```
% ----------------------
% File name: M370_Exam1.m
% Name: Troy Chin
% Date: 10/05/2025
% ----------------------

% Question 1

A = [6 9 0 0 0 0;
    0 5 9 0 0 0;
    0 0 4 9 0 0;
    0 0 0 3 9 0;
    0 0 0 0 2 9;
    0 0 0 0 0 1];

% 1a) Compute and display the eigenvalues and eigenvectors of A
[S, D] = eig(A); % We set this as an array to compute both the eigenvectors
% and eigenvalues at the same time
fprintf('Eigenvalues of A: \n');
disp(D); % This will display the eigenvalues of A in the form of a matrix
% just for organization.
fprintf('Eigenvectors of A: \n');
disp(S); % This will display the eigenvectors of A in the form of a matrix
% just for organization.

% 1b) Display the condition number of A
kappa_inf = cond(S, "inf");
fprintf('Condition number: ')
disp(kappa_inf); % Using the cond() function, this will display the
% condition number for the matrix

% 1c) Perturbation at (6,1) entry
eps = 1e-6; Ae = A;
Ae(6,1) = Ae(6,1) + eps; % Perturb the (6,1) entry
E = Ae - A;
normE_inf = norm(E, inf);
fprintf('Perturbation matrix E: \n');
disp(E);
disp(normE_inf);

% Eigenvalues before and after perturbation
lambda_A = eig(A);
lambda_Ae = eig(Ae);
fprintf('Eigenvalues of A (before perturbation): \n'); disp(lambda_A);
fprintf('Eigenvalues of A (after perturbation): \n'); disp(lambda_Ae);
sorted_lambda_A = sort(lambda_A); sorted_lambda_Ae = sort(lambda_Ae);

% Bauer-Fike Theorem
bound = kappa_inf * normE_inf;
fprintf('Bound: '); disp(bound);

% 1d) Compute and display the left eigenvectors of A
```

```
[W, D] = eig(A.');
fprintf('Left eigenvectors of A: \n'); disp(W);
fprintf('Left eigenvalues of A: \n'); disp(D);

% ------------------------------------------------ %

% Question 2

years = [1926 1937 1939 1959 1970 1979 1989 2002 2010 2021]';
pop = [100.89 103.97 109.40 117.53 130.08 137.55 147.02 145.17 142.86
147.18]';

% 2a) Plot the data

% Interpolate the data
u_plot = linspace(1926, 2030, 1000);
p9 = polyfit(years, pop, 9); poly_vals = polyval(p9, u_plot);

% Piecewise linear
pl_vals = interp1(years, pop, u_plot, 'linear');

% pchip
pchip_pp = pchip(years,pop); pchip_vals = ppval(pchip_pp, u_plot);

% Cubic spline
spline_pp = spline(years, pop); spline_vals = ppval(spline_pp, u_plot);

% 2b) Plot the interpolated data and original data points
figure(1); clf;
hold on; grid on;
plot(years, pop, 'ro', 'MarkerFaceColor', 'r', 'DisplayName', 'Original
Data');
plot(u_plot, poly_vals, 'b-', 'DisplayName', 'Polynomial Fit');
plot(u_plot, pl_vals, 'g--', 'DisplayName', 'Piecewise Linear');
plot(u_plot, pchip_vals, 'm:', 'DisplayName', 'PCHIP');
plot(u_plot, spline_vals, 'c-.', 'DisplayName', 'Cubic Spline');
legend('Data', 'deg-9 poly', 'linear', 'pchip', 'spline', 'Location', 'NW');
hold off;

% 2c) I trust the spline estimate as it is closely fitted across all of the
% interpolated points.

% 2d) The estimation of the population in 1930 is around 90.5.
pop_1930 = interp1(years, pop, 1930, 'linear');
fprintf('Estimated population in 1930: %.2f\n', pop_1930);

% 2e) Modifying the pchiptx.m file
function [v, v_prime] = pchiptx(x,y,u)
%PCHIPTX  Textbook piecewise cubic Hermite interpolation.
%  v = pchiptx(x,y,u) finds the shape-preserving piecewise cubic
%  interpolant P(x), with P(x(j)) = y(j), and returns v(k) = P(u(k)).
%
%  See PCHIP, SPLINETX.
