Stephanie Klumpe
Math 6680
HW 2

5.1. Modify Program 9 to compute and plot the maximum error over $[-1, 1]$ for equispaced and Chebyshev interpolation on a log scale as a function of N. What asymptotic divergence and convergence constants do you observe for these two cases? (Confine your attention to small enough values of N that rounding errors are not dominant.) Now, based on the potential theory of this chapter, determine exactly what these geometric constants should be. How closely do your numerical experiments match the theoretical answers? [1]

Here, we are choosing N to be multiples of 4 between 4 and 28.

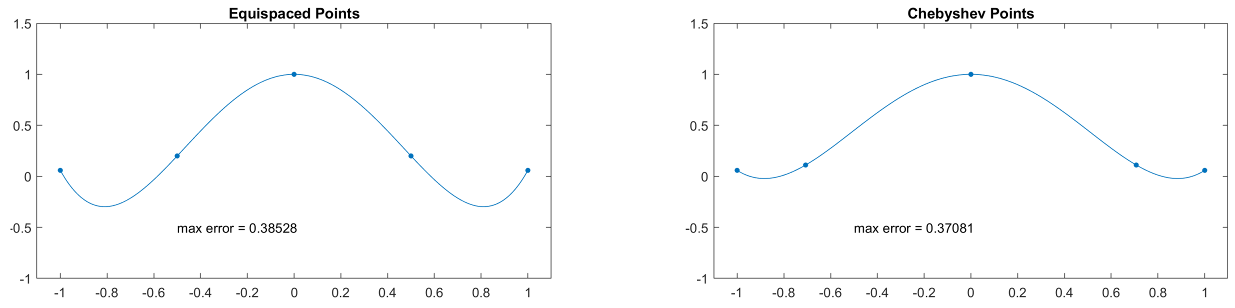


Figure 1: $N = 4$

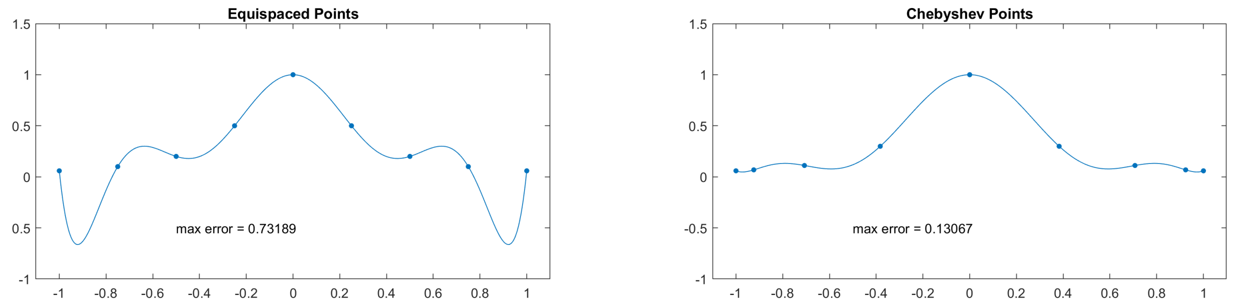


Figure 2: $N = 8$

[1] I accidentally did this problem and didn't want to delete all of the work, so I kept it in.

