

Damian Franco

CS 375 – 001

Homework #7:

*NOTE: All matrices are listed in MATLAB syntax.*

- 1) For number 1, we were given two linear systems to solve in both part (a) and part (b). The  $x$  approximation was also given to us. For both, we had to find the infinity-norm condition number, the relative forward error, relative backward error, and the error magnification factor. Below, you can find the work for both respective parts, alongside an explanation.
  - a) We are given the following matrix for matrix  $A$   $\begin{bmatrix} 1 & 2 \\ 2 & 4.01 \end{bmatrix}$ , the following vector  $b$   $\begin{bmatrix} 3 \\ 6.01 \end{bmatrix}$  and the  $x_A$  vector approximation  $\begin{bmatrix} -100 \\ 52 \end{bmatrix}$ . Looking at the matrix  $A$  and vector  $b$ , I can use intuition to find out the “exact answer” and construct the vector  $x$   $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ . Now since all the parts for the problem have been found, the norms can be taken to find the forward and backward errors. After the norms are found, then the backward and forward errors can be found, which leads into the finding of the error magnification factor. The  $\infty$ -norm condition number is found by is now found by multiplying the norm of  $A$  with the norm of  $A^{-1}$ . Below, you can find two separate answers if values vary with what MATLAB evaluated (says MATLAB) and what I evaluated (says MY WORK). Everything was based directly on the narrations and notes on the cond2 module. All values can be found below. Looking at the infinity-norm condition number, it can be seen to be different from the error magnification factors that were found in both MATLAB and my own work, which shows that there might be a miscalculation.

*Exact answer for  $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$*

*Norm( $x_A - x$ ) = **101***

*Norm( $x$ ) = **1***

*Norm( $b - A * x_A$ ) = **2.51***

*Norm( $b$ ) = **6.01***

*Forward error = **101***

*Backward error = **0.449562349914829** (MATLAB) or **0.4176372712** (MY WORK)*

*Error magnification factor = **2.246629416790236e+02** (MATLAB) or **241.8366534** (MY WORK)*

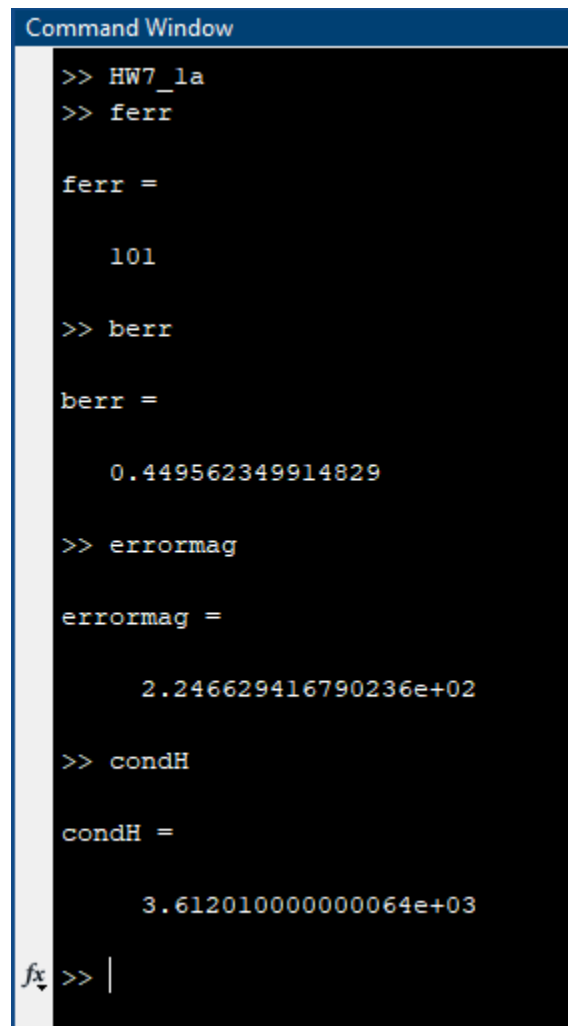
*$\infty$ -norm condition number = **3.612010000000064e+03***

