Part 1: Report

Question 1

Plot of Runge function on interval [-1, 1]:

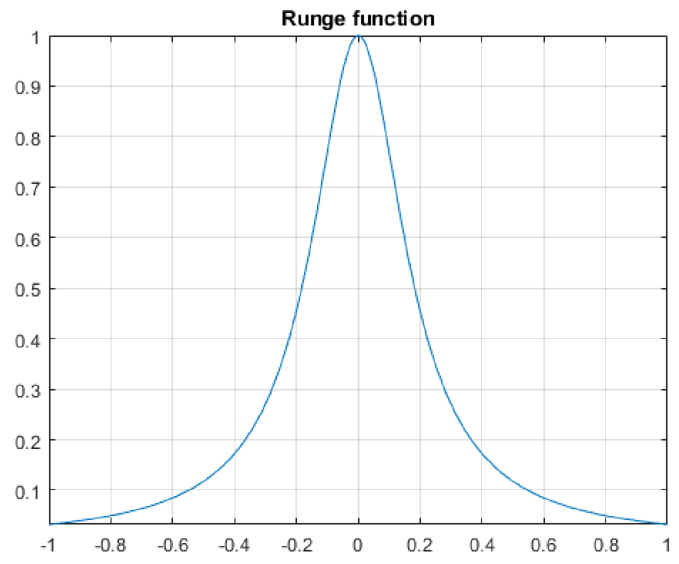


Figure 1. Runge Function plot

Question 2

Execution time of the three functions with n = 5000 equidistant interpolation points is given in the table below. The execution times are given for generation of each of the polynomials themselves, and do not include time to plot the graphs since this was not specified in the instructions (graphs were included in order to verify code correctness for each function and can be viewed by following the link in *Part 2: Code* of this report). The average of 3 runs was taken to get a better approximation since each run varied slightly.

Run	Vandermonde Matrix Form	Barycentric Form	Newton Form
Run 1	1.008s	3.216s	1.112s
Run 2	0.948s	3.260s	1.130s
Run 3	1.041s	3.254s	1.115s
Average Time	**0.999s**	**3.243s**	**1.119s**

The Vandermonde form is quickest, followed closely by Newton's form, then Barycentric. The time that the Barycentric method took was significantly longer than the other two methods and is likely due to the number of matrix multiplications and divisions that occur: two within a for loop to generate the numerator and denominator values, and one for the final polynomial that combines the numerator and denominator values. Newton's form takes approximately 1/3 of the time and has only one matrix multiplications, so it appears that each instance that requires matrix multiplications takes approximately one second. The Vandermonde method has no matrix operations, rather, it uses symbolic math to calculate the polynomial. It is known that this should take more time to execute but is not the case here and may be due to the hardware used for execution.

Question 3

As the Vandermonde form of the polynomial was found to have the fasted execution time it was used for the interpolation point distribution exploration done in this section.

A) Equidistant Interpolation Points

The oscillations near the endpoints that result from choosing equidistant interpolation points increase as more points are used (Figure 2). The middle of the Runge function is well approximated by all n values, with n = 10 beginning to oscillate around [-0.3, 0.3], followed by the other n-value functions around [-0.6, 0.6]. The polynomial using n = 40 interpolation points is a close approximation of the function for most of the interval but has the greatest oscillations. When the y values of the plot are not restricted to [0, 1] it oscillates to –2.5*10^5 near the endpoints of the interval (Figure 3). This oscillation phenomenon indicates that, by choosing an equidistant point distribution, the interpolation polynomial becomes a worse approximation of the function the more data from the function one uses.