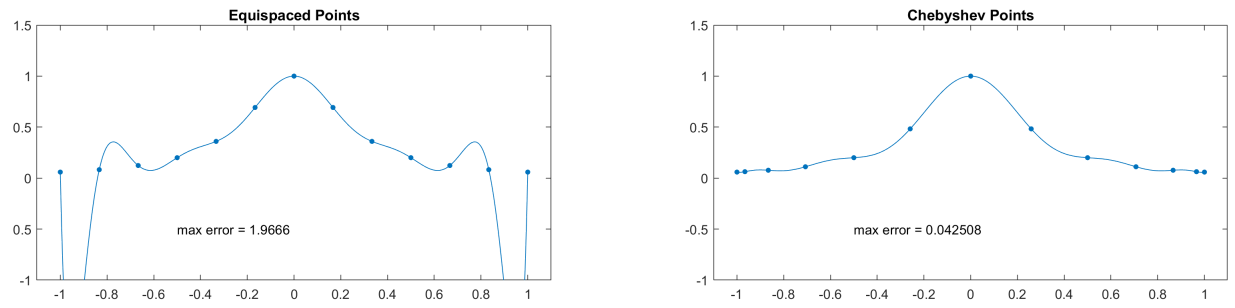


Figure 3: $N = 12$

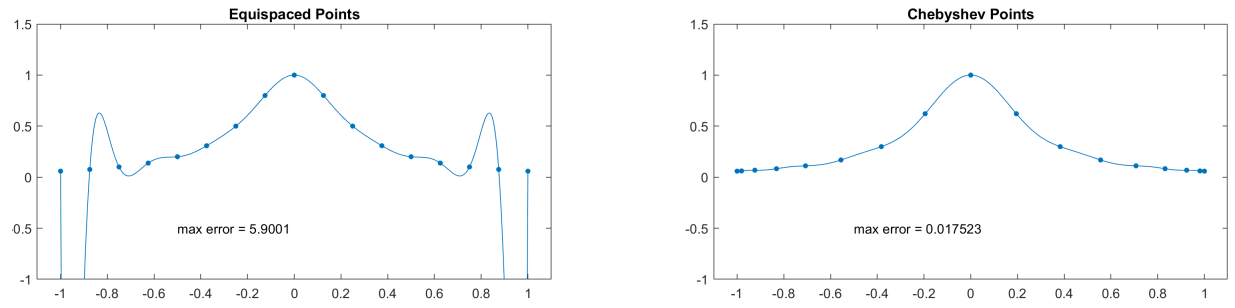


Figure 4: $N = 16$

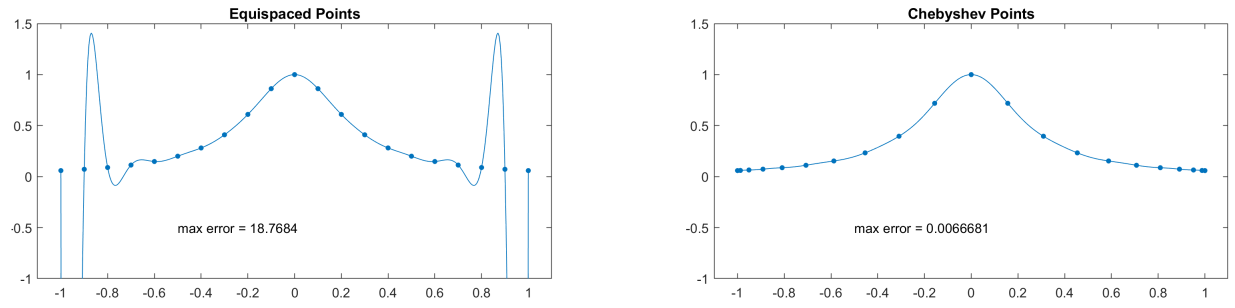


Figure 5: $N = 20$

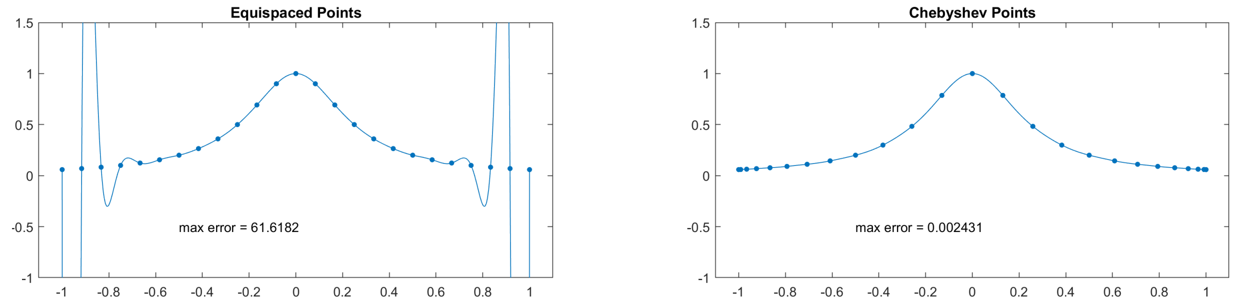


Figure 6: $N = 24$

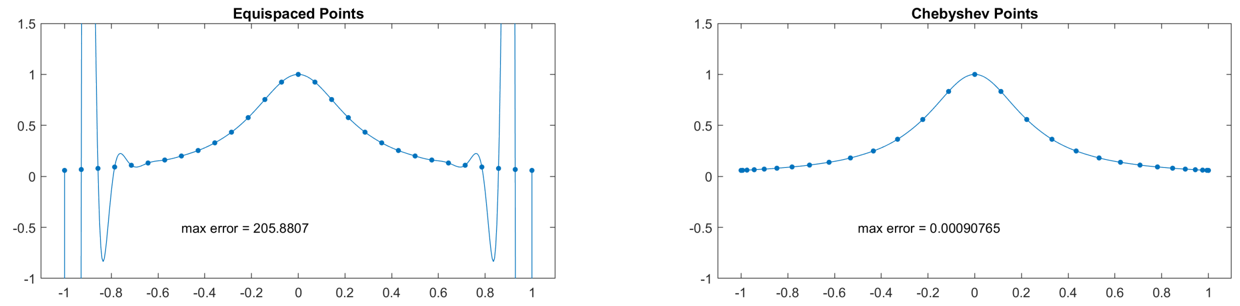
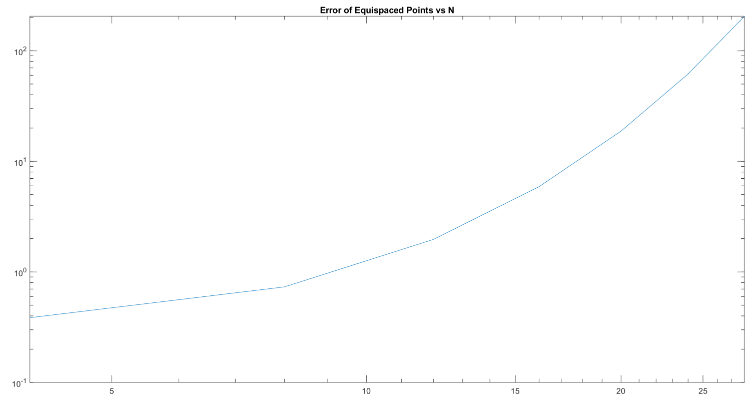
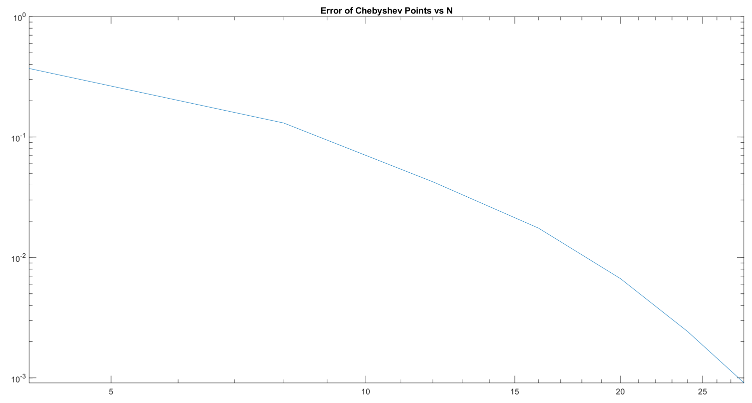


