Midterm – Numerical Computing II

Questions Answered

1. Near the mode and with a larger number of points, the interpolated polynomial is a pretty good match for the Runge function; away from the mode, the interpolation polynomial starts to go pretty wild, especially with higher n.
2. The perturbations of the interpolated polynomial are much less drastic using the Chebyshev inputs!
3. The cubic spline interpolation shows even *less* perturbation than the previous method.

To interpolate the points (0,0), (1/2, 1) and (1,0), we take the intervals [0,1/2] and [1/2,1]. Now in general, the form for cubic equations is

Which admit the following derivatives:

Upon the imposition of the given points and the conditions set on the derivatives, the following system of equations arises:

For convenience, this can be written as a matrix equation and then reduced:

So on take , and on take .

1. The interpolation does become better with increased M, though it seems to begin deviating from the function earlier than the other interpolation types.

The Code

Please note that this program will wait for you to press a key at each step so that you can see each curve being plotted as it happens!

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This script is the lifeblood of the full program. It %

% calls each of the functions, graphing each properly. %

% main.m %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generate a very good looking Runge function.

[runget, rungey] = generate(5000);

[modrunget, modrungey] = generateTrig(5000);

figure(1)

hold on

plot(runget, rungey);

disp('Monomial: Press any key to begin interpolating.');

pause;

for i = 1:4

withMonomial(5\*i);

pause; % This can be removed if you just want the solid figure.

% Pausing will allow you to show each interpolated polynomial

% after calculation.

end

title('Interpolation of Runge function with Monomial Basis and Equally Spaced Inputs');

legend('Runge','n = 5', 'n = 10', 'n = 15', 'n = 20');

hold off

figure(2)

hold on

plot(runget, rungey);

disp('Chebyshev Monomial: Press any key to begin interpolating.');

pause;

for i = 1:4

chebyshevMonomial(5\*i);

pause; % This can be removed if you just want the solid figure.

% Pausing will allow you to show each interpolated polynomial

% after calculation.

end

title('Interpolation of Runge function with Monomial Basis and Chebyshev Inputs');

legend('Runge','n = 5', 'n = 10', 'n = 15', 'n = 20');

hold off

figure(3)

hold on

plot(runget, rungey);

disp('Lagrange: Press any key to begin interpolating.');

pause;

for i = 1:4

withLagrange(5\*i);

pause; % This can be removed if you just want the solid figure.

% Pausing will allow you to show each interpolated polynomial

% after calculation.

end

title('Interpolation of Runge function with Lagrange Basis and Equally Spaced Inputs');

legend('Runge','n = 5', 'n = 10', 'n = 15', 'n = 20');

hold off

figure(4)

hold on

plot(modrunget,modrungey);

disp('Trigonometric Interpolation: Press any key to begin interpolating.');

pause;

for i = 1:21

if (i == 5 || i == 11 || i == 15 ||i == 21)

trigonometricInterpolation(i);

pause; % This can be removed if you just want the solid figure.

% Pausing will allow you to show each interpolated polynomial

% after calculation.

else

continue

end

end

title('Interpolation using a Trigonometric Function');

legend('Modified Runge','M = 2', 'M = 5', 'M = 7', 'M = 10');

hold off

figure(5)

hold on

plot(runget, rungey);

disp('MATLAB Spline: Press any key to begin interpolating.');

pause;

for i = 1:4

useSpline(5\*i);

pause; % This can be removed if you just want the solid figure.

% Pausing will allow you to show each interpolated polynomial

% after calculation.

end

title('Interpolation of Runge function with MATLAB Spline and Equally Spaced Inputs');

legend('Runge','n = 5', 'n = 10', 'n = 15', 'n = 20');

hold off

pause;

close all;

% This function first generates equally spaced points on [-1, 1], then

% evaluates the Runge function at each of the points, returning them

% to vectors.

function [t, y] = generate(n)

% Preallocate space for t and y.

t = zeros(n,1);

y = zeros(n,1);

% First, generate n equally spaced points on [-1,1]

for i = 1:1:n

t(i,1) = -1 + ((2\*i-2)/(n-1));

end

% These tend to come in order, unlike the Chebyshev points. As

% such, sorting the resultant input vector is unnecessary.

% Next, generate the values of the runge function.

for j = 1:1:n

y(j,1) = (1/(1 + 25\*(t(j,1)^2)));

end

end

function [t, y] = generateTrig(N)

% This function first generates equally spaced points, then

% evaluates the a modified Runge function at each of the points,

% returning them to vectors.

% Preallocate space for t and y.

t = zeros(N,1);

y = zeros(N,1);

% First, generate N equally spaced points.

for k = 1:N

% Determine abscissae as defined in (e); fill x.

t(k,1) = (2\*pi\*k)/N;

end

% Next, generate the values of the runge function.

for k = 1:N

% Determine ordinates as defined in (e); fill f.

y(k,1) = 1/(1 + 25\*((t(k,1) - pi)^2));

end

end

% This function first generates a selection of chebyshev points based

% on desired resolution, and then evaluates the Runge function at each

% of the points, returning them to vectors.

function [t, y] = chebyshev(n)

% Chebyshev points seem irrational. Let's not be lossy.

format long

% Preallocate space for t and y vectors.

t = zeros(n,1);

y = zeros(n,1);

% Generate the Chebyshev points.

for k = 1:1:n

r = ((2\*k - 1)\*pi)/(2\*n);

t(k,1) = cos(r);

end

% These tend to come out of order. Sorting will help when using the

% Lagrange basis since it is required that t1 < ... < tn.

t = sort(t);

% Evaluate the Runge function at each chosen Chebyshev point.

for j = 1:1:n

y(j,1) = (1/(1 + (25\*(t(j,1)^2))));

end

end

% Horner's scheme for evaluating a polynomial!

