1. eulersmethod.m

function [tn,xn] = eulersmethod(t0,tt,x0,n)
% Objective
% This function solves the ODE dx/dt = f(x,t) using Euler’s method.
% Inputs
% t0 - initial time
% tt - terminal time
% x0 - initial value (at t0)
% n - number of steps
% Outputs
% tn - vector of time where x(t) is evaluated using Euler’s method, including t0 and tt
% xn - vector of the values of x(t) from t0 to tt
% functions called
% ode_f.m - f(x,t) in dx/dt=f(x,t)
% preallocate vectors tn and xn

% preallocate vectors tn and xn

% calculate the step size
h = (tt-t0)/n;

% Fill in the vectors tn, xn

end

ode_f.m
function [ f ] = ode_f( x,t )
%%% Objective
%%% this function evaluates the RHS of the ODE dx/dt=f(x,t)
%%% Inputs
%%% x,t - state and time
%%% Outputs
%%% f - value of f(x,t) at state x and time t

%%% for problem one, f(x,t) = x/2
f = x/2;
end

hw5p1.m (main script)

%%% This is the main code for homework 5, problem 1.
%%% This script solves the ode dx/dt=x/2 from t = 0 to 2
%%% initial condition is x(0)=3
%%% uses Euler’s method
%%% step sizes n = 10^1, 10^2, ..., 10^5
%%% plots the error at T=2 vs. step sizes on a log10-log10 plot
%%% functions called
%%% eulersmethod.m - euler’s method
%%% ode_f.m - in this problem ode_f = x/2

%%% time domain for this ODE
t0 = 0;
tt = 2;

%%% initial value
x0=3;

%%% no. of steps
j = 1:5;
n = 10.^j;

%%% true solution x(t) at time T=2
xtrue = 3*exp(tt/2);
% preallocate errors
err = zeros(1,5);

% solve for x(t) for each of the step sizes using Euler's method and compute the errors
for k = 1:5
    % Solve for x(t) with Euler's method
    [tn,xn] = eulersmethod(t0,tt,x0,n(k));
    % plot the solution
    figure(1)
    plot(tn,xn)
    hold on
    % compute the error
    err(k) = abs(xtrue-xn(end));
end

% plot the true solution
% x(t) = 3*exp(t/2)
figure(1)
plot(tn,3*exp(tn/2))
legend(’n = 10’,’n = 10^2’,’n = 10^3’,’n = 10^4’,’n = 10^5’,... ’True Solution’,’Location’,’SouthEast’)xlabel(’t’)ylabel(’x(t)’)% plot the errors vs step sizes
figure(2)
plot(log10((tt-t0)./n),log10(err))xlabel(’log10(step size)’)ylabel(’log10(error)’)

3
2. Euler’s method code is the same except the function called is ode_f2.m instead of ode_f2.m.

ode_f2.m

function [ f ] = ode_f2( x,t )
% Objective
% this function evaluates the RHS of the ODE dx/dt=f(x,t)
%%%%%%
% Inputs
% x,t - state and time
%%%%%%
% Outputs
% f - value of f(x,t) at state x and time t
%%%%%%

% for problem two, f(x,t) = (2-sqrt(x^2+2))*(2+t-cos(pi*x))+2*x
f = (2-sqrt(x^2+2))*(2+t-cos(pi*x))+2*x;
end

hw5p2.m (main script)

%%%%%%
% This is the main code for homework 5, problem 2.
% This script solves the ode dx/dt=(2-sqrt(x^2+2))*(2+t-cos(pi*x))+2*x from t = 1 to 4
% initial condition is x(1)=1
% uses Euler’s method
% step sizes n = 10^1, 10^2, ..., 10^5
% use solution with n=10^6 as true solution
% plots the error at T=4 vs. step sizes on a log10-log10 plot
%%%%%%
% functions called
% eulersmethod.m - euler’s method
% ode_f2.m - in this problem ode_f2 = (2-sqrt(x^2+2))*(2+t-cos(pi*x))+2*x
%%%%%%

% time domain for this ODE
t0 = 1;
tt = 4;

% initial value
x0=1;

% no. of steps
j=1:1:5;
n = 10.^j;
% preallocate errors
err = zeros(1,5);