Figure 7: $N = 28$

Now, looking at the two plots of the error vs N, we see that for the equidistal points, there is no convergence. Rather, there is divergence towards infinity and this divergence happens exponentialy. Similarly, for the Chebyshev nodes, there is an exponential plotting, however, this is exponentially convergent.





```
% Stephanie Klumpe
% Problem 5.1 edited code

clear
close all
clc

error = zeros(2,7);        %% Set up an error matrix to hold the max errors
counter = 1;                      %% Counter used for indexing
xx = -1:.005:1; clf               %% x space
v=1; %% Used to split the above error matrix into the two different errors
Nval=[4 8 12 16 20 24 28];        %% N-value vector

 for N = Nval

  for i = 1:2

    if i==1, s = 'Equispaced Points'; x = -1 + 2*(0:N)/N; end
    if i==2, s = 'Chebyshev Points';  x = cos(pi*(0:N)/N); end
    subplot(2,2,i)
    u = 1./(1+16*x.^2);
    uu = 1./(1+16*xx.^2);
    p = polyfit(x,u,N);               % interpolation
    pp = polyval(p,xx);               % evaluation of interpolant
    plot(x,u,'.','markersize',13)
    hold off
    line(xx,pp)
    axis([-1.1 1.1 -1 1.5]), title(s)
    error(i,counter) = norm(uu-pp,inf);
    text(-.5,-.5,['max error = ' num2str(error(i,counter))])

  end

  err1=error(1:v,:);              %% Max error vector of equispaced
  err2=error(v+1:end,:);          %% Max error vector for the Cheb nodes
  figure          %% Opens new figure for each of the different N-values
  counter=counter+1;

 end

loglog(Nval,err1)
title('Error of Equispaced Points vs N')
figure
loglog(Nval,err2)
title('Error of Chebyshev Points vs N')
```

5.3. Let $E_N = \inf_p ||p(x) - e^x||_\infty$, where $||f||_\infty = \sup_{x\in[-1,1]} |f(x)|$, denote the error in degree N minimax polynomial approximation to e^x on $[-1, 1]$. (a) One candidate approximation $p(x)$ would be the Taylor series truncated at degree N. From this approximation, derive the bound $E_N < ((N + 2)/(N + 1))/(N + 1)!$ for $N \geq 0$. (b) In fact, the truncated Taylor series falls short of optimal by a factor of about 2^N, for it is known (see equation (6.75) of [Mei67]) that as $N \to \infty$, $E_N \sim 2^{-N}/(N+1)!$. Modify Program 9 to produce a plot showing this asymptotic formula, the upper bound of (a), the error when $p(x)$ is obtained from interpolation in Chebyshev points, and the same for equispaced points, all as a function of N for $N = 1, 2, 3, \ldots, 12$. Comment on the results.

We know that the Taylor series expansion of e^x about $x = 0$ is given by $\sum \frac{x^n}{n!}$. So, the truncated series is $1 + x + \frac{x^2}{2} + \ldots + \frac{x^N}{N!}$. Hence, $|p(x) - e^x| = |R_N(x)|$ where $R_N(x)$ is the remainder of the exact Taylor expansion of e^x. We also know that such an expansion exists as e^x is an analytic function. So,

$$E_N = \sup_{x\in[-1,1]} |\sum_{k=N+1}^{\infty} \frac{x^k}{k!}|$$

$$= \sum_{k=1}^{\infty} \frac{1}{(N+k)!}$$

$$= \frac{1}{(N+1)!}(1 + \frac{1}{N+2} + \frac{1}{(N+2)(N+3)} + \ldots)$$

$$< \frac{1}{(N+1)!}(\sum_{k=0}^{\infty} \frac{1}{(N+2)^k})$$

Here, we have a geometric series, so we get the equality

$$= \frac{1}{(N+1)!}\frac{1}{1 - \frac{1}{N+2}}$$

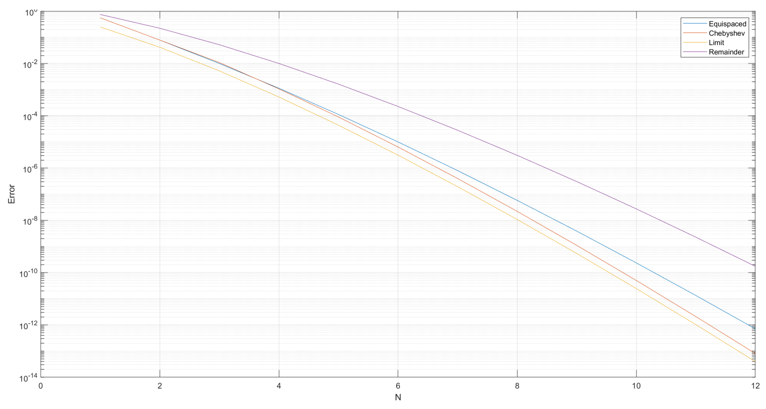
$$= \frac{1}{(N+1)!}\frac{1}{\frac{N+2}{N+2} - \frac{1}{N-2}}$$

