Coding Report 5

Joe Lenning

1 Problem Description

This report presents the results of applying the Fixed Point Method and Newton's Method to non-linear systems. The main difference between finding solutions to linear and nonlinear systems is Linear systems have well-defined and straightforward methods to solve them. They can be solved using substitution, elimination, or matrix methods like Gaussian elimination. These methods provide unique solutions (if one exists) or determine that the system is inconsistent (no solution) or has infinite solutions. In contrast, nonlinear systems are generally more challenging to solve analytically. The methods listed below are some of the ways we can solve nonlinear systems.

Fixed Point Method

The fixed-point method starts with an initial guess or approximation for the solution of the nonlinear system. The original system of equations is transformed into a fixed-point equation of the form x = G(x), where G(x) represents a function derived from the original system. This equation establishes an iterative process, where the value of x in each iteration is updated based on function G. Starting from the initial approximation, the method iteratively applies the fixed-point equation x = G(x) to obtain improved approximations for x. In each iteration, the current value of x is substituted into G(x) to calculate the next approximation. This process is repeated until convergence is achieved.

Newtons Method

Similar to the Newtons method we saw earlier in class, in this version of Newtons we first start by expressing the given nonlinear system of equations as a set of equations of the form F(x) = 0, where F is a vector-valued function and x represents the vector of unknowns. Then Compute the Jacobian matrix J, which is the matrix of partial derivatives of the function F(x) with respect to the variables x. The Jacobian matrix provides information about the local linear behavior of the system near a particular point. We then use our initial approximation and begin to iterate through the system until we find a convergent answer.

2 Results

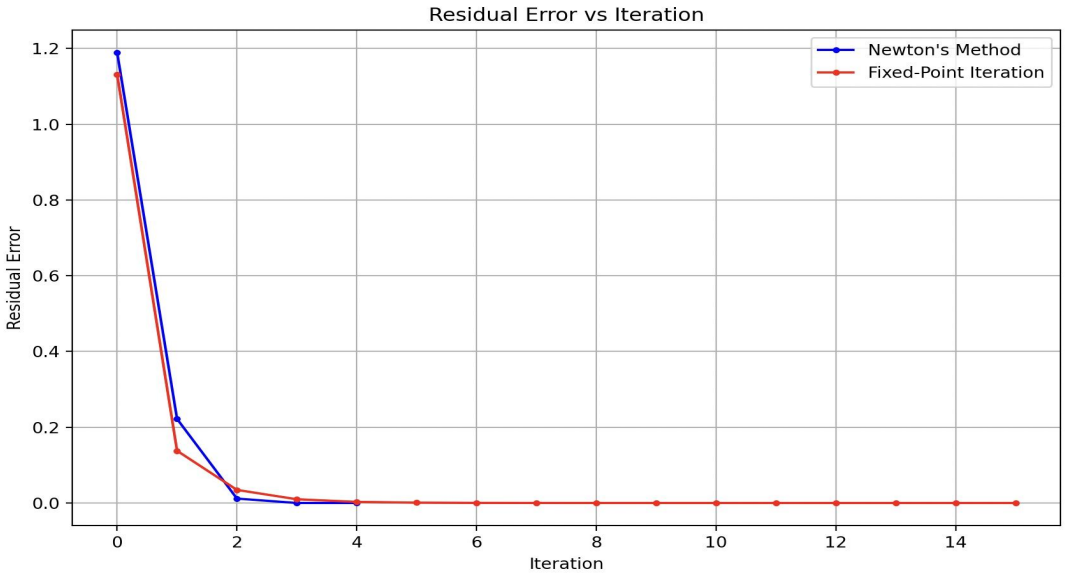
For our given system:

