Computational Methods

Francisco Castillo
Homework 4

March 25, 2018

Problem 1

Derive Newtons iteration for this function. Show that the iteration is well-defined so long as $x_k \neq 1$ and that the convergence rate is expected to be similar to that of the bisection method (and certainly not quadratic).

Newton's method for finding successively better approximations to the roots of a real-valued function is the following,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

Given the function

$$f(x) = (x-1)^2 e^x,$$

and its derivative

$$f'(x) = (x^2-1)e^x,$$

we can obtain the iteration for this problem

$$x_{n+1} = x_n - \frac{(x_n-1)^2 e^{x_n}}{(x_n^2-1)e^{x_n}} = x_n - \frac{(x_n-1)}{(x_n+1)}.$$

Note that this is not defined for $x_n = -1$. Clearly the root of the function is $x^* = 1$. To find the rate of convergence let $\epsilon_n = x_n - x^* = x_n - 1$, and $g(x_n) = \frac{f(x_n)}{f'(x_n)}$. Then, rewriting the first equation in this terms we get

$$\epsilon_{n+1} = \epsilon_n - g(x_n).$$

Further,

$$g(x_n) = g(\epsilon_n + 1) = g(1) + g'(1)\epsilon_n + \frac{\epsilon_n^2}{2} g''(\xi),$$

by Taylor expansion. The values of g and g' are known,

$$g(1) = \frac{f(1)}{f'(1)} = 0, \qquad g'(x) = \frac{2}{(x+1)^2} \Rightarrow g'(1) = \frac{1}{2}.$$

Hence,

$$g(x_n) = \frac{1}{2}\epsilon_n + \frac{\epsilon_n^2}{2} g''(\xi),$$

and

$$\lim_{\epsilon \to 0} \frac{\epsilon_{n+1}}{\epsilon_n} = \frac{1}{2}.$$

Note that the convergence rate is linear and equal to the one of the bisection method. It differes from the quadratic convergence rate talked in class because this function and its derivative have the same root $x^* = 1$. For the convergence rate to be quadratic, the root of the function should not be a root of the derivative.

Implement Newtons method and observe its performance starting from $x_0 = 2$.

After implementing the method in *Matlab* the root $x^* = 1$ was found in 42 iterations, with a time of the order of 0.009 s. In the following figure we can see the function given and the roots calculated by the Newton's method until convergence.

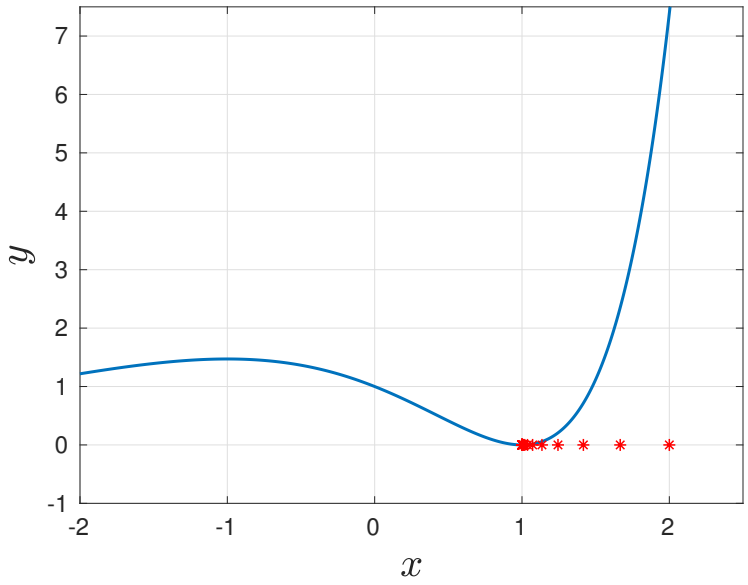


Figure 1: Newton's method performance.

How easy would it be to apply the bisection method? Explain.

For this case the bisection method would not have worked since it relies on a change of sign at both sides of the root, and this is not the case with this function which is nonnegative for all values of x.