$$= \frac{N+2}{N+1}\frac{1}{(N+1)!}$$

as desired.

Now, plotting the error bound we found, the known error bound, and the error of the approximations of e^x yields us graph:

```
% Stephanie Klumpe
% Problem 5.3 edited code

```



```matlab
clear
close all
clc

Nspace = 1:12;
xx = -1.01:.005:1.01; clf

limerr = @(N) 1./(factorial(N+1).*2.^N);        % Define the knwon err and
remerr = @(N) ((N+2)./(N+1)) ./factorial(N+1); % the remainder err as
polyerr = zeros(2, length(Nspace));             % fxns and the polyerr as
                                                % 2x12 matrix for both
                                                % approximations
for n = 1:length(Nspace)
   N=Nspace(n);
  for i = 1:2
    if i==1, s = 'Equispaced Points'; x = -1 + 2*(0:N)/N; end
    if i==2, s = 'Chebyshev Points';  x = cos(pi*(0:N)/N); end

    u = exp(x);
    uu = exp(xx);
    p = polyfit(x,u,N);                % interpolation
    pp = polyval(p,xx);                % evaluation of interpolant
    polyerr(i,n) = norm(uu-pp,inf);    % fill in polyerr matrix for the two
                                       % types and the error values


  end
end

hold on;
grid on;


title('foobar')
semilogy(Nspace,remerr(Nspace));       % semilog plots of N vs the errors
semilogy(Nspace,polyerr(1,:));
semilogy(Nspace,limerr(Nspace));
semilogy(Nspace,polyerr(2,:));

print('5_3.png', '-dpng')
```

6.1. If $x_0, x_1, \ldots, x_N \in \mathbb{R}$ are distinct, then the cardinal function $p_j(x)$ defined by

$$p_j(x) = \frac{1}{a_j} \prod_{k=0,\neq j}^{N} (x - x_k), \quad a_j = \prod_{k=0,\neq j}^{N} (x_j - x_k)$$

is the unique polynomial interpolant of degree N to the values 1 at x_j and 0 at x_k, $k \neq j$. Take the logarithm and differentiate to obtain

$$p_j'(x) = p_j(x) \sum_{k=0,\neq j}^{N} (x - x_k)^{-1},$$

and derive the formulas

$$D_{ij} = \frac{1}{a_j} \prod_{k=0,\neq j}^{N} (x_i - x_k) = \frac{a_i}{a_j(x_i - x_j)} \quad (i \neq j)$$

and

$$D_{jj} = \sum_{k=0,\neq j}^{N} (x_j - x_k)^{-1}$$

Letting $p_j(x)$ and a_j be as given above, we take the natural logarithm of both sides. This gives us

$$\ln(p_j(x)) = \ln(\prod_{k=0,\neq j}^{N} \frac{(x - x_k)}{(x_j - x_k)}) = \sum_{k=0,\neq j}^{N} \ln(\frac{x - x_k}{x_j - x_k})$$

$$= \sum_{k=0,\neq j}^{N} \ln(x - x_k) - \sum_{k=0,\neq j}^{N} \ln(x_j - x_k)$$

Now differentiating both sides yields

$$\frac{p_j'(x)}{p_j(x)} = \sum_{k=0,\neq j}^{N} \frac{1}{(x - x_k)} - 0$$

which gives us the desired equation

$$p_j'(x) = p_j(x) \sum_{k=0,\neq j}^{N} (x - x_k)^{-1}$$

6.8. Let D_N be the usual Chebyshev differentiation matrix. Show that the power $(D_N)^{N+1}$ is identically equal to zero. Now try it on the computer for $N = 5$ and 20 and report the computed 2-norms $||(D_5)^6||_2$ and $||(D_{20})^{21}||_2$. Discuss.

We know that the Chebyshev differentiation matrix, D_N, is a result of differentiating the N^{th} degree Chebyshev polynomial. By the nature of the polynomials, we have that they are given by $\sum_{i=1}^{N} c_i x^i$ where c_i are the coefficients found by interpolating through the Chebyshev nodes. So, taking the derivative would give us an $(N-1)th$ degree polynomial. To find the second order differentiation matrix, we apply D_N to D_N to get D_N^2. That is, we take the second derivative of the Chebyshev polynomial. Continuing the process, we see that

$$(\sum_{i=1}^{N} c_i x^i)^{(N)} = c_N$$

Hence,

$$(\sum_{i=1}^{N} c_i x^i)^{(N+1)} = (c_N)' = 0$$

So, applying D_N to $(D_N)^N$ gives us $(D_N)^{N+1} = 0$ as desired.

