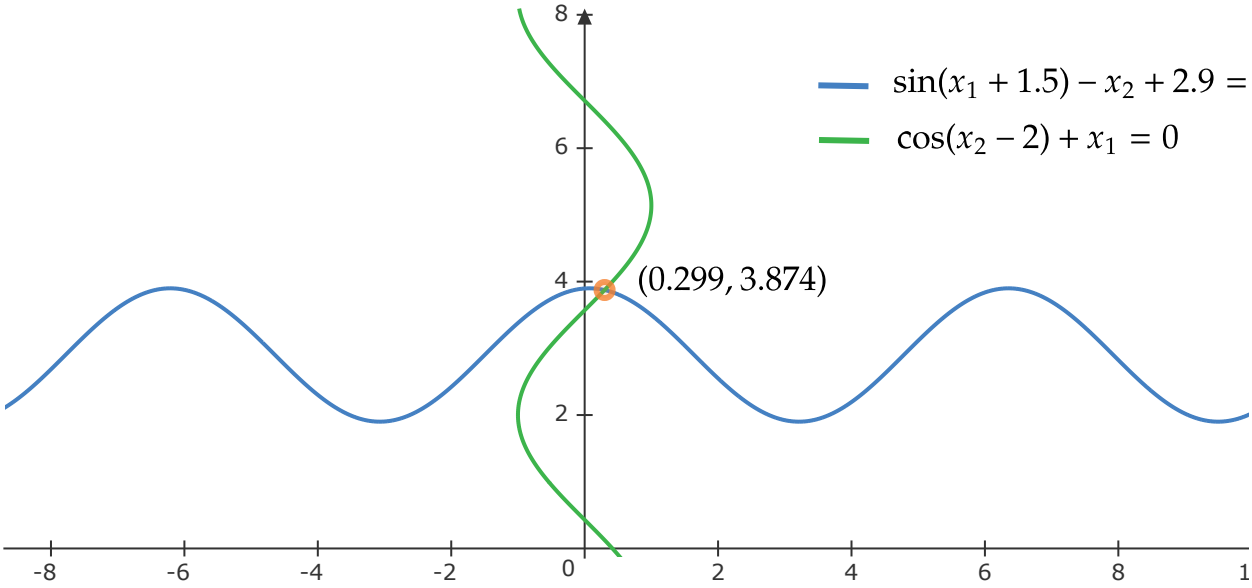
Assignment

Algorithmic and Computational Mathematics

For the system of nonlinear equations

$$\sin(x_1 + 1.5) - x_2 + 2.9 = 0$$
$$\cos(x_2 - 2) + x_1 = 0$$

1. Localise the roots of the system graphically.



The resulting graphs of the two curves of the two equations and indicate the points where they intersect, representing the possible roots of the system. The possible roots of the two curves should be $(0.299,\ 3.874)$.

2. Give the description of the Newton's method of solving systems of nonlinear equations.

Newton's method, also known as the Newton-Raphson method, is an iterative numerical technique used to solve systems of nonlinear equations. The Newton's method for solving systems of nonlinear equations is an extension of the one-dimensional Newton's method for finding roots of a single equation. Instead of solving a single equation, it aims to find the simultaneous solutions to a set of nonlinear equations. The method iteratively refines an initial guess to converge towards the true solution.

1. Formulate the system of equations:

Start with a system of equations in the form $\mathbf{F}(\mathbf{x}) = 0$, where $\mathbf{F}(\mathbf{x})$ represents a vector-valued function and $\mathbf{x}$ is the vector of unknowns.

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \ \mathbf{F}(\mathbf{x}) = \begin{pmatrix} f_1(x_1, x_2 \ \cdots \ x_n) \\ f_2(x_1, x_2 \ \cdots \ x_n) \\ \vdots \\ f_n(x_1, x_2 \ \ldots \ x_n) \end{pmatrix}$$

2. Choose an initial guess:

Start with an initial guess for the solution vector $\mathbf{x}^{(0)}$.

