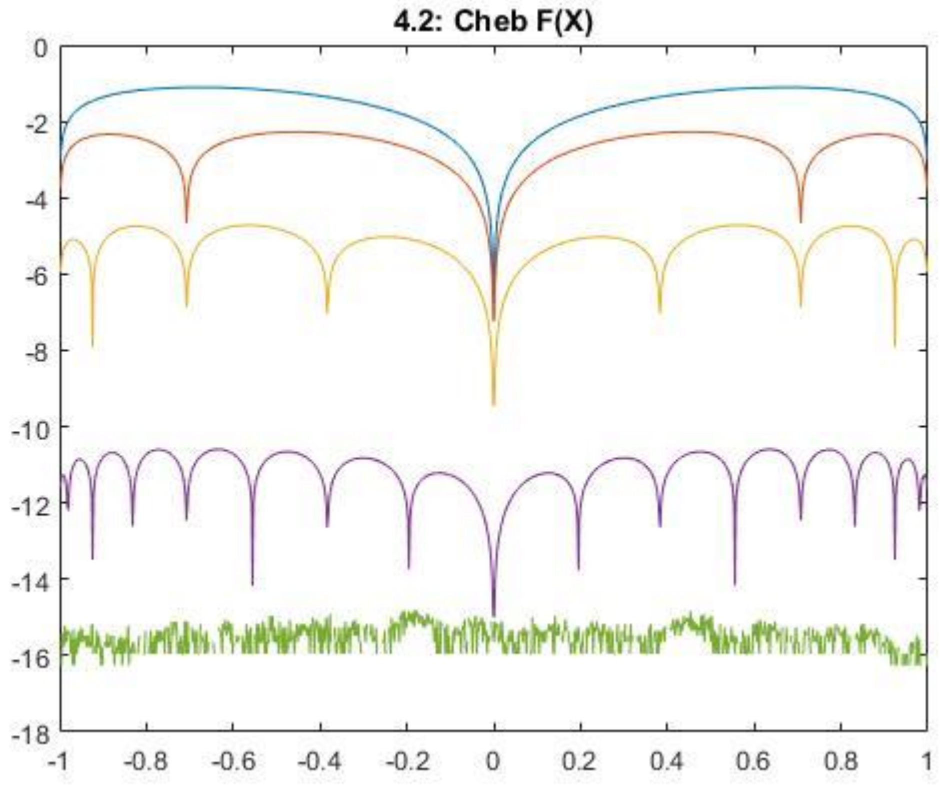
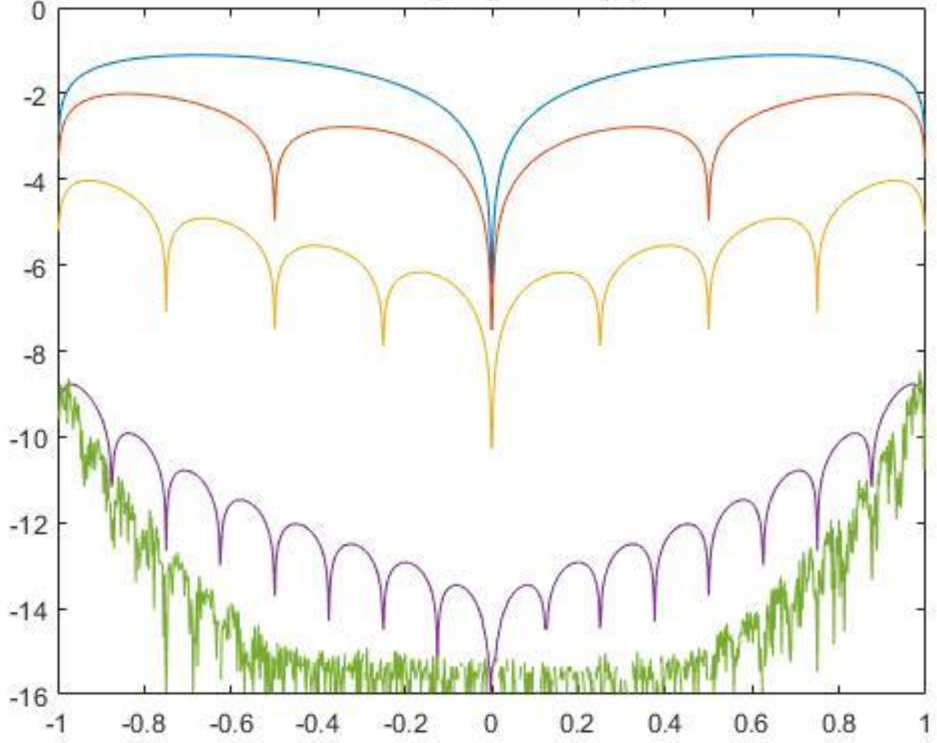
```
% Bhairav Mehta
% HW 4 - Eduardo Corona
% Engineering 371



% 1a.