## Source Code:

*HW7\_1a.m*

```
1 % function HW7_1a
2 % Sets up the all the values for question
3 % 1, part a on HW7 and calculates everything
4 % the question asks us to calculate
5 % Part (a)
6 A = [1 2; 2 4.01]; % A matrix
7 b = [3; 6.01]; % b matrix
8 x = [1; 1]; % x answer "exact"
9 xA = [-100; 52]; % x answer approximation
10 ferr = norm(xA - x, inf) / norm(x, inf); % forward error
11 berr = norm(b - A * xA) / norm(b, inf); % backward error
12 errormag = ferr / berr; % error magnification
13 condH = cond(A, inf); % infinity norm condition #
```

## MATLAB Output:



```
Command Window
>> HW7_1a
>> ferr

ferr =

    101

>> berr

berr =

    0.449562349914829

>> errormag

errormag =

    2.246629416790236e+02

>> condH

condH =

    3.612010000000064e+03

fx >> |
```

- b) For this part, we are given the following matrix for matrix  $A$   $\begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix}$ , the following vector  $b$   $\begin{bmatrix} 3 \\ 7 \end{bmatrix}$  and the  $x_A$  vector approximation  $\begin{bmatrix} -2 \\ -3 \end{bmatrix}$ . Looking at the matrix  $A$  and vector  $b$ , I can use intuition to find out the “exact answer” and construct the vector  $x$   $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ . I found everything the same way, using the same steps with the last part. Below, you can find two separate answers if values vary with what MATLAB evaluated (says MATLAB) and what I evaluated (says MY WORK). The findings are shown below. Quick note, when evaluated the backward error, there was a significant difference that influenced different outcomes for the backward error and error magnification factor. Looking at the infinity-norm condition number, it can be seen to be different from the error magnification factors that were found in both MATLAB and my own work. Looking at the infinity-norm condition number, it can be seen to be the same as my calculations which shows a possible miscalculation with MATLABs function.

*Exact answer for  $x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$*

*Norm( $x_A - x$ ) = 3*

*Norm( $x$ ) = 1*

*Norm( $b - A * x_A$ ) = 1.414213562373095 (MATLAB) or 1 (MY WORK)*

*Norm( $b$ ) = 7*

*Forward error = 3*

*Backward error = 1.414213562373095 (MATLAB) or 0.1428571429 (MY WORK)*

*Error magnification factor = 14.849242404917497 (MATLAB) or 20.99999999 (MY WORK)*

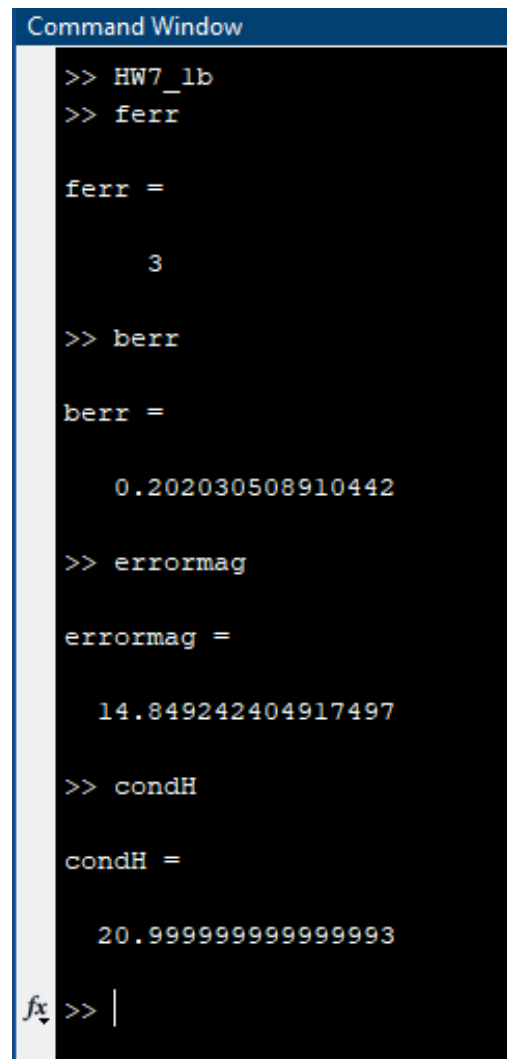
*$\infty$ -norm condition number = 20.999999999999993*

