'k+3(U^n)''\frac{k^2}{2}+3\frac{k^3}{6}(U^n)'''-4U^n+U^n-(U^n)'k+(U^n)''\frac{k^2}{2}-(U^n)'''\frac{k^3}{6})-f(U^{n+1})$$

$$= \frac{1}{2k}(2(U^n)'k. + 4\frac{k^2}{2}(U^n)'' + 2\frac{k^3}{6}(U^n)''') - (U^{n+1})' = -\frac{1}{3}(U^n)'''k^2 + O(k^3),$$

as $k \to 0^+$. □

(c) the Runge-Kutta method (5.30).

Solution. The Runge-Kutta method has a local truncation error given in (5.31):

$$\tau^n = \frac{1}{k}(U^{n+1} - U^n) - f\left(U^n + \frac{1}{2}kf(U^n)\right)$$

So first expanding out the last term:

$$= f(U^n) + \frac{1}{2}k(U^n)'f'(U^n) + \frac{1}{8}k^2(f''(U^n)((U^n)')^2 + (U^n)''f'(U^n)) + O(k^3),$$

so that since $f(U^n) = (U^n)'$ we'll have:

$$f'(U^n) = \frac{(U^n)''}{(U^n)'}$$

and then:

$$f''(U^n) = \frac{(U^n)'''(U^n)' - ((U^n)'')^2}{((U^n)')^2}$$

So introducing these the Taylor expansion above:

$$= (U^n)' + \frac{1}{2}k(U^n)'' + \frac{1}{8}k^2((U^n)'''(U^n)' - ((U^n)'')^2 + \frac{((U'')^n)^2}{(U^n)'}) + O(k^3)$$

Expanding out the first part of the truncation error:

$$\frac{1}{k}(U^n-U^n+k(U')^n+\frac{k^2}{2}(U'')^n+\frac{k^3}{6}(U''')^n)+O(k^2) = (U')^n+\frac{k}{2}(U'')^n+\frac{k^2}{6}(U''')^n+O(k^3).$$

So combining these our truncation error is:

$$\tau^n = k^2\left(\frac{1}{6}(U''')^n - \frac{1}{8}(U''')^n(U')^n + \frac{1}{8}((U'')^n)^2 - \frac{1}{8}\frac{((U'')^n)^2}{(U^n)'}\right) + O(k^3)$$

□

Instructor: P. Choboter
Name: Joseph McGuire

Class: Math 502 (W2022)

Homework # 4

Exercise 6.3 (consistency and zero-stability of LMMs) Which of the following Linear Multistep Methods are convergent? For the ones that are not, are they inconsistent, or not zero-stable, or both?

(a) $U^{n+2} = \frac{1}{2}U^{n+1} + \frac{1}{2}U^n + 2kf(U^{n+1})$

Solution. We'll determine if this is convergent, or not, by testing for zero-stability and consistency. The characteristic polynomial of this difference equation is:

$$\xi^2 - \frac{1}{2}\xi - \frac{1}{2} = 0 \iff 2\xi^2 - \xi - 1 = 0.$$

This has roots of:

$$\xi = \frac{+1 \pm \sqrt{1+4}}{4} = \frac{1 \pm \sqrt{5}}{4} \leq 1.$$

Thus this is zero-stable.

Consistency:

$$\begin{aligned}
\tau^n &= \frac{U^{n+2} - \frac{1}{2}U^{n+1} - \frac{1}{2}U^n}{k} - 2f(U^{n+1}) \\
&= \frac{(U^n + 2k(U')^n) - \frac{1}{2}(U^n + k(U')^n) - \frac{1}{2}U^n}{k} - 2(U')^{n+1} + O(k) \\
&= \frac{(2k(U')^n) - \frac{1}{2}(k(U')^n)}{k} - 2((U')^n + k(U'')^n) + O(k) \\
&= (2(U')^n) - \frac{1}{2}((U')^n) - 2((U')^n) + O(k) \\
&= O(1)
\end{aligned}$$

So this scheme is not consistent.

