Table of Contents

G12ISC 2019-2020 Coursework 4

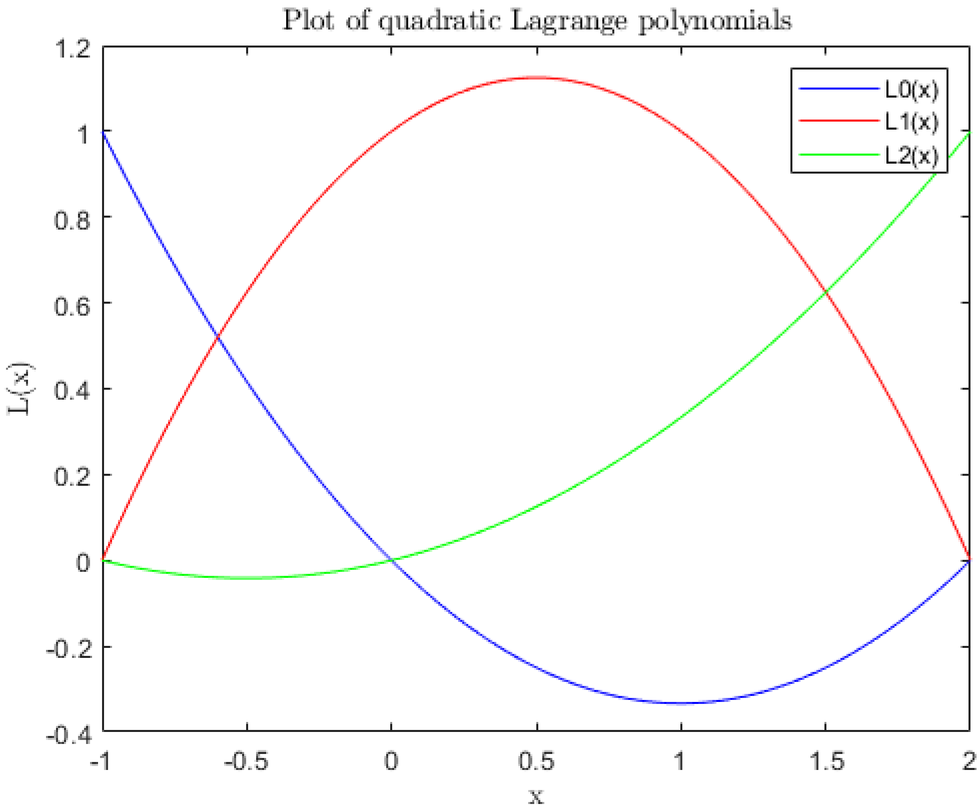
Subject: Polynomial interpolation Student Name: Jake Denton Student ID:14322189

```
clear all
close all
clc
```

Question 1

This code defines the three quadratic Lagrange polynomials based on x0,x1 and x2 respectively and plots them together on one figure.

```
x=linspace(-1,2,1000);%this creates a vector with 1000 values between
 -5 and 5
L0=(1/3).*x.*(x-2);%defines the L0 lagrange polynomial
plot(x,L0,'b')%plots this polynomial in blue
L1=(-1/2).*(x+1).*(x-2);%this line and the next define the L1/L2
 polynomials respectively
L2=(1/6).*x.*(x+1);
hold on %holds the output so that the other two lines can be produced
 on the same plot
plot(x,L1,'r')%plots L1 polynomial in red
plot(x,L2,'g')%plots L2 polynomial in green
hold off
set(groot,'DefaultTextInterpreter','latex')%rest of the code formats
 the graph
xlabel('x');
ylabel('L(x)');
title('Plot of quadratic Lagrange polynomials');
legend('L0(x)','L1(x)','L2(x)');
```