% for i = 1:length(M1)
%     [~,P1(i)] = chol(M1{i});
%     [~,P2(i)] = chol(M2{i});
%     [~,P3(i)] = chol(M3{i});
%     %used the cholesky facrotization function in matlab to confirm that the
%     %matricies were Symetric Positive Definite
% end

%1.b using condition number function we find the condition number for the
%matricies and it is confirmed what is asked.

% for i = 1:length(M1)
%     C1{1,i} = cond(M1{i});
%     C1{2,i} = cond(M2{i});
%     C1{3,i} = cond(M3{i});
%     i = i+1;
% end

%%%1.c -------------------------------------
for i = 1:length(M1)
    tic;
    x1 = M1{i}\b{i};
    Tslash{1,i} = toc;
    x2 = M2{i}\b{i};
    Tslash{2,i} = toc;
    x3 = M3{i}\b{i};
    Tslash{3,i} = toc;
    i=i+1;
end

for i = 1:length(M1);
    Maxit_m1 = length(M1{i});
    tic;
    [x_pcg1,flag,relres{1,i},iter{1,i}] = pcg(M1{i},b{i},1e-10,Maxit_m1);
    Tcg{1,i} = toc;
    tic;
    [x_pcg2,flag2,relres{2,i},iter{2,i}] = pcg(M2{i},b{i},1e-10,Maxit_m1);
    Tcg{2,i} = toc;
    tic;
    [x_pcg3, flag3, relres{3,i},iter{3,i}] = pcg(M3{i},b{i},1e-10,Maxit_m1);
    Tcg{3,i} = toc;
end

%%%1.d for HW4
for i = 1:length(M1)
    hold on;
    d_x{i} = log10(length(M1{i}));
    d_slash{1,i} = log10(Tslash{1,i});
    d_pcg{1,i} = log10(Tcg{1,i});
    d_slash{2,i} = log10(Tslash{2,i});
    d_pcg{2,i} = log10(Tcg{2,i});
    d_slash{3,i} = log10(Tslash{3,i});
    d_pcg{3,i} = log10(Tcg{3,i});
    scatter(d_x{i},d_slash{1,i},'r','x');
    scatter(d_x{i},d_slash{2,i},'r','<');
    scatter(d_x{i},d_slash{3,i},'r','s');
    scatter(d_x{i},d_pcg{1,i},'g','x');
    scatter(d_x{i},d_pcg{2,i},'g','<');
    scatter(d_x{i},d_pcg{3,i},'g', 's');
end

for i = 1:length(M1)
    for j = 1:3
         E{j,i} = log10(relres{i});
         itercomparison{j,i} = 2.*((sqrt(C1{j,i})-
1)./(sqrt(C1{j,i})+1)).^iter{j,i};
    end
end

%1.d
% Backslash's performance is purely based on n, while CG is based on
% condition number
% For small n, backslash is better as it takes the form of c1*n^3 (c1 is
% approx. 2/3) and CG takes the form of c2*iter*n^2, but c2*iter can
% potentially be a very big number.
