PHYS 410 Homework 1

Gavin Pringle, 56401938

September 30, 2024

Introduction

In this homework assignment, two methods for solving nonlinear equations numerically via root finding are explored: bisection and Newton's method. In order to do this, two problems are provided. The first problem involves the 1-dimesional case where the roots of a nonlinear function are found using a hybrid algorithm that first employs bisection followed by Newton's method. The second problem involves the d-dimensional case where a nonlinear system is solved using a d-dimensional Newton iteration.

In both problems it is assumed the function and its derivative in problem 1 as well as the system of equations and its Jacobian in problem 2 are hard-coded as Matlab functions. Specifically, a polynomial of order 10 is provided for the first question and a nonlinear system of three variables is provided for the second question.

Problem 1 - Hybrid Algorithm

Review of Theory

Bisection, also referred to as binary search, is a method used for solving nonlinear equations of the form $f(x) = 0$ for the 1-dimensional case or $\vec{f}(\vec{x}) = \vec{0}$ for the d-dimensional case. The bisection algorithm involves bisecting a search interval and checking whether the root is above or below the bisector. The search interval is then bisected again in the new interval where the root lies, and again it is determined whether the root is above or below the new bisector. This process then repeats until the root is determined to be in an interval of small enough tolerance.

For the 1-dimensional bisection algorithm, it is assumed that there is a root of $f(x) = 0$ in the interval $x_{min} \leq x \leq x_{max}$. From this, it follows that $f(x_{min})f(x_{max}) \leq 0$. As previously described, the interval $[x_{min}, x_{max}]$ which has width $\delta x_0 = x_{max} - x_{min}$ is successively divided into smaller intervals of width $\delta x_1 = \delta x_0/2$, $\delta x_2 = \delta x_0/4$, $\delta x_3 = \delta x_0/8$, $\ldots$ each of contains the root which is checked using the condition $f({x_{min}}^{(n)})f({x_{max}}^{(n)}) \leq 0$. This process is continued until interval is suitably small, which is verified by checking the relative error given by the formula $\frac{|\delta x^{(n)}|}{|x^{(n+1)}|} \leq \epsilon$

Newton's method is another

Numerical Approach

definition of all pertinent problem parameters

exposition of the requisite equations

methodology that is used to solve the problem

function signature

testing method

provided equations

Implementation

refer to appendix

iteration counter

Results

graph and console output

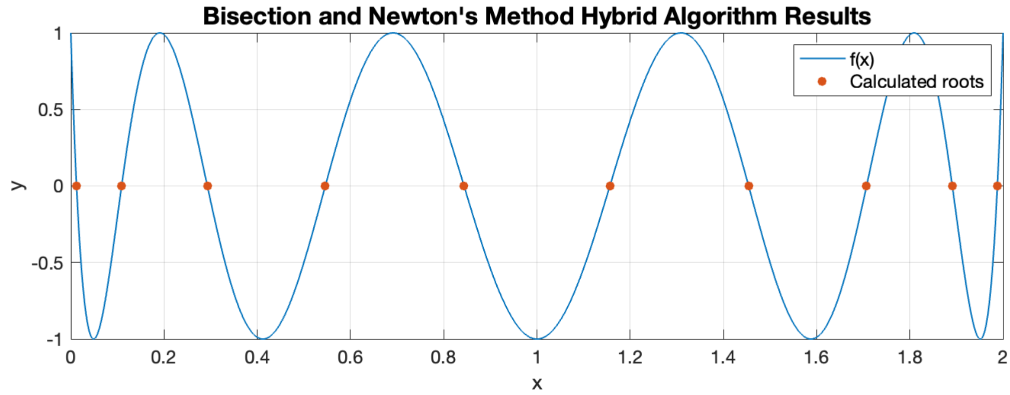

Figure 1: Numerically calculated roots overlaid on example function $f(x)$.

Problem 2 - D-dimesional Newton's Method

Review of Theory

Go over theory for d-d newton's method:

Numerical approach

definition of all pertinent problem parameters

exposition of the requisite equations

methodology that is used to solve the problem

function signature

testing method

provided equations

The Jacobian matrix is

$$\begin{bmatrix} 2x & 4y^3 & 6z^5 \\ -yz^2\sin\left(xyz^2\right)-1 & -xz^2\sin\left(xyz^2\right)-1 & -2xyz\sin\left(xyz^2\right)-1 \\ -2x-2y+2z & -2x+2z & 2x+2y+3z^2-2z \end{bmatrix}$$

Figure 2: Calculated Jacobian matrix for the provided system of equations.

