# Lab 6

David Merrick

```
% 1.
%   a.
lsq(1)
lsq(2)
lsq(3)

% Degree 1: Max error = 0.1548
% Degree 2: Max error = 0.0150
% Degree 3: Max error = 0.0011

%   b.
lsq(4)
lsq(5)
lsq(6)

% Degree 4: Max error = 5.8444e-05
% Degree 5: Max error = 1.9226e-04
% Degree 6: Max error = 0.0015

% Max error stops decreasing at degree 5.

%   c.
lsq(7)
lsq(8)
lsq(9)

% At degree 9, there are visible errors in the approximation.

%   d.
lsq(10)
lsq(11)
lsq(12)

% maxerr on degree 10: 2.3753.
% maxerr on degree 11: 21.8856.
% maxerr on degree 12: 49.4773.
% The maximum error appears to be increasing exponentially.

%   e.
% Polynomial approximations of e^x of degree 1-5 are more accurate than
% higher-degree approximations of this function. This does not necessarily
% mean that,
% in general, low-degree polynomials are better at approximating functions
% than high-degree polynomials. We have only tested this function.
% It might be possible to generate a table of values and use Chebyshev
% nodes to create a high-degree interpolating polynomial that is more
% accurate of an approximation and does not result in an ill-conditioned
% matrix.
```

```
%   f.

% The 4x4 Hilbert matrix is not diagonally dominant, a prerequisite for
% Jacobian convergence, so this should not converge using Jacobian.

m=4
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

m=10
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

m=30
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

m=40
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

m=100
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

m=200
A=hilb(m+1);
N=diag(diag(A));
P=N-A;
norm(inv(N)*P)

% For m=4, result is 4.6738. For m=10, result is 11.2735. For m=30, result
% is 33.2966. For m=40, result is 44.3076. For m=70, result is 77.395. For
% m=100, result is 110.3709. For m=200, result is 270.2244. As predicted,
% this does not converge using Jacobian iteration, so this method could not
% be applied to this linear system.

%   g.
% Gauss-Seidel can be applied to any matrix with non-zero elements on the
% diagonals, but convergence is only guaranteed if the matrix is either
% diagonally dominant,
% or symmetric and positive definite. The Hilbert matrix is not
% diagonally dominant, but is symmetric, so it would be reasonable to
```

```
% predict that this will converge for the Hilbert matrix.

m=4
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=10
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=30
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=40
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=70
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=100
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

m=200
A=hilb(m+1);
N=tril(A);
P=N-A;
norm(inv(N)*P)

% For m=4, result is 1.0003. For m=10, result is 1.0015. For m=30, result
% is 1.0032. For m=40, result is 1.0035. For m=70, result is 1.0039. For
% m=100, result is 1.0041. For m=200, result is 1.0043. Clearly, as
% predicted, this does converge using Gauss-Seidel Iteration, so this
% method could be applied to this linear system.

%   2.
%   a.
n=2
gaussweights(n)
```

```
n=4
gaussweights(n)

n=8
gaussweights(n)

% For n = 2,4,8, the calculated values match the values in Table 5.7 in the
% book.
%   b.
n=32
gaussweights(n)

n=64
gaussweights(n)

% For n=32, the condition number is 1.2827e+12. For n=64, the condition
% number is 1.2338e+20. The condition number relates the reative accuracy
% of the input to the relative accuracy of the output. With a high
% condition number, small changes in the input can produce huge errors,
% and this is clearly demonstrated on the plot.

%   c.
% Convergence for Jacobian iteration requires that the matrix is
% diagonally-dominant. With the Vandermonde matrix, this is not the case.
% So Jacobion iteration will not converge for this linear system.

%   d.
% Convergence for Gauss-Seidel requires  that the matrix is either
% diagonally-dominant, symmetrical, or positive definite. The Vandermonde
% matrix meets none of these criteria, so that method will not converge for
% this linear system.
```

```
maxerr =

    0.1548


ans =

     0


maxerr =

    0.0150


ans =

   1.1102e-16
```

```
maxerr =

    0.0011

ans =

     0

maxerr =

   5.8444e-05

ans =

   2.5438e-16

maxerr =

   1.9226e-04

ans =

   4.0792e-16

maxerr =

    0.0015

ans =

   3.2841e-16

maxerr =

    0.0048

ans =

   2.3009e-15

maxerr =

    0.0051
```

```
ans =

   1.9681e-15


maxerr =

    0.1158


ans =

   2.0337e-12


maxerr =

    2.3753


ans =

   1.7099e-10


maxerr =

   21.8856


ans =

   4.5044e-09


maxerr =

   49.4773


ans =

   8.5473e-08


m =

     4


ans =

    4.6738
```

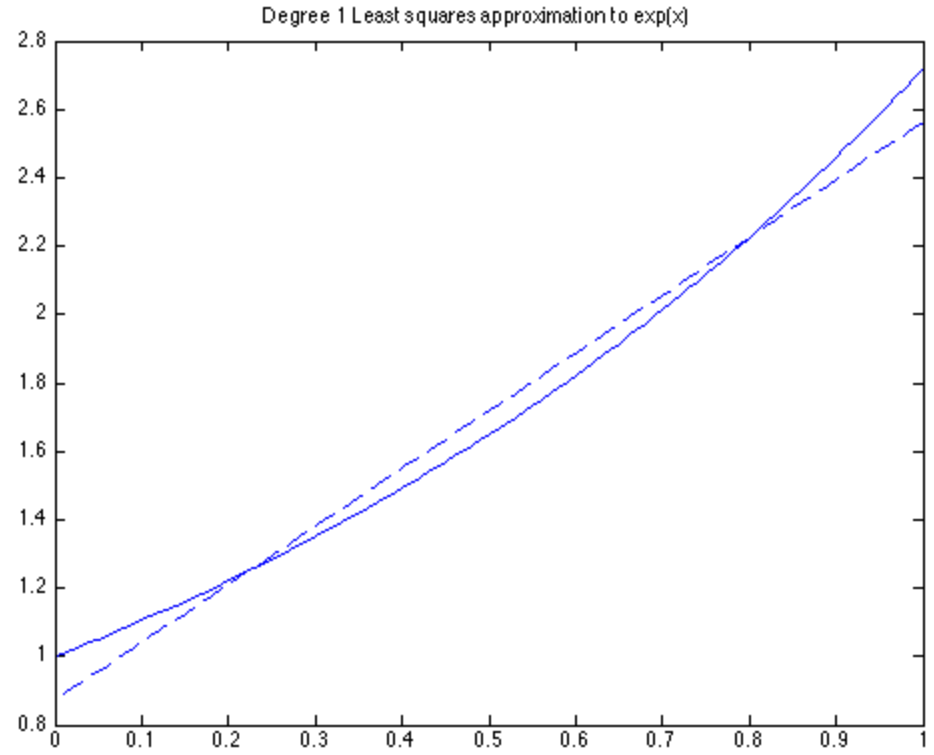
```
m =

    10

ans =

   11.2735

m =

    30

ans =

   33.2966

m =

    40

ans =

   44.3076

m =

   100

ans =

  110.3709

m =

   200

ans =

  220.4744

m =
```

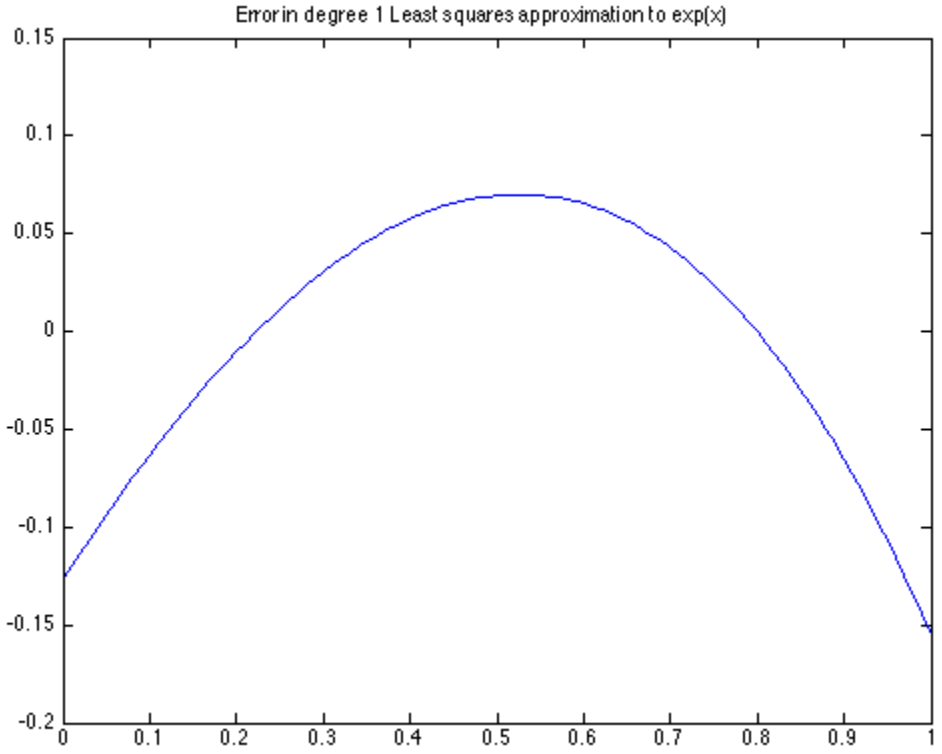
```
     4

ans =

    1.0003

m =

    10

ans =

    1.0015

m =

    30

ans =

    1.0032

m =

    40

ans =

    1.0035

m =

    70

ans =

    1.0039

m =

   100

ans =
```

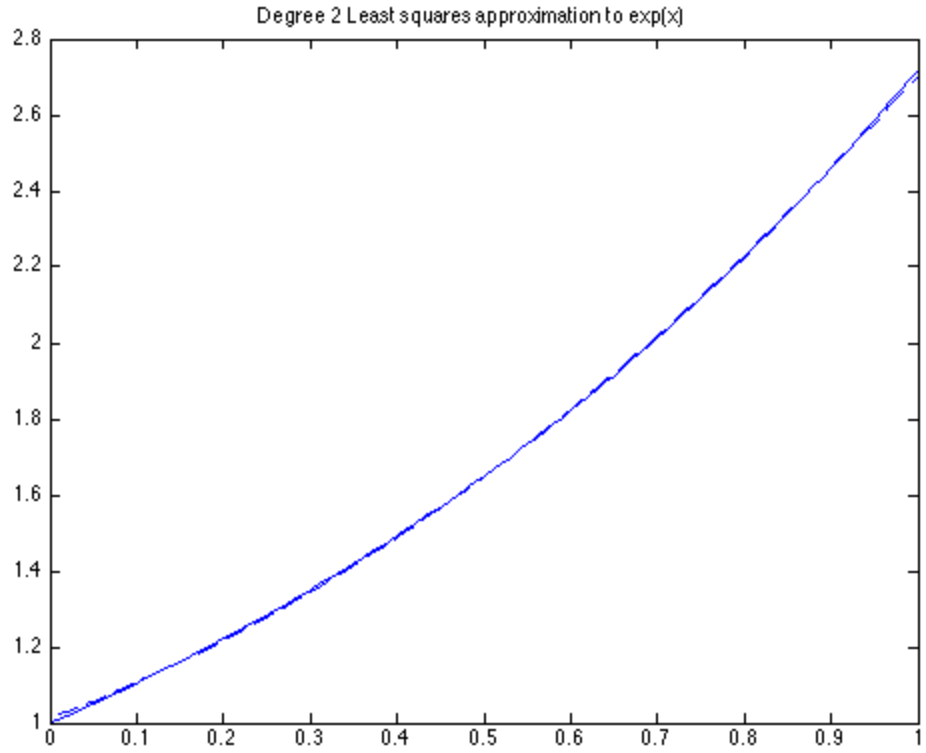
```
    1.0041

m =

   200

ans =

    1.0043

n =

     2

ans =

    2.7321

ans =

     1     1

n =

     4

ans =

   21.9310

ans =

    0.3479    0.6521    0.6521    0.3479

n =

     8

ans =

  798.9577
```