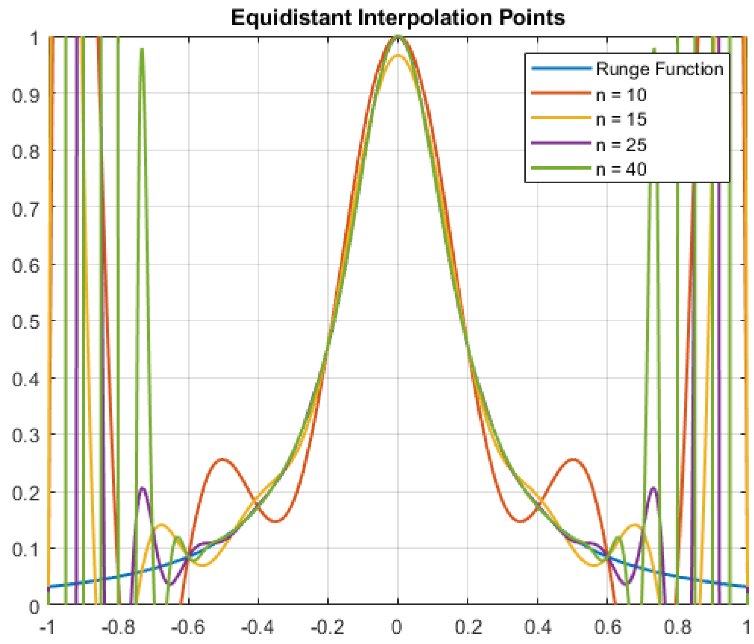


Figure 2. Equidistant interpolation points, y restricted

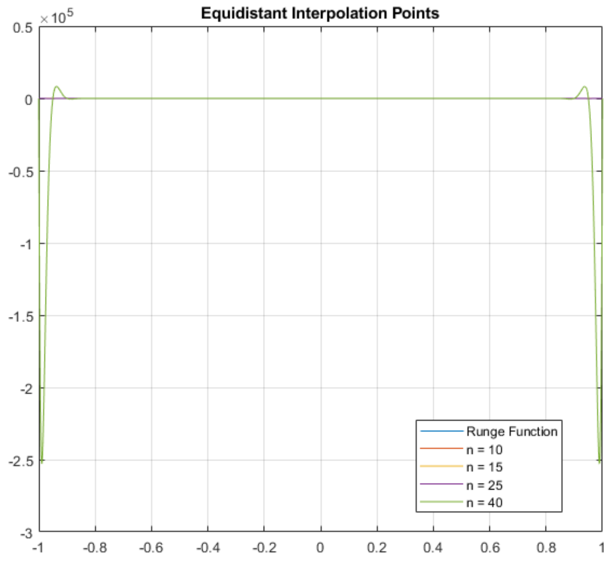


Figure 3. Equidistant interpolation points, y unrestricted

B) Chebyshev Interpolation Points

Compared to the equidistant point distribution Chebyshev points give a much more stable graph (Figure 4). Where the equidistant points oscillate more with a higher n-value the Chebyshev has very little oscillation. This is seen only at the lower n-values of 10 and, to a very small degree, 15. At n = 25 the interpolation polynomial is an almost perfect approximation (the approximation at x = 0 is slightly off), and at n = 40 there is no difference between the original function and the interpolation polynomial. Note that the n-values of 10, 15 and 25 of the equidistant point graph lie much closer to the Runge function than their Chebyshev counterparts in the x interval [-0.2, 0.2]. Despite this, the Chebyshev distribution is much better than equidistant for approximating the function because the polynomial lack the oscillations near the endpoints and become better approximations of the function when more data points are used.

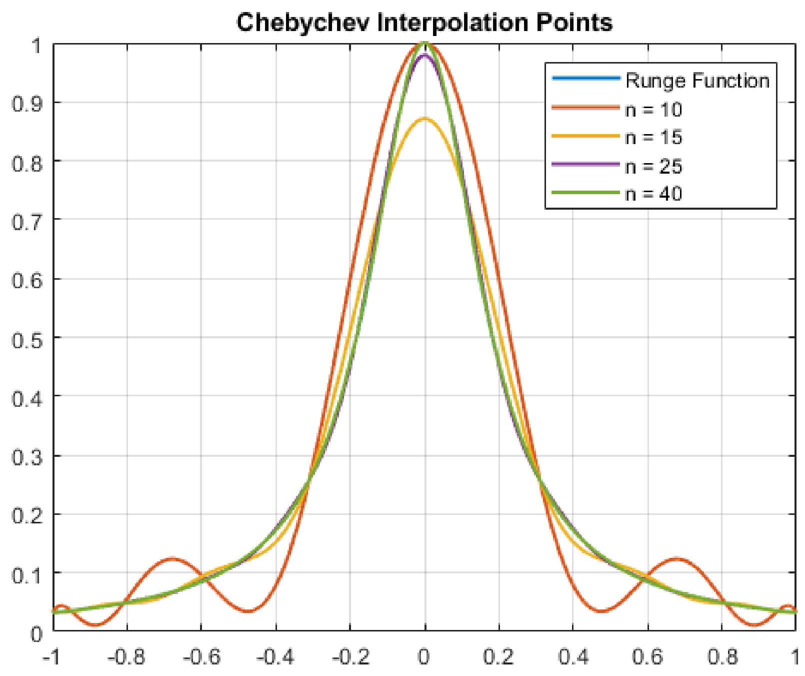


Figure 4. Chebyshev Interpolation Points

C) Other Interpolation Point Distributions

My exploration of other distributions did not yield any better results than the Chebyshev points. I first tried the random distribution as suggested. As can be seen in Figures 5 and 6 these were generally very poor approximations of the Runge function: at n = 40 both approximated it relatively well until about [-0.6, 0.6], which is no better than the equidistant distribution. Random distribution also has the negative effect of being very unpredictable, with each execution yielding different results. Given enough points this may change, though since the oscillations at the endpoints are prominent here, I do not believe that this will improve with more data, as is seen in the equidistant distribution.