To summarize, the scheme is not consistent, is zero stable, thus doesn't converge because of inconsistency. □

(b) $U^{n+1} = U^n$

Solution. Zero-Stability:

$$\xi = 1 \leq 1\checkmark$$

Consistency:

$$\tau^n = \frac{U^{n+1} - U^n}{k} = \frac{(U')^n k}{k} + O(k) = (U')^n + O(k)$$

So that this scheme is not consistent, but is zero-stable, but overall it is not convergent. □

(c) $U^{n+4} = U^n + \frac{4}{3}k(f(U^{n+3}) + f(U^{n+2}) + f(U^{n+1}))$

Solution. Zero-Stability: The characteristic polynomial is:

$$\xi^4 - 1 = 0 \iff \xi = -1, 1, -i, i,$$

each having magnitude less than or equal to 1 hence this is zero-stable!
Consistency:

$$\begin{aligned}\tau^n &= \frac{U^{n+4} - U^n}{k} - \frac{4}{3}((U')^{n+3} + (U')^{n+2} + (U')^{n+1}) \\ &= 4(U')^n - \frac{4}{3}\left((U')^n + (U')^n + (U')^n\right) + O(k) \\ &= 0 + O(k)\end{aligned}$$

Note in the first step I immediately used $f(U^n) = (U')^n$. This shows the scheme is consistent.
Since this is consistent and zero-stable we have that this is convergent. □

(d) $U^{n+3} = -U^{n+2} + U^{n+1} + U^n + 2k(f(U^{n+2}) + f(U^{n+1}))$.

Solution. The characteristic equation is:

$$\xi^3 + \xi^2 - \xi - 1 = 0 \iff (\xi + 1)^2(\xi - 1) = 0.$$

So since $\xi = -1$ is a repeated root, we have that this is no zero-stable!
To determine if this is consistent:

$$\tau^n = \frac{U^{n+3} + U^{n+2} - U^{n+1} - U^n}{k} - 2(f(U^{n+2}) + f(U^{n+1}))$$

We'll need this to go to 0 as $k \to 0$ to be consistent:

$$\begin{aligned}\tau^n &= \frac{U^n + 3k(U')^n + \frac{9}{2}k^2(U'')^n + U^n + 2k(U')^n + 4k^2(U'')^n - U^n - k(U')^n - \frac{1}{2}k^2(U'')^n - U^n}{k} \\ &\quad - 2((U')^{n+2} + (U')^{n+1}) \\ &= \frac{3k(U')^n + \frac{9}{2}k^2(U'')^n + 2k(U')^n + 4k^2(U'')^n - k(U')^n - \frac{1}{2}k^2(U'')^n}{k} \\ &\quad - 2(2(U') + 3k(U'')^n) \\ &= \frac{(3k + 2k - k)(U')^n}{k} - 2(2(U')) + O(k) \\ &= (5 - 2 - 4)(U')^n + O(k) \\ &= -(U')^n + O(k)\end{aligned}$$

So this is not consistent and not zero-stable. □

Exercise 6.5 (Solving a difference equation)

(a) Determine the general solution to the linear difference equation

$$2U^{n+3} - 5U^{n+2} + 4U^{n+1} - U^n = 0.$$

Hint: One root of the characteristic polynomial is at $\xi = 1$.

Solution. The characteristic polynomial of this difference being:

$$2\xi^3 - 5\xi^2 + 4\xi - 1 = 0,$$

factoring this:

$$(\xi - 1)^2 \left(\xi - \frac{1}{2}\right) = 0.$$

Since we have a repeated root, we have the general solution of the difference equation is:

$$\begin{aligned}U^n &= c_1 1^n + c_2 n 1^n + c_3 \left(\frac{1}{2}\right)^n \\ &= c_1 + c_2 n + c_3 \left(\frac{1}{2}\right)^n.\end{aligned}$$

□

(b) Determine the solution to this difference equation with the starting values $U^0 = 11, U^1 = 5$, and $U^2 = 1$. What is U^{10}?