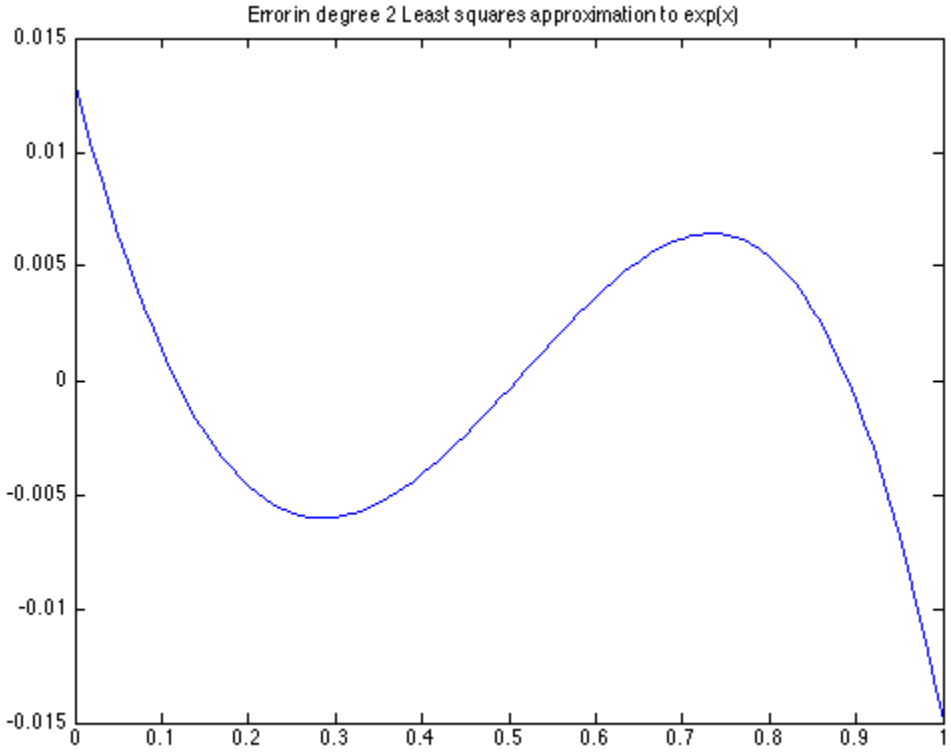
```
ans =

  Columns 1 through 7

    0.1012    0.2224    0.3137    0.3627    0.3627    0.3137    0.2224

  Column 8

    0.1012


n =

    32


ans =

   1.2827e+12


ans =

  Columns 1 through 7

    0.0070    0.0163    0.0254    0.0343    0.0428    0.0510    0.0587

  Columns 8 through 14

    0.0658    0.0723    0.0782    0.0833    0.0877    0.0912    0.0938

  Columns 15 through 21

    0.0956    0.0965    0.0965    0.0956    0.0938    0.0912    0.0877

  Columns 22 through 28

    0.0833    0.0782    0.0723    0.0658    0.0587    0.0510    0.0428

  Columns 29 through 32

    0.0343    0.0254    0.0163    0.0070


n =

    64

Warning: Matrix is close to singular or badly scaled. Results may be inacc
RCOND =  7.132669e-21.

ans =

   1.2338e+20
```

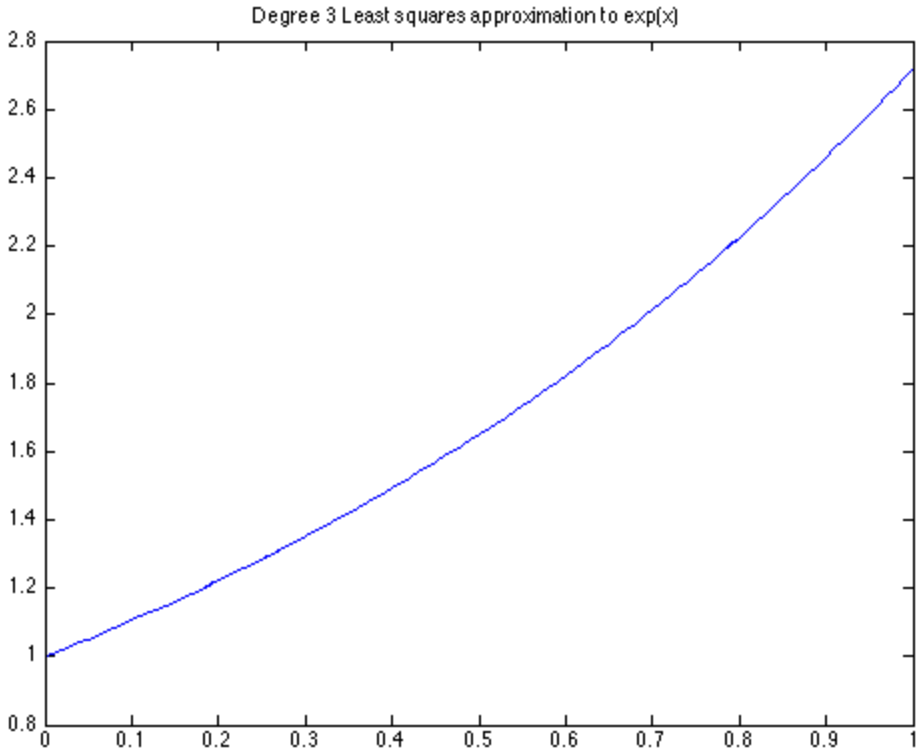
```
ans =

  Columns 1 through 7

    0.0572   -0.1622    0.2423   -0.1803   -0.0105    0.4171   -0.8978

  Columns 8 through 14

    1.4791   -1.9049    2.2038   -2.1179    1.8182   -1.1832    0.6059

  Columns 15 through 21

   -0.0786    0.0529   -0.3540    1.1675   -1.9366    2.6211   -2.6597

  Columns 22 through 28

    2.3834   -1.7498    1.5161   -1.5566    2.0070   -1.8302    0.6451

  Columns 29 through 35

    2.2147   -5.8144    9.2520  -10.7129    9.7656   -6.2526    1.7344

  Columns 36 through 42

    2.6331   -5.2237    5.9092   -4.5707    2.3355    0.2944   -2.2814

  Columns 43 through 49

    3.6344   -3.9850    3.8568   -3.1967    2.5699   -1.8479    1.4037

  Columns 50 through 56

   -0.9705    0.7783   -0.5307    0.4218   -0.2095    0.0916    0.1091

  Columns 57 through 63

   -0.2072    0.3386   -0.3575    0.3771   -0.3045    0.2334   -0.1227

  Column 64

    0.0399
```

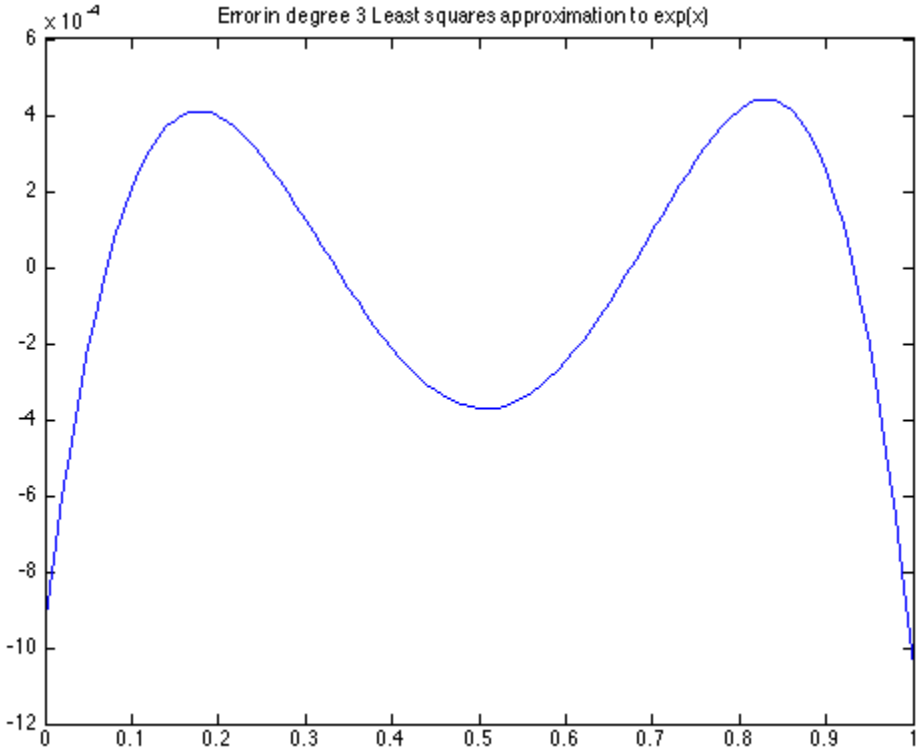

Degree 1 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

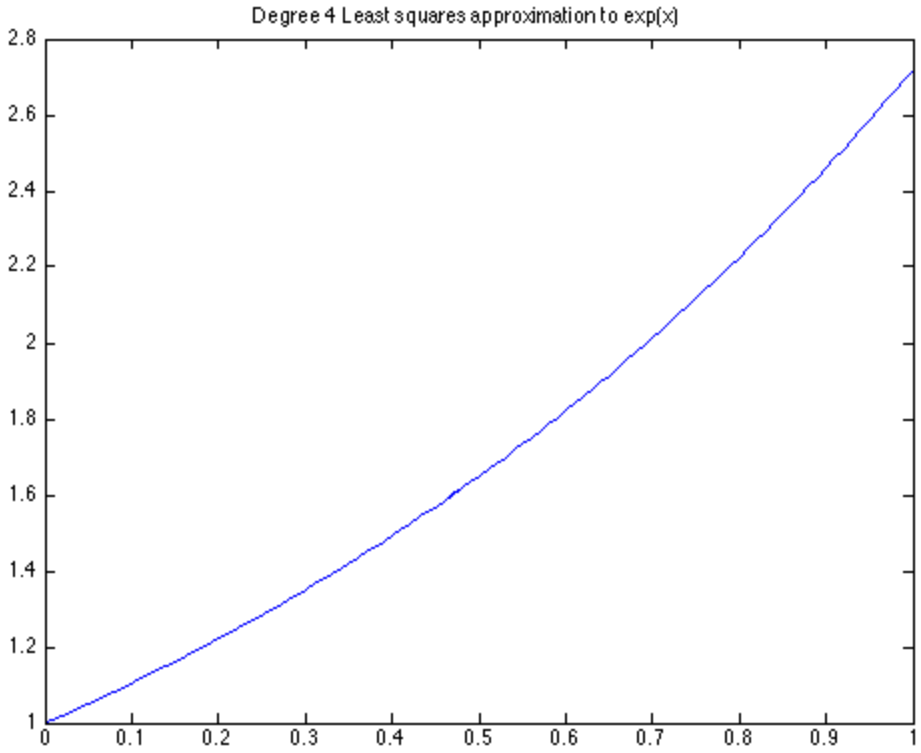

Error in degree 1 Least squares approximation to exp(x)
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

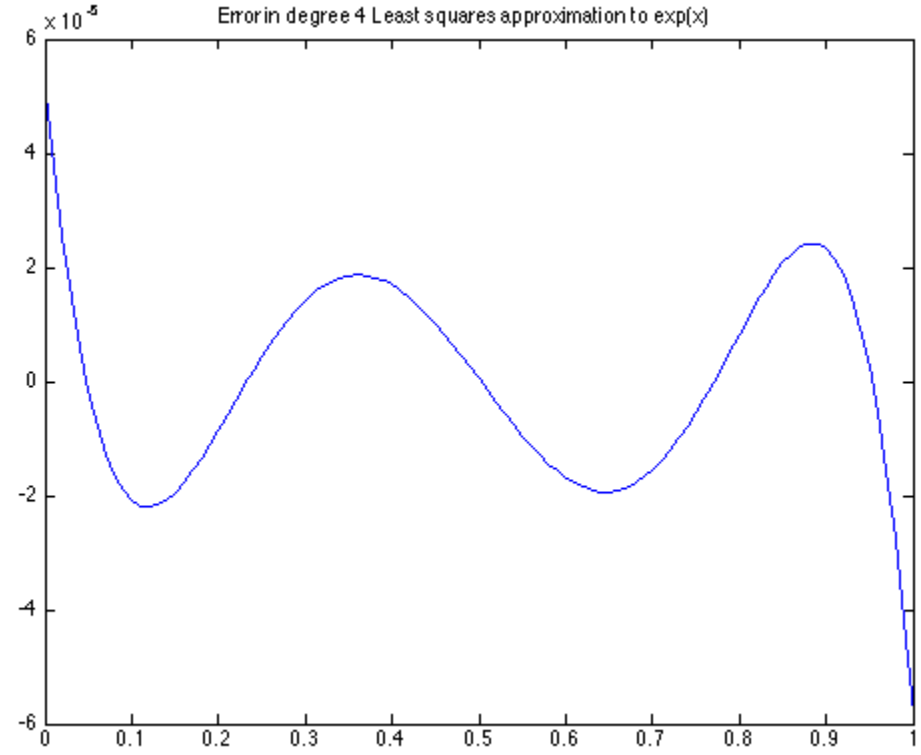

Degree 2 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

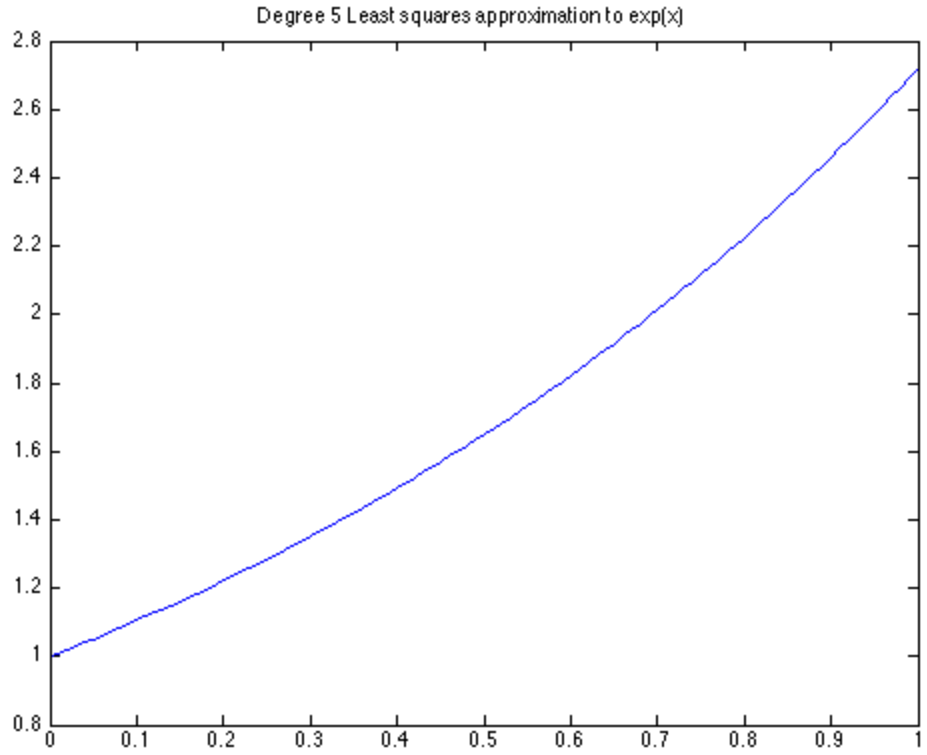

Error in degree 2 Least squares approximation to exp(x)
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