```

```
%   Copyright 2014 Cleve Moler


%  First derivatives
   h = diff(x);
   delta = diff(y)./h;
   d = pchipslopes(h,delta);

%  Piecewise polynomial coefficients

   n = length(x);
   c = (3*delta - 2*d(1:n-1) - d(2:n))./h;
   b = (d(1:n-1) - 2*delta + d(2:n))./h.^2;

%  Find subinterval indices k so that x(k) <= u < x(k+1)

   k = ones(size(u));
   for j = 2:n-1
      k(x(j) <= u) = j;
   end

%  Evaluate interpolant

   s = u - x(k);
   v = y(k) + s.*(d(k) + s.*(c(k) + s.*b(k)));
   v_prime = d(k) + 2*c(k).*s + 3*b(k).*s.^2;
end

% ------------------------------------------------------------

function d = pchipslopes(h,delta)
%  PCHIPSLOPES  Slopes for shape-preserving Hermite cubic
%  pchipslopes(h,delta) computes d(k) = P'(x(k)).

%  Slopes at interior points
%  delta = diff(y)./diff(x).
%  d(k) = 0 if delta(k-1) and delta(k) have opposites
%         signs or either is zero.
%  d(k) = weighted harmonic mean of delta(k-1) and
%         delta(k) if they have the same sign.

   n = length(h)+1;
   d = zeros(size(h));
   k = find(sign(delta(1:n-2)).*sign(delta(2:n-1))>0)+1;
   w1 = 2*h(k)+h(k-1);
   w2 = h(k)+2*h(k-1);
   d(k) = (w1+w2)./(w1./delta(k-1) + w2./delta(k));

%  Slopes at endpoints

   d(1) = pchipend(h(1),h(2),delta(1),delta(2));
   d(n) = pchipend(h(n-1),h(n-2),delta(n-1),delta(n-2));
end
```

```
% ------------------------------------------------------

function d = pchipend(h1,h2,del1,del2)
%  Noncentered, shape-preserving, three-point formula.
   d = ((2*h1+h2)*del1 - h1*del2)/(h1+h2);
   if sign(d) ~= sign(del1)
      d = 0;
   elseif (sign(del1)~=sign(del2))&(abs(d)>abs(3*del1))
      d = 3*del1;
   end
end

% Results
x = 1900:10:2000;  % given years
y = [75.995 91.972 105.711 123.203 131.669 ...
     150.697 179.323 203.212 226.505 249.633 281.422];  % U.S. population
(millions)

% Years to evaluate
u = [1930 1970 1995];

% Call modified pchiptx (which returns v and v_prime)
[v, v_prime] = pchiptx(x, y, u);

% Display results
fprintf('Estimated population values: \n'); fprintf('\n');
disp(table(u', v', v_prime', 'VariableNames', {'Year','Population','dPdu'}))

% Plot the interpolant and derivatives visually
uu = linspace(1900, 2000, 1000);
[pp, pp_prime] = pchiptx(x, y, uu);

figure;
yyaxis left
plot(uu, pp, 'b-', 'LineWidth', 1.5)
hold on
plot(u, v, 'bo', 'MarkerFaceColor', 'b')
ylabel('Population (millions)')
yyaxis right
plot(uu, pp_prime, 'r--', 'LineWidth', 1.5)
ylabel('Rate of Change (millions/year)')
xlabel('Year')
title('PCHIP Interpolation and Derivative')
legend('Interpolant','Data Points','Derivative','Location','northwest')
grid on

 % These are the estimated population values obtained by the PCHIP
 % interpolating polynomial. If v' > 0, positive infinity, the
 % population increases. If v' < 0, the population is decreasing.

% ------------------------------------------------- %

% Question 3
```

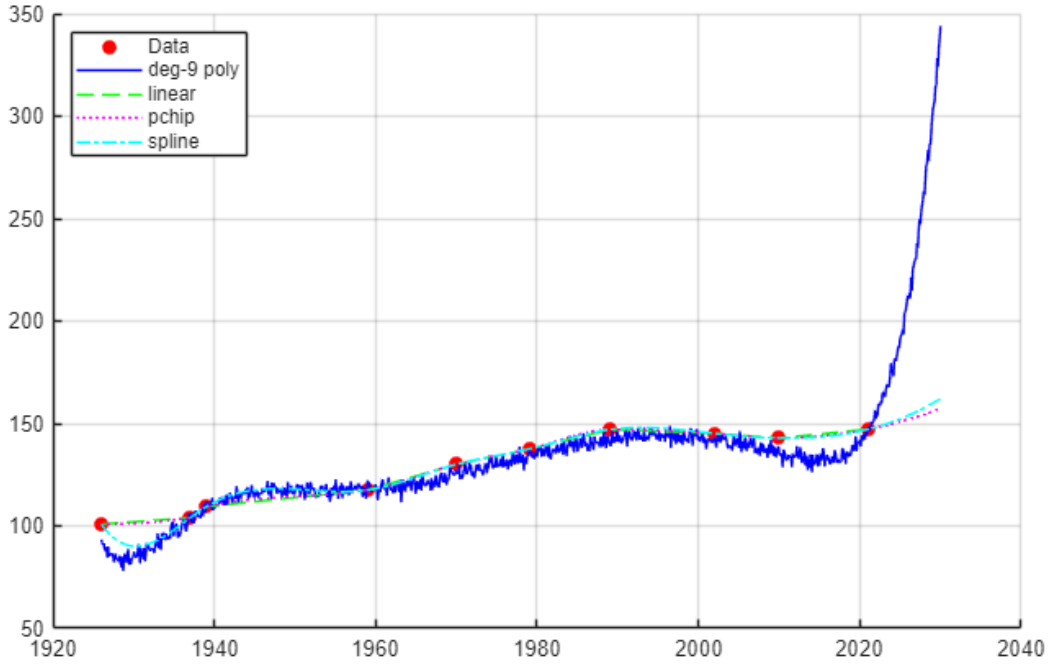
```
% 2a) PROOF: Suppose f(x_n) = x_n - c. If we take the limit as x_n
% approaches infinity, you will see that the limit will always reach c for
% all x_n.