Matlab code for this problem

```
%% Problem 1
xx = linspace(-3,3,1e4);
f = @(x) (x - 1).^2.*exp(x);
df = @(x) (x.^2 - 1).*exp(x);

tic
[x,niter,conv]=newton(f,df,2,1e-12,100)
time=toc
% To obtain the roots, modified newton function
[x,niter,conv]=newton_vector(f,df,2,1e-12,100);

figure
plot(xx,f(xx),'linewidth',1.5)
hold on
plot(x,0,'r*')
grid on
axis([-2 2.5 -1 7.5])
set(gca,'fontsize',axisfontsize)
xlabel('$x$','interpreter','latex','fontsize',labelfontsize)
ylabel('$y$','interpreter','latex','fontsize',labelfontsize)
saveas(gcf,'Latex/FIGURES/prob2roots','epsc')
```

Problem 2

Find all the roots of $f(z) = z^8 + 15z^4 16$ on the complex plane.

In order to use the angle to distinguish the different roots, none of them should share the same angle. To find out if this is the case we compute the roots manually, which is easily done by factorizing the polynomial

$$z^8 + 15z^4 16 = (z-1)(z+1)(z^2+1)(z^4+16) = 0.$$

Hence, the roots are

$$x_k = -1, 1, \pm i, \sqrt{2}(1 \pm i), \sqrt{2}(-1 \pm i).$$

and are shown in the next figure. Note that the angle can be computed as the arc tangent of quotient of the imaginary and real part of the complex roots, and its different for every root.

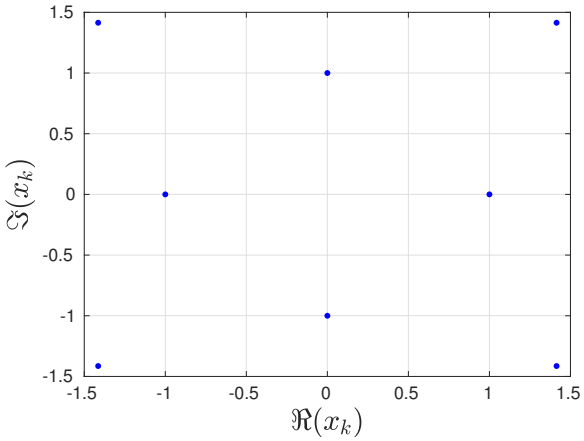
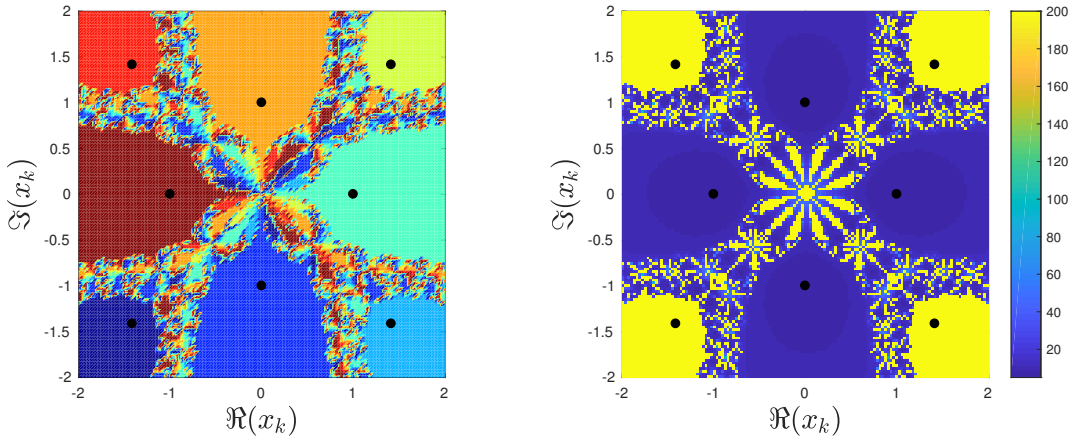


Figure 2: Newton's method performance.

After obtaining the different roots depending on the initial guess we can plot the angle of the Newton's method solution and obtain the following fractals.



(a) Angle fractal

(b) Iterations fractal

Figure 3: Fractals obtained with Newton's method.