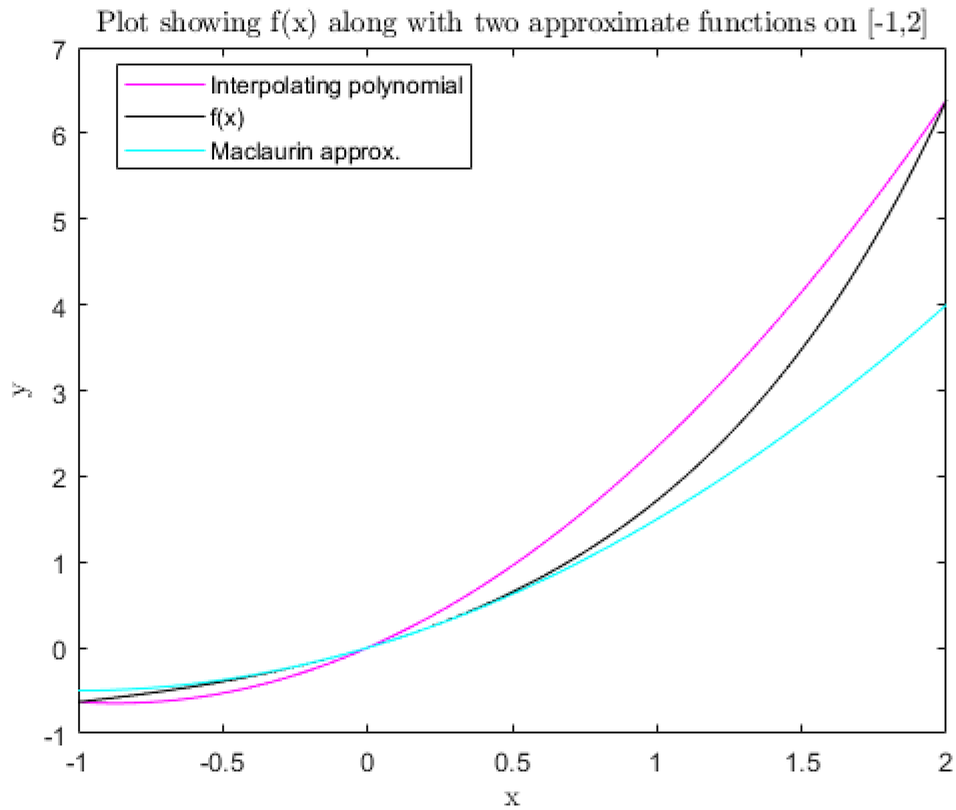


Question 2

This code defines f(x) along with two approximate functions on [-1,2], one being the maclaurin series of f(x) expanded up to the x^2 term, the other the interpolating polynomial generated from x0,x1,x2 given. It then plots the lines in the same figure.

```
f=@(x) exp(x)-1;%creates function handle to use in P(x) formula
x=linspace(-1,2,200);%creates set of 200 equally spaced points in
 [-1,2]
L0=(1/3).*x.*(x-2);%next few lines redefine Lagrange polynomials
L1=(-1/2).*(x+1).*(x-2);
L2=(1/6).*x.*(x+1);
P=f(-1)*L0+f(0)*L1+f(2)*L2;%this line generates the polynomial
 interpolant
plot(x,P,'m')%plots P(x)
fx=exp(x)-1;%defines the function we're interested in approximating
macx=x+(1/2).*x.^2;%defines the maclaurin series of f(x) up to x^2
 term
hold on%holds output of figure
plot(x,fx,'k')%next two lines add fx and maclaurin approximation to
 same plot
plot(x,macx,'c')
hold off
set(groot,'DefaultTextInterpreter','latex')%rest of this code formats
 the figure
xlabel('x');
```

```
ylabel('y');
title('Plot showing f(x) along with two approximate functions on
 [-1,2]');
legend('Interpolating polynomial','f(x)','Maclaurin
 approx.','Location','Best');
```



Question 3

This code defines a general vector of n+1 uniformly distributed points within the interval [a,b] then confirms this vector is as expected for 4 equidistant points in [-1,2].

```
a=-1; %first 3 values needed to go into the linspace function
b=2;
n=3;
z=linspace(a,b,n+1);%the general form of the equation that generates
 the vector of n+1 equally spaced points
disp('4 uniformly distributed points in [-1,2] generated by the
 general form of z vector above are:');
disp(z);%confirms general z vector gives the expected output
```

