Coding Report 5

June 12, 2023

1 Problem Description

In this report, we focus on the implementation and analysis of two numerical methods, the Fixed Point method and Newton's method, for solving a system of nonlinear equations. The system of equations we are considering is given by:

$$x_1^2 - 10x_1 + x_2^2 + 8 = 0, x_1x_2^2 + x_1 - 10x^2 + 8 = 0$$

Our objective is to iterate through these methods until the residual error in each equation is less than 1e-6. We will measure their performance by computing the total wall clock time, total number of iterations, and present the results in a table. Additionally, we will plot the residual error versus the iteration number for both methods to visualize their convergence behavior. To visualize their performance more clearly, we also have plotted a logarithmic plot. The logarithmic scale allows us to better analyze the trend and convergence performance of each method. Thus, the report will present one table containing the total wall clock time and total number of iterations, as well as two plots: one displaying the residual error versus iteration, and another showing the residual error versus iteration on a logarithmic scale for better analysis.

In the code, we implemented both the Fixed-Point method (referred to as "FixedPointMethod" in the code) and Newton's method (referred to as "Newton" in the code) to solve a given system of equations. Our analysis focused on the operation time, number of iterations, and the computation of residual errors at each step. The system was initialized with an approximation of $x^{(0)} = [0, 0]^T$, and the iterative process continued until the residual errors dropped below the defined tolerance of TOL = 1e-6 or the maximum number of iterations N = 15 was reached. To ensure accurate timing measurement, the code for computing residual errors was commented out, allowing us to assess the computational performance of the methods alone. This approach guarantees that the timing solely reflects the computational steps involved in the methods themselves.

2 Results

By observing **Table 1**, we can analyze the overall wall clock time and the total number of iterations for both the Fixed-Point method and Newton's method. Comparing the two methods shows that the Fixed-Point method has a shorter execution time than Newton's method. However, the Fixed-Point method needs 13 iterations, while Newton's method only requires 5 iterations. The reason behind this difference is that in Newton's method, we need to compute the partial derivatives, which usually takes longer time.

Method Name	**Time**	**Iteration**
Fixed-Point Method	0.0030401 seconds	13
Netwon's Method	0.013579 seconds	5

Table 1: One table containing the total wall clock time and total number of iterations for Fixed-Point method and Newton's method.

In Figure 1, we can observe the plot displaying the relationship between the iteration number (k) and the residual errors $\left\|x^{(k)} - x^{(k-1)}\right\|_\infty$. Both the blue and red lines represent the Fixed-Point method and Newton's method, respectively. We can see that both methods start at the same point and show similar trends throughout the iterations. The residual errors consistently decrease for both methods, indicating the convergence of the iterative algorithms. However, Newton's method converges at a faster rate than the Fixed-Point method due to its steeper slope. To visualize their performance more clearly, we have plotted a logarithmic plot in the following page. The logarithmic scale allows us to better analyze the trend and convergence performance of the methods.

In Figure 2, the log plot provides a clearer visualization of the convergence behavior and allows us to compare the convergence rates of the Fixed-Point method and Newton's method. The Newton's method, as observed in the Log plot, has the fastest convergence rate due to the steeper slope.