Now, with the following code, we see that we do get a number very close to zero relative to Matlab's precision for $N = 5$, however, we get an astronomically large number for $N = 20$. This is due to the roundinjg and floating number capapbilities for the program. So while theoretically, the matrix to the 21st power would yield 0, that doesn't quite work in this particular instance of numerical computing.

```matlab
% Stephanie Klumpe
% Problem 6.8
% (Cheb matrix_N)^(N+1)=0 code

clear
close all
clc

[Df,x] = cheb(5);                    % Define the cheb matrices
[Dt,y] = cheb(20);

D6 = (Df)^6;                         % Raise them to powers
D21 = (Dt)^(20);

D5err = norm(D6,2)                   % Compute the L2 norm of the powers
% 6.0832e-11
D20err = norm(D21,2)
% 1.2906e+24
```

7.1. Modify Program 13 so that instead of polyval and polyfit, it uses the more stable formula of barycentric interpolation

$$p(x) = \sum_{j=0}^{N} \frac{a_j^{-1} u_j}{x - x_j} \Big/ \sum_{j=0}^{N} \frac{a_j^{-1}}{x - x_j}$$

where $\{a_j\}$ are defined by (6.7). Experiment with various interpolation problems (such as that of Exercise 5.1) and find evidence of the enhanced stability of this method.

Using the barycentric interpolation, we have the following code. We test it with three different conditions for our ODE and get these three graphs. We can see that this interpolation method works well and is quite robust.