## Source Code:

*HW7\_1b.m*

```
1 % function HW7_1a
2 % Sets up the all the values for question
3 % 1, part b on HW7 and calculates everything
4 % the question asks us to calculate
5 % Part (b)
6 A = [1 -2; 3 -4]; % A matrix
7 b = [3; 7]; % b matrix
8 x = [1; -1]; % x answer "exact"
9 xA = [-2; -3]; % x answer approximation
10 ferr = norm(xA - x, inf) / norm(x, inf); % forward error
11 berr = norm(b - A * xA) / norm(b, inf); % backward error
12 errormag = ferr / berr; % error magnification
13 condH = cond(A, inf); % infinity norm condition #
```

## MATLAB Output:



```
Command Window
>> HW7_1b
>> ferr

ferr =

     3

>> berr

berr =

 0.202030508910442

>> errormag

errormag =

 14.849242404917497

>> condH

condH =

 20.999999999999993

fx >> |
```

2) For number 2, we must solve the linear system given to us and then solve different steps given to us in the next parts a through d. The work for each part is shown below.

a) For part (a), I solved the linear system given to us exactly with all the work below and I also formed some good approximation values based of the exact answers for  $x_1$  and  $x_2$ . I ended up using Gaussian elimination to solve the matrix below and found that the exact matrix is  $[-5; 2]$  where  $x_1 = -5$  and  $x_2 = 2$ . I then approximated  $x_A$  close to the exact matrix which turned out to be  $[-4; 1]$  where  $x_1 = -4$  and  $x_2 = 1$ . The work is shown below.

$$2a) A = \begin{bmatrix} 10^{-16} & 1 \\ 1 & 3 \end{bmatrix} \quad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 10^{-16} & 1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{array}{l} 10^{-16} x_1 + 1x_2 = 2 \\ 1 x_1 + 3x_2 = 1 \end{array} \rightarrow \begin{bmatrix} 10^{-16} & 1 & 2 \\ 1 & 3 & 1 \end{bmatrix}$$

$$R1 \leftrightarrow R2 = \begin{bmatrix} 1 & 3 & 1 \\ 10^{-16} & 1 & 2 \end{bmatrix} \rightarrow R2 - (10^{-16}) \cdot R1 = \begin{bmatrix} 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

$$R2 - R1 \cdot (3) = \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 2 \end{bmatrix} \quad \begin{array}{l} x_1 = -5 \\ x_2 = 2 \end{array} \quad \boxed{x = \begin{bmatrix} -5 \\ 2 \end{bmatrix}}$$

$$\text{A good approximation} = \boxed{x_A = \begin{bmatrix} -4 \\ 1 \end{bmatrix}}$$

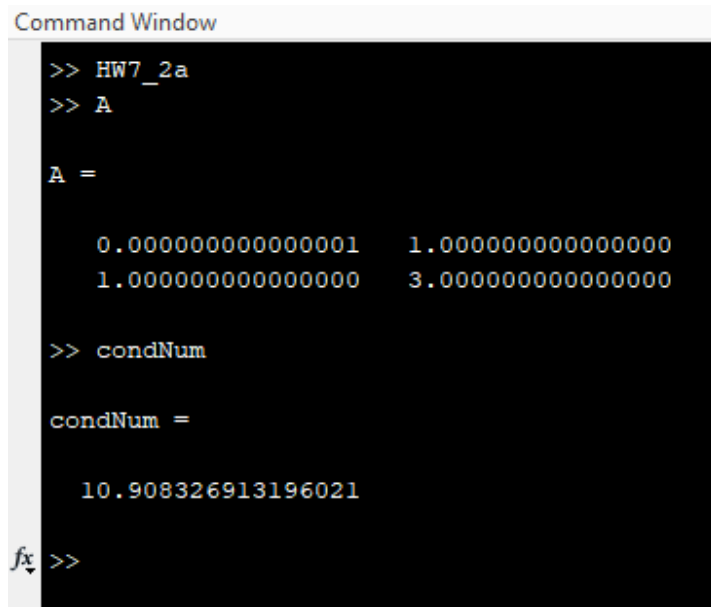
- b) For part (b), I we had to use the MATLAB *cond* function to approximate the 2-norm condition number of the given linear system. The function produced the number **10.908326913196021**. Since the whole number of the condition number is 10, then I can safely say that the condition number for this matrix is well conditioned. There will be small errors to precision and rounding, but overall this linear system is not ill-conditioned because an ill-conditioned condition number will be tens of times larger than this condition number.