% The graphs of these two functions will eventually cross over at a certain
% n, after which CG will have the better performance.
% This code confirms that the conjugate gradients has timings proportion to
% the matrix apply (O(n^2)).

%1.e
% M1 = 35 iterations
% M2 = ~100 iterations
% M3 = ~1000 iterations
% For CG, the performance depends of K(A) where for backslash, it depends
% on the size of n.
% The numbers come extremely close to the theoretical estimate as given by
% the formula discussed in class.
%

%-----------------------------------------------
%HW4.2
%Defining functions
f_x = @(x) exp(1).^(-(x.^2));
g_x = @(y) 1./(1+(2*y.^2));

format long;
n_params = [2 4 8 16 32];
interp_f = [ ];
actual_f = [ ];
diff_f = [ ];
interp_g = [ ];
actual_g = [ ];
diff_g = [ ];
```

```
%points that the function will be evaluated at
points = linspace(-1,1,1000);
for i = 1:1000
%points that the function will be evaluated at
    actual_f(i) = f_x(points(i));
    actual_g(i) = g_x(points(i));
end

%F(X) equispaced
figure
for i = 1:5
        out = lagrange_interp(f_x, n_params(i), points);

        %puts into array
        interp_f = out;

        %gets difference
        diff_f = (abs(interp_f - actual_f));

        plot(points, log10(diff_f));
        hold on;
        title('Equispaced F(X)')
end

%Cheb F(X)
figure
for i = 1:5
        out = lagrange_interp(f_x, n_params(i), points,
chebpts(n_params(i)));