% 2b) Deriving/Implementing Steffenson's Method
function [p, iter] = Steffensen(f, p0, tol)
if nargin < 3, tol = 1e-6; end
    maxIter = 1000;

    for i = 1:maxIter
        f0 = f(p0);
        f1 = f(p0 + f0);      % Evaluate at x + f(x)
        denom = f1 - f0;

        if abs(denom) < eps
            error('Denominator too small; iteration may fail.');
        end

        p = p0 - (f0^2) / denom;  % Steffensen update

        if abs(p - p0) < tol
            iter = i;
            return
        end

        p0 = p;
    end

    error('Steffensen method did not converge within %d iterations.',
maxIter);
end

% Newton's Method
function [root, iter] = Newton(f, df, x0, tol)
    if nargin < 4, tol = 1e-6; end
    maxIter = 1000;

    for i = 1:maxIter
        x1 = x0 - f(x0)/df(x0);
        if abs(x1 - x0) < tol
            root = x1;
            iter = i;
            return
        end
        x0 = x1;
    end
    error('Newton method did not converge.');
end

% Secant Method
function [root, iter] = Secant(f, x0, x1, tol)
    if nargin < 4, tol = 1e-6; end
    maxIter = 1000;
```

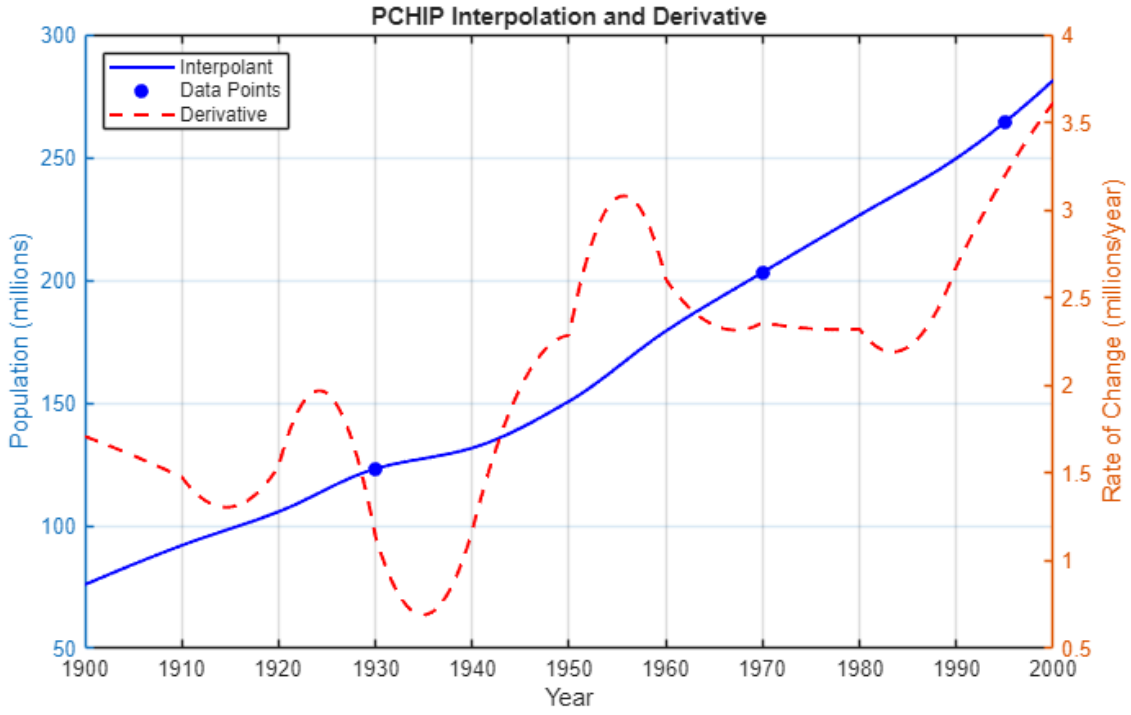
```
    for i = 1:maxIter
        f0 = f(x0);
        f1 = f(x1);
        if abs(f1 - f0) < eps
            error('Denominator too small; iteration fails.');
        end
        x2 = x1 - f1*(x1 - x0)/(f1 - f0);
        if abs(x2 - x1) < tol
            root = x2;
            iter = i;
            return
        end
        x0 = x1;
        x1 = x2;
    end
    error('Secant method did not converge.');
end


% 2c
a_vals = [2, 3, pi];

for a = a_vals
    % Function and derivative
    f = @(x) x^2 - a;      % f(x)
    df = @(x) 2*x;         % derivative f'(x)

    % Initial guesses
    x0 = a/4;  % For Newton & Steffensen
    x1 = 1;    % Second guess for Secant

    % Steffensen's method
    [root_s, iter_s] = Steffensen(f, x0);

    % Newton's method
    [root_n, iter_n] = Newton(f, df, x0);

    % Secant method
    [root_sec, iter_sec] = Secant(f, x0, x1);

    % Display results
    fprintf('a = %.4f\n', a);
    fprintf('Steffensen: Root = %.6f, Iterations = %d\n', root_s, iter_s);
    fprintf('Newton    : Root = %.6f, Iterations = %d\n', root_n, iter_n);
    fprintf('Secant    : Root = %.6f, Iterations = %d\n\n', root_sec,
iter_sec);
end

% I think Newton's Method might be faster because we have an exact
% derivative.
```

```
Eigenvalues of A:
     6     0     0     0     0     0
     0     5     0     0     0     0
```

```
     0     0     4     0     0     0