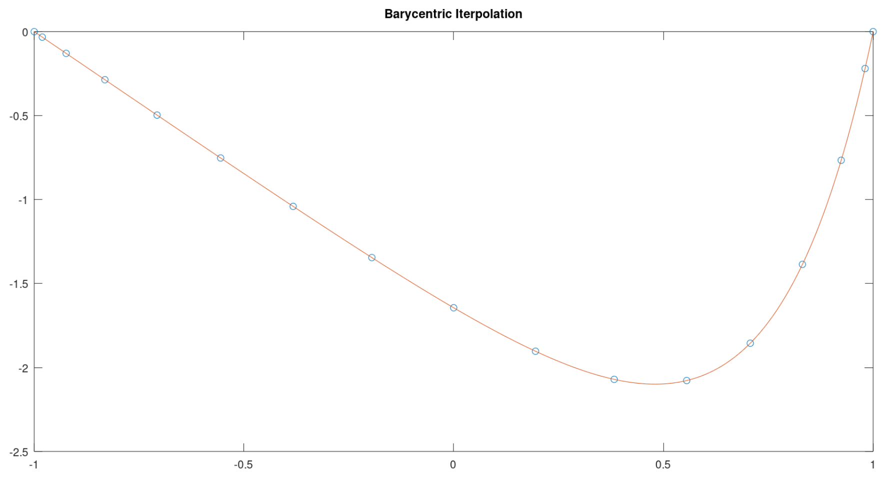


Figure 8: e^{4x}

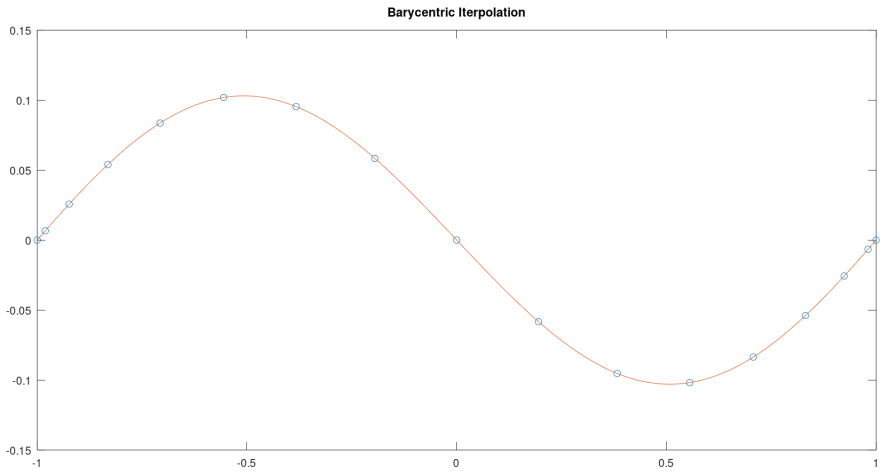


Figure 9: $\sin(x)$

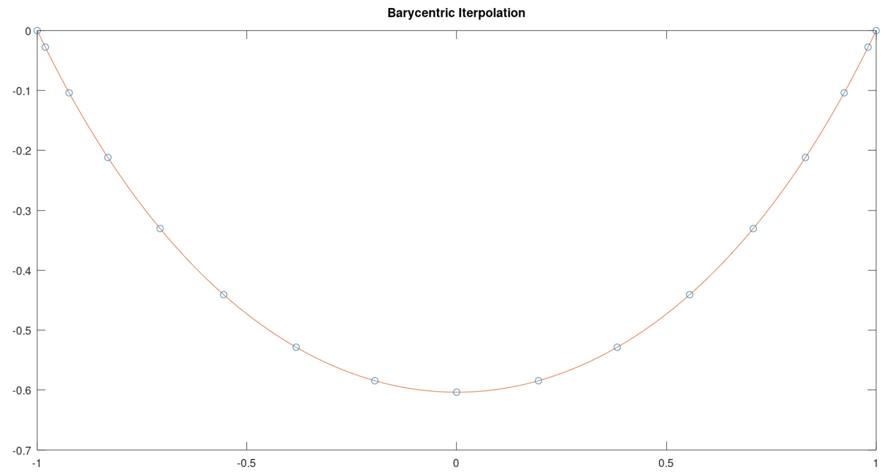
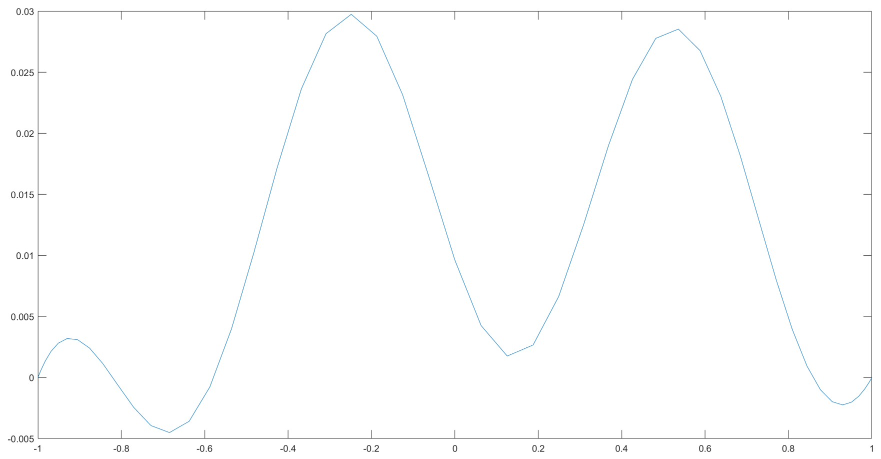


Figure 10: e^{x^2}

```matlab
% Stpehnaie Klumpe
% Problem 7.1

clear
close all
clc

N = 16;
[D,x] = cheb(N);             % Define the Cheb matrix for both first and
D2 = D^2;                    % second order
D2 = D2(2:end-1, 2:end-1);

f = exp(4*x(2:N));
%f = sin(3*x(2:N));          % Different functions to test with
%f = exp((x(2:N)).^2);
u = D2\f;
u = [0; u; 0];               % Define our ODE and plot our Cheb nodes on our
plot(x,u,'o')                % solution
xx = -1:0.01:1;

ainv = ones(N+1,1);
denom = zeros(length(xx),1);     % Define our a^-1, numerator, and ...
    denominator
num = zeros(length(xx),1);       % vectors

for i=1:N+1
    for k=1:N+1
        if k ≠ i
            ainv(i) = ainv(i)*(x(i) - x(k));  % Fill out the a^-1 vector
        end
    end
end


for i = 1:N+1

    denom = denom + ainv(i)^(-1)./(xx.' - x(i));     % Fill out the num and
    num = num + ainv(i)^(-1)*u(i)./(xx.' - x(i));    % denom vectors


end

num = num(2:end-1);          % Take off the ends of the num vector and ...
    replace
num = [0;num;0];           % them with the boundary conditions

u = num./denom;
hold on
plot(xx, u, '-')
title('Barycentric Iterpolation')
```

7.2. Solve the boundary value problem $u_{xx} + 4u_x + e^x u = \sin(8x)$ numerically on $[-1, 1]$ with boundary conditions $u(\pm 1) = 0$. To ten digits of accuracy, what is $u(0)$?

For this problem, we have both a first order and a second order term in our equation, so we will use a first and second order Chebyshev matrix. After running through the code below, we have that $u(0) \approx 0.009597857230$ and we also get the following graph with $N = 50$.



```matlab
% Stephanie Klumpe
% Problem 7.2

clear
close all
clc

N = 50;
[D,x] = cheb(N);
D2 = D^2;                                   % Here, we define our Cheb points and
D2 = D2(2:end-1,2:end-1);                   % our 1st and 2nd order Cheb diff
D1 = D(2:end-1,2:end-1);                    % matrices

evec = exp(x);
evec(1) = [];
evec(end) = [];
f = sin(8*x);                               % We set up our RHS and our e^x vectors
f(1) = [];
f(end) = [];
ematr = diag(evec);

U = D2+4*D1+ematr;
u = U\f;                                    % Solving for our function u(x)
u = [0;u;0];

fprintf('u(0)=%0.12f',u(26));
plot(x,u)
```

8.3. Modify Program 12 (p. 57) to make use of chebfft instead of cheb. The results should be the same as in Output 12, except for rounding errors. Are the effects of rounding errors smaller or larger than before?