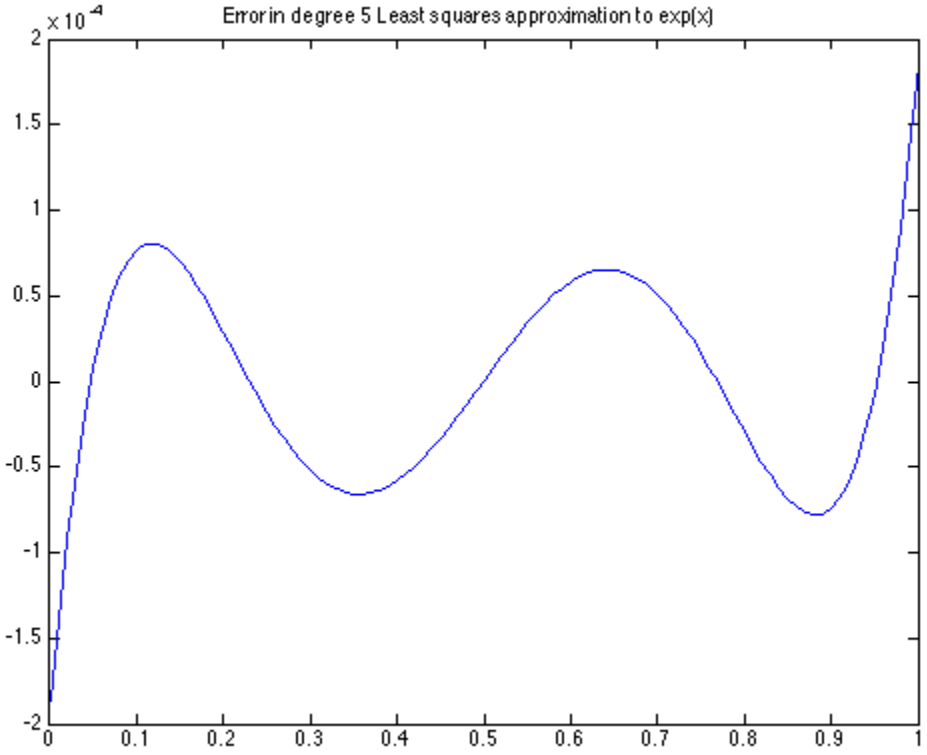

Degree 3 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

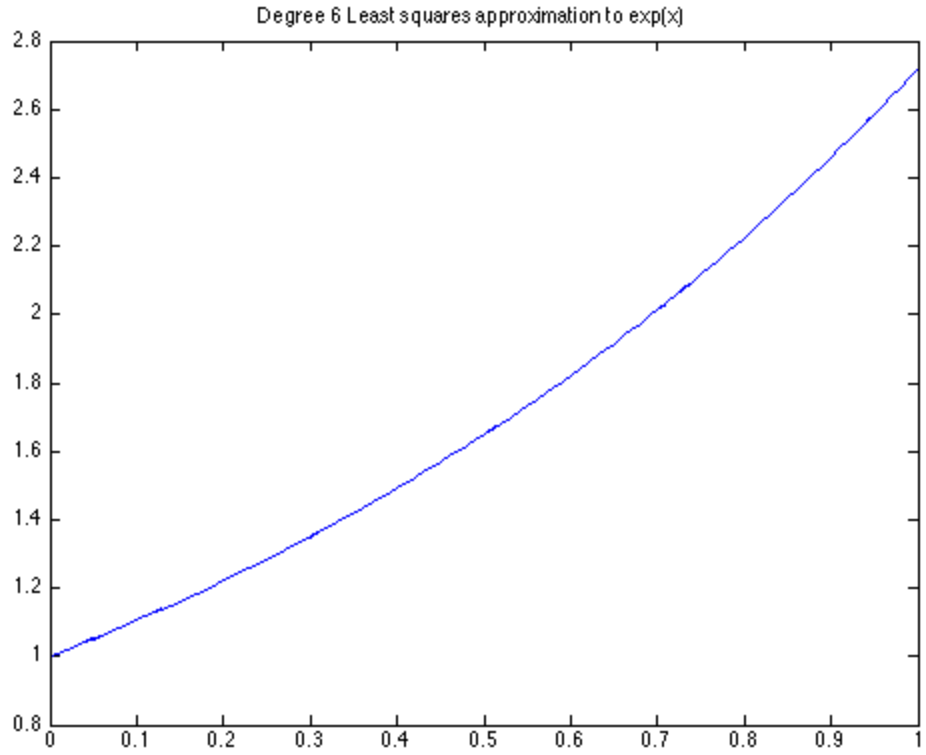

x 10^-4
Error in degree 3 Least squares approximation to exp(x)
6
4
2
0
-2
-4
-6
-8
-10
-12
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

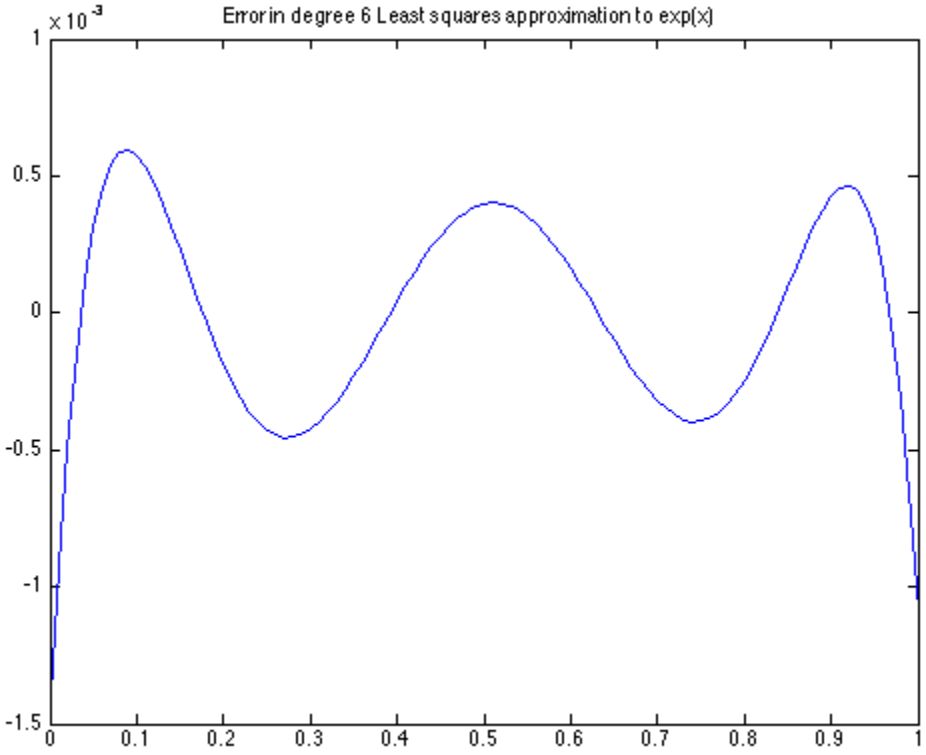

Degree 4 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

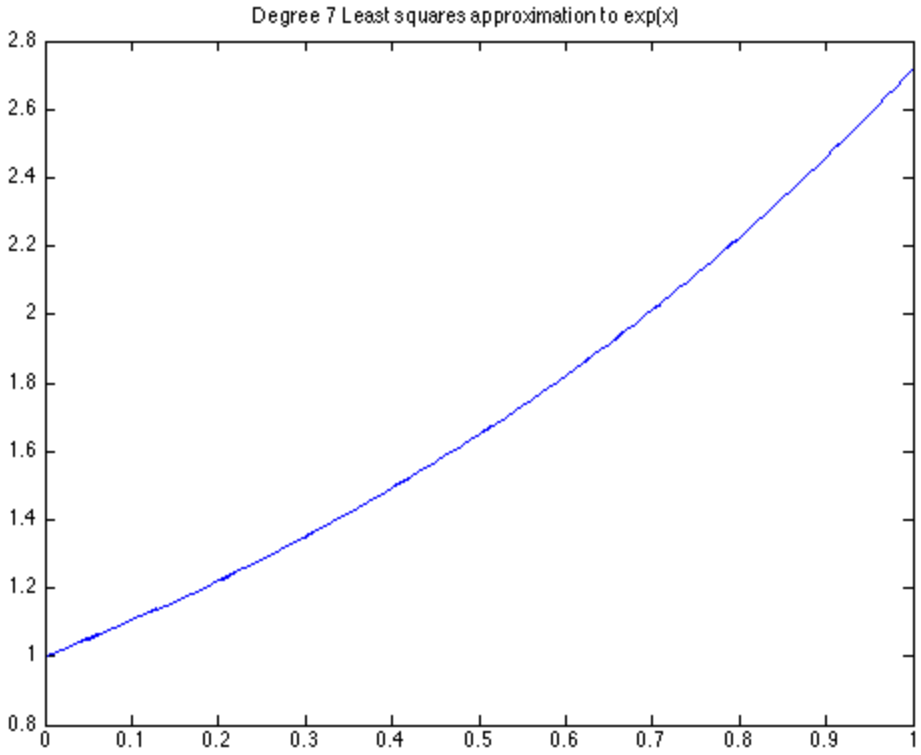

x 10^-5
Error in degree 4 Least squares approximation to exp(x)
6
4
2
0
-2
-4
-6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