% Input x is the coefficient vector for some polynomial,

% Input t is the value at which we evaluate.

% Output f is the gathered function value.

function f = horner(x,t)

n = length(x);

x1 = x(1,1);

result = x(1,1);

% Horner's method is by nature recursive.

for j = 2:1:n

result = result\*t + x(j,1);

end

f = result;

end

% This function takes in abscissae tn, ordinates yn, and evaluates

% the lagrange basis at some abscissa x based on them.

function v = Lagrange(tn,yn,x) % Lagrange Interpolation

% number of interpolating points

n = length(tn);

% number of domain points

k = length(x);

% Initialize output

v = zeros(1,k);

% Initialize storage

L = ones(n,k);

for i=1:n

for j=1:n

if j~=i

L(i,:)=L(i,:).\*(x-tn(j))/(tn(i)-tn(j));

end

end

v=v+yn(i)\*L(i,:);

end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function interpolates and plots with a monomial basis. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function withMonomial(n)

% Generate n domain values and n range values.

[t, y] = generate(n);

% Construct a matrix to become the Vandermonde matrix.

vMatrix = zeros(n);

% Fill the Vandermonde matrix.

for i = 1:1:n

for j = 1:1:n

vMatrix(i,j) = t(i,1)^(n-j);

end

end

% Solve the matrix equation (vMatrix)x = y for x. These will

% be the coefficients of the interpolating polynomial.

x = vMatrix\y;

% Evaluate the polynomial.

domain = -1:0.01:1;

d = length(domain);

output = zeros(d,1);

for k = 1:1:d

f = horner(x, domain(k));

output(k,1) = f;

end

figure(1)

plot(domain,output);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function interpolates using Chebyshev %

% points and a monomial basis. %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function chebyshevMonomial(n)

% Generate n domain values and n range values.

[t, y] = chebyshev(n);

% Preallocate output vectors.

output = zeros(n,1);

% Construct a matrix to become the Vandermonde matrix.

vMatrix = zeros(n);

% Fill the Vandermonde matrix.

for i = 1:1:n

for j = 1:1:n

vMatrix(i,j) = t(i,1)^(n-j);

end

end

% Solve the matrix equation (vMatrix)x = y for x. These will

% be the coefficients of the interpolating polynomial.

x = vMatrix\y;

domain = -1:0.01:1;

d = length(domain);

output = zeros(d,1);

for k = 1:1:d

f = horner(x, domain(k));

output(k,1) = f;

end

figure(2)

plot(domain,output);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function interpolates Runge using trigonometric %

% methods! %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function trigonometricInterpolation(N)

%N = length(x);

if (mod(N,2) == 0)

disp('You need to choose an odd number.')

return;

end

% Set N-1 out of laziness!

P = N-1;

% Initialize x to hold our abscissae

x = zeros(N,1);

for k = 1:N

% Determine abscissae as defined in (e); fill x.

x(k,1) = (2\*pi\*k)/(N);

end

% Sort the abscissae

x = sort(x);

% Initialize f to hold our ordinates

f = zeros(N,1);

for k = 1:N

% Determine ordinates as defined in (e); fill f.

f(k,1) = 1/(1 + (25\*((x(k,1) - pi)^2)));

end

% Initialize A0 to 0.

A0 = 0;

for k = 1:N

% A0 is calculated recursively.

A0 = A0 + ((2/N)\*f(k,1));

end

% Calculate M.

M = (N-1)/2;

% Initialize vectors A and B to store interpolating coefficients

% for psi. There are M components to each.

A = zeros(M,1);

B = zeros(M,1);

% Calculate the interpolating coefficients. They differ by a trig function.

for h = 1:M

for k = 1:N

A(h,1) = A(h,1) + ((2/N)\*(f(k,1)\*(cos((2\*pi\*h\*k)/(N)))));

B(h,1) = B(h,1) + ((2/N)\*(f(k,1)\*(sin((2\*pi\*h\*k)/(N)))));

end

end

% It is now time to evaluate the interpolation polynomial!

% We have N abscissae and M coefficients, plus A0.

% Develop a nice domain over which to evaluate.

domain = 0:0.01:6;

d = length(domain);

psivalues = zeros(d,1);

for i = 1:d

% We will be calculating the ordinates recursively since

% they require summation. As such, we need to initialize

% each ordinate to A0/2 so that we can add on to them.

psivalues(i,1) = A0/2;

for h = 1:M

% Recursively calculate the ordinates.

psivalues(i,1) = psivalues(i,1) + (A(h,1)\*cos(h\*domain(i))) + (B(h,1)\*sin(h\*domain(i)));

end

end

figure(4)

plot(domain,psivalues);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function interpolates Runge using the Lagrange basis! %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function withLagrange(n)

% Generate n domain values and n range values.

[t, y] = generate(n);

% Generate a nice linspace to plot the polynomial over.

domain = -1:0.01:1;

d = length(domain);

% Preallocate an output vector.

output = zeros(d,1);

% Evaluate the Lagrange polynomial.

for k = 1:1:d

f = Lagrange(t,y,domain(k));

output(k,1) = f;

end

figure(3)

plot(domain,output);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function interpolates Runge using MATLAB's Spline! %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function useSpline(n)

[t, y] = generate(n);

xq = -1:0.1:1;

s = spline(t,y,xq);

figure(5)

plot(xq,s);

end