Below are the graph outputs of program 12, the first of which is using cheb and the second using chebfft:

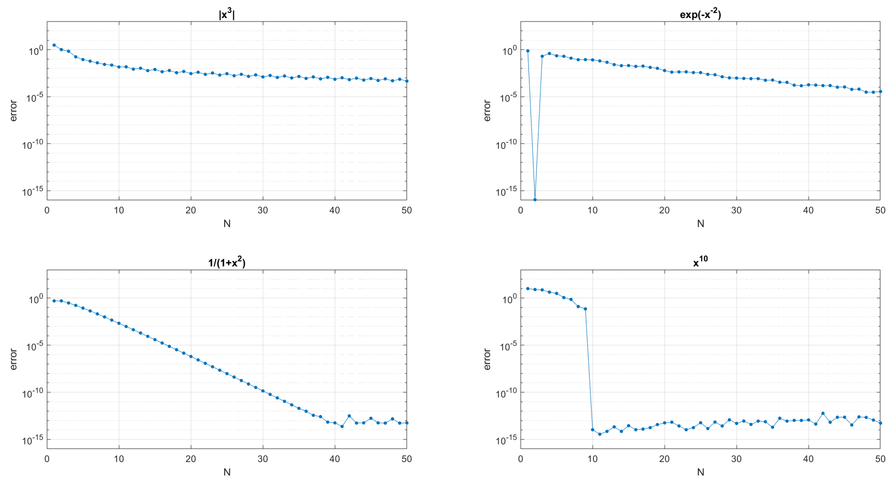


Figure 11: cheb

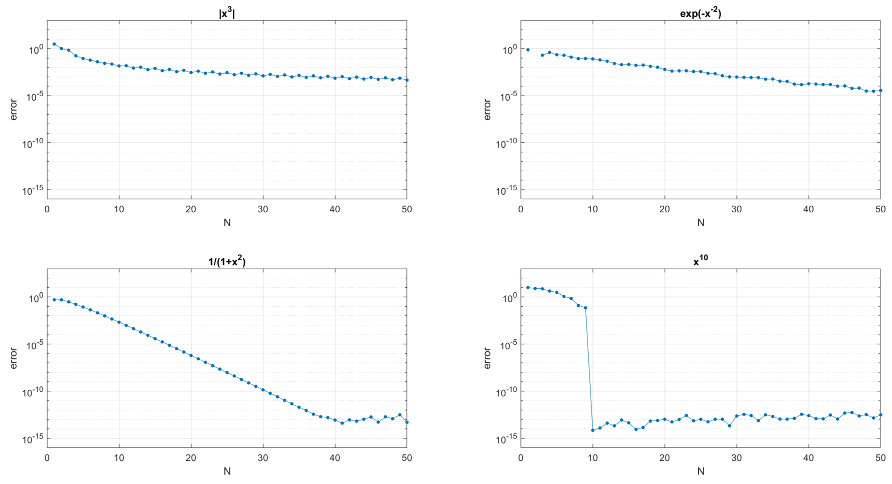


Figure 12: chebfft

We see that the round off error is the only difference between the two graphs. It can also be seen that the rounding error for the chebfft function is slightly less than that of the regular cheb function.

```
% Stephanie Klumpe
% Problem 8.3 edited code

clear
close all
clc

% Compute derivatives for various values of N:
  Nmax = 50; E = zeros(3,Nmax);
  for N = 1:Nmax
    [D,x] = cheb(N);
    v = abs(x).^3; vprime = 3*x.*abs(x);      % 3rd deriv in BV
    E(1,N) = norm(chebfft(v)-vprime,inf);
    v = exp(-x.^(-2)); vprime = 2.*v./x.^3;  % C-infinity
    E(2,N) = norm(chebfft(v)-vprime,inf);
    v = 1./(1+x.^2); vprime = -2*x.*v.^2;     % analytic in [-1,1]
    E(3,N) = norm(chebfft(v)-vprime,inf);
    v = x.^10; vprime = 10*x.^9;              % polynomial
    E(4,N) = norm(chebfft(v)-vprime,inf);
  end                                   %% We replaced the D*v in the norm
                                         % computation with chebfft(v)

% Plot results:
  titles = {'|x^3|','exp(-x^{-2})','1/(1+x^2)','x^{10}'}; clf
  for iplot = 1:4
    subplot(2,2,iplot)
    semilogy(1:Nmax,E(iplot,:),'.','markersize',12)
    line(1:Nmax,E(iplot,:))
    axis([0 Nmax 1e-16 1e3]), grid on
    set(gca,'xtick',0:10:Nmax,'ytick',(10).^(-15:5:0))
    xlabel N, ylabel error, title(titles(iplot))
  end
```

8.5. Find a way to modify your program of Exercise 8.4, as in Exercise 3.8 but now for a second-order problem, to make use of matrix exponentials rather than time discretization. What effect does this have on the computation time?

Using the code that we worked through in class, we have the following code. From the code, we have that the elapsed time is about 0.0065 seconds. For the original code, adding in the tic/toc function, we get that the average elapsed time is about 0.5 seconds.

```
% Stephanie Klumpe
% Problem 8.5

clear
close all
clc

tic

% Grid and initial data:
  N = 24;
  x = cos(pi*(0:N)/N);
  y = x';
  dt = 6/N^2;
  [D,x] = cheb(N);
  D2 = D^2;

  D2 = D2(2:N,2:N);
  I = eye(N-1);
  L = kron(I,D2) + kron(D2,I);

  [xx,yy] = meshgrid(x,y);
  plotgap = round((1/3)/dt);
  dt = (1/3)/plotgap;
  vv = exp(-40*((xx-.4).^2 + yy.^2));
  vv=vv(2:N,2:N);
  vvold = vv;

% Time-stepping by leap frog formula:
  for n = 0:3*plotgap
    t = n*dt;

    U=vv(:); RHS=L*U; RHS=reshape(RHS,N-1,N-1);
    vvnew = 2*vv - vvold + dt^2*RHS;
    vvold = vv;
  end

  toc
% Elapsed time was about 0.0065 seconds
```

8.6. Write a code chebfft2 for second-order differentiation by the FFT, and show by examples that it matches the results obtained by matrices, apart from rounding errors. [2]

Here, we used the functions $\sin(x), e^x$, and e^{-x^2} as examples to demonstrate the chebfft2 function. In order to write this function, all we needed to do is repeat the FFT, inverse FFT, and deriving the algebraic polynomial steps just on the derivative vector. So, we get the following graphs.