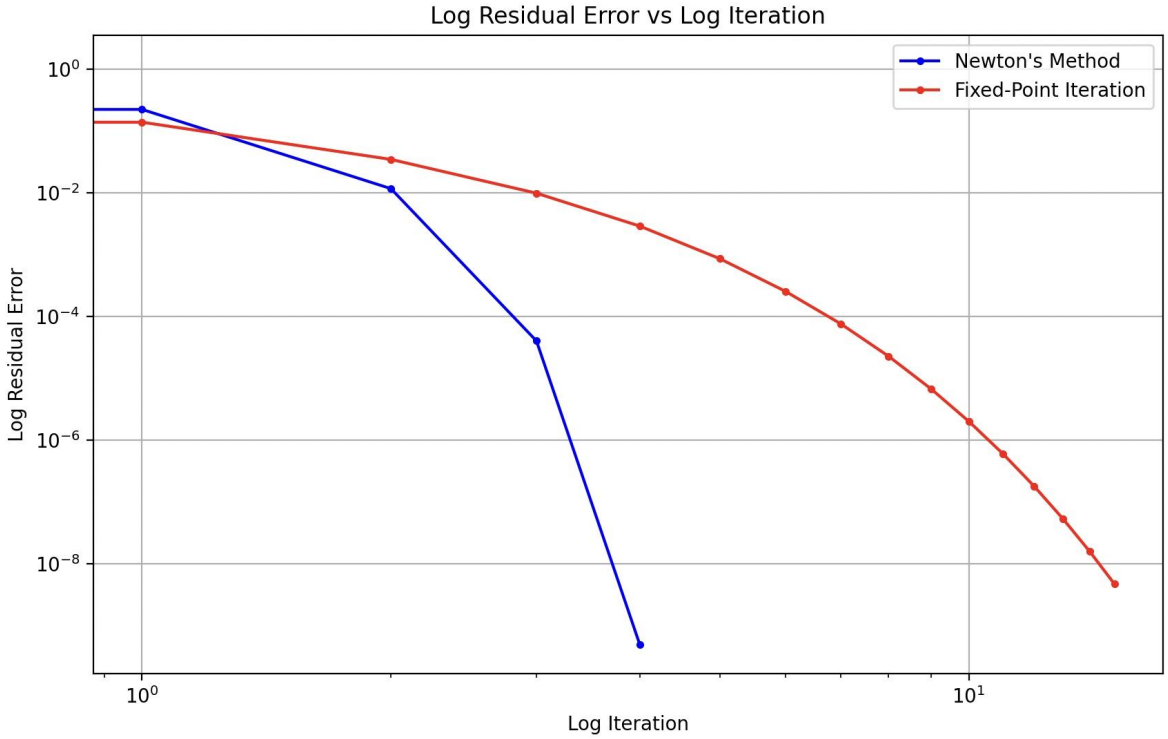
$x_1^2 - 10x_1 + x_2^2 + 8 = 0$ and $x_1 x_2^2 + x_1 - 10x_2 + 8 = 0$

We first Solve for x1 and x2

$$x_1 = \frac{x_1^2 + x_2^2 + 8}{10} \; and \quad x_2 = \frac{x_1 x_2^2 + x_1 + 8}{10}$$

And begin the iteration processes we mentioned above. The results on the graph are noted below.





From the implementation of the Fixed Point Method, we observed that it converged to approximate solutions of x1 = 0.970 and x2 = 0.874 in approximately 15 iteraions. These solutions provided a reasonably accurate approximation to the actual solutions of the system. On

the other hand, Newton's Method exhibited remarkable convergence and was able to achieve the exact solutions of x1 = 1 and x2 = 1 only in 4 iterations. Analyzing the graphs depicting the error versus iteration for both methods, we can clearly observe the contrasting convergence behaviors. Newton's Method displayed a rapid reduction in error with each iteration, indicating its superior convergence speed. The error decreased significantly in just a few iterations, suggesting that Newton's Method quickly approached the solutions. In contrast, the Fixed Point Method showed a slower convergence rate, as the error decreased gradually with each iteration. Despite Newton's Method converging faster, it is essential to note that it required more computational resources and time to converge compared to the Fixed Point Method. Newton's Method involves the computation of the Jacobian matrix and solving a linear system of equations at each iteration, making it more computationally intensive. This additional computational burden resulted in a longer elapsed time for convergence. To conclude, from the implementation of the code, we observed that Newton's Method exhibited faster convergence and achieved the exact solutions of the system. However, this faster convergence came at the cost of increased computational complexity and longer elapsed time. On the other hand, the Fixed Point Method provided reasonable accuracy but fell short of reaching the exact solutions. The choice between the two methods depends on the desired level of accuracy, the trade-off between convergence speed and computational complexity, and the available computational resources.