        %puts into array
        interp_f = out;

        %gets difference
        diff_f = (abs(interp_f - actual_f));

        plot(points, log10(diff_f));
        hold on;
        title('Cheb F(X)')
end

% G(x)Equipsaced

figure
for i = 1:5
        out = lagrange_interp(g_x, n_params(i), points);

        %puts into array
        interp_g = out;

        %gets difference
        diff_g = (abs(interp_g - actual_g));
        plot(points, log10(diff_g));
        hold on;
        title('Equispaced G(X)')
end

figure
for i = 1:5
        out = lagrange_interp(g_x, n_params(i), points,
chebpts(n_params(i)));

        %puts into array
        interp_g = out;

        %gets difference
        diff_g = (abs(interp_g - actual_g));

        plot(points, log10(diff_g));
        hold on;
        title('Cheb G(X)')
end


%2c. I would use the cheb points to approximate both functions because they
%minimze the error of the edges of the function domains (near -1 and 1).
%This is because the Cheb. Points are more densely packed in those areas,
%giving a much better approximation and keeping the error minimized in
%those problematic regions.


% ---------------------------------------------------------------
% HW 4.3


%function definition and points
h_x = @(x) sqrt(1-x^2);
x_i = linspace(-1,1,17);

%evaluate at equspaced points
for i = 1:17
    y(i) = h_x(x_i(i));
end

%points to approximate with
points = linspace(-1,1,1000);