Matlab code for this problem

```
%% Problem 2
n = 200;

[x,y]=meshgrid(linspace(-3,3,n));
z = x+y*1i;
z_roots = [1, -1, -1i, 1i, ...
    sqrt(2)*(1+1i),sqrt(2)*(1-1i),sqrt(2)*(-1+1i),sqrt(2)*(-1-1i)];

% We now compute the roots using Newton's method:
r = nan(size(z)); niter = r; conv = r; % allocate memory
f = @(z) z^8+15*z^4-16;
df = @(z) 8*z^7+60*z^3;
for k = 1:n
    for j = 1:n
        [r(k,j),niter(k,j),conv(k,j)] = newton(f,df,z(k,j),1e-16,200);
    end
end
%%
% Plotting the angle of the Newton's method solution, we obtain a fractal:
close all
figure
pcolor(x, y, round(angle(r),3));
mp = colormap;
colormap jet
shading interp
hold on
plot(z_roots, 'k.','markersize',20); axis equal; axis tight; hold off;
```

```
axis([-2 2 -2 2])
xlabel('$\Re(x_k)$','interpreter','latex','fontsize',labelfontsize)
ylabel('$\Im(x_k)$','interpreter','latex','fontsize',labelfontsize)
saveas(gcf,'Latex/FIGURES/colormap','epsc')

% Plot the number of iterations required in a colormap
figure
pcolor(x, y, niter);
shading flat, hold on
plot(z_roots, 'k.','markersize',20); axis equal; axis tight; hold off;
axis([-2 2 -2 2])
colorbar
xlabel('$\Re(x_k)$','interpreter','latex','fontsize',labelfontsize)
ylabel('$\Im(x_k)$','interpreter','latex','fontsize',labelfontsize)
saveas(gcf,'Latex/FIGURES/colormap2','epsc')

% Plot the roots in the complex plane
figure
plot(real(z_roots),imag(z_roots),'b.','markerSize',12)
grid on
axis([-1.5 1.5 -1.5 1.5])
set(gca,'fontsize',axisfontsize)
xlabel('$\Re(x_k)$','interpreter','latex','fontsize',labelfontsize)
ylabel('$\Im(x_k)$','interpreter','latex','fontsize',labelfontsize)
saveas(gcf,'Latex/FIGURES/prob2roots','epsc')
```

Problem 3

Solve the nonlinear boundary value problem

$$y'' = y' + \cos(y), \quad y(0) = 0, \quad y(\pi) = 1.$$

Use finite differences to discretize the problem and Newtons method to solve the resulting system of equations. Estimate (numerically) the number of discretized points needed to obtain a solution that is accurate to 4 digits at $x = \pi/6$.

We can express the previous differential equation as

$$y'' - y' - \cos(y) = 0,$$

so that its roots will give us the solution. To do so we need to discretize the derivatives using finite differences in order to be able to use Newton's method. Then, the differential equation takes the linear form

$$\begin{aligned} y'' - y_x - cos(y) &= D_2 y - D_1 y - cos(y) \\ &= My - \cos(y) = 0 \ , \end{aligned}$$

where the matrix $M = D_2 - D_1$. The matrices D_1 and D_2 will give us the discrete values of the first and second derivative, respectively, and are obtained using second order centered finite differences. For D_1,

$$y' = \frac{1}{2(\Delta x)} \begin{bmatrix} -1 & 0 & 1 & & \dots & 0 \\ 0 & -1 & 0 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & & -1 & 0 & 1 \end{bmatrix} y + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{2\Delta x} \end{bmatrix} = D_1 y + g_1,$$

where the column vector g_1 contains is found by imposing the boundary conditions. Note that the matrices are calculated in such a way that they give us the derivatives for the interior nodes, since the boundary nodes we know their values thanks to the boundary conditions. For D_2,

$$y'' = \frac{1}{(\Delta x)^2} \begin{bmatrix} 1 & -2 & 1 & & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & & 1 & -2 & 1 \end{bmatrix} y + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \frac{1}{(\Delta x)^2} \end{bmatrix} = D_2 y + g_2 \ ,$$

where the column vector g_2 contains is found by imposing the boundary conditions. We now define

$$\vec{F}(\vec{y}) = My - cos(y) + g_2 - g_1 \ ,$$

where $\vec{y}$ contains the interior nodes. The solution to our differential equations will be the vector y that makes $\vec{F}(\vec{y}) = 0$. To calculate those roots we will use Newton's method. In order to be able to use it, we need to calculate the Jacobian,

$$J_{jk} = \frac{\partial}{y_j} F(y_k).$$

First note that

$$\frac{\partial}{\partial y_j}\sum_{l=1}^{N} M_{kl}y_l = M_{kl}$$

and

$$\frac{\partial}{\partial y_j}\cos(y_k) = \begin{cases} -\sin(y_k) & \text{for } k = j \\ 0 & \text{for } j \neq k \end{cases}$$