Implementation

refer to appendix

iteration counter

Results

console output

Conclusions

Briefly summarize your findings (again, more extensively in the case of projects). Discuss any particular problems you had with the homework/project. Include your statement concerning your use or non-use of generative AI here.

Appendix A - Hybrid Algorithm Code

```
%% Problem 1 - Hybrid algorithm

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A hybrid algorithm that uses bisection and Newton's method
% to locate a root within a given interval [xmin, xmax]. If the
% number of iterations exceeds for either bisection or Newton's
% method returns 50, the function returns NaN.
%
% Arguments:
%  f:       Function whose root is sought (takes one argument).
%  dfdx:    Derivative function (takes one argument).
%  xmin:    Initial bracket minimum.
%  xmax:    Initial bracket maximum.
%  tol1:    Relative convergence criterion for bisection.
%  tol2:    Relative convergence criterion for Newton iteration.
% Returns:
%  x:       Estimate of root.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = hybrid(f, dfdx, xmin, xmax, tol1, tol2)
    overflow_counter = 0;
    MAX_ITERATIONS = 50;

    % Bisection:
    converged = false;
    fmin = f(xmin);
    while not(converged)
        xmid = (xmin + xmax)/2;
        fmid = f(xmid);
        if fmid == 0
            break
        elseif fmid*fmin < 0
            xmax = xmid;
        else
            xmin = xmid;
            fmin = fmid;
        end
        if (xmax - xmin)/abs(xmid) < tol1
            converged = true;
        end

        overflow_counter = overflow_counter + 1;
        if overflow_counter == MAX_ITERATIONS
            x = NaN;
            return;
        end
    end
    bisection_result = xmid;

    overflow_counter = 0;

    % Newton's method:
    converged = false;
    x = bisection_result;
    xprev = bisection_result;
    while not(converged)
        x = xprev - f(xprev)/dfdx(xprev);
```

```
        if abs((x - xprev)/xprev) < tol2
            converged = true;
        end
        xprev = x;

        overflow_counter = overflow_counter + 1;
        if overflow_counter == MAX_ITERATIONS
            x = NaN;
            return;
        end
    end

end
```

Appendix B - D-dimesional Newton's Method Code

```
%% Problem 2 - D-dimensional Newton iteration

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Newton's method for a d-dimensional space. If the number of
% iterations exceeds 50, the function returns NaN.
%
% Arguments:
%  f:    Function which implements the nonlinear system of
%        equations. Function is of the form f(x) where x is a
%        length-d vector, and which returns length-d column
%        vector.
%  jac:  Function which is of the form jac(x) where x is a
%        length-d vector, and which returns the d x d matrix of
%        Jacobian matrix elements for the nonlinear system defined
%        by f.
%  x0:   Initial estimate for iteration (length-d column vector).
%  tol:  Convergence criterion: routine returns when relative
%        magnitude of update from iteration to iteration is
%        <= tol.
% Returns:
%  x:    Estimate of root (length-d column vector).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function x = newtond(f, jac, x0, tol)
    overflow_counter = 0;
    MAX_ITERATIONS = 50;

    x = x0;
    res = f(x0);
    dx = jac(x0)\res;
    while rms(dx) > tol
        res = f(x);
        dx = jac(x)\res;
        x = x - dx;

        overflow_counter = overflow_counter + 1;
        if overflow_counter == MAX_ITERATIONS
            x = NaN;
            return;
        end
    end
end
```