3. Evaluate the Jacobian matrix:

Calculate the Jacobian matrix $\mathbf{J}$, which contains the first-order partial derivatives of the vector function $\mathbf{F}(\mathbf{x})$ with respect to each variable in $\mathbf{x}$

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_1} & & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

4. Solve the linear system:

Solve the linear system $\mathbf{J}\left(\mathbf{x}^{(k)}\right)\delta\mathbf{x}^{(k)} = -\mathbf{F}\left(\mathbf{x}^{(k)}\right)$, where $\mathbf{J}\left(\mathbf{x}^{(k)}\right)$ is the Jacobian matrix evaluated at the kth iteration, $\delta\mathbf{x}^{(k)}$ is the correction vector, and $\mathbf{F}\left(\mathbf{x}^{(k)}\right)$ is the vector function evaluated at the kth iteration.

Since $\mathbf{J}$ is a matrix, instead of dividing by $f'(x)$ we multiply by $\mathbf{J}^{-1}$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathbf{J}^{-1}\left(\mathbf{x}^{(k)}\right)\mathbf{F}\left(\mathbf{x}^{(k)}\right)$$

$$\mathbf{J}\left(\mathbf{x}^{(k)}\right)\delta\mathbf{x}^{(k)} = -\mathbf{F}\left(\mathbf{x}^{(k)}\right)$$

where, $\delta\mathbf{x}^{(k)} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$
using an initial guess $\mathbf{x}^{(0)}$.

5. Update the solution:

Update the solution estimate by computing $\mathbf{x}^{(k+1)} = \mathbf{x}^{k} + \delta\mathbf{x}^{k}$

6. Check convergence

Check if the solution has converged. If the change in the solution is below a specified tolerance or the maximum number of iterations is reached, terminate the process and consider $\mathbf{x}^{(k+1)}$ as the solution. Otherwise, go back to step 3 and repeat the process with the updated solution estimate.

7. Repeat steps 3 to 6:

Iterate until convergence is achieved or until the maximum number of iterations is reached.

3. Solve the system using Newton's method with the accuracy ε = 10−6.

```
#include <iostream>
#include <cmath>

using namespace std;

// Define the vector-valued function
void F(const double x1, const double x2, double& f1, double& f2) {
    f1 = sin(x1 + 1.5) - x2 + 2.9;
    f2 = cos(x2 - 2) + x1;
}

// Define the Jacobian matrix
void Jacobian(const double x1, const double x2, double& j11, double&
j12, double& j21, double& j22) {
    j11 = cos(x1 + 1.5);
```

```
    j12 = -1.0;
    j21 = 1.0;
    j22 = -sin(x2 - 2);
}

// Solve the linear system J(x) * Δx = -F(x) using Gaussian elimination
void SolveLinearSystem(const double j11, const double j12, const double
j21, const double j22,
                       const double f1, const double f2, double& dx1,
double& dx2) {
    double det = j11 * j22 - j12 * j21;
    dx1 = (j22 * f1 - j12 * f2) / det;
    dx2 = (-j21 * f1 + j11 * f2) / det;
}

// Perform Newton's method to solve the system of equations
void NewtonMethod(double& x1, double& x2, const double epsilon, const
int maxIterations) {
    double dx1, dx2;
    double f1, f2;
    double j11, j12, j21, j22;

    for (int i = 0; i < maxIterations; ++i) {
        // Evaluate the function and Jacobian at the current point
        F(x1, x2, f1, f2);
        Jacobian(x1, x2, j11, j12, j21, j22);

        // Solve the linear system J(x) * Δx = -F(x)
        SolveLinearSystem(j11, j12, j21, j22, f1, f2, dx1, dx2);

        // Update the solution estimate
        x1 -= dx1;
        x2 -= dx2;

        // Check convergence
        double norm = sqrt(dx1 * dx1 + dx2 * dx2);
        if (norm < epsilon) {
            cout << "Converged after " << i+1 << " iterations." << endl;
```

```
            return;
        }
    }

    cout << "Did not converge after " << maxIterations << "
iterations." << endl;
}

int main() {
    // Set initial guess, epsilon, and maximum iterations
    double x1 = 1.0;
    double x2 = 2.0;
    double epsilon = 1e-6;
    int maxIterations = 100;

    // Solve the system using Newton's method
    NewtonMethod(x1, x2, epsilon, maxIterations);

    // Print the solution
    cout << "Solution: x1 = " << x1 << ", x2 = " << x2 << endl;

    return 0;
}
```

Output: