Exercise P14: Equation 10.1 on page 77 is a cartoon of how Householder triagularization works on a 5x3 matrix. Turn this cartoon into a specific calculation by showing the stages A, Q_1A, Q_2Q_1A, and $Q_3Q_2Q_1A = R$ on the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 0 & -2 \\ 1 & 3 & 5 \\ 4 & 5 & 6 \\ 3 & -3 & 3 \end{bmatrix}$$

Solution:
First we need to construct the first Householder reflector F_1. Recall equation 10.4 and note that geometrically we want $v = ||x||e_1 - x$ for each x pulled from A,

$$F = I - 2\frac{vv^*}{v^*v}.$$

To construct each unitary recall equation 10.2 and we get,

$$Q_k = \begin{bmatrix} I & 0 \\ 0 & F \end{bmatrix}.$$

Consider the following MATLAB script which computes each unitary Q_i and stores them in a Matlab cell,
Code:

```
function [C] = P14(A)
%This function takes a tall matrix A
%and returns a storage array C that contains the
%Unitary matrices used to perform householder QR


%Pulling dimensions and initializing storage array.
[m,n] = size(A);
C = {1, n};

    for i = 1:n
        %Computing v.
        x = A(i:m, i);
        v = (norm(x)*eye(m+1-i,1))-x;
        %Computing F.
        F = eye(m+1-i, m+1-i) - 2*((v*v')/(v'*v));
        %Form Q_i.
        Q = eye(m,m);
        Q(i:m, i:m) = F;
        %Store Q_i.
        C{1,i} = Q;
        %Apply Q_i to rest of matrix for the next step.
```

```
            A = Q*A;
        end
    end
```

Console:

```
>> A

A =
       1       2       3
       2       0      -2
       1       3       5
       4       5       6
       3      -3       3

>> Q = P14(A);
>> Q{1,1}*A

    5.5678    2.8737    6.6454
   -0.0000   -0.3825   -3.5961
   -0.0000    2.8087    4.2019
   -0.0000    4.2349    2.8077
   -0.0000   -3.5738    0.6058

>> Q{1,2}*Q{1,1}*A

    5.5678    2.8737    6.6454
   -0.0000    6.2243    3.6796
   -0.0000   -0.0000    1.1088
   -0.0000    0.0000   -1.8560
   -0.0000    0.0000    4.5415

>> Q{1,3}*Q{1,2}*Q{1,1}*A

    5.5678    2.8737    6.6454
   -0.0000    6.2243    3.6796
   -0.0000    0.0000    5.0298
   -0.0000    0.0000    0.0000
   -0.0000   -0.0000    0.0000
```

Exercise P15: The matlab built-in qr() computes the *QR* factorization using Householder

transformations as described in lecture 10. This problem asks you to go ahead and use it! The purpose of this problem, is to show that *QR* had a completely different purpose.

a. By searching for 'Unsolvable quintic polynomials, ' for example, confirm that there is a theorem which shows that fifth and higher-degree polynomials cannot be solved using finitely many operations including roots. In other words, there is no finite formula for the roots of such polynomials. Who proved this theorem and when? Show a quintic polynomials for which it is know that there is no finite formula.

Solution:
The theorem regarding the solvability of quintic polynomials goes as follows: 'there exists no solution in radicals to general polynomial equations of degree five or higher with arbitrary coefficients. From what I was able to find online the theorem is called the Abel-Ruffini theorem and apparently it first appeared in Paolo Ruffini's *Teorie generale delle equazioni* in 1799. In 1824 Norwegian mathematician Niels Henrik Abel would go on to produce a more rigorous proof of the theorem. Later Evariste Galois would go on to prove a more generalized form of the theorem (I'd need to do some more reading before going into further detail) but died in a duel before publishing any of his work. Galois' work showed that $x^5 - x - 1 = 0$ was the simplest polynomial which cannot be solved using finitely many operations.

b. At the Matlab/Octave command line, try the following...
Console:

```
>> A

A =
     1     2     3
     2     0    -2
     1     3     5
     4     5     6
     3    -3     3

>> Q = P14(A);
>> Q{1,1}*A

    5.5678    2.8737    6.6454
   -0.0000   -0.3825   -3.5961
   -0.0000    2.8087    4.2019
   -0.0000    4.2349    2.8077
   -0.0000   -3.5738    0.6058

>> Q{1,2}*Q{1,1}*A

    5.5678    2.8737    6.6454
```

```
   -0.0000    6.2243    3.6796
   -0.0000   -0.0000    1.1088
   -0.0000    0.0000   -1.8560
   -0.0000    0.0000    4.5415

>> Q{1,3}*Q{1,2}*Q{1,1}*A

    5.5678    2.8737    6.6454
   -0.0000    6.2243    3.6796
   -0.0000    0.0000    5.0298
   -0.0000    0.0000    0.0000
   -0.0000   -0.0000    0.0000
```

We can see that as we increase the number of iterations the value of A_i converges to the eigenvalue matrix of A.

c. Too see a bit more of what is going on in part (b.) show that

$$A_{i+1} = Q_i^* A_i Q_i.$$

This shows that A_{i+1} and A_i have the same eigenvalues, explain.

Solution:
First consider the QR decomposition of A_i, and note that,

$$A_i = Q_i R_i$$

Know left multiply by Q_i^* and right multiply by Q_i to get,

$$Q_i^* A_i Q_i = Q_i^* Q_i R_i Q_i = R_i Q_i.$$

Applying our definition of A_{i+1} we get,

$$Q_i^* A_i Q_i = A_{i+1}.$$

To show that A_{i+1} and A_i have the same eigenvalues, let's suppose the eigenvalue decomposition of $A_i = X^{-1}\Lambda X$. By substitution into our last equation we get,

$$Q_i^* X^{-1} \Lambda X Q_i = A_{i+1}.$$

Let $Y = Q_i^* X^{-1}$ and note that,

$$Y^{-1} = (Q_i^* X^{-1})^{-1} = X^{-1^{-1}} Q_i^{*^{-1}} = X Q_i.$$

Thus we get the following,

$$Y \Lambda Y^{-1} = A_{i+1}.$$

Therefore A_{i+1} must have the same eigenvalues as A_i.

d. Write a few sentences which relate parts a and b.

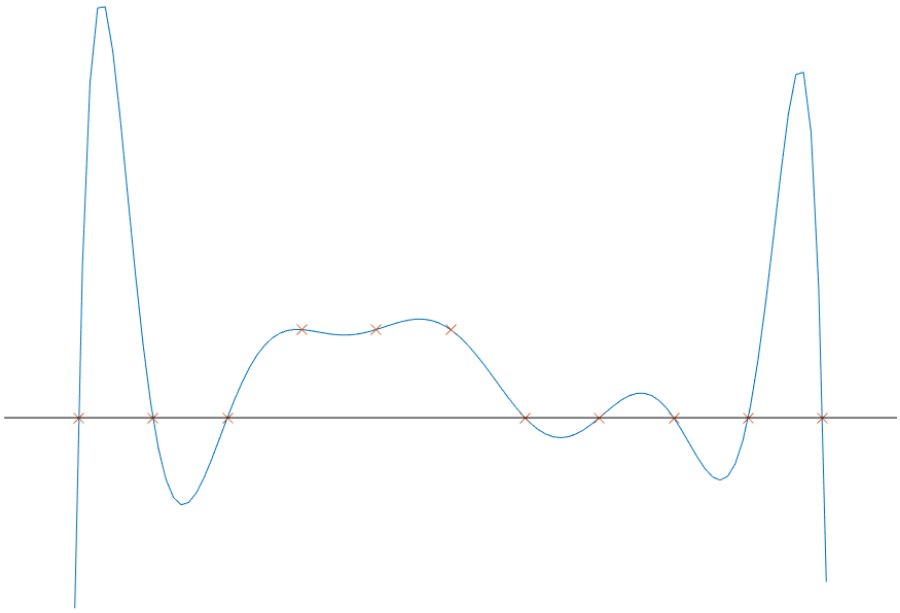
Solution:
What we want to highlight here is the relationship between polynomial root finding and eigenvalue problems. When we computed eigenvalues by-hand it involved creating the characteristic equation for a matrix, and solving for the roots of that polynomial gave us the eigenvalues. Part a shows us that there is no way to compute the roots of a quintic polynomial using exact arithmetic, and therefore there is no way of computing the eigenvalues of a rank 5 matrix. Part b shows us an iterative method for solving eigenvalue problems. Note that by back-substituting the recurrence relation in part a (with some matrix algebra) we get to product described in 25.4 of lecture 25.

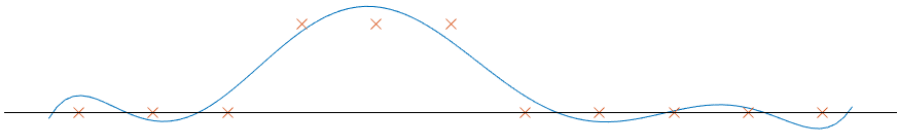
Exercise P16: Either by using the built in funciton polyfit() and polyval(), or by setting up linear systems and solving using Matlab's backslash command, reproduce Figures 11.1 and 11.2.

Solution:
Consider the following matlab code.
Code:

```
y = [0 0 0 1 1 1 0 0 0 0 0]
x = [-5 -4 -3 -2 -1 0 1 2 3 4 5]
%Fitting degree 10 polynomial
c = polyfit(x, y, 10)
xx = linspace(-5.1, 5.1, 100);
yy = polyval(c, xx)

%Plotting degree 10 polynomial
plot(xx,yy,x,y, 'x')
hold on
axis off
plot([-6 6], [0 0], 'k')
hold off

%Fitting degree 7 polynomial
c = polyfit(x, y, 7)
xx = linspace(-5.4, 5.4, 100);
yy = polyval(c, xx)
```

```
$Plotting degree 7 polynomial
plot(xx,yy,x,y, 'x')
hold on
axis off
plot([-6 6], [0 0], 'k')
ylim([-4 4])
```





Exercise P17: How closely, as measured in the L^2 norm on the interval $[1, 2]$ can the function $f(x) = x^{-1}$ be fitted by a linear combination of the functions e^x, $cos(x)$, and $\sqrt{x}$? Write a program that determines the answer to at least two digit of relative accuracy using a discretization of $[1, 2]$ and a discrete least squares problem. Write down your estimate of the error and also of the coefficients of the optimal linear combination, and produce a plot showing both $f(x)$ and the optimal approximation.

Solution:
Using MATLAB to fit the function we get the following function, a maximum residual of $\approx .038$ and an accuracy under the L^2 norm of $\approx .0105$,

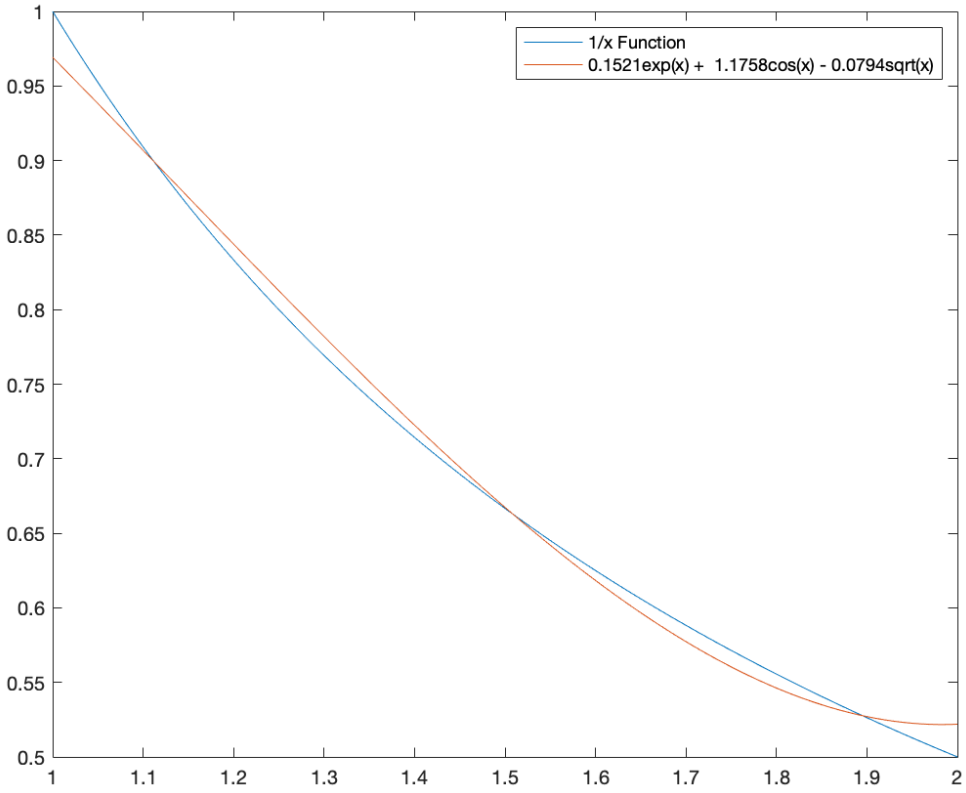
$$F(x) = 0.1521e^x + 1.1758\cos x - 0.0794\sqrt{x}$$

Console:

```
% Discretizing interval
x = (1:.01:2)';

% Setting up Design Matrix
A = [exp(x) cos(x) sqrt(x)];
```

```
% Defining the function
y = 1./x;

% Solving Least-Squares
c = A\y

    0.1521
    1.1758
   -0.0794

% Computing accuracy under the L2 Norm
norm(y - A*c)/sqrt(100)

    0.0105

% Maximum Residual
norm(y - A*c, 'inf')


    0.0308

%Plotting Curves
plot(x, y, x, A*c)
```

Figure 1: Plot of $f(x)$ and Best Approximation



Exercise 10.1: Determine the (a.)eigenvalues, (b.)determinant, and (c)singular values of a householder reflector, For the eigenvalues, give a geometric argument, as well as an algebraic proof.

a. Recall that a Householder reflector F is defined by $F = I - 2P$ where P is some orthogonal projector. Recall that in P12 we showed that projectors have eigenvalues of either 1 or 0. Furthermore, by Theorem 6.1 (equation 6.5) when an orthogonal

projector is formed by a single vector outer-product, like in F we know that,

$$P = Q\Lambda Q^*$$

$$\begin{bmatrix} | & | & | & \dots \\ v & q_1 & q_2 & \dots \\ | & | & | & \dots \end{bmatrix} \begin{bmatrix} 1 & \dots & \dots & \dots \\ \vdots & 0 & \dots & \dots \\ \vdots & \vdots & \ddots & \\ \vdots & \vdots & & 0 \end{bmatrix} \begin{bmatrix} | & | & | & \dots \\ v & q_1 & q_2 & \dots \\ | & | & | & \dots \end{bmatrix}^* .$$

Suppose P has the following eigenvalue decomposition, $P = Q^*\Lambda Q$. Let $I = Q^*Q$ and note that by substitution we get,

$$F = I - 2P = Q^*Q - 2Q^*\Lambda Q = Q^*(I - 2\Lambda)Q.$$

Consider $I - 2\Lambda$ and note that by substitution $\lambda_1 = -1$ and $\lambda_{>1} = 1$.
Consider Figure 10.2. It shows how a vector x is transformed under two possible reflectors F based on choice of P (or v) via the addition of $v = \|x\|e_1 - x$. Suppose we had some vector x which lies in the output space of F. By definition x must be of the form $\|x\|e_1$, and therefore when passed through F again we must get the same vector since $v = \|x\|e_1 - x$. When v is defined by $v = -\|x\|e_1 - x$ we get the similar $\lambda_1 = -1$ case.

b. From the previous problem we found an eigenvalue decomposition of $F = Q^*(I - 2\Lambda)Q$. Applying the determinant we get,

$$det(F) = det(Q^*(I - 2\Lambda)Q) = det(Q^*)det(I - 2\Lambda)det(Q) = 1 * -1 * 1 = -1.$$

c. For the singular values of F simply consider the eigenvalue decomposition from part a. Taking the absolute value of the $I - 2\Lambda$ term and adjusting Q accordingly we get an SVD where the singular values are all $\sigma = 1$.

Exercise 10.2: a. Write a Matlab function $[W, R] = house(A)$ that computes an implicit representation of a full QR factorization $A = QR$ of an mxn matrix A with $m \geq n$ using Householder reflections. The output variables are the lower-triangular matrix $W \in \mathbb{C}^{mxn}$ whose columns are the vectors v_k defining the successive Householder reflections, and a triangular matrix $R \in \mathbb{C}^{n,n}$

Solution:
Consider the following Matlab function,
Code:

```
function [W,R] = house(A)
%This function takes a tall matrix A and
%returns a matrix W with columns describing
%the householder reflectors and R.

%Pull dimensions
[m,n] = size(A);
if m < n
    error('Size Error')
end
%Initialize W
W = zeros(m, n);
    for i = 1:n
        %Computing v
        x = A(i:m, i);
        v = sign(x(1))*(norm(x)*eye(m+1-i,1)) + x;
        %Store v in W
        W(i:m, i) = v;
        %Computing R
        vnorm = v./norm(v);
        A(i:m, i:n) =  A(i:m, i:n) - 2*vnorm*(vnorm'* A(i:m, i:n));
    end
%return A as R
R = A(1:n, 1:n);
end
```

Console:

```
% Testing with A from P14
>> A = [3 1 4; 1 5 9;2 6 5 ;3 5 8]

>> [W,R] = house(A)

W =
    7.7958         0         0
    1.0000    9.7441         0
    2.0000    3.8712   -8.3579
    3.0000    1.8068   -0.3804


R =
   -4.7958   -7.2980  -11.4683
         0   -5.8085   -6.0780
         0         0    4.1876

% Testing R matrix
>> [Q, Rqr] = qr(A);
```

```
>> norm(Q*R - A)
ans =

    1.6012e-15
```

b. Write a Matlab function $Q = formQ(W)$ that takes the matrix W produced by *house*() as input and generates a corresponding *mxm* orthogonal matrix Q.

Solution:
Consider the following Matlab function,
Code:

```
function [Q] = formQ(W)
% This function takes a matrix W of vectors v
% forms the corresponding householder reflectors
% and computes the orthogonal matrix Q

%Pull dimensions
[m,n] = size(W);

if m < n
    error('Size Error')
end

Q = eye(m,m);
    for i = 1:n
        %Computing F.
        v = W(i:m,i);
        F = eye(m+1-i, m+1-i) - 2*((v*v')/(v'*v));

        %Form Q_i.
        QSubI = eye(m,m);
        QSubI(i:m, i:m) = F;
        Q = QSubI*Q;
    end
%Q = Q(1:n,:)'; I added this later
end
```

Console:

```
% Testing formQ() with matrix A from P14
>> [W,R] = house(A)
```

```
W =
    7.7958         0         0
    1.0000    9.7441         0
    2.0000    3.8712   -8.3579
    3.0000    1.8068   -0.3804

R =
   -4.7958   -7.2980  -11.4683
         0   -5.8085   -6.0780
         0         0    4.1876

>> Q = formQ(W)

Q =
   -0.6255   -0.2085   -0.4170   -0.6255
    0.6138   -0.5988   -0.5090   -0.0749
    0.1329    0.7090   -0.6869    0.0886
   -0.4629   -0.3086   -0.3086    0.7715

% formQ() computes Q' and R is in reduced form.
>> Q(1:3,:)'*R

    3.0000    1.0000    4.0000
    1.0000    5.0000    9.0000
    2.0000    6.0000    5.0000
    3.0000    5.0000    8.0000

>> norm(Q(1:3,:)'*R - A)

   2.6666e-15
```

Exercise 11.3: Take $m = 50$, $n = 12$. Using Matlabs's linspace, define t to be the m-vector corresponding to linearly spaced grid points from 0 to 1. Using Matlab's vander and fliplr, define A to be the mxn matrix associated with least squares fitting on this grind by a polynomial of degree $n - 1$. Take b to be the function $\cos(4t)$ evaluated on the grid. Now calculate and print(to 16 digit precision) the least squares coefficients vector x by six methods:

a-f. Compare the various algorithims used to solve least squares problems.