### Source Code:

*HW7\_2.m*

```
1      % Problem 2 (b)
2      A = [10e-16 1; 1 3];
3      b = [2; 1];
4      condNum = cond(A);
```

### MATLAB Output:



```
Command Window
>> HW7_2a
>> A

A =

    0.0000000000000001    1.0000000000000000
    1.0000000000000000    3.0000000000000000

>> condNum

condNum =

    10.908326913196021

fx >>
```

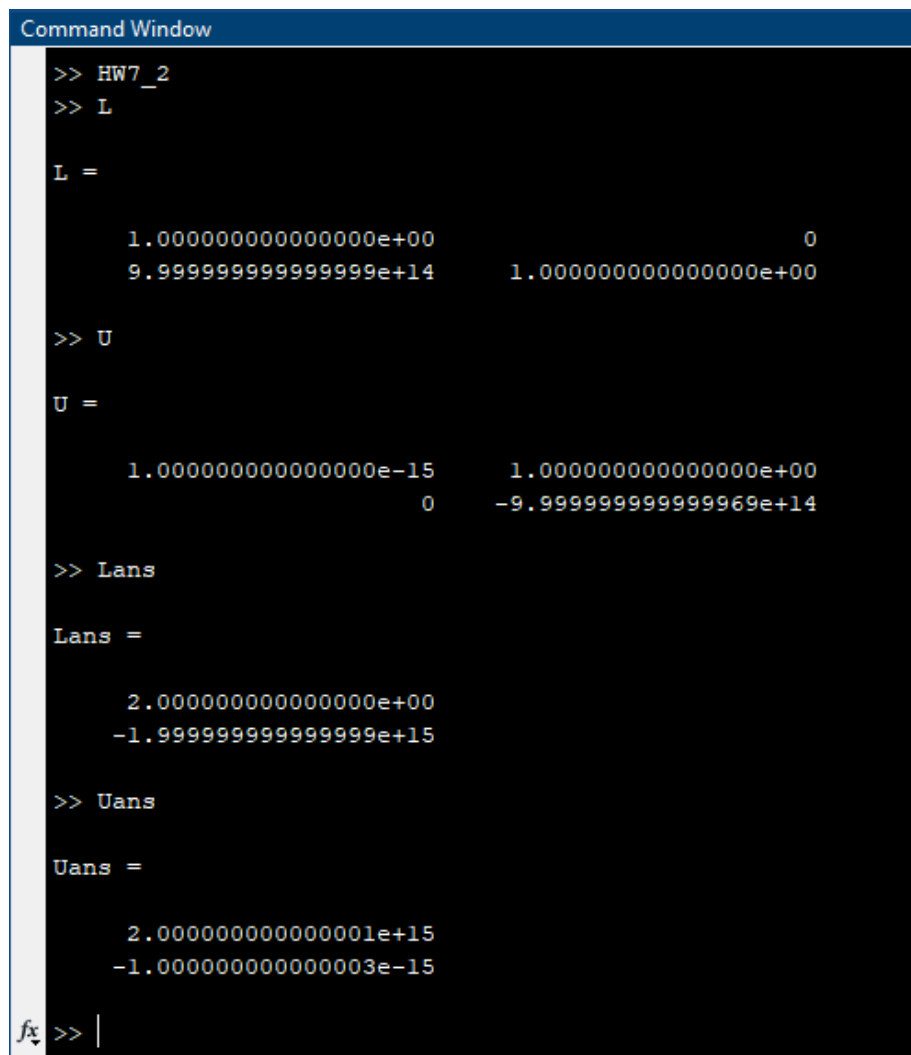
- c) For part (c), we had to solve the linear system numerically with the MATLAB functions *GE.m*, *LTriSol.m*, and *UTriSol.m*. Below you can find the results of each function with screenshots from the MATLAB console. For the Gaussian elimination without pivoting technique, there are a lot of precision errors here. This method generates  $x_1 = 2.0$  and  $x_2 = -1.9999999999999999e+15$  for the lower triangular solve and  $x_1 = 2.0000000000000001e+15$  and  $x_2 = -1.0000000000000003e-15$  for the upper triangular solve, which produces both significant errors compared to the exact answer -5 and 2. The non-pivoting nature comes up with very prominent precision errors compared to when you solve with pivots.