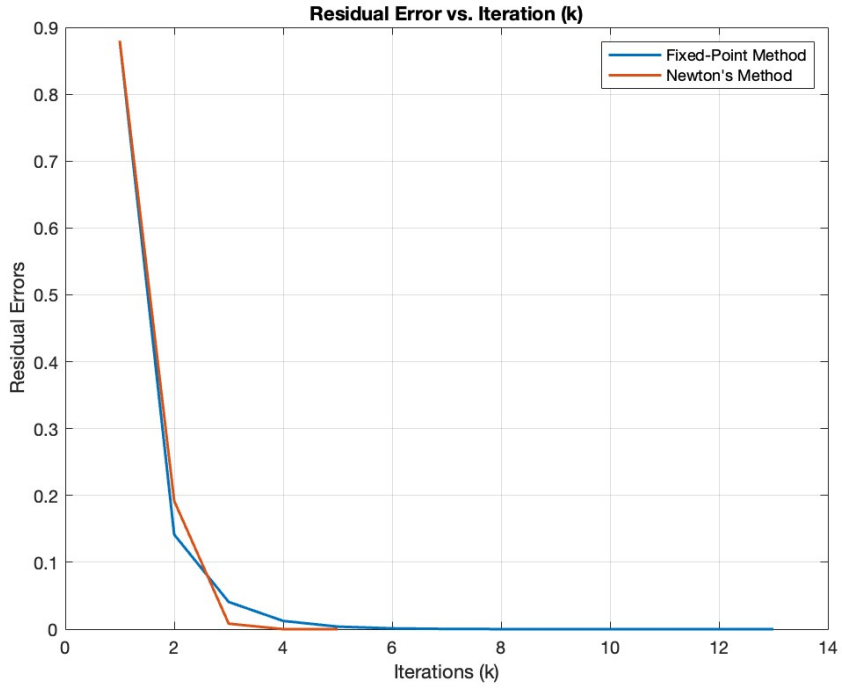

Figure 1: Plot of Iteration numvber (k) vs. residual error for Fixed-Point method and Newton's method.

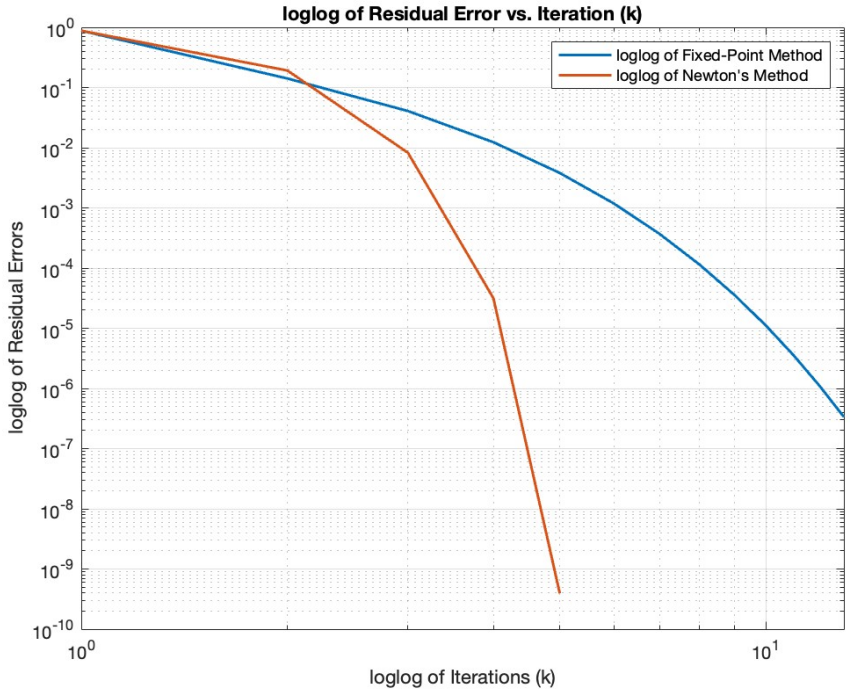

Figure 2: Log-log plot showing the number of iterations number (k) vs. the residual error for both the Fixed-Point method and Newton's method.

3 Collaboration

Yirong Xu and Sebrina Zhu

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not given nor received any unauthorized assistance on this assignment.