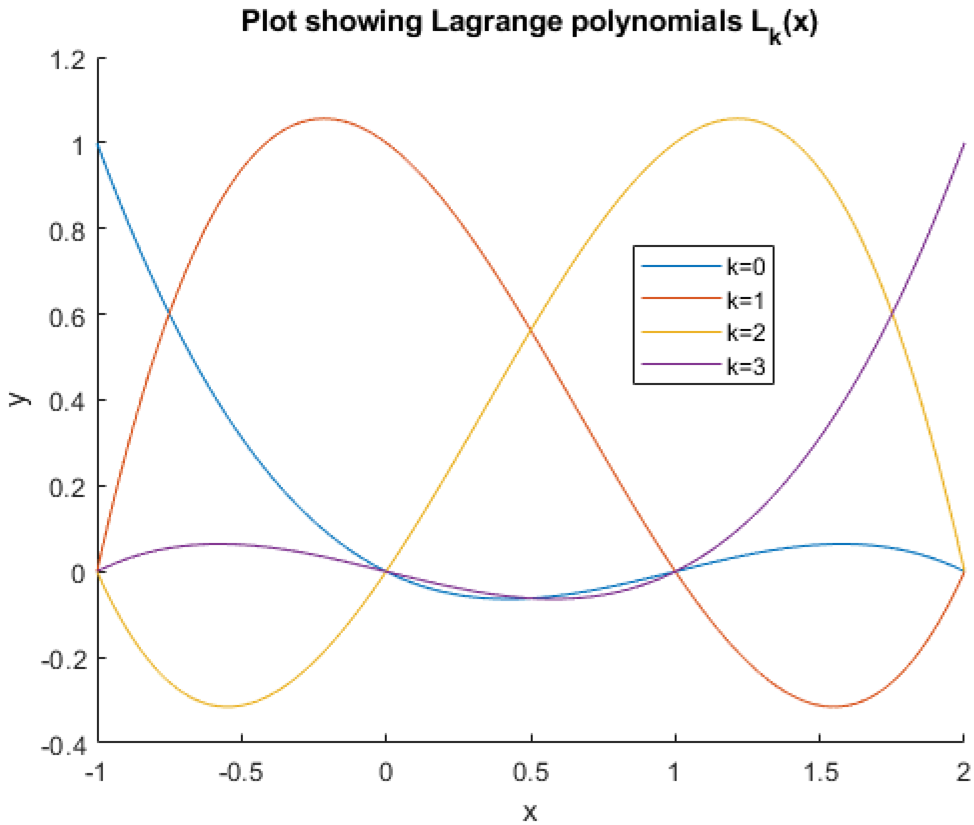
```
4 uniformly distributed points in [-1,2] generated by the general form
 of z vector above are:
    -1     0     1     2
```

Question 5

This code uses a for loop to plot the Lagrange polynomials calculated from 4 nodes in [-1,2] on the same figure.

```
close all hidden
axes

z=linspace(-1,2,4);%creates a vector of n+1 (4) equally spaced points
 (the nodes)
x=linspace(-1,2,200);%creates a vector for use in evaluating points on
 the curves
for k=0:3 %for loop has an index from zero to n (each node has a
 Lagrange polynomial)
    y=LagrPolyn(k,x,z);%uses LagrPolyn to evaluate Lk polynomial at x
    hold all %allows all the curves to be plotted on the same axes
    plot(x,y);
    Legend{k+1}=strcat('k=',num2str(k));%labels each curve with its
 appropriate k value
end
hold off
set(groot,'DefaultTextInterpreter','tex')%this text format allowed me
 to use a subscript in the title
title('Plot showing Lagrange polynomials L_{k}(x)');%rest of the code
 formats the plot
xlabel('x');
ylabel('y');
legend(Legend,'Location','Best');
```