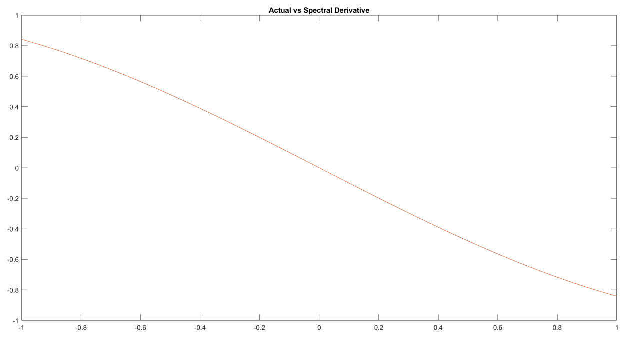


Figure 13: $\sin(x)$

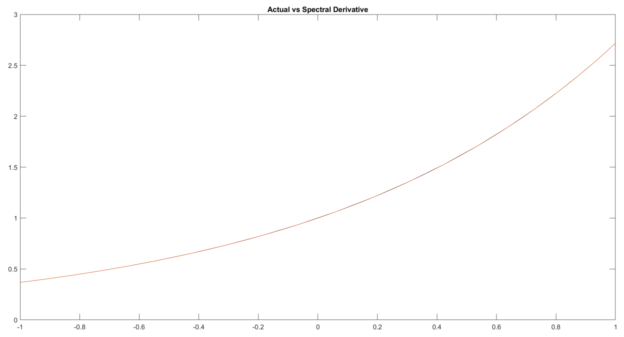


Figure 14: e^x

[2]Again, this was not an assigned problem, however I did quite a bit of work for it and didn't want to delete it.

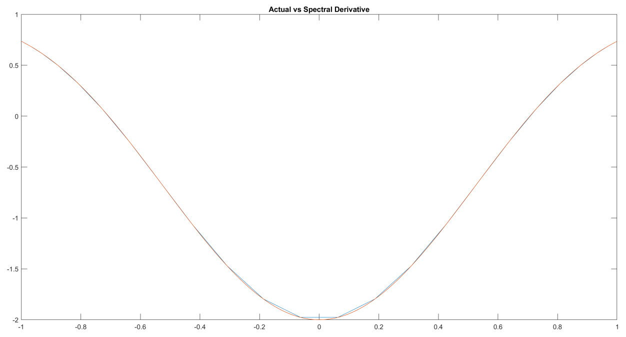


Figure 15: e^{-x^2}

```
% Stephanie Klumpe
% Problem 8.5
% chebfft2

clear
close all
clc

t=-1:0.001:1;
[D,x]=cheb(25);                         % Compute Cheb points
v=sin(x);                               % Choose a function
%v=exp(x);
%v=exp(-x.^2);
d2v=-sin(t);                            % Define the second derivative
%d2v=exp(t);
%d2v=exp(-t.^2).*(4.*t.^2-2);
dw=chebfft2(v);                         % Call the new function
plot(x,dw)                              % Plot the actual deriv and the
hold on                                 % spectral deriv to compare
plot(t,d2v)
title('Actual vs Spectral Derivative')



  function dw = chebfft2(v)
  N = length(v)-1; if N==0, w=0; return, end
  x = cos((0:N)'*pi/N);
  ii = 0:N-1;
  v = v(:); V = [v; flipud(v(2:N))];      % transform x -> theta
  U = real(fft(V));
  W = real(ifft(1i*[ii 0 1-N:-1]'.*U));
  w = zeros(N+1,1);
  w(2:N)  = -W(2:N)./sqrt(1-x(2:N).^2);    % transform theta -> x
  w(1)  = sum(ii'.^2.*U(ii+1))/N + .5*N*U(N+1);
  w(N+1)  = sum((-1).^(ii+1)'.*ii'.^2.*U(ii+1))/N + ...
              .5*(-1)^(N+1)*N*U(N+1);
  dV=[w; flipud(w(2:N))];                 % Repeat the above process
  dU=real(fft(dV));                       % by defininig a vector to be
  dW = real(ifft(1i*[ii 0 1-N:-1]'.*dU)); % the derivative vector. Then
  dw = zeros(N+1,1);                      % compute the spectral deriv.
  dw(2:N) = -dW(2:N)./sqrt(1-x(2:N).^2);
  dw(1) = sum(ii'.^2.*dU(ii+1))/N + .5*N*U(N+1);
  dw(N+1) = sum((-1).^(ii+1)'.*ii'.^2.*dU(ii+1))/N + ...
              .5*(-1)^(N+1)*N*dU(N+1);
  end

% From this, we can say that if we repeate the same idea as above
% n times, we can get the nth derivative using Cheb spectral diff via FFT.
```