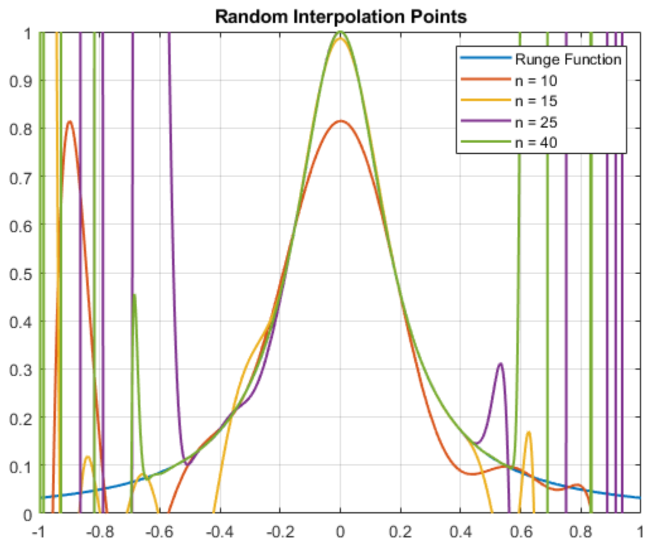


Figure 5. Random Point Distribution 1

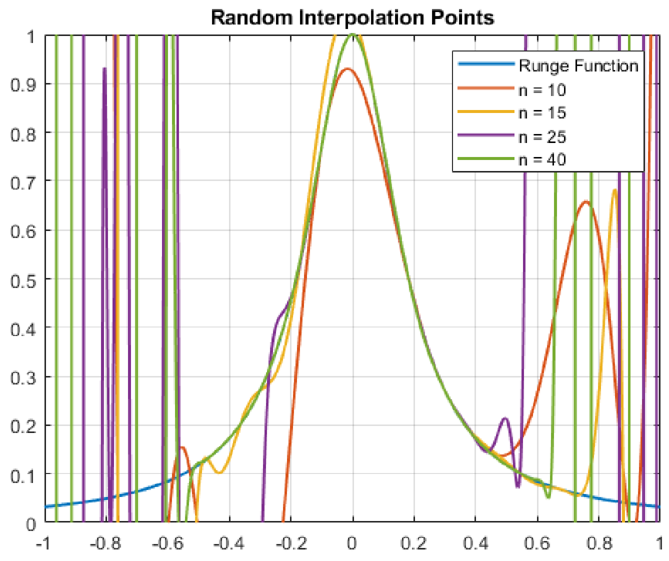


Figure 6. Random Point Distribution 2

Since the Chebyshev distribution was the best distribution seen thus far, it follows that multiplying the points by some scalar constant could yield interesting results, so I tested the interpolation polynomials with a 2*Chebyshev distribution (Figure 7). The results for polynomials with interpolation points of n>40 are quite poor, though a positive is the lack of oscillations near the endpoints. With n = 40 the function is well approximated for much of the interval, though near the ends there is some slight wobbling that is absent in the regular Chebyshev distribution. Thus, of all distributions examined, the Chebyshev distribution of interpolation points with n >=25 yields the most accurate approximation of the function.

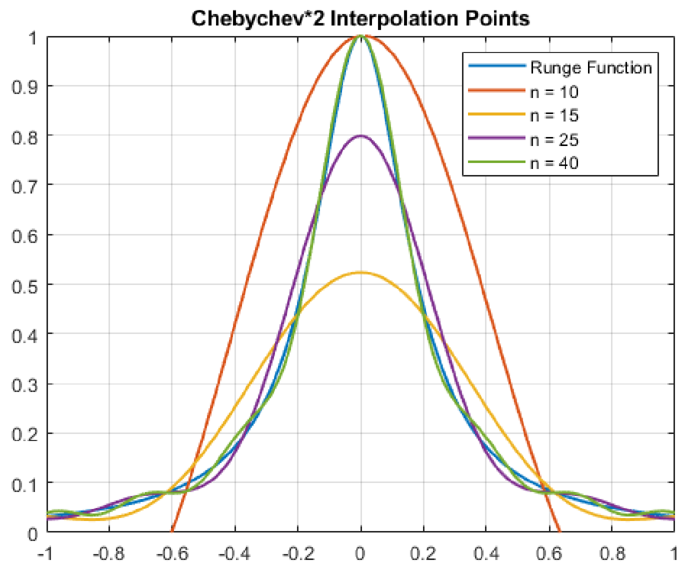


Figure 7. Chebyshev*2 Distribution

Question 4

Plot of the natural spline of the Runge function with n = 5 and n = 10 is given in Figure 8. The interpolation function with n = 5 is not a very good approximation of the function, but with n = 10 it is much closer. Comparing this to the Chebyshev interpolation graph (Figure 4), it takes the Vandermonde form of the interpolation polynomial 25 data points to have as accurate an approximation as the natural spline has in 10. Thus, it appears that the natural spline function is the best method for approximation because it needs so few data to give an accurate representation of the function.

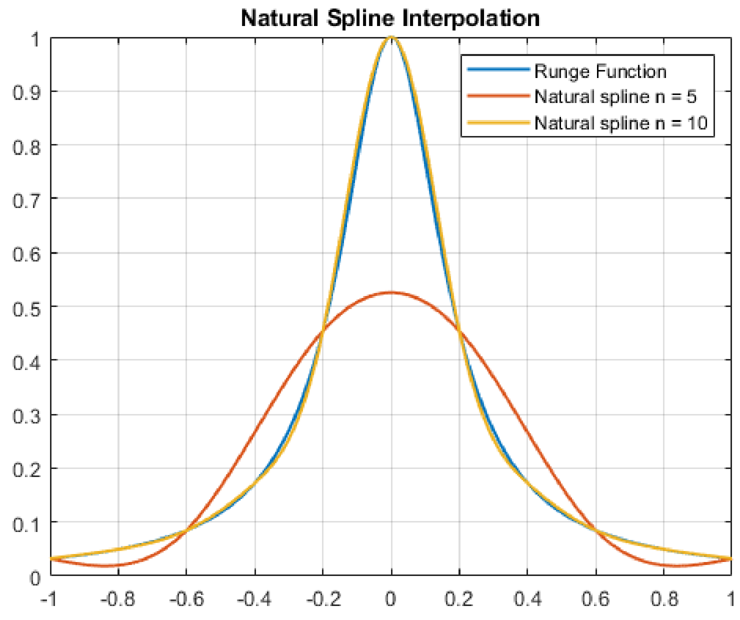
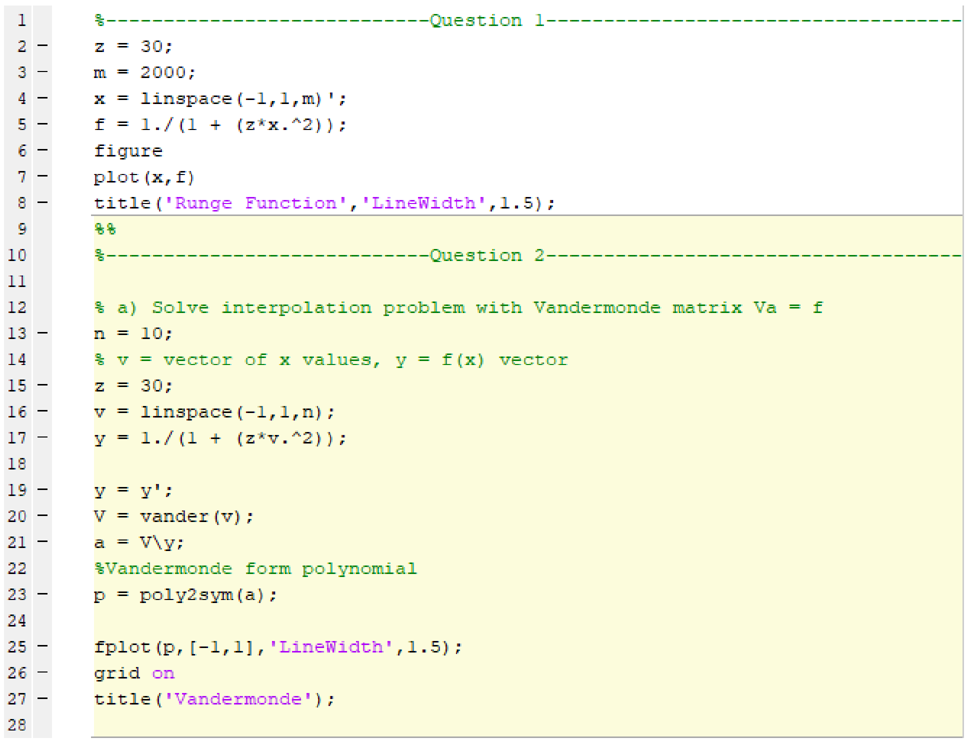


