For the source code for the coding problems, see the attached Jupyter notebook, Matlab live script. Alternatively, the entire git repository is attached as a zip archive, and is available on GitHub. The comments in the code have been omitted here for brevity. They are present in the Jupyter notebook and Matlab live script.

(1) **Problem Statement:** Perform Gaussian Elimination in order to solve the system $Ax = b$.

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right] \begin{array}{l} \text{R2} = \text{R2} + \frac{3}{2}\text{R1} \\ \text{R3} = \text{R3} + \text{R1} \end{array}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 2 & 1 & 5 \end{array}\right] \text{R3} = \text{R3} - 4\text{R2}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

$$\begin{aligned} -1x_3 &= 1 \\ x_3 &= -1 \end{aligned}$$

$$\begin{aligned} \frac{1}{2}x_2 + \frac{1}{2}x_3 &= 1 \\ \frac{1}{2}x_2 - \frac{1}{2} &= 1 \\ \frac{1}{2}x_2 &= \frac{3}{2} \\ x_2 &= 3 \end{aligned}$$

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 8 \\ 2x_1 + 3 + 1 &= 8 \\ 2x_1 &= 4 \\ x_1 &= 2 \end{aligned}$$

(2) **Problem Statement:** Consider the following linear system. Perform row and column swapping to create a system which is diagonally dominant.

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ -7 & -3 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & -4 \\ 0 & 0 & 10 & 2 & -2 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} -8 \\ -7 \\ 3 \\ 3 \\ -5 \end{bmatrix}$$

$$\left[\begin{array}{ccccc|c} 1 & 2 & 0 & 0 & 0 & -8 \\ -7 & -3 & 0 & 0 & 0 & -7 \\ 0 & 0 & 2 & 0 & -4 & 3 \\ 0 & 0 & 10 & 2 & -2 & 3 \\ 0 & 0 & 0 & 1 & 0 & -5 \end{array}\right] \begin{array}{l} \\ \\ \text{swap(R1, R2)} \\ \text{swap(R3, R4)} \\ \\ \end{array}$$

$$\left[\begin{array}{ccccc|c} -7 & -3 & 0 & 0 & 0 & -7 \\ 1 & 2 & 0 & 0 & 0 & -8 \\ 0 & 0 & 10 & 2 & -2 & 3 \\ 0 & 0 & 2 & 0 & -4 & 3 \\ 0 & 0 & 0 & 1 & 0 & -5 \end{array}\right] \begin{array}{l} \\ \\ \\ \\ \text{swap(C4, C5)} \end{array}$$

$$\left[\begin{array}{ccccc|c} -7 & -3 & 0 & 0 & 0 & -7 \\ 1 & 2 & 0 & 0 & 0 & -8 \\ 0 & 0 & 10 & -2 & 2 & 3 \\ 0 & 0 & 2 & -4 & 0 & 3 \\ 0 & 0 & 0 & 0 & 1 & -5 \end{array}\right]$$

(3) **Problem Statement:** Determine L and U such that $A = LU$.

$$A = \begin{bmatrix} 5 & 4 & 2 \\ 4 & 2 & 4 \\ 2 & 1 & 4 \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\ell_{11} = 5$$

$$\ell_{21} = 4$$

$$\ell_{31} = 2$$

$$\ell_{11}u_{13} = 2$$
$$5u_{13} = 2$$
$$u_{13} = 0.4$$

$$\ell_{11}u_{12} = 4$$
$$5u_{12} = 4$$
$$u_{12} = 0.8$$

$$\ell_{21}u_{12} + \ell_{22} = 2$$
$$4(0.8) + \ell_{22} = 2$$
$$\ell_{22} = -1.2$$

$$\ell_{21}u_{13} + \ell_{22}u_{23} = 4$$
$$4(0.4) - 1.2u_{23} = 4$$
$$-1.2u_{23} = 2.4$$
$$u_{23} = -2$$

$$\begin{aligned} \ell_{31}u_{12} + \ell_{32} &= 1 \\ 2(0.8) + \ell_{32} &= 1 \\ \ell_{32} &= -0.6 \end{aligned}$$

$$\begin{aligned} \ell_{31}u_{13} + \ell_{32}u_{23} + \ell_{33} &= 4 \\ 2(0.4) - 0.6(-2) + \ell_{33} &= 4 \\ 0.8 + 1.2 + \ell_{33} &= 4 \\ \ell_{33} &= 2 \end{aligned}$$

$$L = \begin{bmatrix} 5 & 0 & 0 \\ 4 & -1.2 & 0 \\ 2 & -0.6 & 2 \end{bmatrix} \qquad U = \begin{bmatrix} 1 & 0.8 & 0.4 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

(4) **Problem Statement:** Write a function called `jacobi` that has as inputs an $n \times n$ matrix, A, a column vector, b, an initial guess $x^(0)$, an error tolerance ϵ, and a maximum number of iterations, and as outputs an approximate solution obtained using the Jacobi method, the residual error and the number of iterations. Use the method to find approximate solutions to the linear system from problem 2 after you have made it diagonally dominant. If you chose not to complete problem 2, you may use the 5×5 matrix of your choice. Find the solution within an accuracy of $\epsilon = 10^{-5}$, with a maximum of $N = 100$ iterations. If your method succeeds, report the number of iterations needed. If your method fails, offer a possible reason why.

```python
def jacobi(A, b, x0, epsilon, max_):
    n = len(A)
    x = np.copy(x0)
    D = np.diag(A)
    R = A - np.diagflat(D)
    iterations = 0
    for _ in range(max_):
        x_new = (b - np.dot(R, x)) / D
        residual = np.linalg.norm(x_new - x, ord=np.inf)
        if(residual < epsilon):
            break
        x = x_new
        iterations += 1
    final_residual = np.linalg.norm(np.dot(A, x) - b, ord=np.inf)
    return(x, final_residual, iterations)
#
```

LISTING 1. Python

It took 17 iterations.

(5) **Problem Statement:** Write a function called `gauss_seidel` that has as inputs an $n \times n$ matrix, A, a column vector, b, an initial guess $x^{(0)}$, an error tolerance ϵ, and a maximum number of iterations, and as outputs an approximate solution obtained using the Gauss-Seidel method, the residual error and the number of iterations. Use the method to find approximate solutions to the linear system from the previous problem to within an accuracy of $\epsilon = 10^{-5}$, with a maximum of $N = 100$ iterations. If your method succeeds, report the number of iterations needed. If your method fails, offer a possible reason why.

```python
def gauss_seidel(A, b, x0, epsilon, max_):
    n = len(A)
    x = x0.copy()
    iterations = 0
    for _ in range(max_):
        x_old = x.copy()
        for i in range(n):
            sigma = sum(A[i][j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sigma) / A[i][i]
        error = np.linalg.norm(x - x_old, ord=2)
        iterations += 1
        if(error < epsilon):
            break
    return(x, error, iterations)
```

Listing 2. Python

It took 10 iterations.