## Source Code:

*HW7\_2.m*

```
1      % Problem 2 (c)
2      A = [10e-16 1; 1 3];
3      b = [2; 1];
4      [L, U] = GE(A);
5      Lans = LTriSol(L, b);
6      Uans = UTriSol(U, b);
```

## MATLAB Output:



```
Command Window
>> HW7_2
>> L

L =

    1.000000000000000e+00    0
    9.999999999999999e+14    1.000000000000000e+00

>> U

U =

    1.000000000000000e-15    1.000000000000000e+00
         0    -9.999999999999999e+14

>> Lans

Lans =

    2.000000000000000e+00
   -1.999999999999999e+15

>> Uans

Uans =

    2.000000000000001e+15
   -1.000000000000003e-15

fx >> |
```

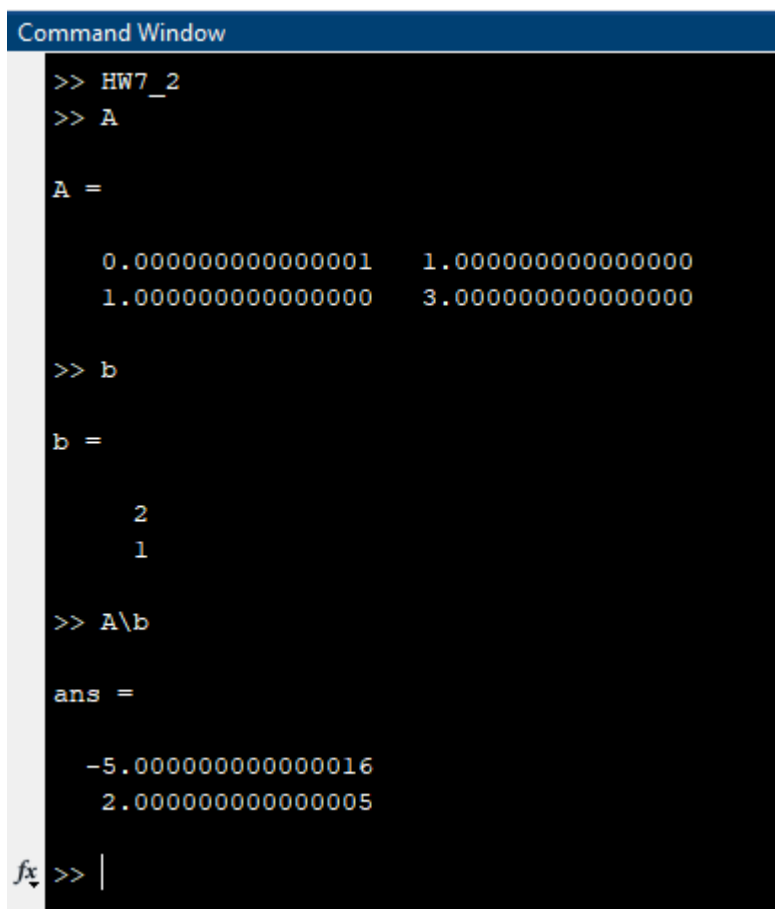
- d) For part (d), we had to repeat what we did in part (c), but with MATLAB's built in `\`, "backslash function." The function solves the linear system for the vector  $x$ . The MATLAB output is shown below with screenshots. This function ends up printing the "exact" answer of the linear system. The output is  $[-5.0000000000000016; 2.0000000000000005]$ , which is almost the same as I got in part (a) but has minor precision errors on both outputs. This function does use partial pivoting, which could also cost some time and precision issues, which could be a reason why the precision errors have occurred here. This a good example of numerical computing and how everything you write and perform on a computer can have/cause precision errors.

