1 Implement, debug and test a MATLAB function \texttt{pleg.m} of the form

\begin{verbatim}
function p = pleg(t, n)
  \% t: evaluation point
  \% n: degree of polynomial

  This function evaluates a monic Legendre polynomial \( P_n \) of degree \( n \), at evaluation point \( t \) with \(|t| \leq 1 \). Here \( P_0 = 1 \), \( P_1(t) = t \) and \( P_n \) is determined by the recurrence

  \[ P_n(t) = tP_{n-1}(t) - c_nP_{n-2}(t) \]

  for \( n \geq 2 \), where \( c_n = (n-1)^2/(4(n-1)^2-1) \). Be sure to iterate forward from \( n = 0 \) rather than recurse backward from \( n \), and do not generate any new function handles. Test that your function gives the right values for small \( n \) where you know \( P_n \).
\end{verbatim}

2 Implement a MATLAB function \texttt{gaussint.m} of the form

\begin{verbatim}
function [w, t] = gaussint( n )
  \% n: Number of Gauss weights and points

  which computes weights \( w \) and points \( t \) for the \( n \)-point Gaussian integration rule

  \[ \int_{-1}^{1} f(t) \, dt \approx \sum_{j=1}^{n} w_j f(t_j). \]

  (a) Find the points \( t_j \) to as high precision as possible, by applying your code \texttt{bisection.m} to \texttt{pleg.m}. Bracket each \( t_j \) initially by the observation that the zeroes of \( P_{n-1} \) separate the zeroes of \( P_n \) for every \( n \). Thus the single zero of \( P_1 = t \) separates the interval \([-1, 1]\) into two intervals, each containing exactly one zero of \( P_2 \). The two zeroes of \( P_2 \) separate the interval \([-1, 1]\) into three intervals, and so forth. Thus you will find all the zeroes of \( P_1, P_2, \ldots, P_{n-1} \) in the process of finding all the zeroes of \( P_n \).

  (b) Find the weights \( w_j \) to as high precision as possible by applying your code \texttt{gadap.m} to

  \[ w_j = \int_{-1}^{1} L_j(t)^2 \, dt \]

  where \( L_j \) is the \( j \)th Lagrange basis polynomial for interpolating at \( t_1, t_2, \ldots, t_n \).

  (c) For \( 1 \leq n \leq 20 \), test that your weights and points integrate monomials \( f(t) = t^j \) exactly for \( 0 \leq j \leq 2n - 1 \).
\end{verbatim}

3 (a) For arbitrary real \( s \) find the exact solution of the initial value problem

\[ y'(t) = \frac{1}{2} \left( y(t) + y(t)^3 \right) \]

with \( y(0) = s > 0 \).

(b) Show that the solution blows up when \( t = \log(1 + 1/s^2) \).
4 (a) Find the general solution of the difference equation
\[ u_{j+2} = u_{j+1} + u_j. \]

(b) Find all initial values \( u_0 \) and \( u_1 \) such that \( u_j \) remains bounded by a constant as \( j \to \infty \).

5 (a) Write, test and debug a matlab function

\[ \text{function } u = euler(a, b, ya, f, r, n) \]
\[
\% a,b: interval endpoints with a < b \\
\% n: number of steps with h = (b-a)/n \\
\% ya: vector y(a) of initial conditions \\
\% f: function handle f(t, y, r) to integrate \\
\% r: parameters to f \\
\% u: output approximation to the final solution vector y(b) \\
\]
which approximates the final solution vector \( y(b) \) of the vector initial value problem
\[ y' = f(t, y, r) \]
\[ y(a) = y_a \]
by the numerical solution vector \( u_n \) of Euler’s method
\[ u_{j+1} = u_j + hf(t_j, u_j, r) \quad j = 0, 1, \ldots, n - 1 \]
with \( h = (b-a)/n \) and \( u_0 = y_a \).

(b) Use \texttt{euler.m} to approximate the solution \( z(T) \) at \( T = 4\pi \) of the initial value problem

\[ z' = \begin{bmatrix} x \\ y \\ u \\ v \end{bmatrix}' = f(t, z) = \begin{bmatrix} u \\ v \\ -x/(x^2 + y^2) \\ -y/(x^2 + y^2) \end{bmatrix} \]
with initial conditions
\[ z = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \]
at \( t = 0 \) which cause the solution to move in a unit circle forever. Measure the maximum error
\[ E_N = \max(|x_N - \cos t_N|, |y_N - \sin t_N|, |u_N + \sin t_N|, |v_N - \cos t_N|) \]
after 2 revolutions \( (T = 4\pi) \) with time steps \( h = T/N \) for \( N = 1000, 2000, \ldots, 16000 \).
Estimate the constant \( C \) such that the error behaves like \( Ch \). Measure the CPU time for each run and estimate the total CPU time necessary to obtain the solution to three–digit, six–digit and twelve–digit accuracy. Plot the solutions.

(c) Use \texttt{euler.m} with \( s = [512, 64, 8, 1] \) and \( N = [10^3, 10^4, 10^5, 10^6] \) to verify conclusion (b) of problem 3.
(See GGK 10.1) The position \((x(t), y(t))\) of a satellite orbiting around the earth and moon is described by the second-order system of ordinary differential equations:

\[
x'' = x + 2y' - b \frac{x + a}{((x + a)^2 + y^2)^{3/2}} - a \frac{x - b}{((x - b)^2 + y^2)^{3/2}}
\]

\[
y'' = y - 2x' - b \frac{y}{((x + a)^2 + y^2)^{3/2}} - a \frac{y}{((x - b)^2 + y^2)^{3/2}}
\]

where \(a = 0.012277471\) and \(b = 1 - a\). When the initial conditions

\[
x(0) = 0.994
\]

\[
x'(0) = 0
\]

\[
y(0) = 0
\]

\[
y'(0) = -2.00158510637908
\]

are satisfied, there is a periodic orbit with period \(T = 17.06521656015796\).

(a) Convert this problem to a 4 \times 4 first-order system \(u' = f(t, u, r)\), \(u(0) = u_0\), by introducing

\[
u = [x, x', y, y'] = [u_1, u_2, u_3, u_4]
\]

as a new vector unknown function and defining \(f\) appropriately.

(b) Use euler.m to approximate \(u(T)\) and plot the error vs. \(N\) for \(N = 1000, 2000, \ldots, 1024000\) steps. Measure the CPU time for each run and estimate the total CPU time necessary to obtain an orbit which is periodic to three-digit, six-digit and twelve-digit accuracy.