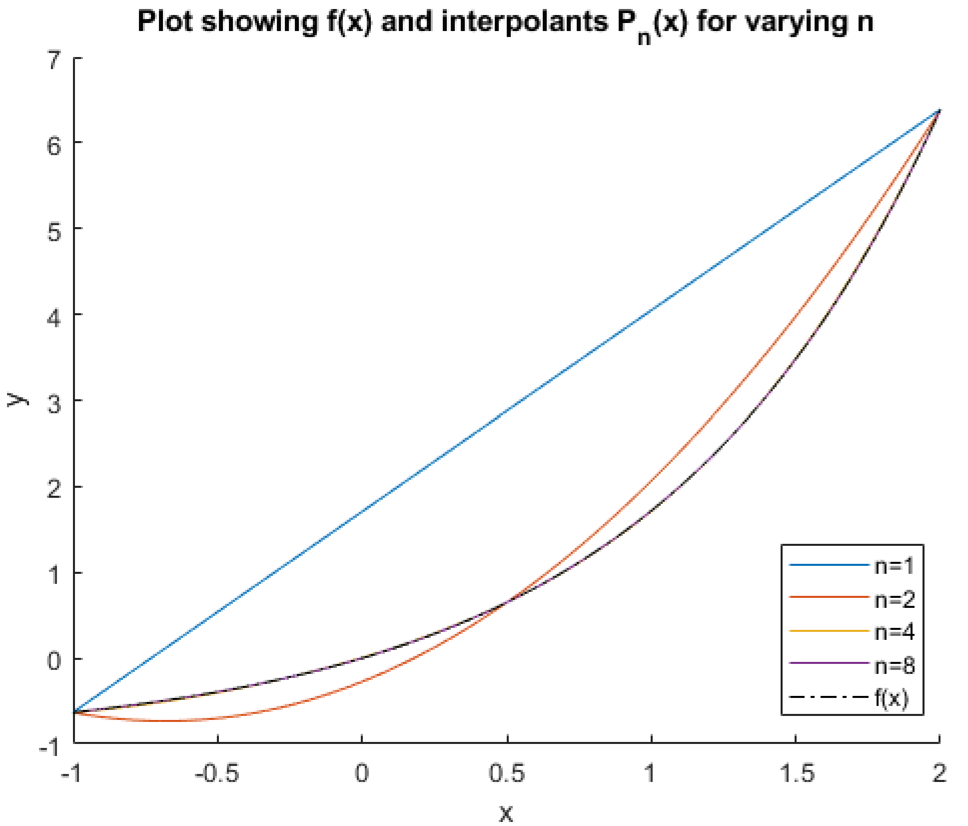
Question 7

This code plots on one figure the curve f(x)=exp(x)-1 along with interpolants of varying numbers of nodes. Note that to see each curve clearly, the zoom in tool must be used as the interpolants with n=4,8 and f(x) curves coincide very closely.

```
 close all hidden
 axes

f=@(x) exp(x)-1;%defines the function handle as f(x) to go into
 PolynInterp function
x=linspace(-1,2,150);%creates a vector for us to evaluate our
 functions over (interpolants have nodes in [-1,2])
for i=0:3 %this for loop defines the interpolant for varying n and
 plots them on the figure
    n=2^(i);%this defines the degree of each interpolant
    z=linspace(-1,2,n+1);%for this n, this gives n+1 equally spaced
 points in [-1,2]
    Px=PolynInterp(f,x,z);%evaluates value of interpolating polynomial
 with n+1 points
    hold all %allows all the curves to be put in one figure
    plot(x,Px);%plots the curve
end
fx=exp(x)-1;%defines the function given, f(x)
plot(x,fx,'k-.');%plots f(x) with a black dash-dot line (this is done
 as the lines with n=4,8 coincide very closely to f(x))
```

```
hold off
set(groot,'DefaultTextInterpreter','tex')%rest of this code formats
 the figure
title('Plot showing f(x) and interpolants P_{n}(x) for varying n');
xlabel('x');
ylabel('y');
legend('n=1','n=2','n=4','n=8','f(x)','Location','Best');
```