Figure 8. Natural Spline Interpolation

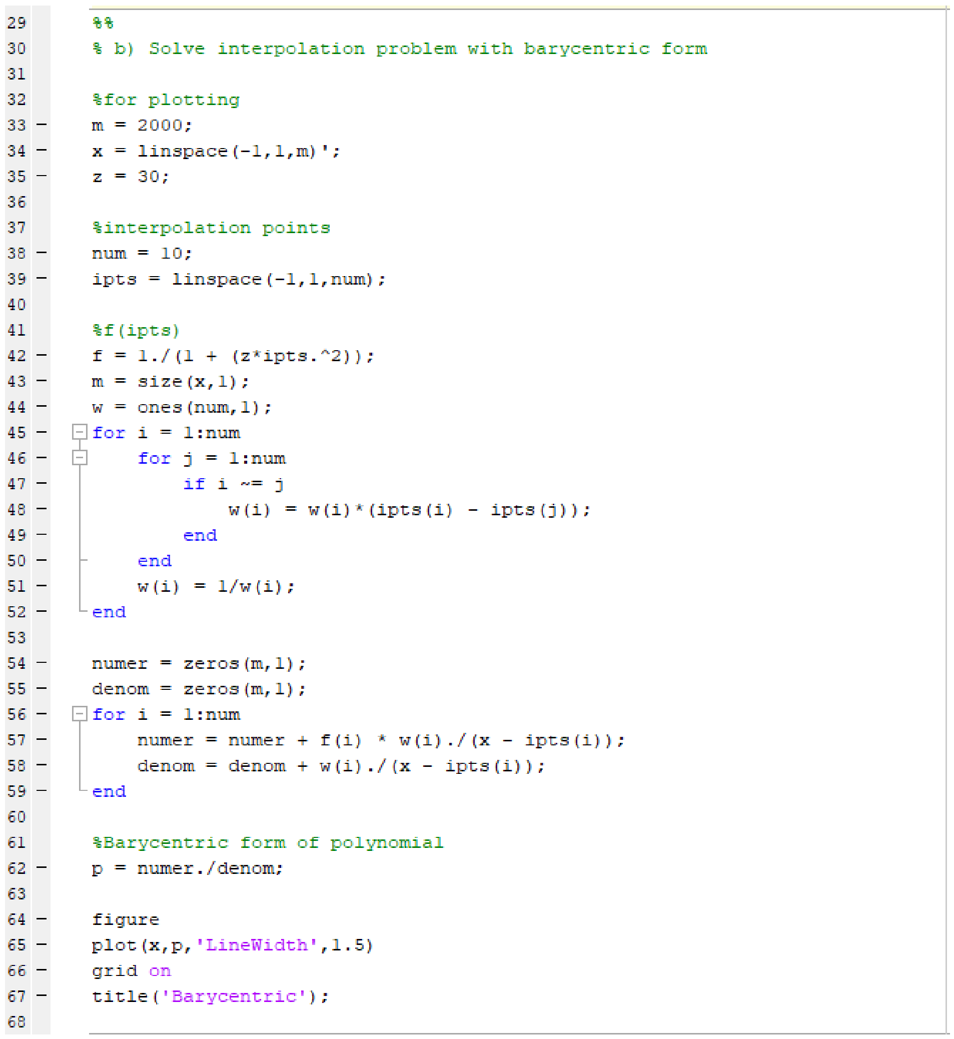
Part 2: Code

Assignment 5 Code

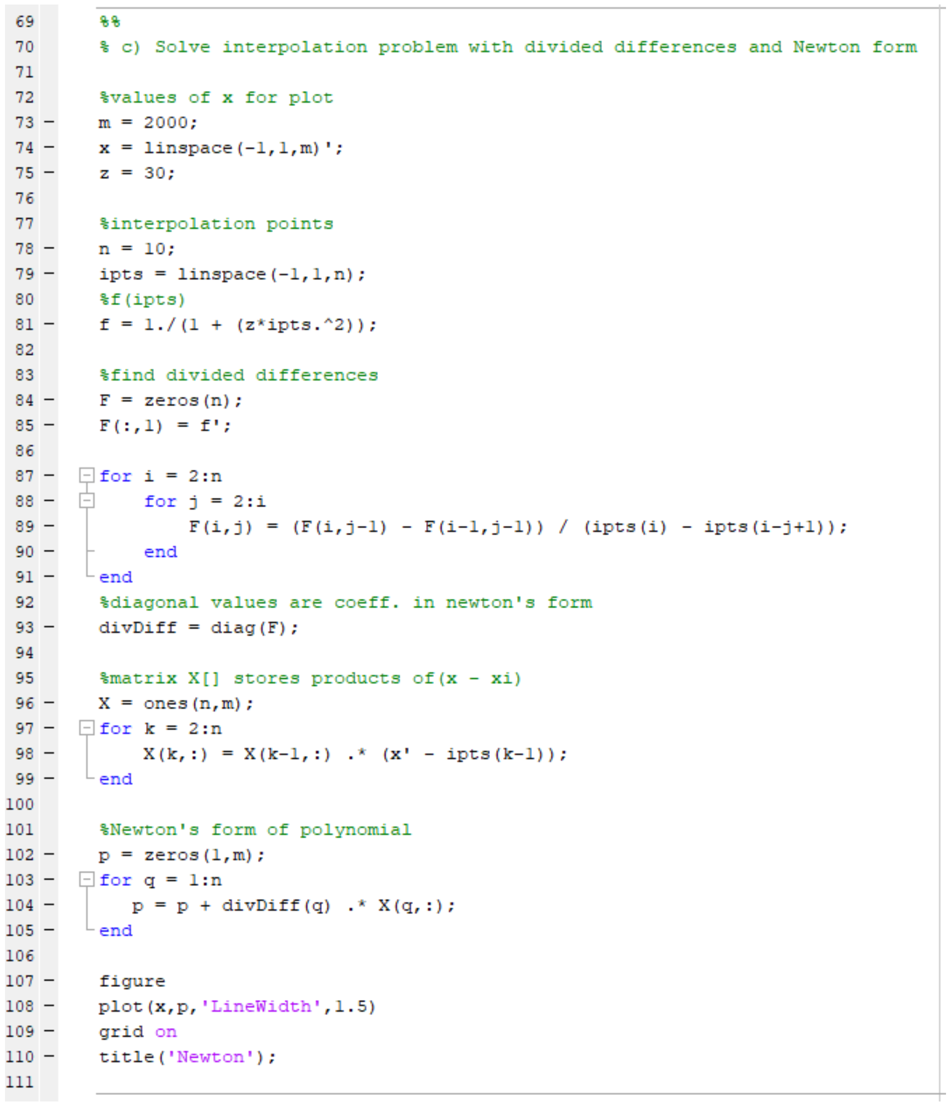


```matlab
%----------------------------Question 1------------------------------------
z = 30;
m = 2000;
x = linspace(-1,1,m)';
f = 1./(1 + (z*x.^2));
figure
plot(x,f)
title('Runge Function','LineWidth',1.5);
%%
%----------------------------Question 2------------------------------------

% a) Solve interpolation problem with Vandermonde matrix Va = f
n = 10;
% v = vector of x values, y = f(x) vector
z = 30;
v = linspace(-1,1,n);
y = 1./(1 + (z*v.^2));

y = y';
V = vander(v);
a = V\y;
%Vandermonde form polynomial
p = poly2sym(a);

fplot(p,[-1,1],'LineWidth',1.5);
grid on
title('Vandermonde');

```

Note: barycentric form code is adapted from code provided in class in intpoly.m



```
%%
% b) Solve interpolation problem with barycentric form

%for plotting
m = 2000;
x = linspace(-1,1,m)';
z = 30;

%interpolation points
num = 10;
ipts = linspace(-1,1,num);

%f(ipts)
f = 1./(1 + (z*ipts.^2));
m = size(x,1);
w = ones(num,1);
for i = 1:num
    for j = 1:num
        if i ~= j
            w(i) = w(i)*(ipts(i) - ipts(j));
        end
    end
    w(i) = 1/w(i);
end

numer = zeros(m,1);
denom = zeros(m,1);
for i = 1:num
    numer = numer + f(i) * w(i)./(x - ipts(i));
    denom = denom + w(i)./(x - ipts(i));
end

%Barycentric form of polynomial
p = numer./denom;

figure
plot(x,p,'LineWidth',1.5)
grid on
title('Barycentric');