Hence, the Jacobian can be coded as

$$J(\vec{F}(\vec{y})) = M + \text{diag}(\sin(y)).$$

We will first run the code in a while loop to find out how many nodes we need to have an L_1 norm of $y^{new}(\pi/6) - y^{old}(\pi/6)$ less than a $tol = 10^{-4}$. Using 31 nodes I obtained an L_1 norm of $7.269 \cdot 10^{-5}$. To be safe, since it is a low number of nodes and the code runs fast, we could run it for double number of elements N. Using then 61 nodes we obtain an L_1 norm of the error of $7.603 \cdot 10^{-6}$. Using 61 nodes we have obtained the solution showed in the next figure

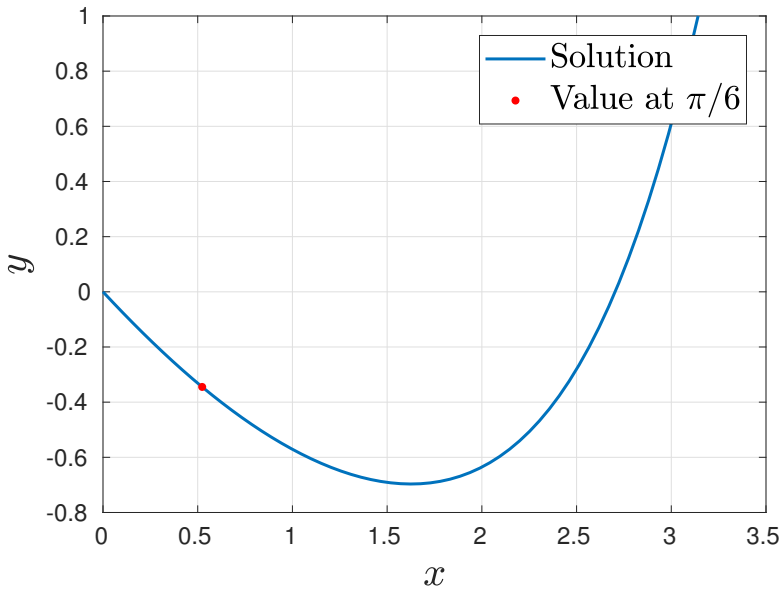


Figure 4: Solution for 61 points

Matlab code for this problem

```
%% Problem 3
close all
clc
format long
% Find an answer with sufficient level of convergence
```

```
tol=1e-4;
i=1;
N=6;
err=1;
yy=zeros(6+1,1);
while err>tol
    i=i+1;
    y_old=yy(N/6);
    N = i*6;
    xi=linspace(0,pi,N+1)';
    [F,J,y0]=FandJ(N);
    [yy,niter] = newton(F,J,y0,1e-14);
    y_new=yy(N/6);
    % Check solution
    err=max(abs(y_new-y_old));
end
% Results
nodes=N+1
err
% Run it for N=60
N=60;
xi=linspace(0,pi,N+1)';
[F,J,y0]=FandJ(N);
[yy,niter] = newton(F,J,y0,1e-14);
y_new=yy(N/6);
% Plot the solution
figure
plot(xi,[0; yy; 1],'linewidth',1.5)
hold on
plot(xi(N/6+1),y_new,'r.','markerSize',12)
grid on
leg1= legend('Solution', 'Value at $\pi/6$');
set(leg1,'interpreter','latex','FontSize',17);
set(gca,'fontsize',axisfontsize)
xlabel('$x$','interpreter','latex','fontsize',labelfontsize)
ylabel('$y$','interpreter','latex','fontsize',labelfontsize)
saveas(gcf,'Latex/FIGURES/prob3sol','epsc')
```