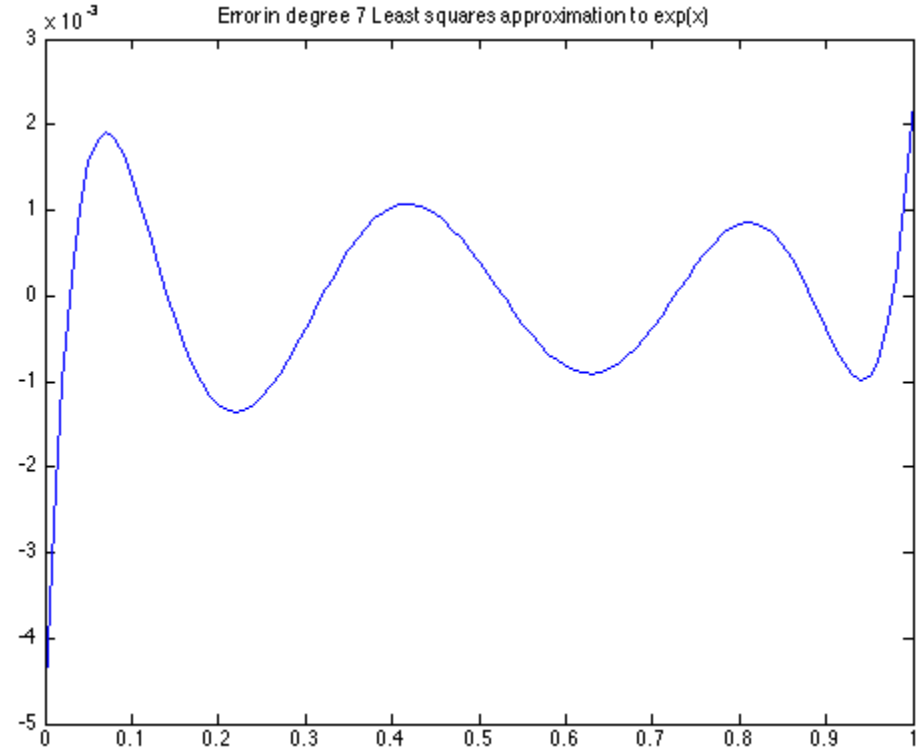

Degree 5 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

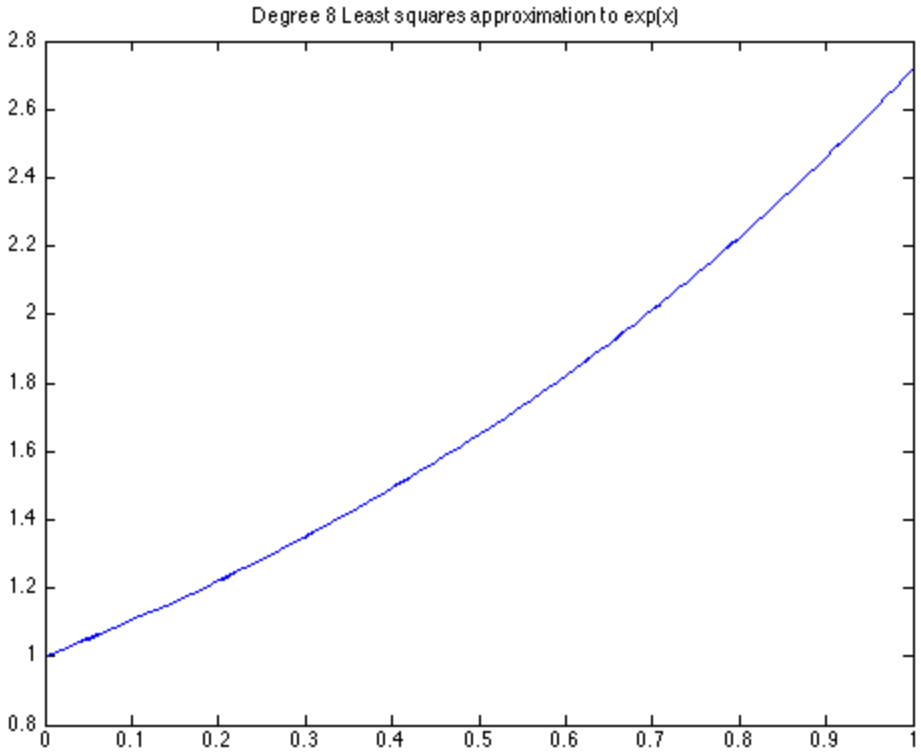

Error in degree 5 Least squares approximation to exp(x)
x 10^-4
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

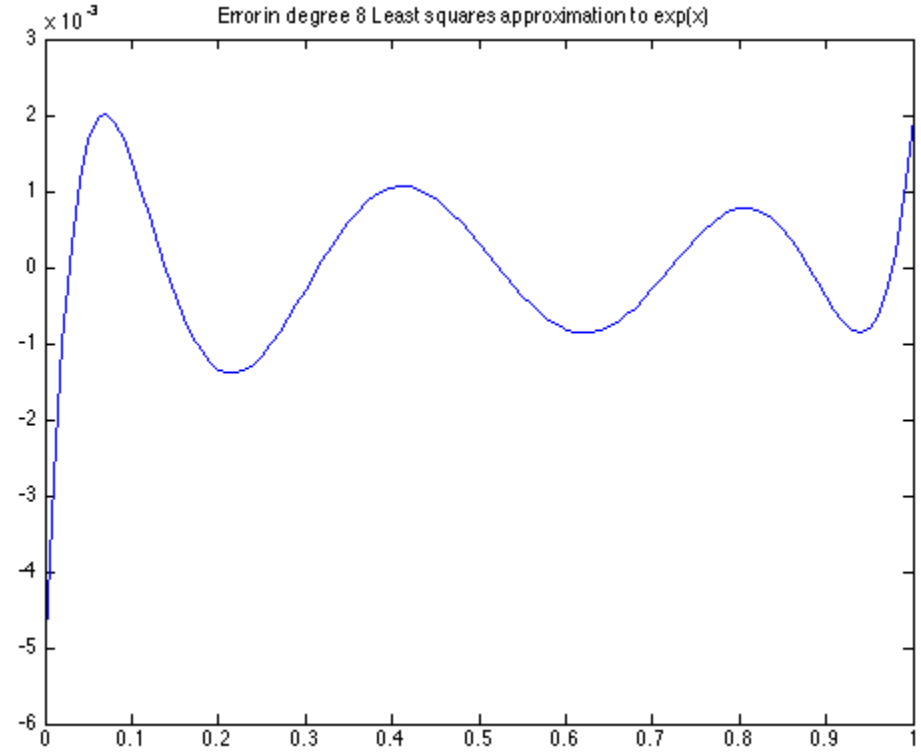

Degree 6 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

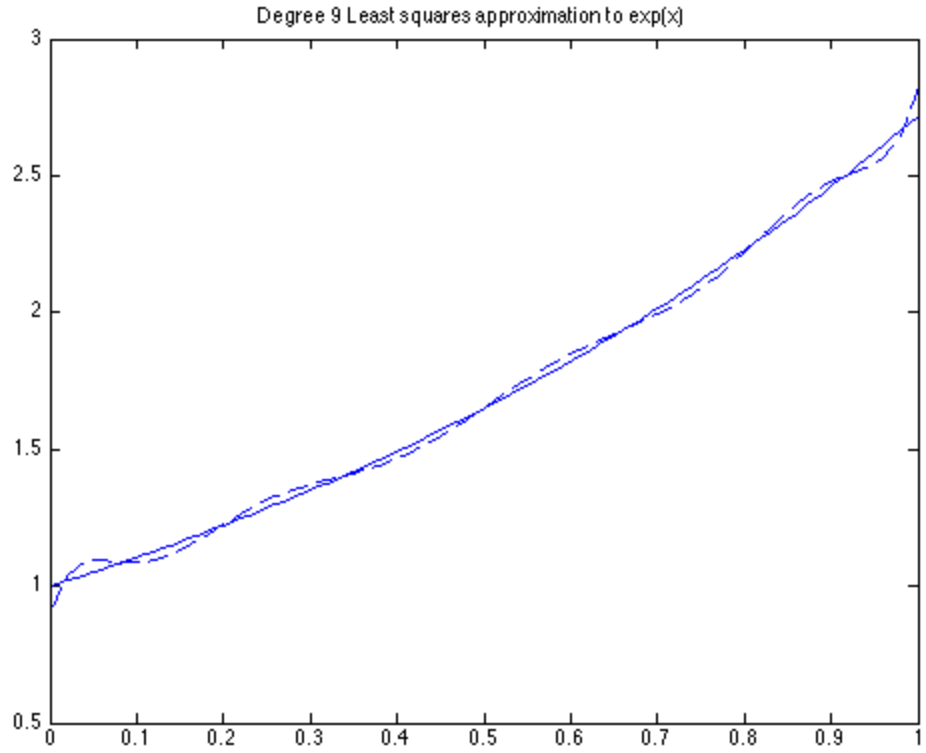

Error in degree 6 Least squares approximation to exp(x)
x 10^-3
1
0.5
0
-0.5
-1
-1.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

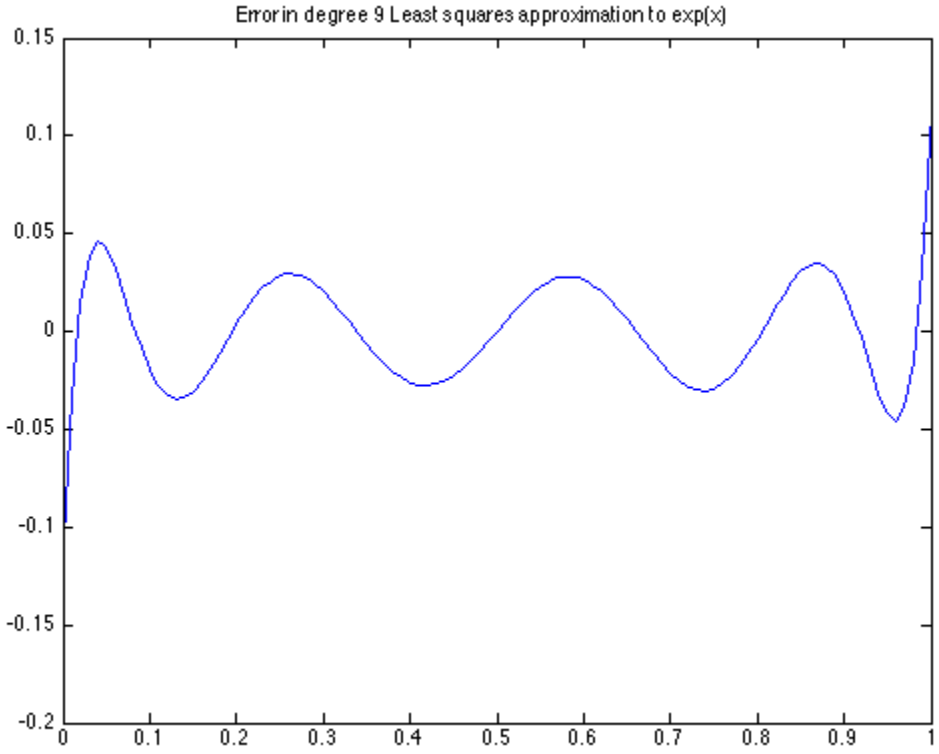

Degree 7 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

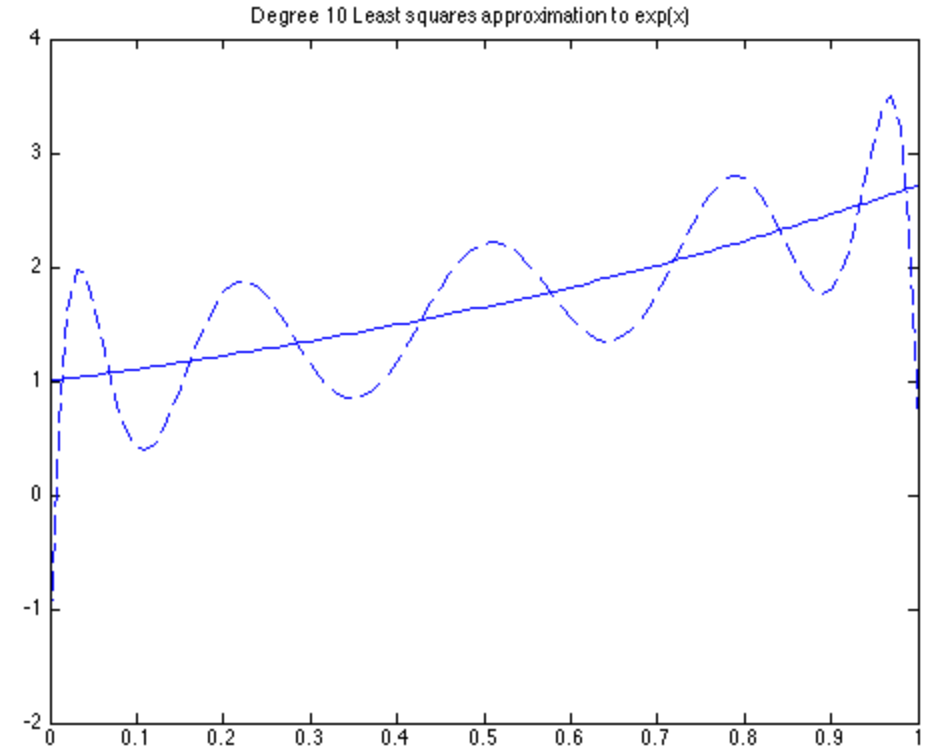

x 10^-3
Error in degree 7 Least squares approximation to exp(x)
3
2
1
0
-1
-2
-3
-4
-5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