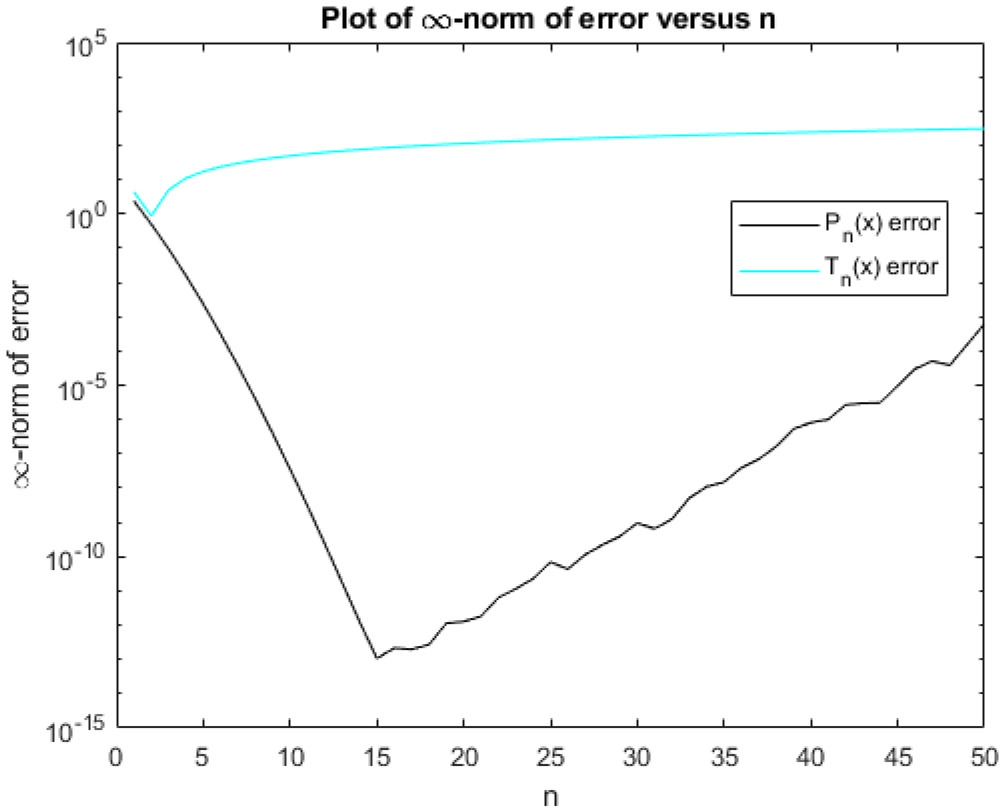
Question 9

This code plots the error in the polynomial interpolant against n.

```
f=@(x) exp(x)-1;%defines function handle as f(x) given
N=1:50;%creates a row vector with 50 elements for us to use in the
 plot
E1=zeros(1,50);%next two lines initialise error vectors
E2=zeros(1,50);
Tn=0;%initialises Tn which will be the maclaurin series of f(x)
x=linspace(-1,2,2500);%defines a vector x which we will use with the
 maclaurin series
fx=f(x);%defines f(x) which we will use to compute the error with the
 maclaurin series
for n=1:50%for loop from 1 to 50 as this is the range of n I decided
 to investigate
    z=linspace(-1,2,n+1);%defines a vector with n+1 uniformly
 distributed points in [-1,2]
```

```
    E1(n)=PolyInterpolError(-1,2,f,z);%assigns the nth element in E1
 with the max absolute error for this n
    for i=1:n%this for loop generates the maclaurin series of f(x)
        Term=x.^(i)./factorial(i);
        Tn=Tn+Term;
    end
    emac=abs(fx-Tn);%defines a vector with of absolute errors
    E2(n)=max(emac);%assigns nth element in E2 with max absolute error
 for this n (for T(x))
end
semilogy(N,E1,'k');%plots error curve for interpolating polynomial on
 logarithmic axes
hold on
semilogy(N,E2,'c');%also plots error curve for maclaurin series on the
 same figure
set(groot,'DefaultTextInterpreter','tex')%rest of the code formats the
 figure
title('Plot of \infty-norm of error versus n');
xlabel('n');
ylabel('\infty-norm of error');
legend('P_{n}(x) error','T_{n}(x) error','Location','Best');
hold off

% BONUS
% We observe that the Maclaurin series has increasing error with
 increasing
% n, which is expected since we are investigating the interval [-1,2],
 and
% so the Maclaurin series is not a good estimate of the function at
 x=2.
% This can be analysed by considering f(2)-Tn(2); as n increases more
% positive terms are added to Tn(2) so the absolute value
 monotonically
% increases beyond a specific n by 2^(n)/factorial(n). Since
 factorials
% grow quicker than powers, the graph levels off as observed.
```