Function to calculate F and J

```
function [F,J,y0]=FandJ(N)
x = linspace(0,pi,N+1)';
h=pi/N;
xi = x(2:N);
y0 = xi/pi;
```

```
D1 = (gallery('tridiag',N+1,-1,0,1))/(2*h);
D1(1,:) = [];
D1(end,:) = [];
D1(:,1) = [];
D1(:,end) = [];

D2 = (gallery('tridiag',N+1,1,-2,1))/(h^2);
D2(1,:) = [];
D2(end,:) = [];
D2(:,1) = [];
D2(:,end) = [];

% g1 = [zeros(N-2,1)];
% g1(N-2) = (1/(2*h));
% g2 = [zeros(N-2,1)];
% g2(N-2) = (1/(h^2));
% e1 = g2 - g1;

M = D2 - D1;
e1 = [zeros(N-1,1)];
e1(N-1) = ((2-h)/(2*h^2));

F = @(y) (M*y)-cos(y)+e1;
J = @(y) M + diag(sin(y));
end
```

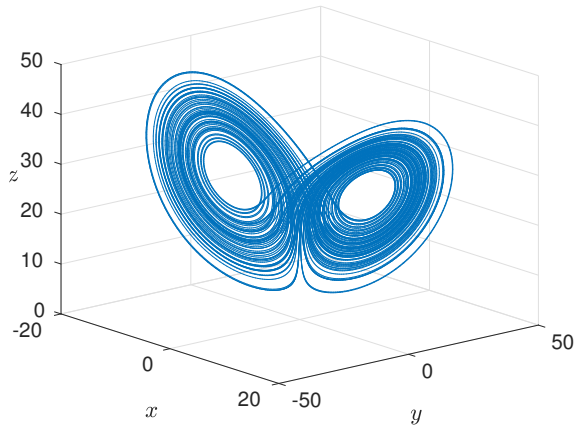
Problem 4

Solve the system of equations

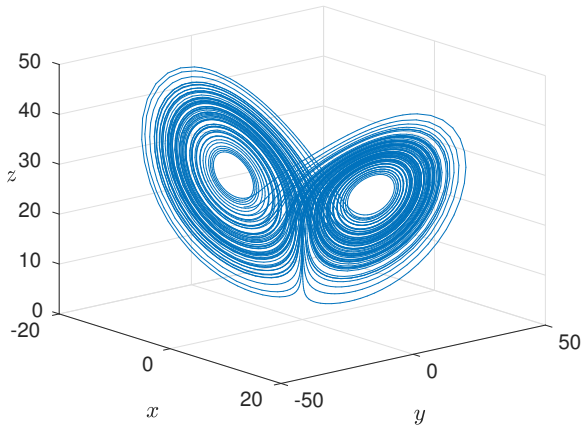
$$\frac{dx}{dt} = 10(y - x)$$
$$\frac{dy}{dt} = 28x - y - xz$$
$$\frac{dz}{dt} = xy - \frac{8}{3}z$$

using ode45, ode113, ode15s.Use the initial conditions $x(0) = 3$, $y(0) = 3$, $z(0) = 20$. **Set absolute and relative tolerances to** 10^{-6}. **use plot3 to plot your solution in 3D for** $t \in [0, 100]$. **The trajectory of your solution should have a butterfly shape. How accurate are your solutions at** $t = 100$.

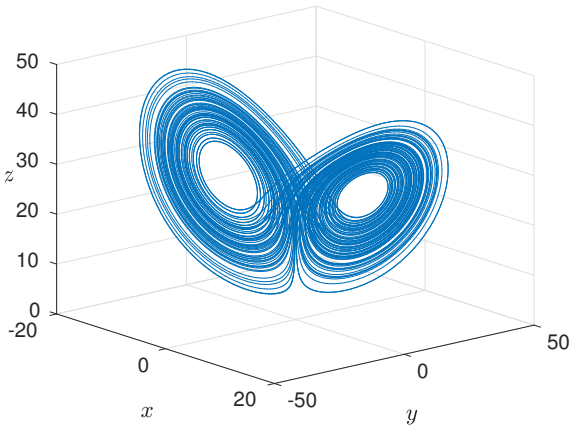
We implement the code given in the link to run with the three solvers and we perturb the initial conditions to check if our solution remains the same. In the following figure the Lorentz atractor obtained using the three different solvers.