5 Appendix

```
clc; clear; close all;

x = [0; 0]; % Initial approximation
TOL = 1e-6;
N = 15;

tic;
[x_fp, n_fp] = FixedPointMethod(x, TOL, N);
fp_time = toc;

tic;
[x_new, n_new] = Newton(x, TOL, N);
new_time = toc;

% ----- Create a table for total time and iterations------
method = ["Fixed-Point Method"; "Newton's Method"];
time = [fp_time; new_time];
iterations = [n_fp; n_new];
T = table(method, time, iterations);
disp(T);

% ------ Plot residual error vs. iteration ----
figure;
fixedpoint_x = 1:n_fp;
[~, ~, fp_err] = FixedPointMethod(x, TOL, N);
plot(fixedpoint_x, fp_err, "LineWidth", 1.5);
hold on;
new_x = 1:n_new;
[~, ~, new_err] = Newton(x, TOL, N);
plot(new_x, new_err, "LineWidth", 1.5);
hold on;
grid on;
xlabel('Iterations (k)');
ylabel('Residual Errors');
legend("Fixed-Point Method", "Newton's Method");
title('Residual Error vs. Iteration (k)')

% ----- Log-log plot -------
figure;
loglog(fixedpoint_x, fp_err, "LineWidth", 1.5);
hold on;
loglog(new_x, new_err, "LineWidth", 1.5);
hold on;
grid on;
xlabel('loglog of Iterations (k)');
ylabel('loglog of Residual Errors');
```

```
legend("loglog of Fixed-Point Method", "loglog of Newton's
   Method");
title('loglog of Residual Error vs. Iteration (k)')


% ----- FixedPointMethod -------
function [x, k, err] = FixedPointMethod(x, TOL, N)
    k = 1;
    err = [];

    while k <= N
        y = x;
        x(1) = (x(1)^2 + x(2)^2 + 8) / 10;
        x(2) = (x(1) * x(2)^2 + x(1) + 8) / 10;

        err(end+1) = norm(x - y, inf);

        if norm(x - y, inf) < TOL
            return;
        end

        k = k + 1;
    end

    fprintf("Maximum number of iterations exceeded for the
       Fixed-Point Method\n");
end

% ----- Newton's Method --------
function [x, k, err] = Newton(x, TOL, N)
    k = 1;
    err = [];

    while k <= N
        F_x = my_fun(x(1), x(2));
        J_x = Jacobian(x(1), x(2));

        [L, U] = LUFactorization(J_x);
        L_v = [L, -F_x];
        L_v_sol = ForSubstitution(2, L_v);
        U_y = [U, L_v_sol];

        y = BackSubstitution(2, U_y);
        x = x + y;

        err(end+1) = norm(y, Inf);

        if norm(y) < TOL
```

```
            return;
        end

        k = k + 1;
    end

    fprintf("Maximum number of iterations exceeded for Newton
       's Method\n");
end

function F = my_fun(x1, x2)
    f1 = x1^2 - 10 * x1 + x2^2 + 8;
    f2 = x1 * x2^2 + x1 - 10 * x2 + 8;
    F = [f1; f2];
end

function J = Jacobian(x1, x2)
    f11 = 2 * x1 - 10;
    f12 = 2 * x2;
    f21 = x2^2 + 1;
    f22 = 2 * x1 * x2 - 10;
    J = [f11, f12; f21, f22];
end

% ----- Back Substitution -------
function x = BackSubstitution(n, b)
    x = zeros(n, 1);
    x(n, 1) = b(n, n+1) / b(n, n);

    for i = n-1:-1:1
        sum = 0;

        for j = i+1:n
            sum = sum + b(i, j) * x(j, 1);
        end

        x(i, 1) = (b(i, n+1) - sum) / b(i, i);
    end
end

% ----- Forward Substitution -------
function x = ForSubstitution(n, b)
    x = zeros(n, 1);
    x(1, 1) = b(1, n+1) / b(1, 1);

    for i = 2:n
        sum = 0;
```

```
        for j = 1:i-1
            sum = sum + b(i, j) * x(j, 1);
        end

        x(i, 1) = (b(i, n+1) - sum) / b(i, i);
    end
end

%--------LU factorization-------------------
function [L, U] = LUFactorization(A)
    n = size(A,1);
    % Initialize L and U matrices
    L = eye(n);
    U = A;
    % Gaussian elimination without pivoting
    for k = 1:n-1
        if U(k,k) == 0
            disp('Factorization impossible');
        end
        for i = k+1:n
            factor = U(i,k) / U(k,k);
            L(i,k) = factor;
            U(i,k:n) = U(i,k:n) - factor * U(k,k:n);
        end
    end
end
```