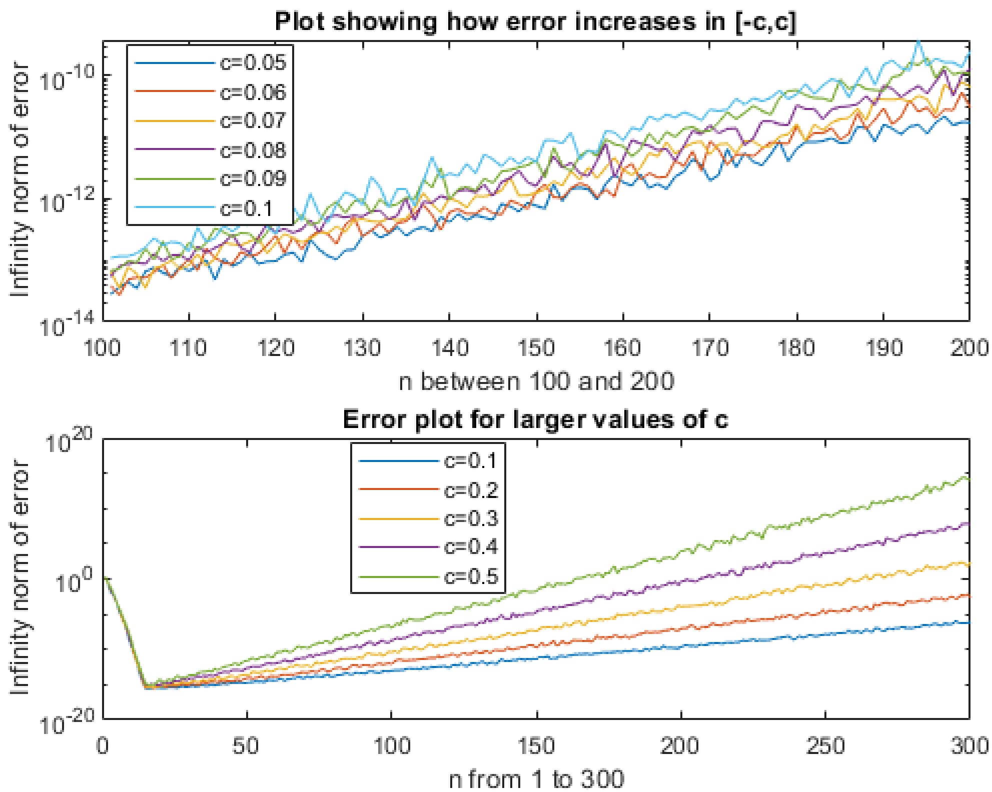


Q9 Answer For Convergence

Clearly from the graph produced above, Pn does not converge to f(x) in the interval [-1,2]. The code below takes our previous interpolants over [-1,2] and investigates how the error evolves in the sub-interval [-c,c] as n approaches infinity. The first plot displays how, even for tiny c, as n gets larger the error begins to increase after a certain point (therefore not satisfying the formal definition of convergence). However, although there is an increase, the error (if considered by plotting the interpolant and comparing to f(x) by eye) is negligible and would not be noticed. This prompted me to produce the second plot, which has greater values of c and again displays divergence. Despite this, at n=300 the errors for c=0.1,02 are still very small. We can conclude the polynomial interpolant remains very close to f(x) (would appear to have converged) for large n if a c value of 0.2 is used.

```
f=@(x) exp(x)-1;%define function handle
E=zeros(1,100);%initialises error vector
N=101:200;%this will be the range of n we investigate over (chosen
 because large and useful for analysis)
for c=0.05:0.01:0.1%these values of c are investigated (they're very
 small)
    x=linspace(-c,c,200);%defines a vector with values in [-c,c]
    fx=f(x);%evaluates f(x) with the above x
    for n=101:200
        z=linspace(-1,2,n+1);%defines the uniform distributed points
 in interval
        y=PolynInterp(f,x,z);%evaluates the interpolant with nodes in
 [-1,2] at the [-c,c] sub interval
```

```
        evec=abs(fx-y);%next two lines find the infinity norm of the
 absolute error and assign them to element of E
        E(n-100)=max(evec);
    end
    hold all
    subplot(2,1,1);
    semilogy(N,E);%plots the error against n for this value of c
end
hold off
title('Plot showing how error increases in [-c,c]');%rest of this code
 formats the first graph
xlabel('n between 100 and 200');
ylabel('Infinity norm of error');
legend('c=0.05','c=0.06','c=0.07','c=0.08','c=0.09','c=0.1','Location','Best');
E2=zeros(1,300);%initialises second error vector
N2=1:300;%creates vector to plot error against
for c=0.1:0.1:0.5%these values of c are investigated (they're larger
 than previous plot)
    x=linspace(-c,c,200);%defines a vector with values in [-c,c]
    fx=f(x);%evaluates f(x) with the above x
    for n=1:300
        z=linspace(-1,2,n+1);%defines the uniform distributed points
 in interval
        y=PolynInterp(f,x,z);%evaluates the interpolant with nodes in
 [-1,2] at the [-c,c] sub interval
        evec=abs(fx-y);%next two lines find the infinity norm of the
 absolute error and assign them to element of E2
        E2(n)=max(evec);
    end
    hold all
    subplot(2,1,2);
    semilogy(N2,E2);%plots the error against n for this value of c
end
hold off
title('Error plot for larger values of c');%rest of this code formats
 the second graph
xlabel('n from 1 to 300');
ylabel('Infinity norm of error');
legend('c=0.1','c=0.2','c=0.3','c=0.4','c=0.5','Location','Best');
```