(a) Using ode45



(b) Using ode113



(c) Using ode15s

Figure 5: Lorentz atractor using different solvers.

Perturbing the initial condition by $\epsilon = 10^{-2}$, we have obtain a solution that differs a bit from the obtained with the original initial conditions, since the system is very sensitive to the initial conditions. The infinity norm of the difference is, for each solver

- Solver ode45: $L_1(err) = 0.020995541619556$.
- Solver ode113:$L_1(err) = 0.020998997008590$.
- Solver ode15s: $L_1(err) = 0.020998978476641$.

We can see how the solver ode45 seemed to be the less sensitive to the perturbation.

Problem 5

In this problem you will derive the barycentric weights for Chebyshev points of the second kind. Let

$$x_j = \cos(j\pi/n),\ j = 0, \ldots, n, \quad \text{and} \quad w_j = 1/\left(\prod_{k=0, k\neq j}^{n} (x_j - x_k) \right)$$

In Matlab, use the formula above to evaluate w_j, $j = 0, \ldots, n$ with $n = 10$. Verify that the weights returned in Matlab are

$$w_0 = 2^{n-1}/(2n),\ w_j = 2^{n-1}(-1)^j/n,\ j = 1, \ldots, n-1,\ w_n = 2^{n-1}(-1)^n/(2n)\ . \tag{1}$$

Explain why the factor $2^{n-1}/n$ can be ignored when evaluating the barycentric formula. That is, one can use, instead, the weights

$$\tilde{w}_0 = 1/2,\ \tilde{w}_j = (-1)^j,\ j = 1, \ldots, n-1,\ \tilde{w}_n = (-1)^n/2\ .$$

By coding matlab both ways of calculating the weights we found the claim to be true since we obtained an infinity norm of the difference of order 10^{-14}. We can prove the previous by simply substituting the definition of the weights in the baricentric formula that when using the barycentric formula,

$$\begin{aligned} p(x) &= \frac{\sum_{j=0}^{n} \frac{w_j}{(x-x_j)} f_j}{\sum_{j=0}^{n} \frac{w_j}{(x-x_j)}} \\ &= \frac{\sum_{j=0}^{n} \frac{\frac{2^{n-1}(-1)^j}{n}}{(x-x_j)}}{\sum_{j=0}^{n} \frac{\frac{2^{n-1}(-1)^j}{n}}{(x-x_j)}} f_j \\ &= \frac{\frac{2^{n-1}}{n}}{\frac{2^{n-1}}{n}} \frac{\sum_{j=0}^{n} \frac{(-1)^j}{(x-x_j)}}{\sum_{j=0}^{n} \frac{(-1)^j}{(x-x_j)}} f_j \\ &= \frac{\sum_{j=0}^{n} \frac{(-1)^j}{(x-x_j)}}{\sum_{j=0}^{n} \frac{(-1)^j}{(x-x_j)}} f_j. \end{aligned}$$

Hence, since the factor $2^{n-1}/n$ gets cancelled, it can be ignored from the weights.

Show that, for $x_j = cos(j\pi/n)$,

$$\prod_{k=0}^{n}(x - x_k) = \frac{1}{2^{n-1}}(x^2 - 1)U_{n-1}(x)$$

and

$$w_j = 1/\left(\prod_{k=0,k\neq j}^{n}(x_j - x_k)\right) = \lim_{x\to x_j}\frac{2^{n-1}(x - x_j)}{(x^2 - 1)U_{n-1}(x)}\ .$$

Here $U_{n-1}(x) = \frac{sin(n\arccos x)}{sin(\arccos x)}$ is the Chebyshev polynomial of the second kind of degree $n - 1$.

