

$$\begin{aligned} 5x_1 + 1x_2 - 0.5x_3 &= 13.5 \\ -6x_1 - 12x_2 + 4x_3 &= -123 \\ 2x_1 + 2x_2 + 10x_3 &= -43 \end{aligned}$$

Q1

$$\begin{bmatrix} 5 & 1 & -0.5 & 13.5 \\ -6 & -12 & 4 & -123 \\ 2 & 2 & 10 & -43 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -2 & -1 & 2.7 \\ -6 & -12 & 4 & -123 \\ 2 & 2 & 10 & -43 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.2 & -1 & 2.7 \\ 0 & -10.8 & 3.4 & -106.8 \\ 0 & 1.6 & 10.2 & -48.4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.2 & -1 & 2.7 \\ 0 & 1 & -3.125 & 9.888 \\ 0 & 1.6 & 10.2 & -48.4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -0.37 & 7.22 \\ 0 & 1 & -3.15 & 9.88 \\ 0 & 0 & 10.74 & -64.22 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & -0.37 & 7.222 \\ 0 & 1 & -3.15 & 9.88 \\ 0 & 0 & 1 & -6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 8 \\ 0 & 0 & 1 & -6 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 1 & -5 \\ -6 & -12 & 4 \\ 2 & 2 & 10 \end{bmatrix} \begin{bmatrix} -5 \\ 8 \\ 6 \end{bmatrix} = \begin{bmatrix} 13.5 \\ -123 \\ -43 \end{bmatrix}$$

$$X = \begin{bmatrix} .5 \\ 8 \\ -6 \end{bmatrix}$$

```

import numpy as np

array = np.array([[5, 1, -.5, 13.5],
                  [-6, -12, 4, -123],
                  [2, 2, 10, -43]])

def reduce(array, printSteps=False):
    rows = len(array)
    cols = len(array[0])
    if array[0][0] == 0:
        index = 1
        while array[index][0] == 0:
            index += 1
            if index > len(array):
                raise KeyError
        temp = array[0]
        array[index] = array[0]
        array[0] = temp

    if printSteps:
        print(array)
        print("\nConverting to upper triangular")

    #Convert to RREF
    for k in range(0, rows):
        if printSteps:
            print(f"Pivot row: {k}")
        for i in range(k+1, rows):
            s = array[i][k] / array[k][k]
            for j in range(k, cols):
                array[i][j] = array[i][j] - s * array[k][j]
        if printSteps:
            print(array)
            print()

    if printSteps:
        print("\nScaling rows")

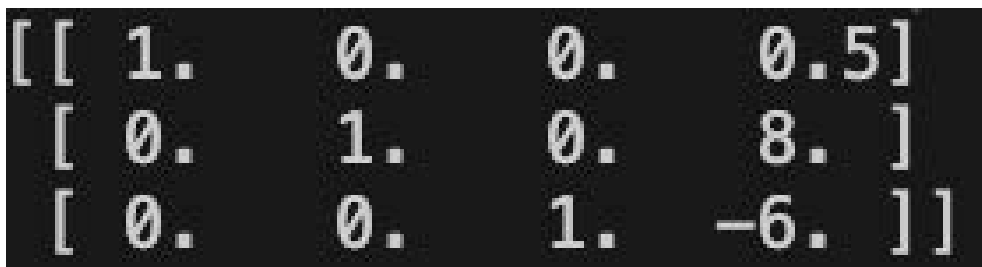
    #Make leading values equal to one
    for m in range(0, rows):
        leadingValue = array[m][m]
        for n in range(m, cols):
            array[m][n] /= leadingValue
        if printSteps:
            print(array)
            print()

    if printSteps:
        print("\nEliminating rows")

    #Eliminate other rows
    for k in range(1, rows):
        #All rows other than m
        if printSteps:
            print(f"Pivot row: {k}")
        for m in list(range(0,k)) + list(range(k+1,rows)):
            s = array[m][k]/array[k][k]
            for n in range(k,cols):
                array[m][n] -= s*array[k][n]
        if printSteps:
            print(array)
            print()
    return array

print(reduce(array))

```



```

[[ [ 1.  0.  0.  0.5]
   [ 0.  1.  0.  8. ]
   [ 0.  0.  1. -6. ]]]

```

Since my code and solution by hand produce the same results my code is verified. My conceptual model is proven to work

Q2

$$\begin{bmatrix} 8 & 4 & -1 \\ -2 & 5 & 1 \\ 2 & -1 & 6 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ 1 \end{bmatrix} = \begin{bmatrix} 11 \\ 4 \\ 7 \end{bmatrix}$$

U:

$$\begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & .75 \\ 0 & -2 & 6.75 \end{bmatrix}$$

$$U = \begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & .75 \\ 0 & 0 & 6.5 \end{bmatrix}$$