% solve for x(t) for each of the step sizes using Euler's method and compute the errors
% solve the ODE with n=10^6 first
[t6,x6] = eulersmethod(t0,tt,x0,10^6);
% use the solution at tt=4 as the true solution
xtrue = x6(end);
for k = 1:5
    % Solve for x(t) with Euler's method
    [tn,xn] = eulersmethod(t0,tt,x0,n(k));
    % plot the solution
    figure(1)
    plot(tn,xn)
    hold on
    err(k) = abs(xtrue-xn(end));
end
% plot the true solution
figure(1)
plot(t6,x6)
% add legend
legend('n = 10','n = 10^2','n = 10^3','n = 10^4','n = 10^5',...  
'True Solution','Location','SouthEast')
xlabel('t')
ylabel('x(t)')
% plot the errors
figure
plot(log10((tt-t0)./n),log10(err))
xlabel('log10(step size)')
ylabel('log10(error)')
With \( n = 10^6 \), the approximated solution is very close to the true solution (compared to much smaller \( n \)'s), and we can expect a linear behavior in the log-log plot.

3. The magnitude of the error at 1000 steps, \( e_{1000} \), is roughly \( 100/1000 \) the size of the magnitude of that with 100 steps, \( e_{100} \). We have the following approximate relation between a step size \( n \) and the error \( e_n \) at \( n \) steps.

\[
e_n \simeq C/n
\]

Or equivalently,

\[
\log_{10}(e_n) \simeq \log_{10}(C) - \log_{10}(n)
\]

We can use \( \bar{x}(27) \) at \( n = 1000 \) as the true solution \( x(27) \) to estimate \( e_{100} \).

\[
\log_{10}(|125.866 - 124.156|) \simeq \log_{10}(C) - \log_{10}(100)
\]

\[
\Rightarrow \log_{10}(C) \simeq 2.233
\]

For an error of \( 10^{-3} \), we can solve for the number of steps, \( n \):

\[
\log_{10}(10^{-3}) \simeq 2.233 - \log_{10}(n)
\]

\[
\Rightarrow \log_{10}(n) \simeq 5.233
\]

\[
\Rightarrow n \simeq 10^{5.233} = 1.7 \times 10^5
\]

The run time required for 1000 steps is 2 seconds. Therefore, the run time for \( 1.7 \times 10^5 \) steps is

\[
t \simeq \frac{2s}{1000} \cdot 1.7 \times 10^5 = 340s
\]

4. Linear interpolation for \( f(x) = \exp(2x - 1), \ x_0 = 1, \ x_1 = 2. \)

\[
p(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)
\]

\[
= \exp(2 \cdot 1 - 1) + \frac{\exp(2 \cdot 2 - 1) - \exp(2 \cdot 1 - 1)}{2 - 1}(x - 1)
\]

\[
= 2.7138 + 17.3673(x - 1)
\]

\[
p(x) = 17.3673x - 14.6535
\]

Error bound:

\[
|e(x)| \leq \frac{1}{8}(x_1 - x_0)^2 \max_{\xi \in [x_0, x_1]} |f^{(2)}(\xi)|
\]

\[
\leq \frac{1}{8}(2 - 1)^2 \max_{\xi \in [1, 2]} |4 \exp(2 \xi - 1)|
\]

\( f^{(2)}(x) = 4 \exp(2x - 1) \) is increasing on \([1, 2] \), therefore, the error bound is

\[
|e(x)| \leq \frac{1}{8} \cdot 4 \exp(2 \cdot 2 - 1) = 10.0428
\]
Actual max errors:

\[ e(x) = |f(x) - p(x)| = |\exp(2x - 1) - (17.3673x - 14.6535)| \]

\[
\max_{x \in [1,2]} e(x) = \max_{x \in [1,2]} |\exp(2x - 1) - (17.3673x - 14.6535)| \\
= \max_{x \in [1,2]} |\exp(2x - 1) - 17.3673x + 14.6535| \\
= 4.1157
\]

5. **Piecewise linear interpolation with max error < 0.02**

Use a uniform grid on \([1,3]\), where we denote the grid size by \(h\), the error bound for linear interpolation is

\[ |e(x)| \leq \frac{1}{8} h^2 \max_{\xi \in [1,3]} |f^{(2)}(\xi)| \]

Substitute in the second order derivative \(f^{(2)}\), we need \(h\) such that

\[ |e(x)| \leq \frac{1}{8} h^2 \max_{\xi \in [1,3]} |2\exp(\xi) + (1 + \xi)\exp(\xi)| < 0.02 \]