```



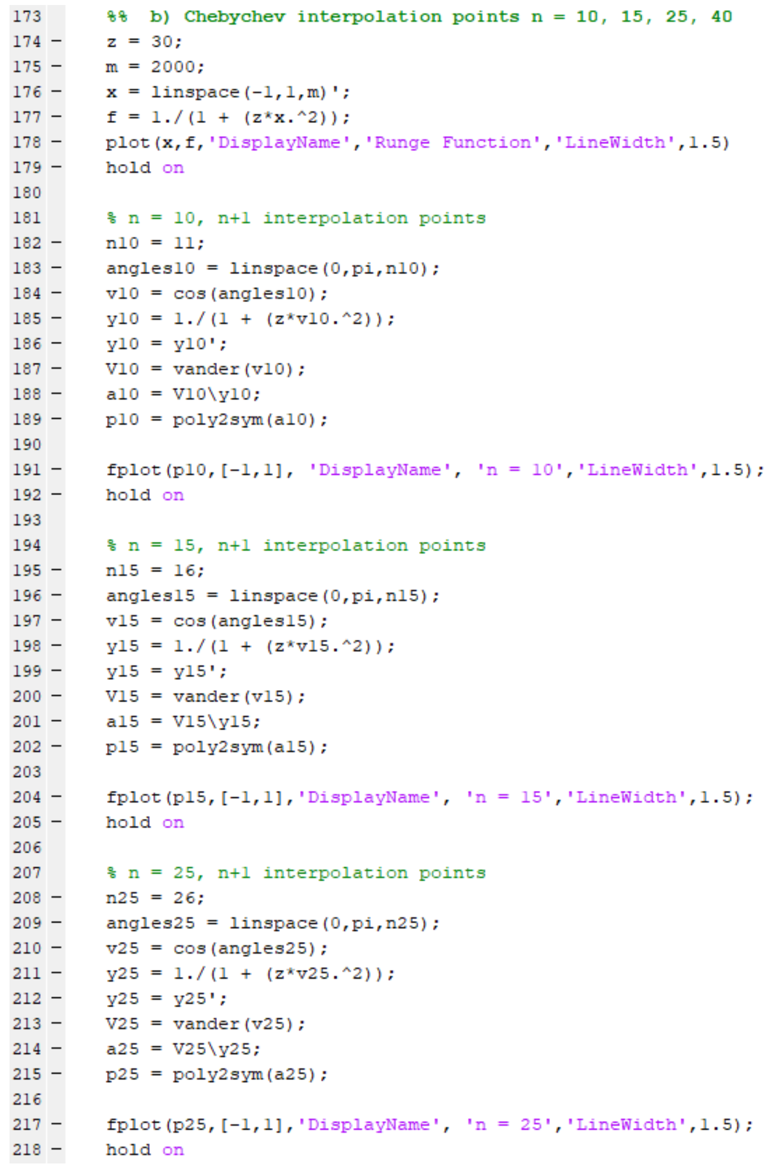
```matlab
%%
% c) Solve interpolation problem with divided differences and Newton form

%values of x for plot
m = 2000;
x = linspace(-1,1,m)';
z = 30;

%interpolation points
n = 10;
ipts = linspace(-1,1,n);
%f(ipts)
f = 1./(1 + (z*ipts.^2));

%find divided differences
F = zeros(n);
F(:,1) = f';

for i = 2:n
    for j = 2:i
        F(i,j) = (F(i,j-1) - F(i-1,j-1)) / (ipts(i) - ipts(i-j+1));
    end
end
%diagonal values are coeff. in newton's form
divDiff = diag(F);

%matrix X[] stores products of(x - xi)
X = ones(n,m);
for k = 2:n
    X(k,:) = X(k-1,:) .* (x' - ipts(k-1));
end

%Newton's form of polynomial
p = zeros(1,m);
for q = 1:n
   p = p + divDiff(q) .* X(q,:);
end

figure
plot(x,p,'LineWidth',1.5)
grid on
title('Newton');

```

```matlab
%% ----------------------------Question 3---------------------------------
%a) equidistant interpolation points n = 10, 15, 25, 40
z = 30;
m = 2000;
x = linspace(-1,1,m)';
f = 1./(1 + (z*x.^2));
plot(x,f,'DisplayName','Runge Function','LineWidth',1.5)
hold on

% n = 10, n+1 interpolation points
n10 = 11;
v10 = linspace(-1,1,n10);
y10 = 1./(1 + (z*v10.^2));
y10 = y10';
V10 = vander(v10);
a10 = V10\y10;
p10 = poly2sym(a10);

fplot(p10,[-1,1], 'DisplayName', 'n = 10','LineWidth',1.5);
hold on

% n = 15, n+1 interpolation points
n15 = 16;
v15 = linspace(-1,1,n15);
y15 = 1./(1 + (z*v15.^2));
y15 = y15';
V15 = vander(v15);
a15 = V15\y15;
p15 = poly2sym(a15);

fplot(p15,[-1,1],'DisplayName', 'n = 15','LineWidth',1.5);
hold on

% n = 25, n+1 interpolation points
n25 = 26;
v25 = linspace(-1,1,n25);
y25 = 1./(1 + (z*v25.^2));
y25 = y25';
V25 = vander(v25);
a25 = V25\y25;
p25 = poly2sym(a25);

fplot(p25,[-1,1],'DisplayName', 'n = 25','LineWidth',1.5);
hold on

% n = 40, n+1 interpolation points
n40 = 41;
```