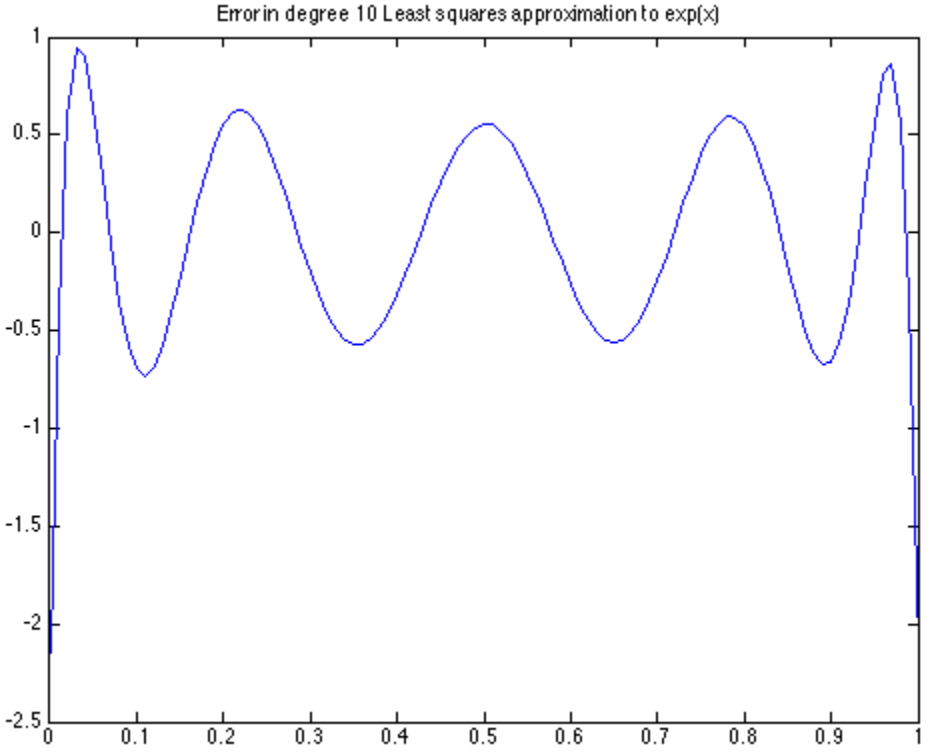

Degree 8 Least squares approximation to exp(x)
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

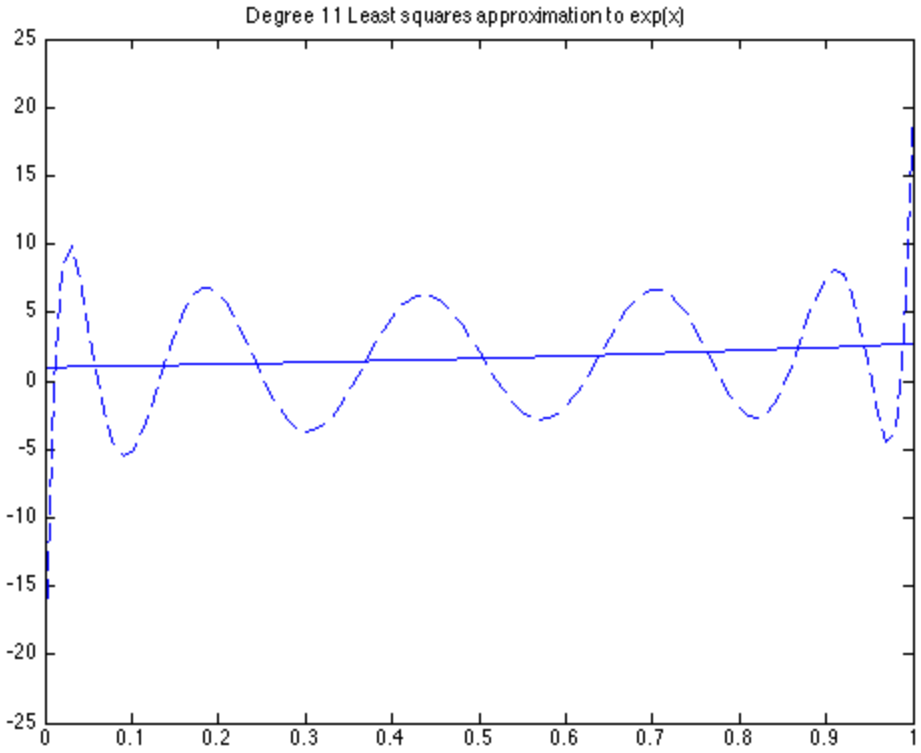

x 10^-3
Error in degree 8 Least squares approximation to exp(x)
3
2
1
0
-1
-2
-3
-4
-5
-6
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

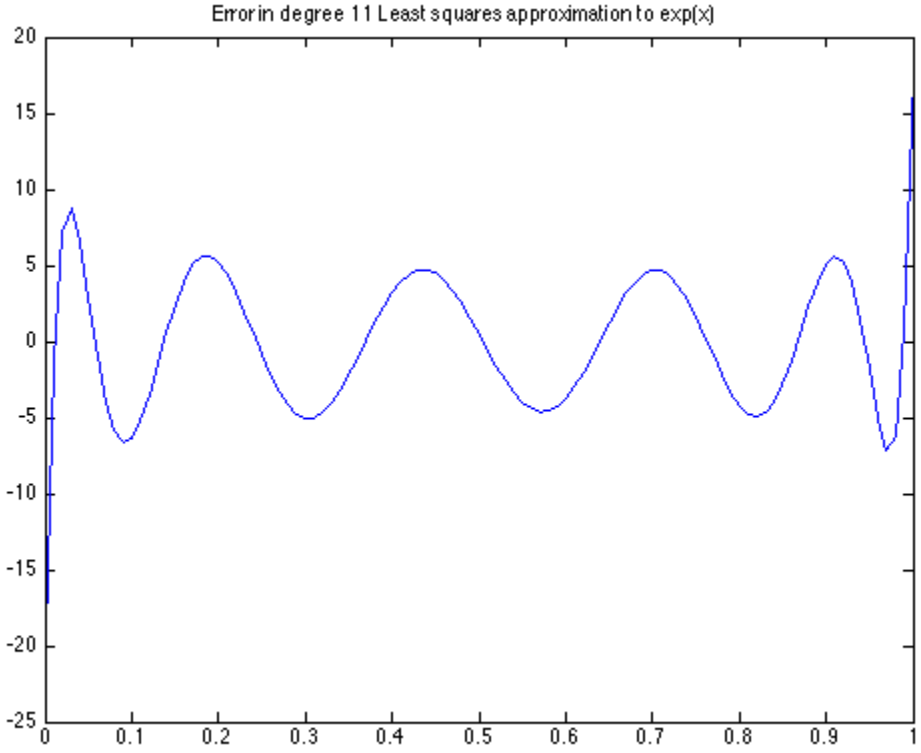

Degree 9 Least squares approximation to exp(x)
3
2.5
2
1.5
1
0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

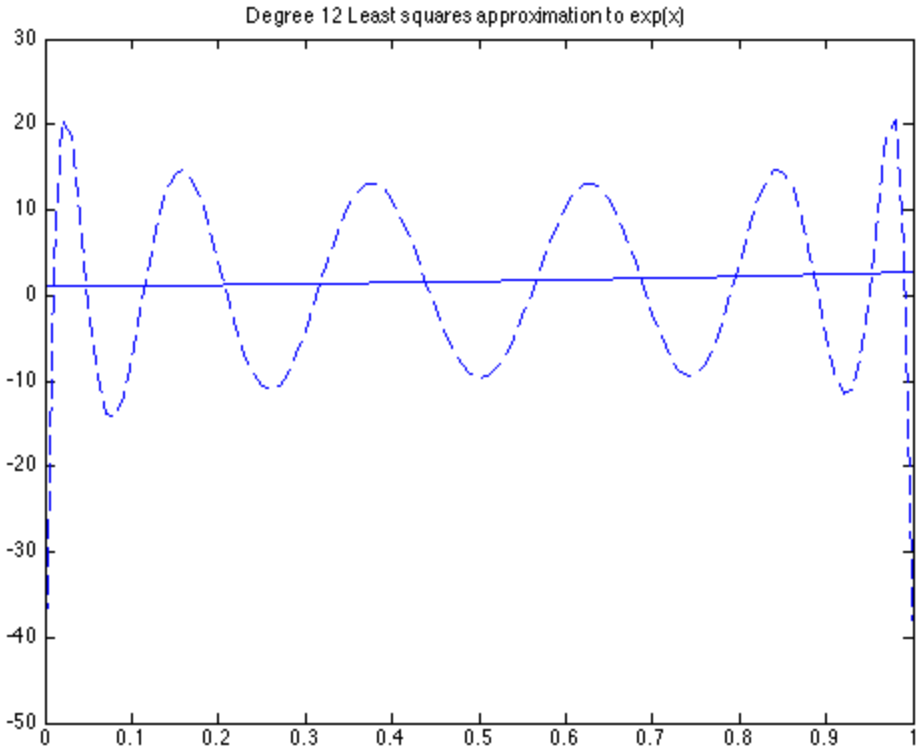

Error in degree 9 Least squares approximation to exp(x)
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

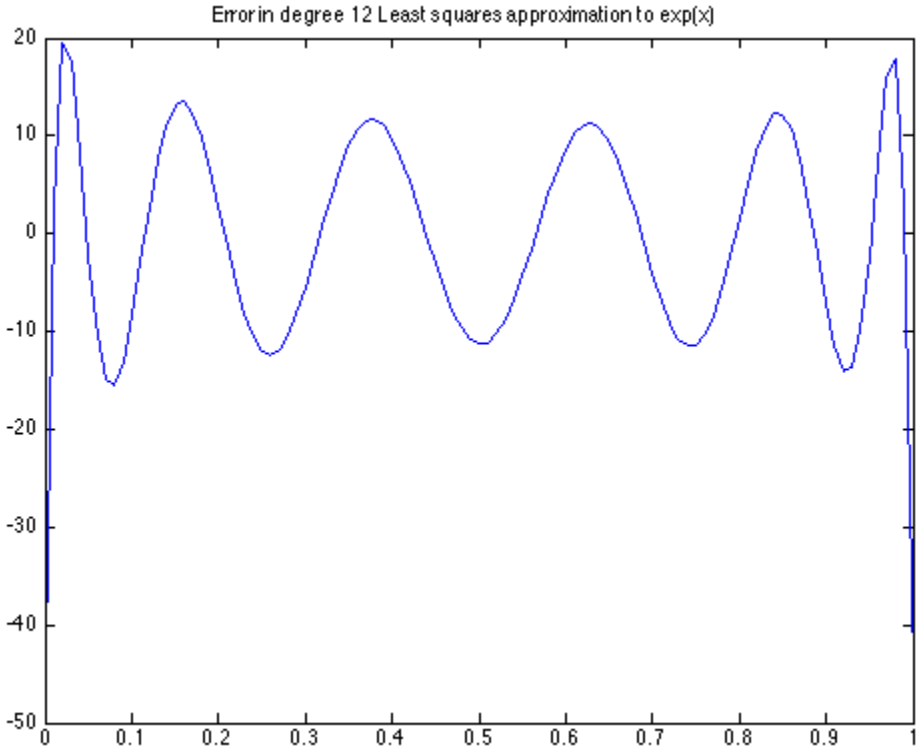

Degree 10 Least squares approximation to exp(x)
4
3
2
1
0
-1
-2
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

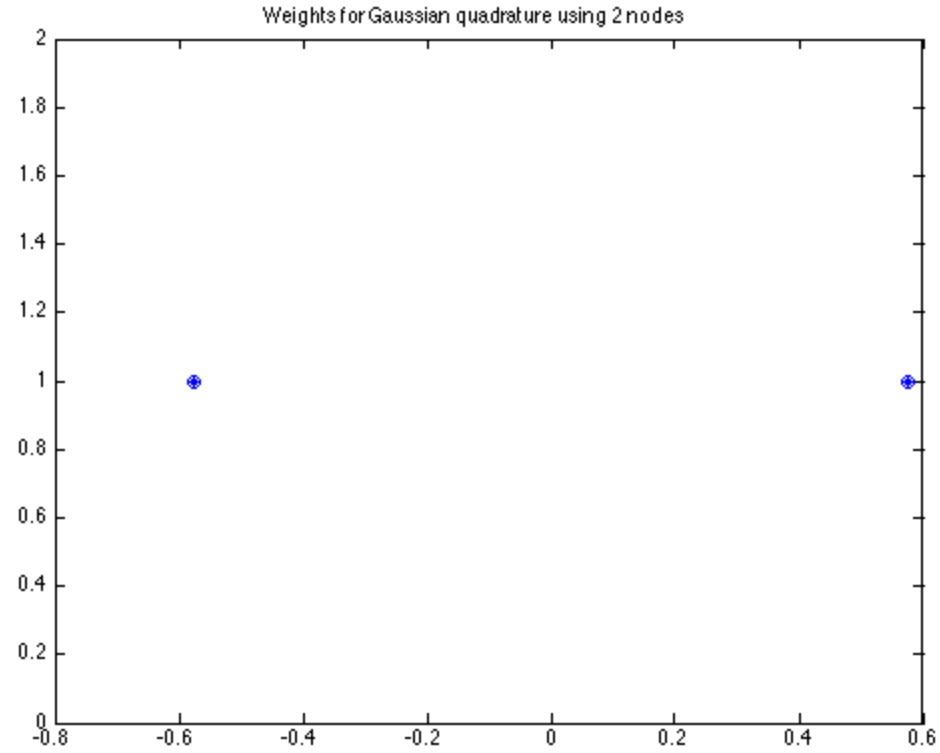

Error in degree 10 Least squares approximation to exp(x)
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

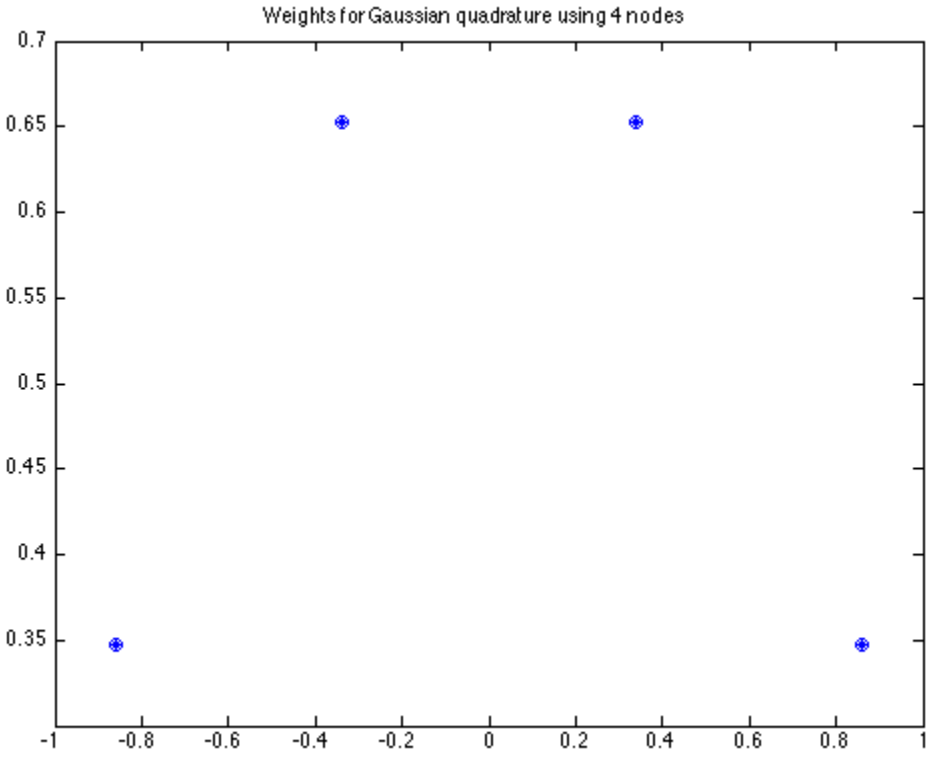

Degree 11 Least squares approximation to exp(x)
25
20
15
10
5
0
-5
-10
-15
-20
-25
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

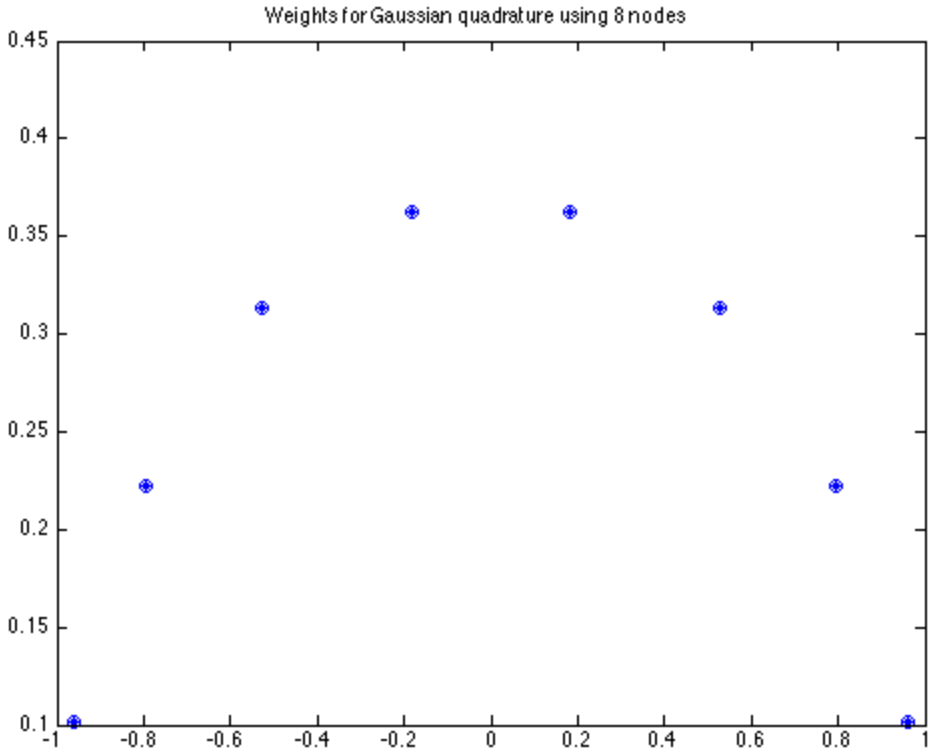

Error in degree 11 Least squares approximation to exp(x)
20
15
10
5
0
-5
-10
-15
-20
-25
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

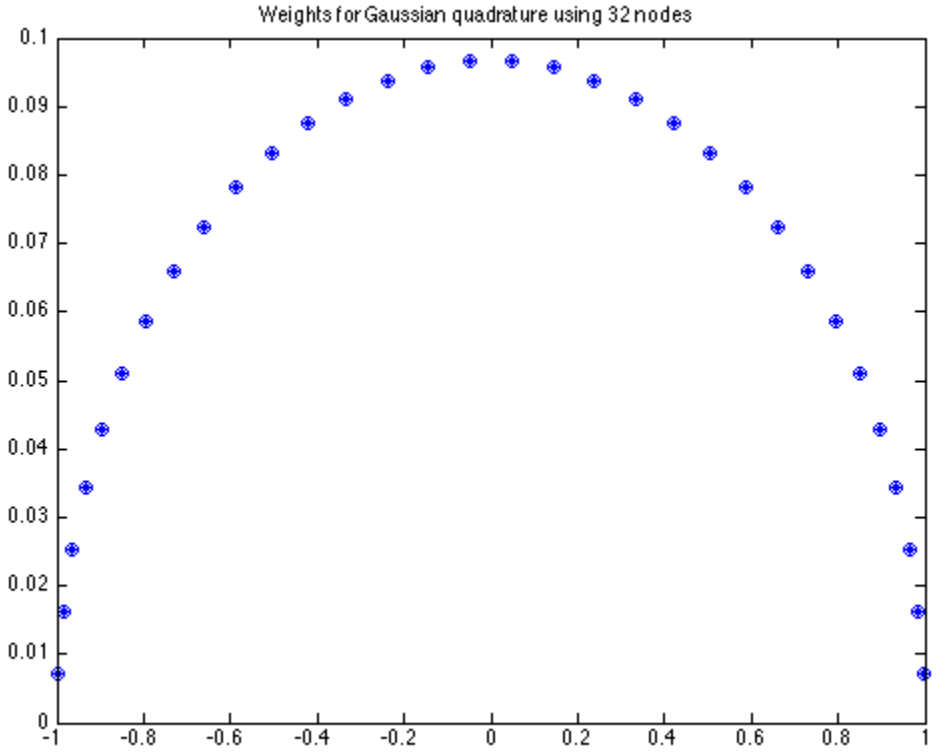

Degree 12 Least squares approximation to exp(x)
30
20
10
0
-10
-20
-30
-40
-50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

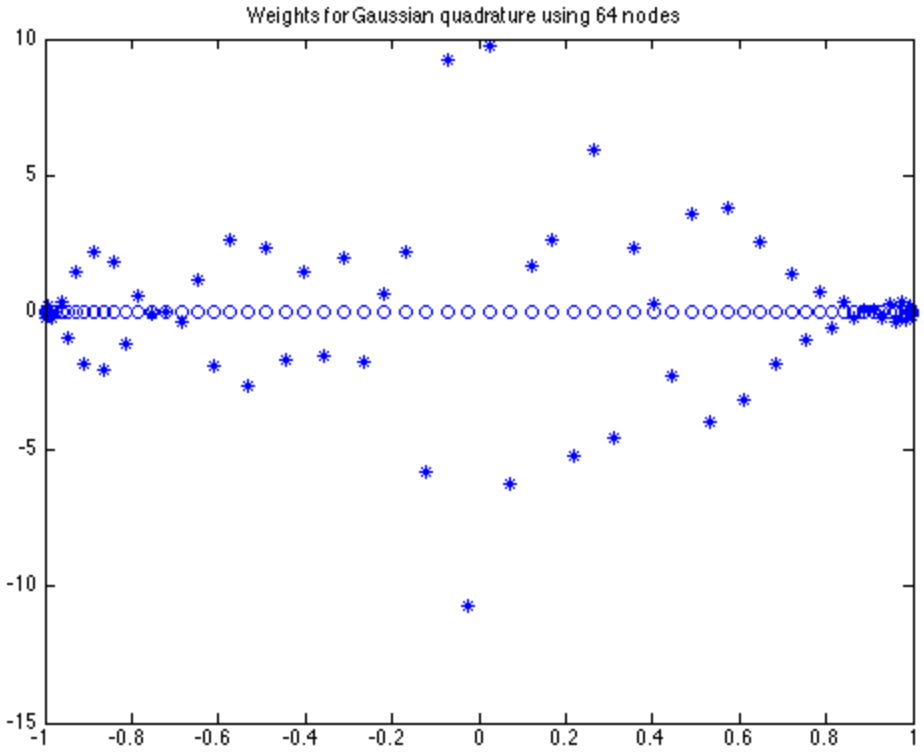

Error in degree 12 Least squares approximation to exp(x)
20
10
0
-10
-20
-30
-40
-50
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Weights for Gaussian quadrature using 2 nodes
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


Weights for Gaussian quadrature using 4 nodes
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Weights for Gaussian quadrature using 8 nodes
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Weights for Gaussian quadrature using 32 nodes
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1