Homework # 4

Solution. With initial information $U^0 = 11, U^1 = 5, U^2 = 1$, we get the following system of equations:

$$\begin{aligned} c_1 + 0c_2 + 1c_3 &= 11 \\ c_1 + c_2 + \frac{1}{2}c_3 &= 5 \\ c_1 + 2c_2 + \frac{1}{4}c_3 &= 1. \end{aligned}$$

Solving for $c_1 = 3$, $c_2 = -2, c_3 = 8$, and this gives a solution for U^{10} of

$$U^{10} \approx -16.9922$$

□

(c) Consider the LMM

$$2U^{n+3} - 5U^{n+2} + 4U^{n+1} - U^n = k(\beta_0 f(U^n) + \beta_1 f(U^{n+1})).$$

For what values of β_0 and β_1 is the local truncation error $O(k^2)$?

Solution. The local truncation error for this difference equation will be given by:

$$\tau^n = \frac{1}{k}(2U^{n+3} - 5U^{n+2} + 4U^{n+1} - U^n) - (\beta_0 f(U^n) + \beta_1 f(U^{n+1})),$$

since at the true solution we have $f(U^n) = (U')^n$ this simplifies to:

$$\begin{aligned} \tau^n &= \frac{2}{k}(U^n + (U')^n 3k + \frac{9}{2}k^2(U'')^n + \frac{27}{6}k^3(U''')^n) \\ &\quad - \frac{5}{k}(U^n + (U')^n 2k + (U'')^n 2k^2 + (U''')^n \frac{8}{6}k^3) \\ &\quad + \frac{4}{k}(U^n + (U')^n k + (U'')^n \frac{k^2}{2} + (U''')^n \frac{k^3}{6}) - \frac{1}{k}U^n \\ &\quad - (\beta_0 (U')^n + \beta_1 (U')^{n+1}) \\ &= \frac{1}{k}(2 - 5 + 4 - 1)U^n + (6 - 10 + 4)(U')^n \\ &\quad + k\,(9 - 10 + 2)\,(U'')^n + k^2(U''')^n(\frac{54}{6} - \frac{40}{6} + \frac{4}{6}) \\ &\quad - (\beta_0 (U')^n + \beta_1 (U')^{n+1}) \\ &= k\,(9 - 10 + 2)\,(U'')^n - (\beta_0 (U')^n + \beta_1 (U')^{n+1}) + O(k^2) \\ &= k\,(9 - 10 + 2)\,(U'')^n - (\beta_0 (U')^n + \beta_1 ((U')^n + (U'')^n k) + O(k^2) \\ &= -(\beta_0 + \beta_1)(U')^n + (1 - \beta_1)k(U'')^n + O(k^2). \end{aligned}$$

So we need $\beta_1 = 1$ and $\beta_0 = -\beta_1 = -1$. □

(d) Suppose you use the values of β_0 and β_1 just determined in this LMM. Is this a convergent method?

Solution. We have that this is consistent by our previous work (we showed $\tau^n \to 0$ as $k \to 0$), so we just need to show that this is zero-stable.

$$2\xi^3 - 5\xi^2 + 4\xi - 1 = 0 \iff (\xi - 1)^2 \left(\xi - \frac{1}{2}\right),$$

so since we have a repeated root of $\xi = 1$, we have an issue, namely we need this to be a strict inequality which it is not. So this is not zero-stable.
So this not a convergent scheme. □

Exercise 6.6 (Solving a difference equation)

(a) Find the general solution of the linear difference equation

$$U^{n+2} - U^{n+1} + 0.25U^n = 0.$$

Proof. The characteristic polynomial is:

$$\xi^2 - \xi + 0.25 = 0 \iff \left(\xi - \frac{1}{2}\right)^2 = 0,$$

so this has a general solution:

$$U^n = A\frac{1}{2^n} + Bn\frac{1}{2^n}.$$

□

(b) Determine the particular solution with initial data $U^0 = 2, U^1 = 3$. What is U^{10}?

Proof. Using the initial data:

$$2 = A \qquad 3 = 1 + B\frac{1}{2} \iff B = 4$$

so that gives us:

$$U^n = \frac{1}{2^{n-1}} + \frac{n}{2^{n-2}}.$$

So that

$$U^{10} = \frac{1}{2^9} + \frac{10}{2^8} \approx 0.041$$

□