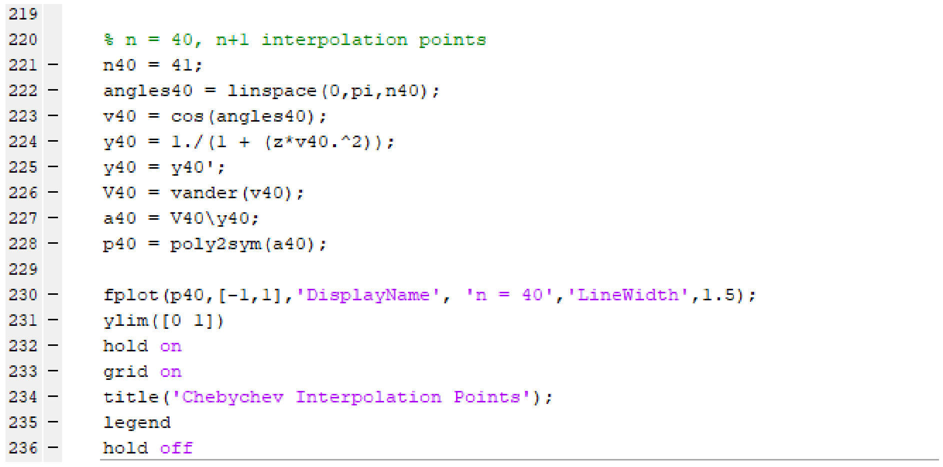
```matlab
% n = 40, n+1 interpolation points
n40 = 41;
```

```matlab
v40 = linspace(-1,1,n40);
y40 = 1./(1 + (z*v40.^2));
y40 = y40';
V40 = vander(v40);
a40 = V40\y40;
p40 = poly2sym(a40);

fplot(p40,[-1,1],'DisplayName', 'n = 40','LineWidth',1.5);
ylim([0 1])
hold on
grid on
title('Equidistant Interpolation Points');
legend
hold off
```

Note: code for determining Chebyshev interpolation points was adapted from code provided in class in ellx.m



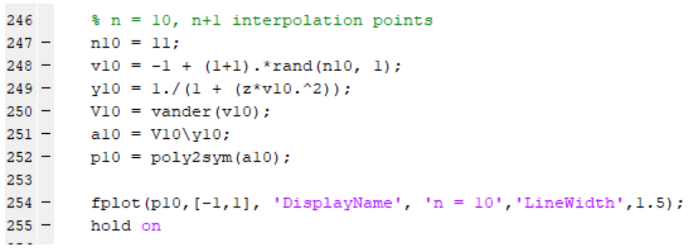
```
173      %%  b) Chebychev interpolation points n = 10, 15, 25, 40
174      z = 30;
175      m = 2000;
176      x = linspace(-1,1,m)';
177      f = 1./(1 + (z*x.^2));
178      plot(x,f,'DisplayName','Runge Function','LineWidth',1.5)
179      hold on
180
181      % n = 10, n+1 interpolation points
182      n10 = 11;
183      angles10 = linspace(0,pi,n10);
184      v10 = cos(angles10);
185      y10 = 1./(1 + (z*v10.^2));
186      y10 = y10';
187      V10 = vander(v10);
188      a10 = V10\y10;
189      p10 = poly2sym(a10);
190
191      fplot(p10,[-1,1], 'DisplayName', 'n = 10','LineWidth',1.5);
192      hold on
193
194      % n = 15, n+1 interpolation points
195      n15 = 16;
196      angles15 = linspace(0,pi,n15);
197      v15 = cos(angles15);
198      y15 = 1./(1 + (z*v15.^2));
199      y15 = y15';
200      V15 = vander(v15);
201      a15 = V15\y15;
202      p15 = poly2sym(a15);
203
204      fplot(p15,[-1,1],'DisplayName', 'n = 15','LineWidth',1.5);
205      hold on
206
207      % n = 25, n+1 interpolation points
208      n25 = 26;
209      angles25 = linspace(0,pi,n25);
210      v25 = cos(angles25);
211      y25 = 1./(1 + (z*v25.^2));
212      y25 = y25';
213      V25 = vander(v25);
214      a25 = V25\y25;
215      p25 = poly2sym(a25);
216
217      fplot(p25,[-1,1],'DisplayName', 'n = 25','LineWidth',1.5);
218      hold on
```



```
219
220      % n = 40, n+1 interpolation points
221      n40 = 41;
222      angles40 = linspace(0,pi,n40);
223      v40 = cos(angles40);
224      y40 = 1./(1 + (z*v40.^2));
225      y40 = y40';
226      V40 = vander(v40);
227      a40 = V40\y40;
228      p40 = poly2sym(a40);
229
230      fplot(p40,[-1,1],'DisplayName', 'n = 40','LineWidth',1.5);
231      ylim([0 1])
232      hold on
233      grid on
234      title('Chebychev Interpolation Points');
235      legend
236      hold off
```

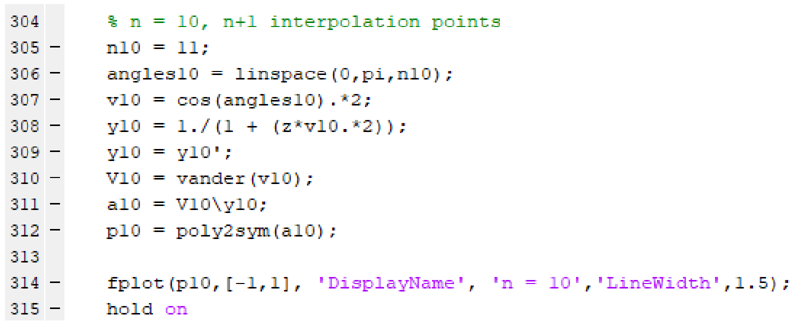
Random Distribution (code is duplicate of equidistant points (lines 114-158) except lines 248, 259, 270, 281 which fill the vectors v10, v15, v25, and v40 with random numbers).