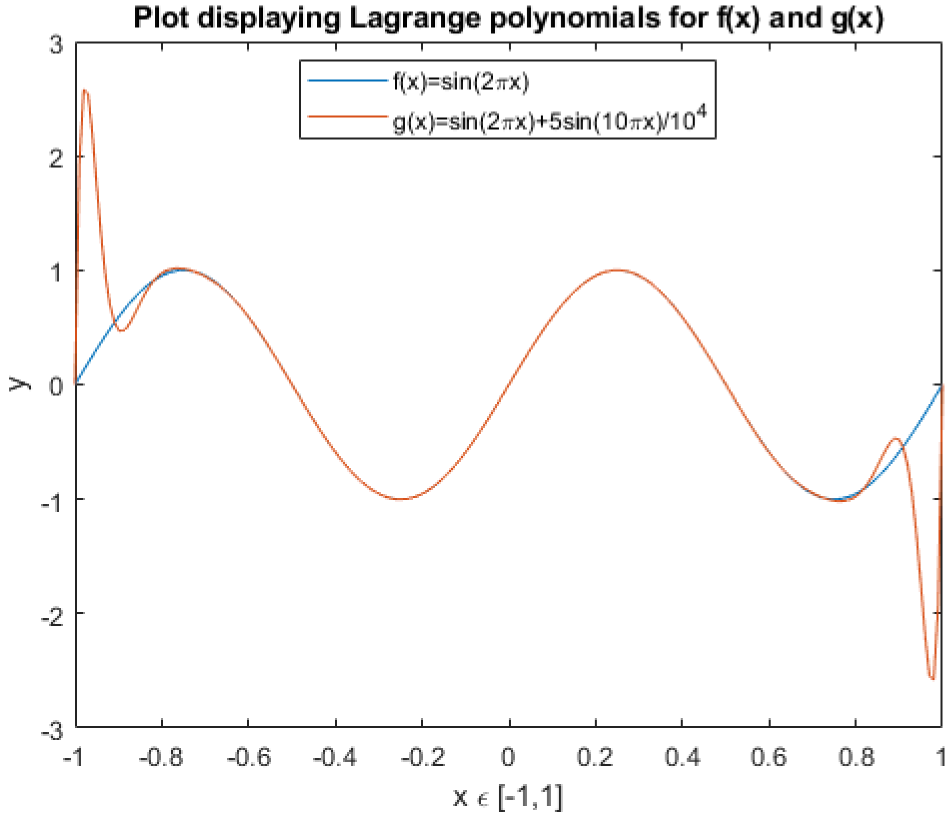


Question 10

This code evaluates and plots the Lagrange polynomials of order 22 for f(x) and g(x) given in the region [-1,1].

```
close all hidden
axes
f=@(x) sin(2*pi*x);%first two lines define function handles for f(x)
 and g(x)
g=@(x) sin(2*pi*x)+(5/[10^4])*sin(10*pi*x);
z=linspace(-1,1,23);%this creates the vector of nodes to put into
 PolynInterp
x=linspace(-1,1,200);%creates a vector for the polynomial to be
 evaluated at
y1=PolynInterp(f,x,z);%defines polynomial degree 22 and evaluates at x
plot(x,y1);%plots f(x) curve
hold on
y2=PolynInterp(g,x,z);%as above but using g(x)
plot(x,y2);
set(groot,'DefaultTextInterpreter','tex')%rest of this code formats
 the figure
title('Plot displaying Lagrange polynomials for f(x) and g(x)');
xlabel('x \epsilon [-1,1]');
ylabel('y');
legend('f(x)=sin(2\pix)','g(x)=sin(2\pix)+5sin(10\pix)/10^{4}','Location','Best');
hold off
```

```
% The plot shows that for x approximately in [-0.7,0.7], the Lagrange
% polynomials of the functions with n=22 coincide. The effect of the
% perturbation in g(x) presents itself near the endpoints of the
 interval
% where there is oscillation around the curve for f(x) which itself
% resembles a sine curve very closely. The g(x) polynomial behaving
 this
% way is an example of Runge's phenomenon, since the nodes are
 equidistant
% and the interpolant has a high degree.
```