### Source Code:

*HW7\_2.m*

```
1      % Problem 2 (d)
2      A = [10e-16 1; 1 3];
3      b = [2; 1];
4      ans = A \ b;
```

### MATLAB Output:



```
Command Window

>> HW7_2
>> A

A =

    0.0000000000000001    1.0000000000000000
    1.0000000000000000    3.0000000000000000

>> b

b =

     2
     1

>> A\b

ans =

-5.0000000000000016
 2.0000000000000005

fx >> |
```



- 3) For number 3, we were asked to create a Hilbert matrix that would be based off a  $n$  variable that would dictate how big the matrix is ( $n$  by  $n$ ) and every entry within the matrix  $A_{ij}$  ( $A_{ij} = 1/(|i - j|^2 + 1)$ ). The entries will have an effect of  $1/n$ . We then had to solve for  $n = 50, 100, 200, 300$  and  $400$ . I set this up through MATLAB and based my code off the last example in the notes for cond2. For every  $n$  entry the exact vector matrix will equal to  $(1, 1, 1 \dots 1, 1)$  to the  $n$  times. I first found the right-hand side vector  $b$  with the equation  $b = A*x$ . Next, I computed an approximate solution with MATLAB using MATLAB's `\` "backslash function. After that, both backward and forward errors are computed by finding the infinity norms of  $b$ ,  $x$ ,  $xA - x$  and  $b - A*x$ . After the errors are found, then the error magnification factor is found by dividing the forward error by the backward error. Lastly, the condition number is found with MATLAB's `cond` function. Looking at the data retrieved, I can notice that both forward and backward errors seem to be increasing as the input number goes up. I did expect this because with more input and different input, the errors seem to increase. The error magnification factors seem to be fluctuating when increasing the input  $n$  size. This is expected since the errors are both getting significantly bigger, then the error magnification factor will fluctuate like this, but also increase as the errors increase. The condition number is increasing and that is expected because this is an ill-conditioned linear system that will continue to grow as the errors and  $n$  input increases. Overall, this is a great example of something I would like to avoid in the future because of all the precision issues and size of the errors and the condition number.

### Source Code:

HW7\_3.m

```

1      % Problem 3 - HW7
2      n = 50; % set n
3      A = hilb(n); % create Hilbert matrix
4      x = ones(n,1); % the "exact" answer
5      b = A*x; % the right-hand vector
6      xA = A\b; % solve with MATLAB's backslash
7      ferr = norm(xA-x,inf)/norm(x,inf); % forward error
8      berr = norm(b-A*xA)/norm(b,inf); % backward error
9      errormag = ferr/berr; % error magnification
10     H = cond(A,inf); % infinity-norm condition number

```

### Table of outputs:

n	f.errors	b.errors	m.factor	cond.num
50	1.6877e+02	7.5373e-16	2.2391e+17	3.2157e+19
100	3.9942e+02	2.5439e-15	1.5701e+17	2.0866e+21
200	3.7776e+02	2.8447e-15	1.3279e+17	6.9251e+20
300	3.4912e+03	2.3852e-14	1.4637e+17	3.4244e+20
400	4.3159e+03	3.2746e-14	1.3180e+17	9.8482e+20