Sections are identical so only one is posted here for brevity. Full code can be seen via the link above.



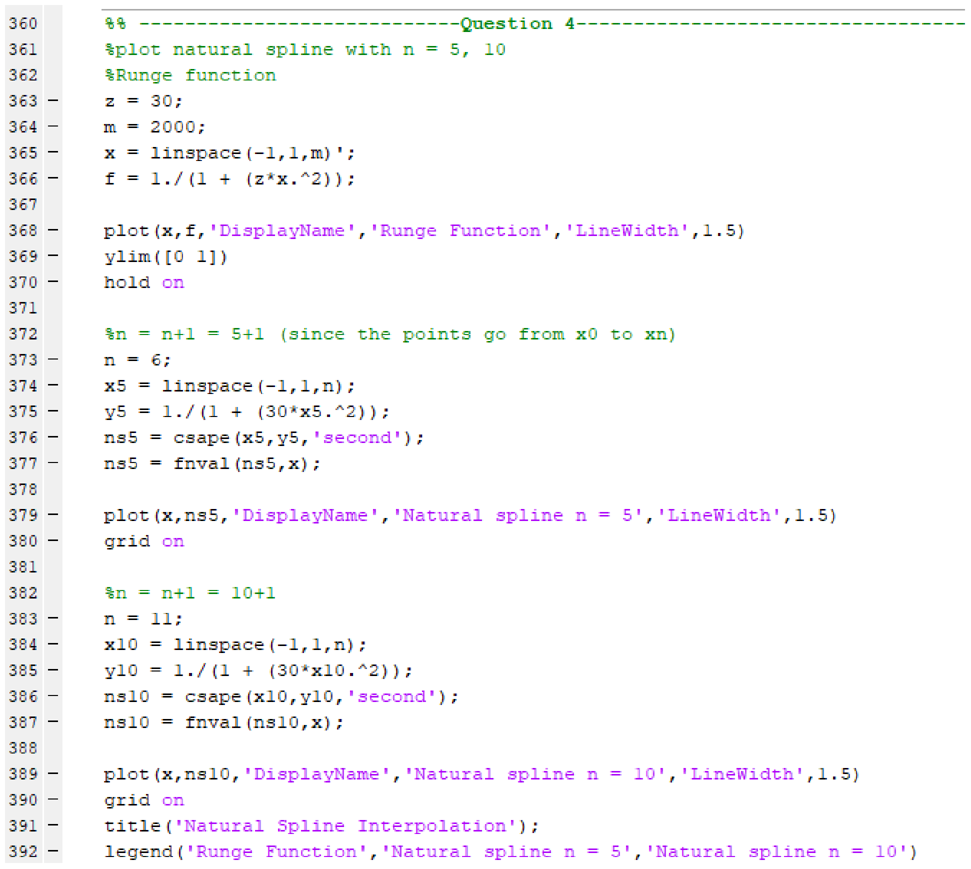
```
% n = 10, n+1 interpolation points
n10 = 11;
v10 = -1 + (1+1).*rand(n10, 1);
y10 = 1./(1 + (z*v10.^2));
V10 = vander(v10);
a10 = V10\y10;
p10 = poly2sym(a10);

fplot(p10,[-1,1], 'DisplayName', 'n = 10','LineWidth',1.5);
hold on
```

Chebyshev * 2 (code is duplicate of Chebyshev points from lines 173-236 except lines 306, 319, 332, 345 which fill the vectors v10, v15, v25, and v40 with 2 * each Chebyshev point). Since these sections are also identical only one is posted here.



```
304       % n = 10, n+1 interpolation points
305 -     n10 = 11;
306 -     angles10 = linspace(0,pi,n10);
307 -     v10 = cos(angles10).*2;
308 -     y10 = 1./(1 + (z*v10.*2));
309 -     y10 = y10';
310 -     V10 = vander(v10);
311 -     a10 = V10\y10;
312 -     p10 = poly2sym(a10);
313
314 -     fplot(p10,[-1,1], 'DisplayName', 'n = 10','LineWidth',1.5);
315 -     hold on
```

Note: code for natural spline was adapted from code provided in class in spline1.m and spline2.m



```
%% ----------------------------Question 4----------------------------------
%plot natural spline with n = 5, 10
%Runge function
z = 30;
m = 2000;
x = linspace(-1,1,m)';
f = 1./(1 + (z*x.^2));

plot(x,f,'DisplayName','Runge Function','LineWidth',1.5)
ylim([0 1])
hold on

%n = n+1 = 5+1 (since the points go from x0 to xn)
n = 6;
x5 = linspace(-1,1,n);
y5 = 1./(1 + (30*x5.^2));
ns5 = csape(x5,y5,'second');
ns5 = fnval(ns5,x);

plot(x,ns5,'DisplayName','Natural spline n = 5','LineWidth',1.5)
grid on

%n = n+1 = 10+1
n = 11;
x10 = linspace(-1,1,n);
y10 = 1./(1 + (30*x10.^2));
ns10 = csape(x10,y10,'second');
ns10 = fnval(ns10,x);

plot(x,ns10,'DisplayName','Natural spline n = 10','LineWidth',1.5)
grid on
title('Natural Spline Interpolation');
legend('Runge Function','Natural spline n = 5','Natural spline n = 10')
```