Solution:
Consider the following Matlab script.
Code:

```
LeastSquaresMethods = [];
m = 50;
n = 12;
t = linspace(0,1,m);
FullVandermonde = fliplr(vander(t));
A = FullVandermonde(:, 1:n);
b = cos(4*t');

%Part A Normal Equation
LeastSquaresMethods = [LeastSquaresMethods (A'*A)\(A'*b)];

%Part B Modified Gram Shmidt
[Qmgs, Rmgs] = mgs(A);
LeastSquaresMethods = [LeastSquaresMethods Rmgs\(Qmgs'*b)];

%Part C Householder
[WHouse,RHouse] = house(A);
QHouse = formQ(WHouse);
LeastSquaresMethods = [LeastSquaresMethods RHouse\(QHouse'*b)];

%Part D Householder(Matlab)
[Q, R] = qr(A);
LeastSquaresMethods = [LeastSquaresMethods R\(Q'*b)];

%Part E Matlab solve
LeastSquaresMethods = [LeastSquaresMethods A\b];

%Part F SVD solve
[U, S, V] = svd(A);
LeastSquaresMethods = [LeastSquaresMethods V*(S\(U'*b))];
```

```
LeastSquaresMethods =

   0.999999996787553   1.000000001037689   1.000000000996603   1.000000000996608   1.000000000996607   1.000000000996608
   0.000000350916732  -0.000000426525960  -0.000000422742768  -0.000000422743080  -0.000000422743364  -0.000000422743090
  -8.000003028795119  -7.999981225479934  -7.999981235691345  -7.999981235685203  -7.999981235676154  -7.999981235684716
  -0.000077893877909  -0.000317425649726  -0.000318763174192  -0.000318763231287  -0.000318763346323  -0.000318763237912
  10.668084035612900  10.669411140597621  10.669430795558023  10.669430795858052  10.669430796641096  10.669430795902734
  -0.009615585352545  -0.013695419352547  -0.013820286728296  -0.013820287698367  -0.013820290914619  -0.013820287873982
  -5.654480747960067  -5.647518455098814  -5.647075630417310  -5.647075628404193  -5.647075619959385  -5.647075627971106
  -0.068885958631546  -0.074362540368604  -0.075316019359204  -0.075316022079922  -0.075316036589419  -0.075316022772098
   1.693354534665567   1.692331412818159   1.693606958189665   1.693606960559036   1.693606976803618   1.693606961276897
   0.001461547101732   0.007068808893883   0.006032112337315   0.006032111063859   0.006032099645104   0.006032110596673
  -0.370576739064076  -0.374710650596597  -0.374241704837825  -0.374241704456940  -0.374241699881279  -0.374241704283244
   0.087095866530693   0.088131160246938   0.088040576307572   0.088040576259675   0.088040575462356   0.088040576231470
```

The scripts computes the least squares coefficient for a degree 11 polynomial approximating the function $\cos(x)$ on the interval $[0, 1]$ with 50 equally spaced points. Each column represents the coefficients in ascending order of one solving method in the order they where computed ie. column 6 was computed via SVD.

g. The calculations above will produce six lists of twelve coefficient. In each list, highlight the digits that appear wrong, Comment on what differences you observe. Do the normal equations exhibit instability? You do not have to explain your observations.

Solution:
Consider the solutions given from the previous Matlab script. In our discussion of these algorithms we concluded that the SVD is the most stable algorithm, in comparing each algorithm I used the SVD as the baseline/true answer. We can see that the Normal Equations method seems to produce the most rounding error.

```
 0.9999999967875530   1.0000000001037689   1.0000000000996603   1.0000000000996608   1.0000000000996607   1.0000000000996608
-0.0000000350916732  -0.0000004265259600  -0.0000004227427680  -0.0000004227430800  -0.0000004227433640  -0.0000004227430900
-8.0000030287951190  -7.9999812254799340  -7.9999812356913450  -7.9999812356852030  -7.9999812356761540  -7.9999812356847160
-0.0000778938779090  -0.0003174256497260  -0.0003187631741920  -0.0003187632312870  -0.0003187633463230  -0.0003187632379120
 10.668084035612900   10.669411140597621   10.669430795558023   10.669430795858052   10.669430796641096   10.669430795902734
-0.0096155853525450  -0.0136954193525470  -0.0138202867282960  -0.0138202876983670  -0.0138202909146190  -0.0138202878739820
-5.6544807479600670  -5.6475184550988140  -5.6470756304173100  -5.6470756284041930  -5.6470756199593850  -5.6470756279711060
-0.0688859586315460  -0.0743625403686040  -0.0753160193592040  -0.0753160220799220  -0.0753160365894190  -0.0753160227720980
 1.6933545346655670   1.6923314128181590   1.6936069581896650   1.6936069605590360   1.6936069768036180   1.6936069612768970
 0.0014615471017320   0.0070688088933830   0.0060321123373150   0.0060321110638590   0.0060321099645104   0.0060321105966730
-0.3705767390640760  -0.3747106505965970  -0.3742417048378250  -0.3742417044569400  -0.3742416998812790  -0.3742417042832440
 0.0870958665306930   0.0881311602469380   0.0880405763075720   0.0880405762596750   0.0880405754623560   0.0880405762314700
```