3 Collaboration:

None

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 Attachments

```
import time
import numpy as np
import matplotlib.pyplot as plt

# let x1= x and x2= y
def f1(x, y):
   return x**2 - 10*x + y**2 + 8

def f2(x, y):
   return x * (y**2) + x - 10*y + 8

def g1(x, y):
   return (x ** 2 + y ** 2 + 8) / 10
```

```
def g2(x, y):
   return (x * (y ** 2) + + 8) / 10

# Define the derivatives
def f1_derivative_x(x, y):
   return 2*x - 10

def f1_derivative_y(x, y):
   return 2*y

def f2_derivative_x(x, y):
   return y**2 + 1

def f2_derivative_y(x, y):
   return 2*x*y - 10

def fixedPointIteration(x0, y0, tol, max_iterations):
   start_time = time.time()
   iterations = 0
   errors = []
   for i in range(max_iterations):
       x = g1(x0, y0)
       y = g2(x0, y0)
       my_error = np.linalg.norm(np.array([x, y]) - np.array([x0, y0]))
       errors.append(my_error)
       if my_error < tol:
           end_time = time.time()
           elapsed_time = end_time - start_time
           return x, y, errors, iterations, elapsed_time
       x0 = x
       y0 = y
       iterations += 1
   end_time = time.time()
   elapsed_time = end_time - start_time
   return x, y, errors, iterations, elapsed_time

# Set the initial guess, tolerance, and maximum iterations
initial_guess = (0, 0)
tolerance = 1e-8
max_iterations = 100
def newtonMethod(x0, y0, tol, max_iterations):
   start_time = time.time()
   iterations = 0
   errors = []
   for i in range(max_iterations):
       J = np.array([[f1_derivative_x(x0, y0), f1_derivative_y(x0, y0)],
                     [f2_derivative_x(x0, y0), f2_derivative_y(x0, y0)]])
       F = np.array([-f1(x0, y0), -f2(x0, y0)])
       delta = np.linalg.solve(J, F)
```

```
        x = x0 + delta[0]
        y = y0 + delta[1]
        my_error = np.linalg.norm(np.array([x, y]) - np.array([x0, y0]))
        errors.append(my_error)
        if my_error < tol:
            end_time = time.time()
            elapsed_time = end_time - start_time
            return x, y, errors, iterations, elapsed_time
        x0 = x
        y0 = y
        iterations += 1
    end_time = time.time()
    elapsed_time = end_time - start_time
    return x, y, errors, iterations, elapsed_time

# Call the Newton's method and fixed-point iteration functions with the
updated arguments
newton_solution1, newton_solution2, newton_errors, newton_num_iterations,
newton_elapsed_time = newtonMethod(initial_guess[0], initial_guess[1],
tolerance, max_iterations)
fixed_solution1, fixed_solution2, fixed_errors, fixed_num_iterations,
fixed_elapsed_time = fixedPointIteration(initial_guess[0], initial_guess[1],
tolerance, max_iterations)

# Print the results
print("Newton's Method:")
print("Solution 1:", newton_solution1)
print("Solution 2:", newton_solution2)
print("Number of iterations:", newton_num_iterations)
print("Elapsed time:", newton_elapsed_time, "seconds")
print("Fixed-Point Iteration:")
print("Solution 1:", fixed_solution1)
print("Solution 2:", fixed_solution2)
print("Number of iterations:", fixed_num_iterations)
print("Elapsed time:", fixed_elapsed_time, "seconds")

# Plotting the error versus iteration for both methods
newton_iterations = np.arange(newton_num_iterations + 1)
fixed_iterations = np.arange(fixed_num_iterations + 1)

plt.figure(figsize=(10, 6))
plt.plot(newton_iterations, newton_errors, 'b.-', label="Newton's Method")
plt.plot(fixed_iterations, fixed_errors, 'r.-', label="Fixed-Point Iteration")
plt.xlabel('Iteration')
plt.ylabel('Residual Error')
plt.title('Residual Error vs Iteration')
plt.grid(True)
plt.legend()
plt.show()
```

```
# Plotting the error versus iteration on a log-log scale
plt.figure(figsize=(10, 6))
plt.loglog(newton_iterations, newton_errors, 'b.-', label="Newton's Method")
plt.loglog(fixed_iterations, fixed_errors, 'r.-', label="Fixed-Point
Iteration")
plt.xlabel('Log Iteration')
plt.ylabel('Log Residual Error')
plt.title('Log Residual Error vs Log Iteration')
plt.grid(True)
plt.legend()
plt.show()
```