L:

$$\begin{bmatrix} 1 & 0 & 0 \\ -.14 & 1 & 0 \\ .14 & 0 & 1 \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ -.25 & 1 & 0 \\ .25 & -.333 & 1 \end{bmatrix}$$

$$L \times \delta = b$$

$$\begin{bmatrix} 1 & 0 & 0 \\ -.25 & 1 & 0 \\ -.25 & -.333 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 11 \\ 4 \\ 7 \end{bmatrix}$$

$$\delta_1 = 11$$

$$\delta_2 = 6.75$$

$$\delta_3 = 6.5$$

$$\begin{bmatrix} 8 & 4 & -1 \\ 0 & 6 & .75 \\ 0 & 0 & 6.5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ 1 \end{bmatrix} = \begin{bmatrix} 11 \\ 6.75 \\ 6.5 \end{bmatrix}$$

$$x_3 = 1$$

$$x_2 = 1$$

$$x_1 = 1$$

Verification: My results by hand match the results of my conceptual model, therefore my solution is verified.  
I proved my conceptual model is functional.

```

import numpy as np

array = np.array([[8, 4, -1, 11],
                  [-2, 5, 1, 4],
                  [2, -1, 6, 7]])

def lu_decomposition(A):
    n = len(A)
    # Create zero matrices for L and U
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    # Decomposing matrix into Upper and Lower triangular matrices
    for i in range(n):
        #Find upper triangle
        for k in range(i, n):
            sum_upper = sum(L[i][j] * U[j][k] for j in range(i))
            U[i][k] = A[i][k] - sum_upper

        #Find lower triangle
        for k in range(i, n):
            if i == k:
                L[i][i] = 1 # Diagonal as 1
            else:
                sum_lower = sum(L[k][j] * U[j][i] for j in range(i))
                L[k][i] = (A[k][i] - sum_lower) / U[i][i]

    return L, U

def __forward_substitution(L, U, b):
    n = len(L)
    d = np.zeros(n)
    d[0] = b[0]
    #forward substitution
    for i in range(1, n):
        d[i] = b[i] - (sum(L[i][j]*d[j] for j in range(0, i)))

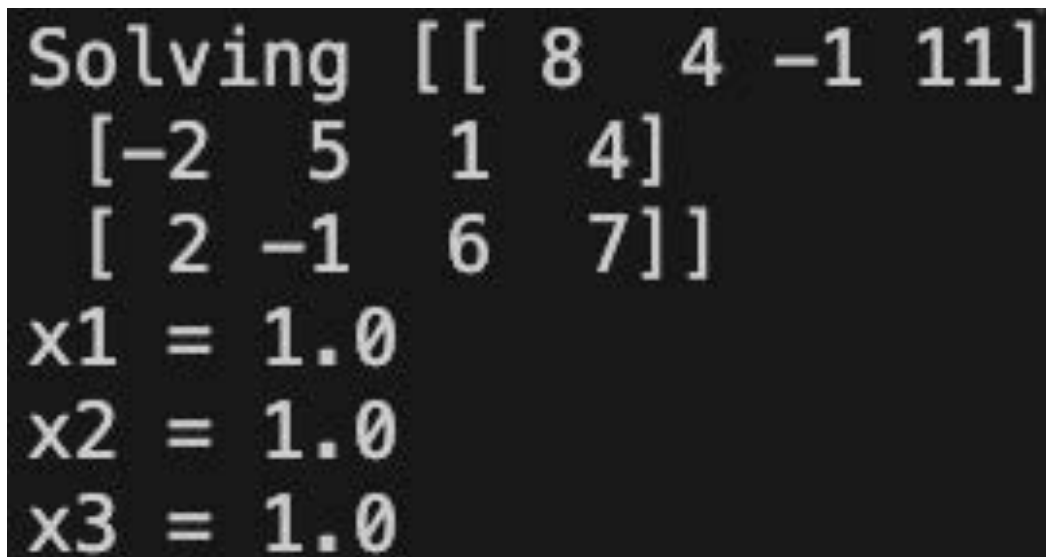
    #backwards substitution
    x = np.zeros(n)
    x[n-1] = d[n-1]/U[n-1][n-1]
    for i in range(n-2, -1, -1):
        x[i] = (d[i] - sum(U[i][j]*x[j] for j in range(i+1, n)))/U[i][i]

    return x

def solve(A):
    b = A[:, -1]
    A = A[:, :-1]
    L, U = lu_decomposition(A)
    return __forward_substitution(L, U, b)

print(f'Solving {array}')
x = solve(array)
print(f'x1 = {x[0]}\nx2 = {x[1]}\nx3 = {x[2]}')

```



```

Solving [[ 8  4 -1 11]
         [-2  5  1  4]
         [ 2 -1  6  7]]
x1 = 1.0
x2 = 1.0
x3 = 1.0

```

```

import numpy as np

A = np.array([[6, -1, -1],
              [6, 9, 1],
              [-3, 1, 12]])
b = np.array([3, 40, 50])

solution = np.linalg.solve(A, b)
print(f'Solution using built-in function: x = {solution}')

def solve(A, b, x, tol=1e-6, max_iter=10):
    n = len(b)
    iter = 0
    while iter < max_iter:
        previous_x = x.copy() # Make a full copy of the current solution
        for i in range(n):
            sigma = 0
            for j in range(n):
                if i != j:
                    sigma += A[i][j] * x[j]
            x[i] = (b[i] - sigma) / A[i][i] # Gauss-Seidel update
        if max(np.abs(np.subtract(previous_x, x))) < tol:
            return x
        iter += 1

    print("Warning: Max iterations exceeded without convergence")
    return x

initial_guess = np.zeros(len(b))
x = solve(A, b, initial_guess, tol=0.0005, max_iter=20)
print(f'Solution using gauss-seidel: x = {x}')

'''If the equations are not reordered, the algorithm is not guaranteed to converge.
This emanates from updating of the x vector, where dividing a by a small diagonal value
A[i][i] will cause the x value to "blow up"
'''

```

```

Solution using built-in function: x = [1.69736842 2.82894737 4.35526316]
Solution using gauss-seidel: x = [1.69736916 2.82894576 4.35526348]

```