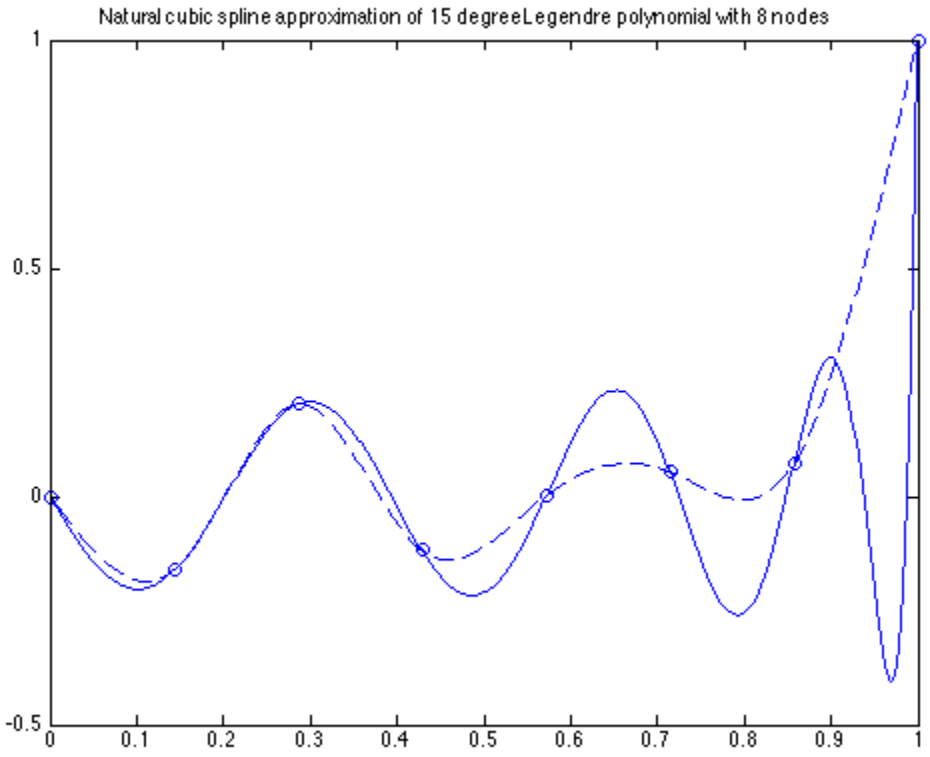

# 3.

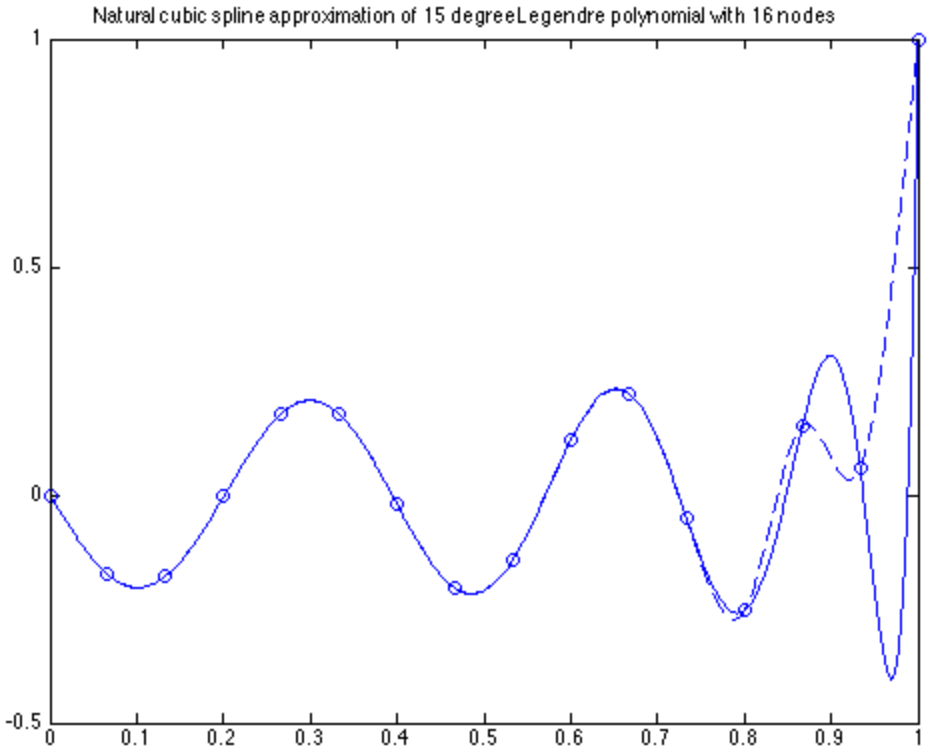
```
cubspline(8)
cubspline(16)

% Run each of the methods 3 times, find the total, and divide by 3
% to display the average times.
%   a. GEPivot
tot641=0;
tot1281=0;
tot2561=0;
for i=1:3
    [res64, time64, itnum64] = cubspline(64);
    [res128, time128, itnum128] = cubspline(128);
    [res256, time256, itnum256] = cubspline(256);
    tot641 = tot641 + time64;
    tot1281 = tot1281 + time128;
    tot2561 = tot2561 + time256;
end

%   b. Jacobian
tot642=0;
tot1282=0;
tot2562=0;
for i=1:3
```

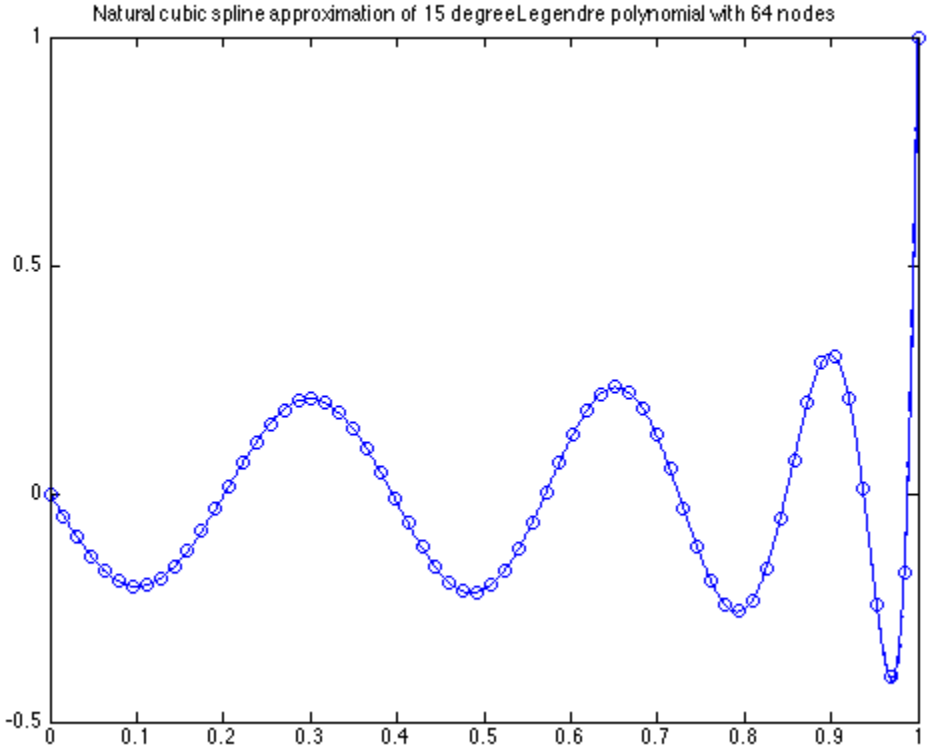
```
    [res64, time64, itnum64] = cubspline2(64);
    [res128, time128, itnum128] = cubspline2(128);
    [res256, time256, itnum256] = cubspline2(256);
    tot642 = tot642 + time64;
    tot1282 = tot1282 + time128;
    tot2562 = tot2562 + time256;
end

%   c. Gauss-Seidel: ignore timings
tot643=0;
tot1283=0;
tot2563=0;
for i=1:3
    [res64, time64, itnum64] = cubspline3(64);
    [res128, time128, itnum128] = cubspline3(128);
    [res256, time256, itnum256] = cubspline3(256);
    tot643 = tot643 + time64;
    tot1283 = tot1283 + time128;
    tot2563 = tot2563 + time256;
end

%   d. Conjugate-Gradient
tot644=0;
tot1284=0;
tot2564=0;
for i=1:3
    [res64, time64, itnum64] = cubspline4(64);
    [res128, time128, itnum128] = cubspline4(128);
    [res256, time256, itnum256] = cubspline4(256);
    tot644 = tot644 + time64;
    tot1284 = tot1284 + time128;
    tot2564 = tot2564 + time256;
end

%   e. Tridiag
tot645=0;
tot1285=0;
tot2565=0;
for i=1:3
    [res64, time64, itnum64] = cubspline5(64);
    [res128, time128, itnum128] = cubspline5(128);
    [res256, time256, itnum256] = cubspline5(256);
    tot645 = tot645 + time64;
    tot1285 = tot1285 + time128;
    tot2565 = tot2565 + time256;
end


disp(sprintf('Runtimes for solving tridiagonal systems using \n'))
disp(sprintf('n\t GEPivot\t Jacobi\t Conjugate-Gradient\t Tridiagonal\n'))
disp(sprintf('64\t %d\t %d\t %d\t %d\n', tot641/3, tot642/3, tot644/3, tot645/3));
disp(sprintf('128\t %d\t %d\t %d\t %d\n', tot1281/3, tot1282/3, tot1284/3, tot1285
disp(sprintf('256\t %d\t %d\t %d\t %d\n', tot2561/3, tot2562/3, tot2564/3, tot2565
```

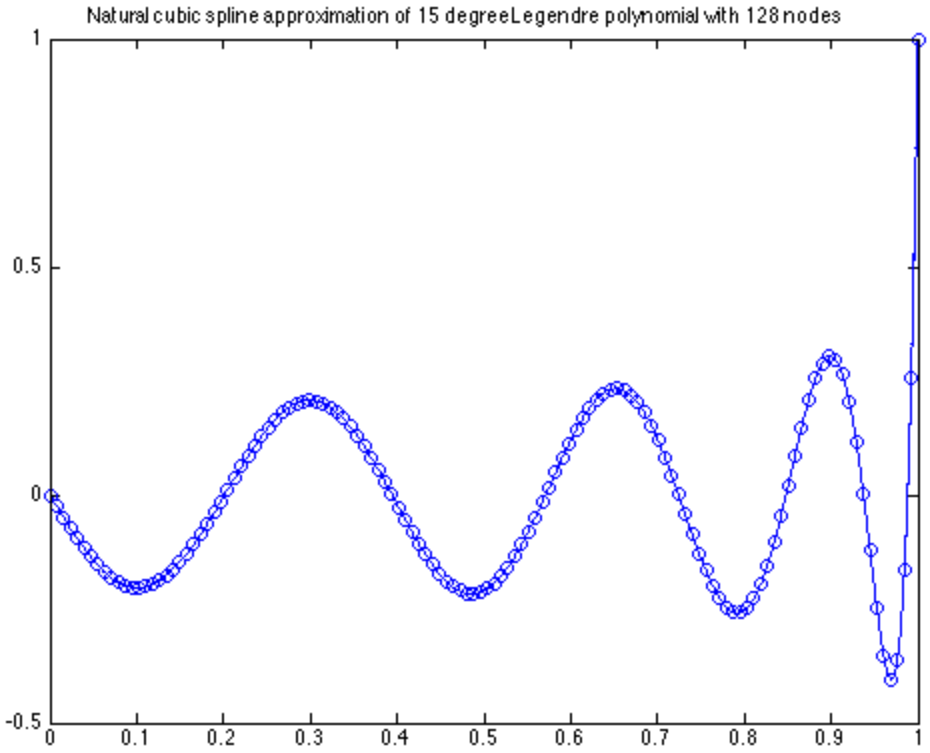
```
% Tridiagonal had the lowest average runtimes, followed by Jacobi, then
% Conjugate-Gradient, and GEPivot was the slowest.
```

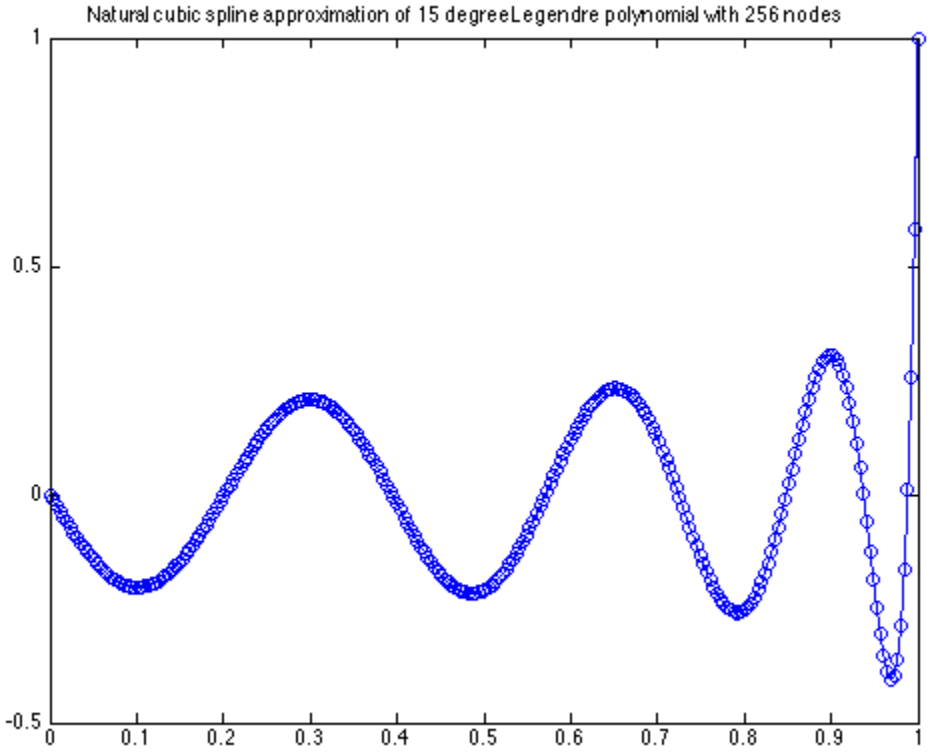
```
ans =

   1.0923e-14


ans =

   4.4702e-14

Runtimes for solving tridiagonal systems using

n  GEPivot  Jacobi  Conjugate-Gradient  Tridiagonal

64  1.730935e-02  4.215353e-04  2.429209e-03  5.137500e-05

128  8.051737e-02  7.223910e-04  3.633219e-03  5.584033e-05

256  3.496016e-01  1.648796e-03  5.013996e-03  1.086917e-04
```