We start with the relation between the Chebishev's polynomials of first and second kind

$$T_n'(x) = nU_{n-1}(x).$$

Using that

$$T_n(x) = 2^{n-1}\prod_{k=0}^{n-1}(x - x_k),$$

and the previous relation we get

$$nU_{n-1}(x) = T_n'(x) = n2^{n-1}\prod_{k=1}^{n-1}(x - \tilde{x}_k)\ .$$

Note that now the roots have changed, now they are the roots of the Chebishev's polynomial of the second kind and the roots of the boundaries are missing. We can include them by multiplying

$$\prod_{k=0}^{n}(x - \tilde{x}_k) = (x - x_0)(x - x_n)\prod_{k=1}^{n-1}(x - \tilde{x}_k) = (x - x_0)(x - x_n)\frac{1}{2^{n-1}}U_{n-1}(x),$$

where we have made $\tilde{x}_0 = x_0$ and same for x_n. Lastly, recall that

$$(x - x_0)(x - x_n) = (x + 1)(x - 1) = (x^2 - 1),$$

and we obtain the desired expression

$$\prod_{k=0}^{n}(x - \tilde{x}_k) = \frac{1}{2^{n-1}}(x^2 - 1)U_{n-1}(x).$$

Further, observe that

$$\frac{d}{dx}\prod_{k=0}^{n}(x - x_k) = \sum_{k=0}^{n}\left(\prod_{j=0,j\neq k}^{n}(x - x_j)\right) = \prod_{k=0,k\neq j}^{n}(x_j - x_k).$$

Hence,

$$\frac{1}{\frac{d}{dx}\prod_{k=0}^{n}(x-x_k)} = \lim_{x\to x_j}\frac{(x-x_j)}{\prod_{k=0}^{n}(x-x_k)} = \lim_{x\to x_j}\frac{2^{n-1}(x-x_j)}{(x^2-1)U_{n-1}(x)}.$$

Use L'Hopital's Rule to evaluate the limit in (2) and derive the expressions in equation (1).

Given the expression for $U_{n-1}(x)$ above we obtain the derivative

$$U'_{n-1}(x) = \frac{-n\cos(n\arccos(x))\sin(\arccos(x)) + \sin(n\arccos(x))\cos(\arccos(x))}{\left(\sqrt{1-x^2}\right)\sin^2(\arccos(x))}.$$

Recall the definition of $x_j = \cos(j\pi/n)$. Then,

$$\begin{aligned} U'_{n-1}(x_j) &= \frac{-n\cos(j\pi)}{\left(\sqrt{1-x^2}\right)\sin(\frac{j\pi}{n})} + \frac{\sin(j\pi)\cos(\frac{j\pi}{n})}{\left(\sqrt{1-x^2}\right)\sin^2(\frac{j\pi}{n})} \\ &= \frac{1}{\sqrt{1-x^2}}\left(\frac{n\cos(j\pi)}{\sin(\frac{j\pi}{n})}\right) \\ &= \frac{n(-1)^j}{\sin^2(\frac{j\pi}{n})}\ . \end{aligned}$$

To finish, use L'Hopital's rule and the result from the previous part,

$$\begin{aligned} w_j &= \lim_{x\to x_j}\frac{2^{n-1}(x-x_j)}{(x^2-1)U_{n-1}(x)} \\ &= \lim_{x\to x_j}\frac{2^{n-1}}{2xU_{n-1}(x) + (x^2-1)U'_{n-1}(x)} \\ &= \frac{2^{n-1}}{2x_jU_{n-1}(x_j) + (x_j^2-1)U'_{n-1}(x_j)} \\ &= \frac{2^{n-1}}{\sin^2(\frac{j\pi}{n})\left(\frac{n(-1)^j}{\sin^2(\frac{j\pi}{n})}\right)} \\ &= \frac{2^{n-1}(-1)^j}{n}. \end{aligned}$$

Matlab code for this problem

```
%% Problem 5
n=10;
j=0:n;
x=cos(j*pi/n);
w_a=zeros(n+1,1);
w_b=zeros(n+1,1);
for j=0:n
```

```
    xj=x(j+1);
    xk=x;
    xk(j+1)=[];
    w_a(j+1)=1/prod(xj-xk);
end
w_b(1)=2^(n-1)/(2*n);
for j=2:n
    w_b(j)=2^(n-1)*(-1)^(j-1)/n;
end
w_b(n+1)=2^(n-1)*(-1)^n/(2*n);
err=norm(w_a-w_b,inf)
```