     0     0     0     3     0     0
     0     0     0     0     2     0
     0     0     0     0     0     1

Eigenvectors of A:
    1.0000   -0.9939    0.9759   -0.9463    0.9051   -0.8523
         0    0.1104   -0.2169    0.3154   -0.4023    0.4735
         0         0    0.0241   -0.0701    0.1341   -0.2104
         0         0         0    0.0078   -0.0298    0.0701
         0         0         0         0    0.0033   -0.0156
         0         0         0         0         0    0.0017

Condition number:    4.0498e+04

Perturbation matrix E:
   1.0e-06 *

         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
         0         0         0         0         0         0
    1.0000         0         0         0         0         0

   1.0000e-06

Eigenvalues of A (before perturbation):
     6
     5
     4
     3
     2
     1

Eigenvalues of A (after perturbation):
    6.0005
    0.9995
    2.0025
    2.9951
    4.0049
    4.9975

Bound:     0.0405

Left eigenvectors of A:
         0         0         0         0         0    0.0017
         0         0         0         0    0.0033    0.0156
         0         0         0    0.0078    0.0298    0.0701
         0         0    0.0241    0.0701    0.1341    0.2104
         0    0.1104    0.2169    0.3154    0.4023    0.4735
    1.0000    0.9939    0.9759    0.9463    0.9051    0.8523

Left eigenvalues of A:
```

```
     1     0     0     0     0     0
     0     2     0     0     0     0
     0     0     3     0     0     0
     0     0     0     4     0     0
     0     0     0     0     5     0
     0     0     0     0     0     6

Warning: Polynomial is badly conditioned. Add points
with distinct X values, reduce the degree of the
polynomial, or try centering and scaling as described
in HELP POLYFIT.
Estimated population in 1930: 102.01
Estimated population values:

    Year    Population     dPdu
    ____    __________    ______

    1930       123.2       1.141
    1970      203.21      2.3587
    1995      264.36       3.196

a = 2.0000
Steffensen: Root = -1.414214, Iterations = 6
Newton    : Root = 1.414214, Iterations = 6
Secant    : Root = 1.414214, Iterations = 6

a = 3.0000
Steffensen: Root = -1.732051, Iterations = 5
Newton    : Root = 1.732051, Iterations = 6
Secant    : Root = 1.732051, Iterations = 6

a = 3.1416
Steffensen: Root = -1.772454, Iterations = 5
Newton    : Root = 1.772454, Iterations = 5
Secant    : Root = 1.772454, Iterations = 7
```





Published with MATLAB® R2025a