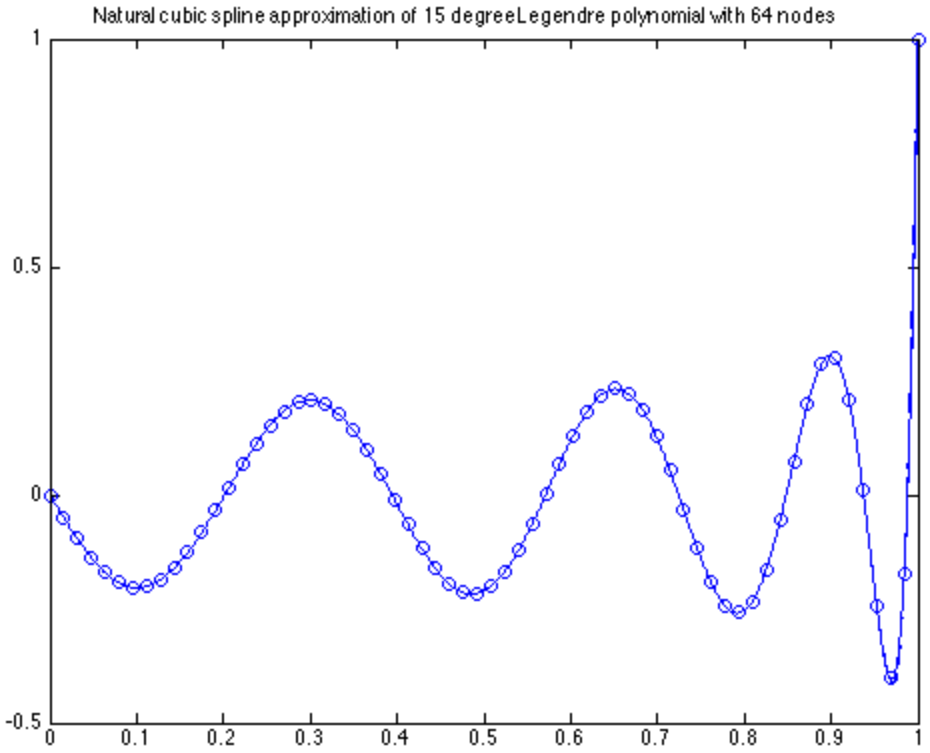


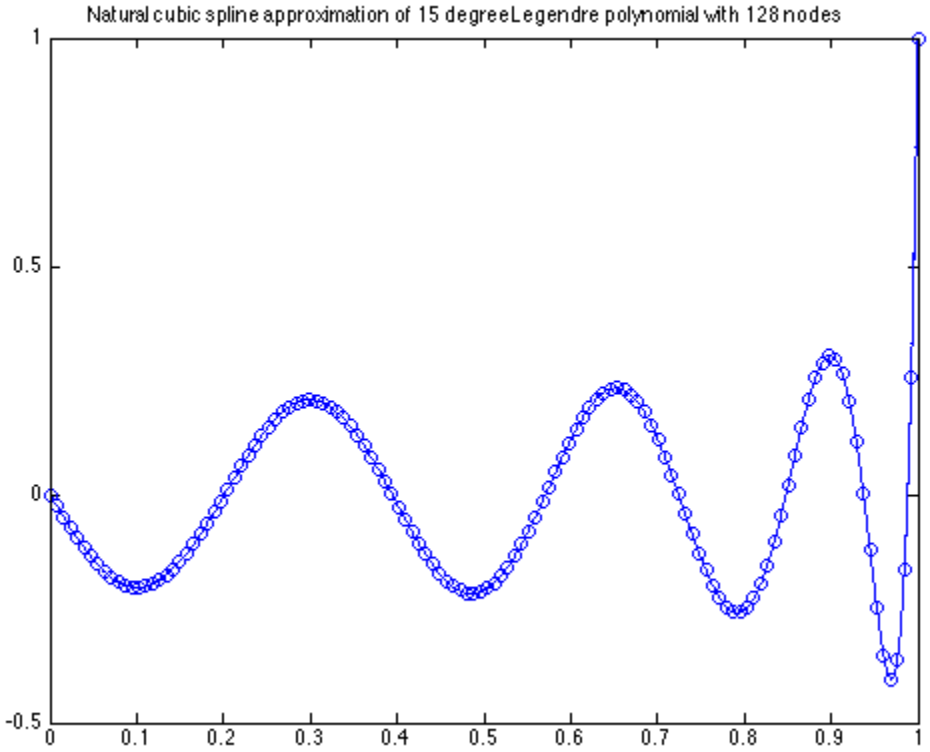

Natural cubic spline approximation of 15 degree Legendre polynomial with 16 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

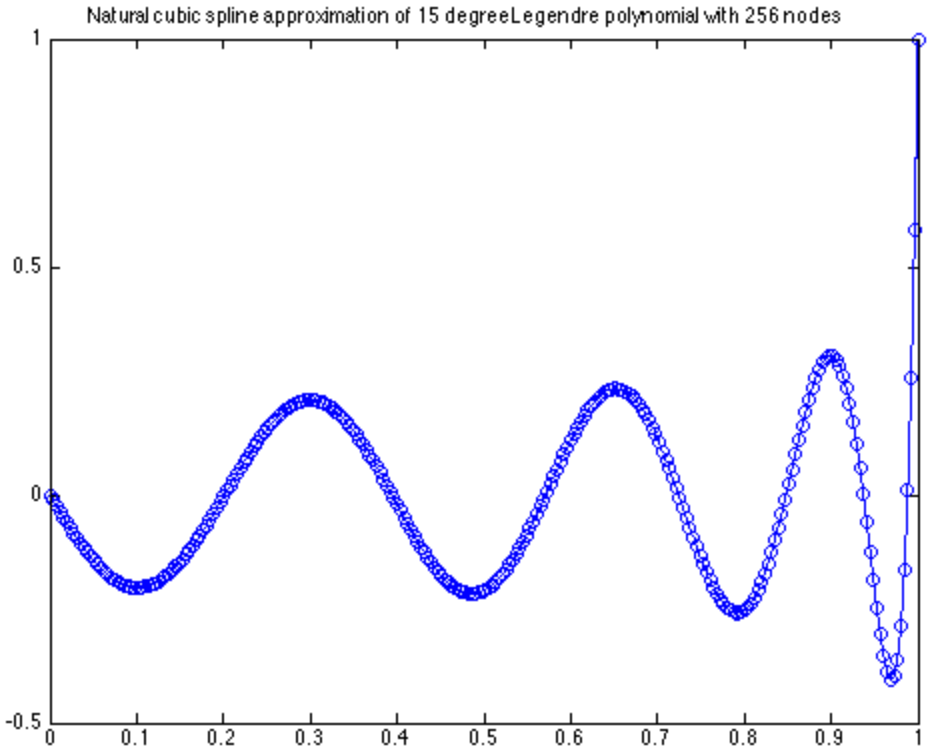

Natural cubic spline approximation of 15 degree Legendre polynomial with 64 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

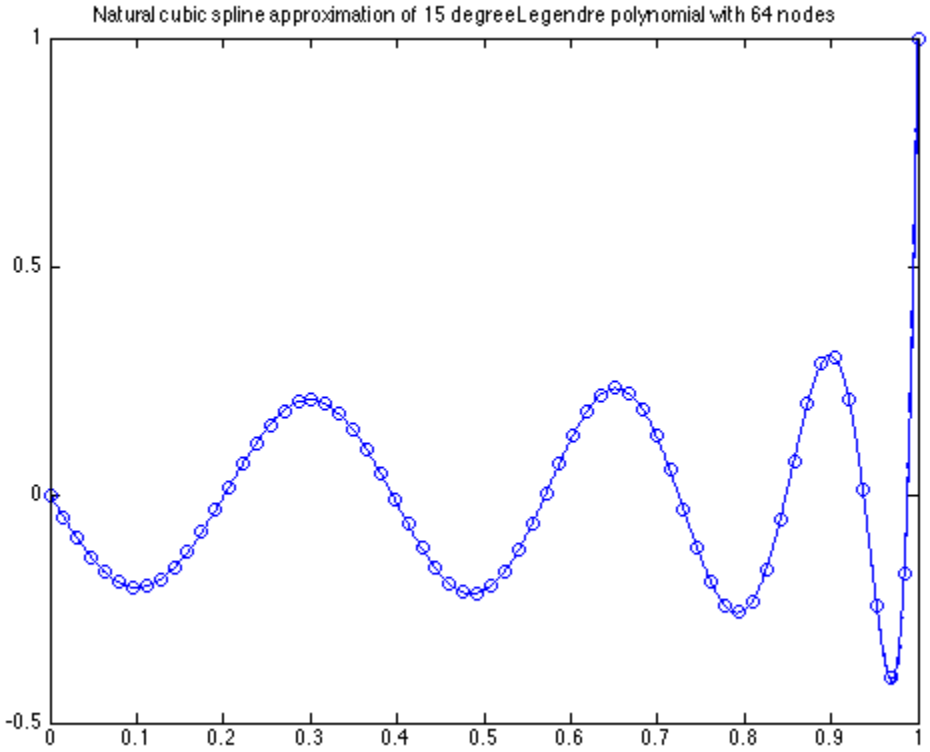

Natural cubic spline approximation of 15 degree Legendre polynomial with 128 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

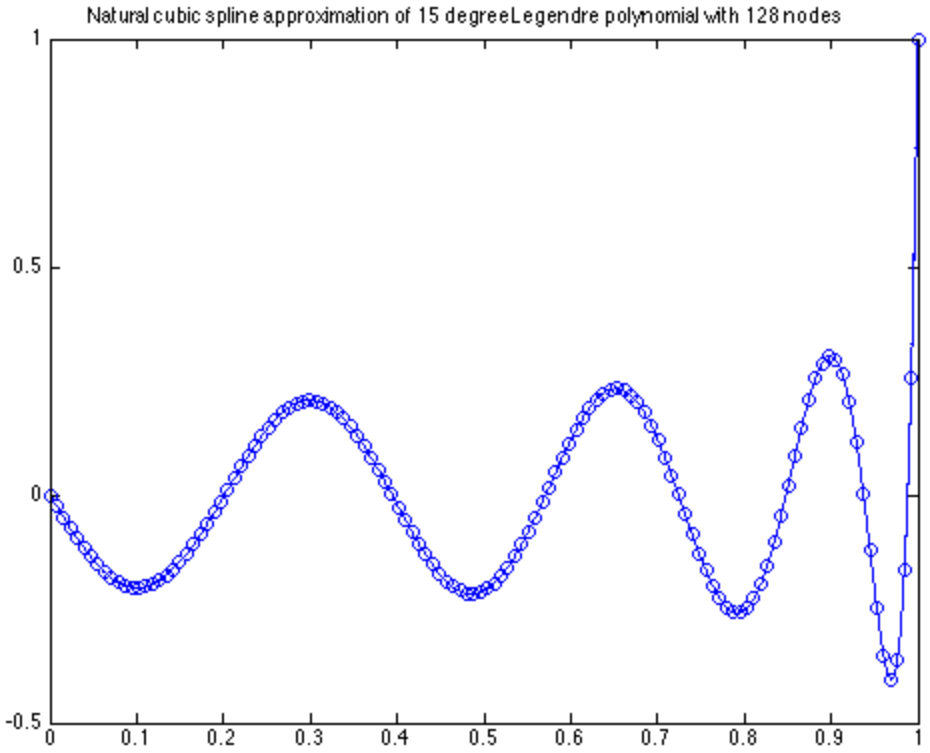

Natural cubic spline approximation of 15 degree Legendre polynomial with 256 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

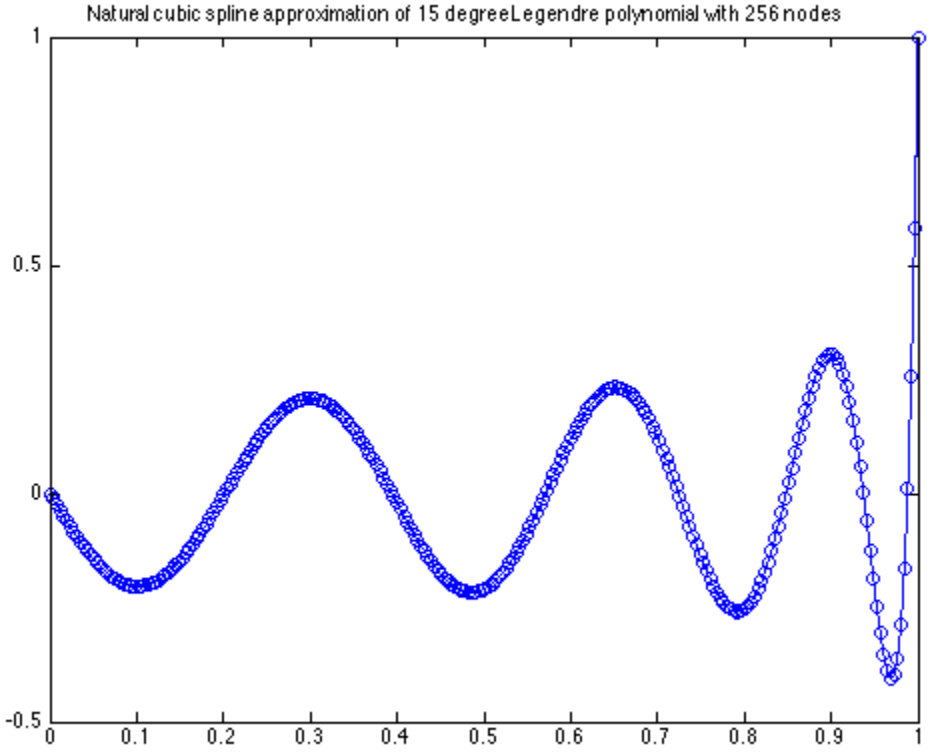

Natural cubic spline approximation of 15 degree Legendre polynomial with 64 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Natural cubic spline approximation of 15 degree Legendre polynomial with 128 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Natural cubic spline approximation of 15 degree Legendre polynomial with 256 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Natural cubic spline approximation of 15 degree Legendre polynomial with 64 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Natural cubic spline approximation of 15 degree Legendre polynomial with 128 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Natural cubic spline approximation of 15 degree Legendre polynomial with 256 nodes
1
0.5
0
-0.5
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1