The 2nd order derivative is increasing from \(f^{(2)}\), so it attains its maximum on \([1,3]\) at \(\xi = 3\). Therefore,

\[
\frac{1}{8} h^2 \cdot |2\exp(3) + (1 + 3)\exp(3)| < 0.02 \\
\Rightarrow h < 0.0364
\]

Therefore, the number of pieces is

\[ n \geq \frac{3 - 1}{0.0364} = 54.9451 \Rightarrow n \geq 55 \]

6. **Numerically integrate** \(\int_{0}^{\pi/3} \tan(x) \, dx\) **using the left endpoint rule**, with step size \(h = \frac{\pi}{6}\) and \(\frac{\pi}{12}\). Find the actual errors and error bounds for these values of \(h\).

True value of integral (use calculator): \(\int_{0}^{\pi/3} \tan(x) \, dx = 0.6932\)

\(h = \frac{\pi}{6}, n = (\pi/3 - 0)/(\pi/6) = 2\)

\[ x_0 = 0, \tan(x_0) = 0 \]
\[ x_1 = \pi/6, \tan(x_1) = 0.5774 \]

\[ \int_{0}^{\pi/3} \tan(x) \, dx \simeq (0 + 0.5774)(\pi/6) = 0.3023 \]

True error:

\[ e_{h=\frac{\pi}{6}} = |0.6932 - 0.3023| = 0.3909 \]

Error bound:

\[ e_{bound} = \frac{(b - a) \cdot h}{2} \max_{\xi \in [0,\pi/3]} |f'(\xi)| = \frac{(\pi/3) \cdot (\pi/6)}{2} \max_{\xi \in [0,\pi/3]} |\sec^2(\xi)| = 1.0966 \]
\[ h = \frac{\pi}{12}, \quad n = (\pi/3 - 0)/(\pi/12) = 4 \]

\[ x_0 = 0, \tan(x_0) = 0 \]
\[ x_1 = \pi/12, \tan(x_1) = 0.2679 \]
\[ x_2 = \pi/6', \tan(x_2) = 0.5774 \]
\[ x_3 = \pi/4', \tan(x_3) = 1 \]

\[ \int_0^{\pi/3} \tan(x) \, dx \simeq (0 + 0.2679 + 0.5774 + 1)(\pi/12) = 0.4831 \]

True error:
\[ e_{h=\pi/12} = |0.6932 - 0.4831| = 0.2101 \]

Error bound:
\[ e_h^{\text{bound}} = \frac{(b - a) \cdot h}{2} \max_{\xi \in [0, \pi/3]} |f'(\xi)| = \frac{(\pi/3) \cdot (\pi/12)}{2} \max_{\xi \in [0, \pi/3]} |\sec^2(\xi)| = 0.5483 \]

Step sizes such that error is no greater than \(10^{-3}\) and \(10^{-5}\)

Error bound formula:
\[ e_h^{\text{bound}} = \frac{(b - a) \cdot h}{2} \max_{\xi \in [0, \pi/3]} |f'(\xi)| \]

We have \( \max_{\xi \in [0, \pi/3]} |f'(\xi)| = 4 \), \( b - a = \pi/3 \). For error to be no greater than \(10^{-3}\), we need

\[ \frac{(\pi/3) \cdot h}{2} \cdot 4 < 10^{-3} \Rightarrow h < 4.7746 \times 10^{-4} \]

For error to be no greater than \(10^{-5}\), we need

\[ \frac{(\pi/3) \cdot h}{2} \cdot 4 < 10^{-5} \Rightarrow h < 4.7746 \times 10^{-6} \]

7. Left endpoint rectangle rule error estimate

We are given \( e_{k,n}^{LE} < 7h^2 \), that is, the error in one rectangle is bounded by \(7h^2\). To find the total error bound, we need to multiply the error bound in one rectangle by the total number of rectangles. Since the interval has length \( \pi - 0 = \pi \), we have

\[ e_n^{LE} \leq e_{k,n}^{LE} \cdot n \leq 7h^2 \cdot \frac{\pi}{h} \leq 7\pi h \]

We need \( e_n^{LE} < 10^{-5} \). Solve for \( h \):

\[ e_n^{LE} \leq 7\pi h < 10^{-5} \Rightarrow h < 4.5473 \times 10^{-7} \]

Therefore, the minimal number of steps (rectangles) is \( n = \pi/h = 6.9 \times 10^6 \)