Appendix C - Testing Code

```
%% Test script for Problem 1 and Problem 2

close all; clear; clc;

format long;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test Script - Problem 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example polynomial function given in problem 1 of Homework 1
% document.
%
% Arguments:
%   x:  Polynomial independent variable
% Returns:
%   example_f_out:  Function evaluated at x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function example_f_out = example_f(x)
    example_f_out = 512*x^10 - 5120*x^9 + 21760*x^8 - 51200*x^7 + ...
    72800*x^6 - 64064*x^5 + 34320*x^4 - 10560*x^3 + 1650*x^2 - 100*x + 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Derivative of example polynomial function given in problem 1 of
% Homework 1 document.
%
% Arguments:
%   x:  Polynomial independent variable
% Returns:
%   example_dfdx_out:  Derivative evaluated at x
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function example_dfdx_out = example_dfdx(x)
    example_dfdx_out = 20*(-5 + 165*x - 1584*x^2 + 6864*x^3 - 16016*x^4 ...
    + 21840*x^5 - 17920*x^6 + 8704*x^7 - 2304*x^8 + 256*x^9);
end

% Root finding
roots = zeros([1,10]);

BS_tol = 1.0e-4;
NM_tol = 1.0e-12;

roots(1) = hybrid(@example_f, @example_dfdx, 0.0, 0.04, BS_tol, NM_tol);
roots(2) = hybrid(@example_f, @example_dfdx, 0.05, 0.15, BS_tol, NM_tol);
roots(3) = hybrid(@example_f, @example_dfdx, 0.23, 0.35, BS_tol, NM_tol);
roots(4) = hybrid(@example_f, @example_dfdx, 0.47, 0.6, BS_tol, NM_tol);
roots(5) = hybrid(@example_f, @example_dfdx, 0.77, 0.9, BS_tol, NM_tol);
roots(6) = hybrid(@example_f, @example_dfdx, 1.11, 1.22, BS_tol, NM_tol);
roots(7) = hybrid(@example_f, @example_dfdx, 1.65, 1.75, BS_tol, NM_tol);
roots(8) = hybrid(@example_f, @example_dfdx, 1.86, 1.90, BS_tol, NM_tol);
roots(9) = hybrid(@example_f, @example_dfdx, 1.4, 1.5, BS_tol, NM_tol);
roots(10) = hybrid(@example_f, @example_dfdx, 1.98, 2.0, BS_tol, NM_tol);

function_at_roots = transpose(arrayfun(@example_f, roots))
```

```
% Plotting
xvec = linspace(0,2,10000);

fig = figure;
plot(xvec, arrayfun(@example_f, xvec), 'LineWidth', 1, 'DisplayName', 'f(x)');
hold on;
scatter(roots, zeros([1,10]), 'filled', 'DisplayName', 'Calculated roots', 'Color', 'r');
lgd = legend;
ax = gca;
fontsize(lgd,12,'points');
fontsize(ax,12,'points');
title('Bisection and Newton''s Method Hybrid Algorithm Results', 'FontSize', 16);
xlabel('x');
ylabel('y');
grid on;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Test Script - Problem 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example nonlinear system given in problem 2 of Homework 1
% document.
%
% Arguments:
%  x:  Vector of length 3. x, y, z independent variables in the
%      system.
% Returns:
%  example_sys_out:  Column ector of length 3. f1, f2, f3
%      outputs of each function in the system.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function example_sys_out = example_sys(x)
    example_sys_out = zeros(3,1);
    example_sys_out(1) = x(1)^2 + x(2)^4 + x(3)^6 - 2;
    example_sys_out(2) = cos(x(1)*x(2)*x(3)^2) - x(1) - x(2) - x(3);
    example_sys_out(3) = x(2)^2 + x(3)^3 - (x(1) + x(2) - x(3))^2;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Jacobian matrix of example nonlinear system given in problem 2
% of Homework 1 document.
%
% Arguments:
%  x:  Vector of length 3. x, y, z independent variables in the
%      System.
% Returns:
%  example_jac_out:  3x3 matrix. Entries of the Jacobian matrix
%      for f1(x,y,z), f2(x,y,z), f3(x,y,z).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function example_jac_out = example_jac(x)
    example_jac_out(1,1) = 2*x(1);
    example_jac_out(1,2) = 4*x(2)^3;
    example_jac_out(1,3) = 6*x(3)^5;
    example_jac_out(2,1) = -x(2)*x(3)^2*sin(x(1)*x(2)*x(3)^2) - 1;
```

```
    example_jac_out(2,2) = -x(1)*x(3)^2*sin(x(1)*x(2)*x(3)^2) - 1;
    example_jac_out(2,3) = -2*x(1)*x(2)*x(3)*sin(x(1)*x(2)*x(3)^2) - 1;
    example_jac_out(3,1) = -2*x(1)-2*x(2)+2*x(3);
    example_jac_out(3,2) = -2*x(1)+2*x(3);
    example_jac_out(3,3) = 2*x(1)+2*x(2)+3*x(3)^2-2*x(3);
end

% Root finding
initial_guess = [-1.0; 0.75; 1.50];
NM_3D_tol = 1.0e-12;

solution = newtond(@example_sys, @example_jac, initial_guess, NM_3D_tol);

system_at_solution = example_sys(solution)
```