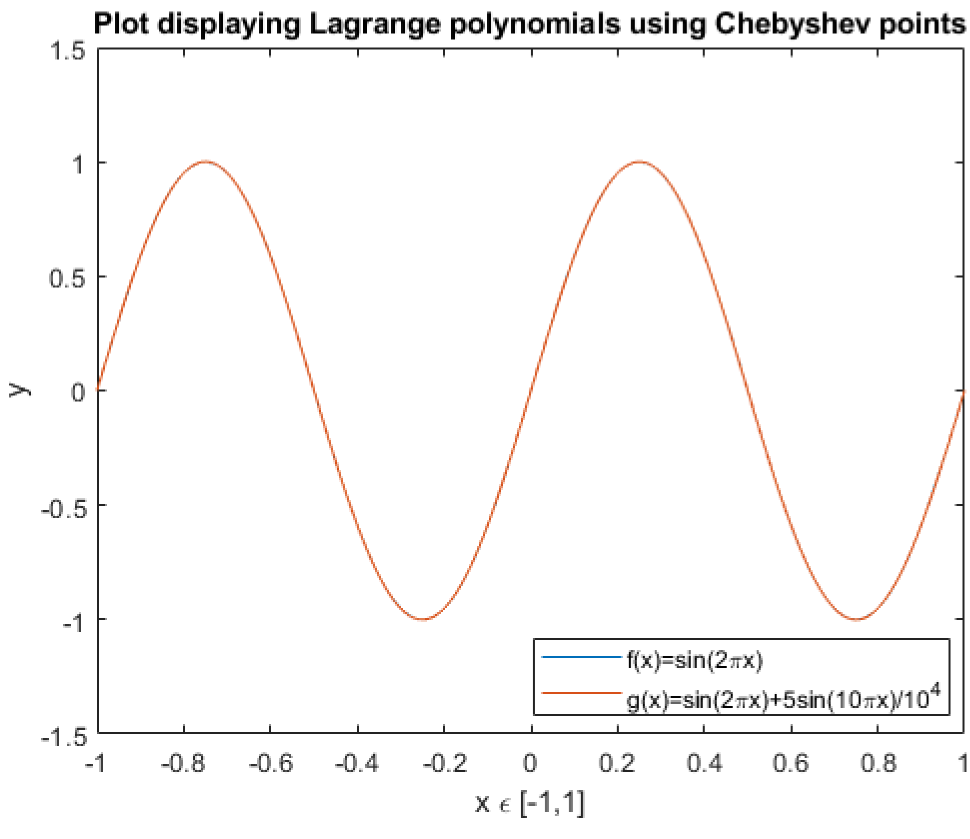
Question 11

This code plots the Lagrange polynomials of f and g with degree 22, where 23 Chebyshev points are used as the nodes generating each polynomial instead of 23 uniformly distributed points in the interval [-1,1].

```
f=@(x) sin(2*pi*x);%first two lines define function handles for f(x)
 and g(x)
g=@(x) sin(2*pi*x)+(5/[10^4])*sin(10*pi*x);
z=GridCheb(22,-1,1);%this creates the vector of Chebyshev points as
 required
x=linspace(-1,1,200);%creates a vector for the polynomial to be
 evaluated at
y1=PolynInterp(f,x,z);%defines polynomial degree 22 and evaluates at x
plot(x,y1);%plots f(x) curve
```

```
hold on
y2=PolynInterp(g,x,z);%as above but using g(x)
plot(x,y2);
set(groot,'DefaultTextInterpreter','tex')%rest of this code formats
 the figure
title('Plot displaying Lagrange polynomials using Chebyshev points');
xlabel('x \epsilon [-1,1]');
ylabel('y');
legend('f(x)=sin(2\pix)','g(x)=sin(2\pix)+5sin(10\pix)/10^{4}','Location','Best');
hold off

% We now observe that when Chebyshev points are used instead of
 uniformly
% distributed points on the interval that the Lagrange polynomials of
 order
% n=22 for f(x) and g(x) coincide very closely at all points in [-1,1]
 i.e.
% the effect of the perturbation is mitigated. the zoom in tool must
 be
% used to observe any differences between the curves.
```



Published with MATLAB® R2019a