Table that shows all that was asked for int problem 3 on HW7

## MATLAB Output:

$n = 50$

```
Command Window

>> ferr

ferr =

    1.6877e+02

>> berr

berr =

    7.5373e-16

>> errormag

errormag =

    2.2391e+17

>> H

H =

    3.2157e+19

fx >> |
```

```
Command Window

>> xA

xA =

    1.0000e+00
    1.0002e+00
    9.9337e-01
    1.1177e+00
   -9.0119e-02
    6.7671e+00
   -1.6800e+01
    3.1489e+01
   -2.3197e+01
    1.0807e+01
   -3.4895e+01
    6.4245e+01
    3.2812e+00
   -5.3407e+01
   -1.6760e+01
    4.1553e+01
    2.2682e+01
   -2.6677e+01
    3.1049e+01
   -5.1478e+00
   -5.2046e+01
    2.6560e+01
    1.5225e+01
   -2.8303e+01
    7.1814e+00
    7.4833e+01
   -3.9287e+01
   -6.3255e+01
    3.3204e+01
    8.4963e+00
    9.1767e+01
   -1.4935e+01
   -1.0551e+02
   -7.4714e+01
    5.4165e+01
    1.5248e+02
   -6.1544e+01
   -3.1717e+01
   -1.4475e+00

fx
```

$$n = 100$$

```

Command Window

>> ferr

ferr =

    3.9942e+02

>> berr

berr =

    2.5439e-15

>> errormag

errormag =

    1.5701e+17

>> H

H =

    2.0866e+21

fx >> |

```

```

>> xA

xA =

    1.0000e+00
    9.9978e-01
    1.0106e+00
    7.9065e-01
    3.1234e+00
   -1.1156e+01
    4.1036e+01
   -6.8837e+01
    3.7381e+01
    6.7465e+01
   -7.5890e+01
   -5.3061e+01
    8.0744e+01
    1.5671e+01
   -1.2128e+01
    3.5022e+01
   -7.1798e+01
   -1.7457e+01
    1.8734e+01
    7.3576e+01
   -1.0183e+02
    5.0233e+01
    8.3993e+01
   -8.0625e+01
    1.5468e+01
    2.5909e+01
   -4.4306e+01
    4.1768e+01
   -7.0616e+01
    5.7926e+01
   -1.0476e+02
    5.8173e+01

fx

```



$$n = 300$$

```
Command Window
>> ferr
ferr =
    3.4912e+03
>> berr
berr =
    2.3852e-14
>> errormag
errormag =
    1.4637e+17
>> H
H =
    3.4244e+20
fx >> |
```

```
Command Window

>> xA

xA =

    9.9999e-01
    1.0023e+00
    9.0710e-01
    2.6185e+00
   -1.3853e+01
    7.9658e+01
   -2.4500e+02
    4.2970e+02
   -3.0103e+02
   -1.4947e+02
    1.7062e+02
    4.5649e+02
   -5.6412e+02
    4.4243e+01
   -5.6239e+01
    1.3975e+02
    3.5426e+02
   -7.3922e+02
    9.5803e+02
   -6.3727e+02
   -2.7437e+02
    6.1824e+02
   -2.8278e+02
   -6.7844e+01
   -2.7735e+02
    3.7864e+02
   -1.4790e+02
    9.5317e+02
   -5.0483e+02
   -3.3196e+02
    5.7532e+02
   -7.5816e+02
   -1.4218e+02
   -7.6833e+02
```

$n = 400$

```
Command Window

>> ferr

ferr =

    4.3159e+03

>> berr

berr =

    3.2746e-14

>> errormag

errormag =

    1.3180e+17

>> H

H =

    9.8482e+20

fx >> |
```

```
Command Window

>> xA

xA =

    1.0000e+00
    9.9806e-01
    1.0838e+00
   -5.3632e-01
    1.5660e+01
   -7.8664e+01
    2.5120e+02
   -4.1237e+02
    1.7585e+02
    4.7322e+02
   -5.3993e+02
   -2.9746e+02
    5.7378e+02
    7.0560e+01
   -2.5686e+02
    6.8914e+01
    5.0054e+01
   -2.1977e+02
    2.0423e+01
    9.9193e+01
   -1.7562e+00
    1.8503e+02
   -1.4092e+01
   -4.7993e+02
    8.5403e+01
    9.4304e+02
   -2.2222e+02
   -8.0633e+02
   -7.0234e+01
    4.8328e+02
   -7.2781e+01
   -7.4398e+01
   -1.0513e+02
   -3.0032e+02
   - - - - -
```