%interpolations
cubic = interp1(x_i, y , points, 'CUBIC');
linear = interp1(x_i, y , points, 'LINEAR');

%true values
for i = 1:1000
    actual(i) = h_x(points(i));
end
```

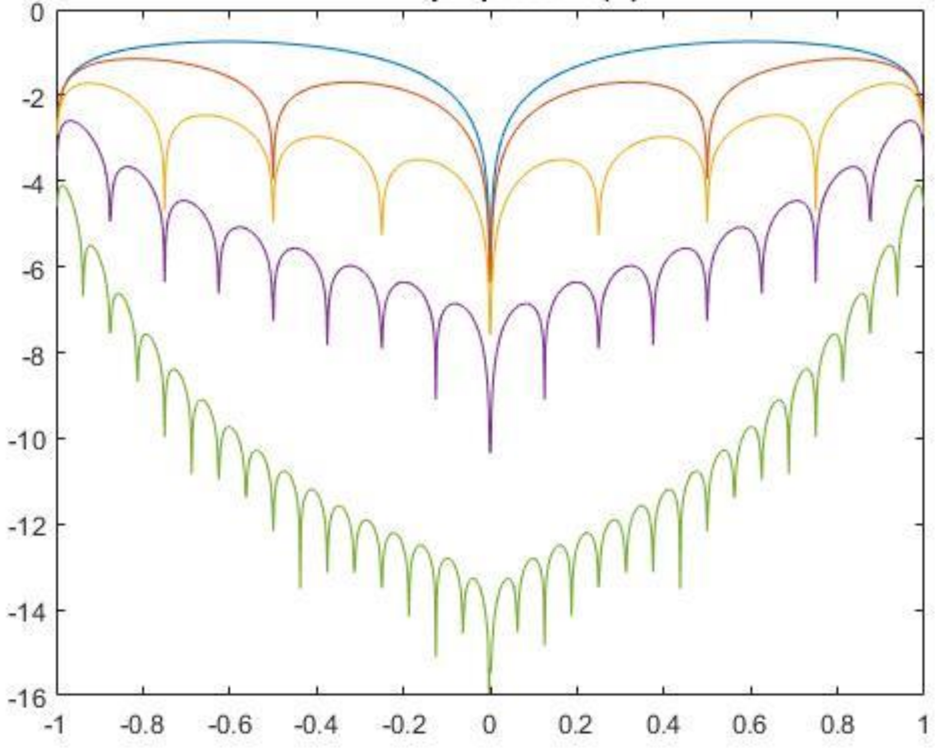
```
%plots
plot(linspace(-1,1,1000), log10(actual-cubic),'b');
hold on;
plot(linspace(-1,1,1000), log10(actual-linear), 'r');
title('Cubic in Blue, Linear In Red');

% ANSWER TO 3a:
% -----------------
% From these plots, you can see for both the error blows up at the edges,
% but the cubic interpolate (in blue) is a much better interpolant because
% it has more data to interpolate with, making it a better approximation.
% It clearly has a lower error for almost all points when compared with the
% true values.

cubpp = interp1(x_i, y, 'CUBIC', 'pp');
linpp = interp1(x_i, y, 'LINEAR', 'pp');

% The linear approximation to h'(x)
h_prime_lin = linpp.coefs(:,1);

%Cubic approximation to h'(x)

h_prime_cub = zeros(17,3);
for i = 1:16
   h_prime_cub(i,1) = 3*cubpp.coefs(i,1);
   h_prime_cub(i,2) = 2*cubpp.coefs(i,2);
   h_prime_cub(i,3) = cubpp.coefs(i,3);
end
```

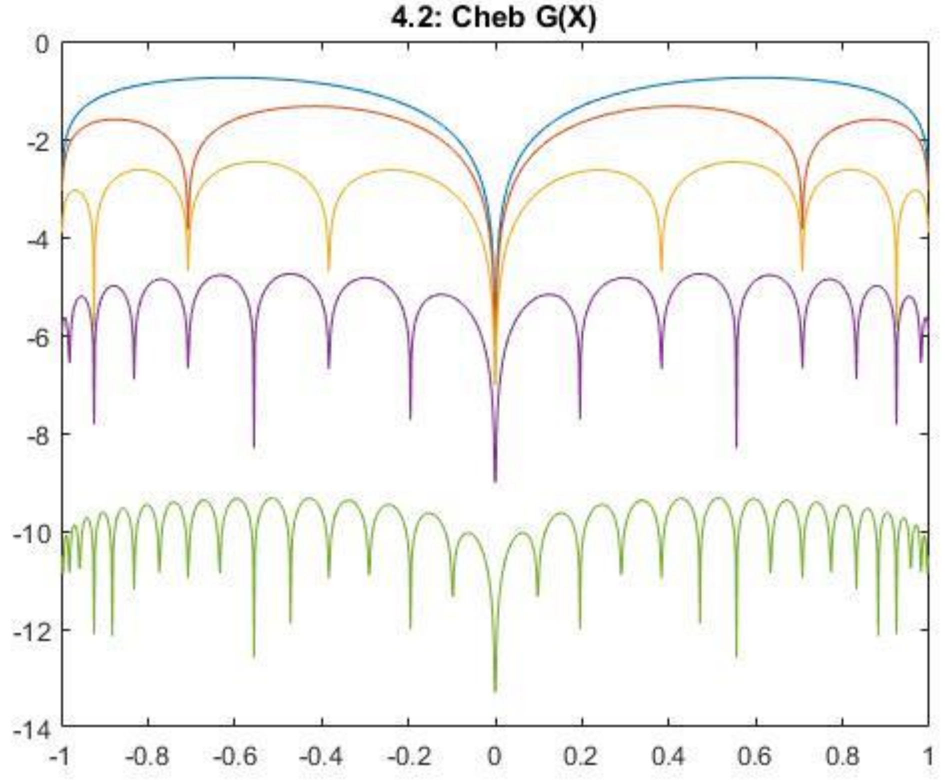


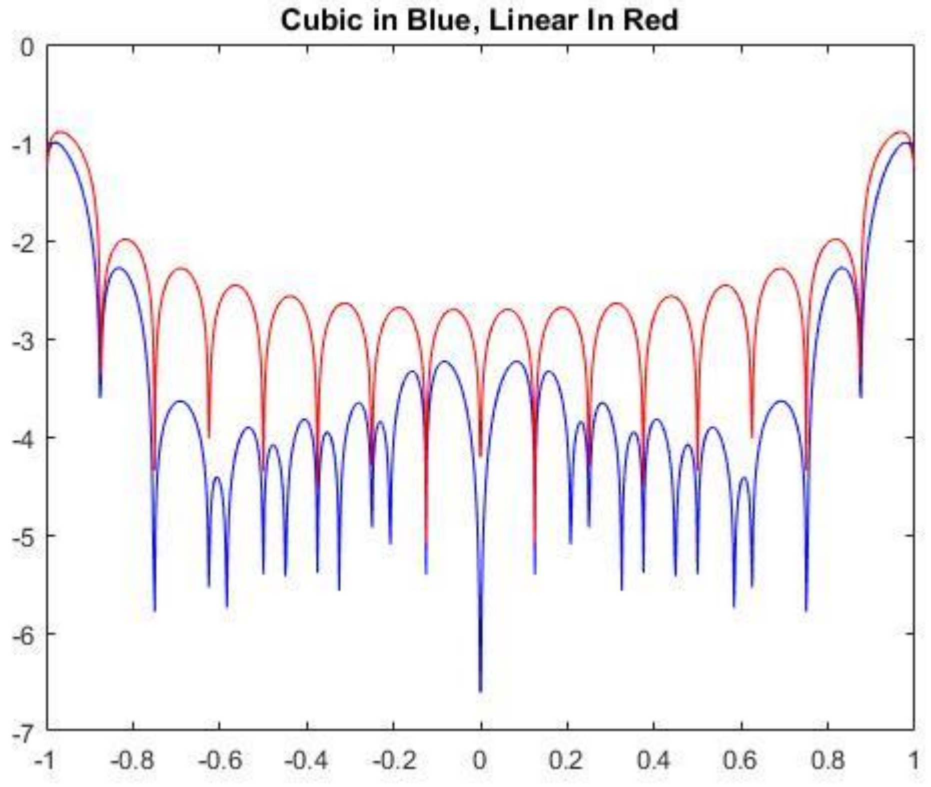
4.2: Equispaced F(X)



4.2: Equispaced G(X)




4.2: Cheb G(X)
0
-2
-4
-6
-8
-10
-12
-14
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Cubic in Blue, Linear In Red
0
-1
-2
-3
-4
-5
-6
-7
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1