```
#Question 1

import numpy as np

# Coefficients matrix
A = np.array([[10, 2, -1],
              [-3, -6, 2],
              [1, 1, 5]])

# Constants vector
B = np.array([27, -61.5, -21.5])

# Augmented matrix [A|B]
augmented_matrix = np.column_stack((A, B))

# Number of equations
n = len(B)

# Print the initial augmented matrix
print("Initial Augmented Matrix:")
print(augmented_matrix)
print()

# Perform Gaussian elimination
for i in range(n):
    # Pivot element
    pivot = augmented_matrix[i, i]

    # Divide the current row by the pivot element to make the diagonal 1
    augmented_matrix[i, :] /= pivot

    # Eliminate elements below the pivot
    for j in range(i + 1, n):
        factor = augmented_matrix[j, i]
        augmented_matrix[j, :] -= factor * augmented_matrix[i, :]

    # Print the current state of the augmented matrix
    print(f"Step {i + 1}:")
    print(augmented_matrix)
    print()

# Back-substitution to find the solutions
solutions = np.zeros(n)
for i in range(n - 1, -1, -1):
    solutions[i] = augmented_matrix[i, -1]
    for j in range(i + 1, n):
        solutions[i] -= augmented_matrix[i, j] * solutions[j]

# Print the solutions
print("Solutions:")
for i in range(n):
    print(f"x_{i + 1} = {solutions[i]:.4f}")
```

```
Initial Augmented Matrix:
[[ 10.    2.   -1.   27. ]
 [ -3.   -6.    2.  -61.5]
 [  1.    1.    5.  -21.5]]

Step 1:
[[  1.    0.2  -0.1   2.7]
 [  0.   -5.4   1.7 -53.4]
 [  0.    0.8   5.1 -24.2]]

Step 2:
[[  1.           0.2         -0.1          2.7       ]
 [ -0.           1.          -0.31481481   9.88888889]
 [  0.           0.           5.35185185 -32.11111111]]

Step 3:
[[ 1.          0.2        -0.1         2.7       ]
 [-0.          1.         -0.31481481  9.88888889]
 [ 0.          0.          1.         -6.        ]]

Solutions:
x_1 = 0.5000
x_2 = 8.0000
x_3 = -6.0000
```

```
#Question 2

import numpy as np

# Coefficients matrix
A = np.array([[8, 2, -2],
              [10, 2, 4],
              [12, 2, 2]], dtype=float)

# Constants vector
B = np.array([8, 16, 16], dtype=float)

# Augmented matrix [A|B]
augmented_matrix = np.column_stack((A, B))

# Number of equations
n = len(B)

# Print the initial augmented matrix
print("Initial Augmented Matrix:")
print(augmented_matrix)
print()

# Perform Gaussian elimination with partial pivoting
for i in range(n):
    # Find the pivot element (the largest element in the current column)
    max_row = np.argmax(abs(augmented_matrix[i:, i])) + i
    augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

    # Divide the current row by the pivot element to make the diagonal element 1
    pivot = augmented_matrix[i, i]
    augmented_matrix[i, :] /= pivot

    # Eliminate elements below the pivot
    for j in range(i + 1, n):
        factor = augmented_matrix[j, i]
        augmented_matrix[j, :] -= factor * augmented_matrix[i, :]

    # Print the current state of the augmented matrix
    print(f"Step {i + 1}:")
    print(augmented_matrix)
    print()

# Back-substitution to find the solutions
solutions = np.zeros(n)
for i in range(n - 1, -1, -1):
    solutions[i] = augmented_matrix[i, -1]
    for j in range(i + 1, n):
        solutions[i] -= augmented_matrix[i, j] * solutions[j]

# Print the solutions
print("Solutions:")
for i in range(n):
    print(f"x_{i + 1} = {solutions[i]:.4f}")
```

```
Initial Augmented Matrix:
[[ 8.  2. -2.  8.]
 [10.  2.  4. 16.]
 [12.  2.  2. 16.]]

Step 1:
[[ 1.          0.16666667  0.16666667  1.33333333]
 [ 0.          0.33333333  2.33333333  2.66666667]
 [ 0.          0.66666667 -3.33333333 -2.66666667]]

Step 2:
[[ 1.          0.16666667  0.16666667  1.33333333]
 [ 0.          1.         -5.         -4.        ]
 [ 0.          0.          4.          4.        ]]

Step 3:
[[ 1.          0.16666667  0.16666667  1.33333333]
 [ 0.          1.         -5.         -4.        ]
 [ 0.          0.          1.          1.        ]]

Solutions:
```

```
x_1 = 1.0000
x_2 = 1.0000
x_3 = 1.0000
```

```
#Question 3

import numpy as np

# Coefficients matrix
A = np.array([[2, 1, -1],
              [5, 2, 2],
              [3, 1, 1]], dtype=float)

# Constants vector
B = np.array([2, 9, 5], dtype=float)

# Augmented matrix [A|B]
augmented_matrix = np.column_stack((A, B))

# Number of equations
n = len(B)

# Print the initial augmented matrix
print("Initial Augmented Matrix:")
print(augmented_matrix)
print()

# Perform Gauss-Jordan elimination without pivoting
for i in range(n):
    # Divide the current row by the pivot element to make the diagonal element 1
    pivot = augmented_matrix[i, i]
    augmented_matrix[i, :] /= pivot

    # Eliminate elements above and below the pivot
    for j in range(n):
        if j != i:
            factor = augmented_matrix[j, i]
            augmented_matrix[j, :] -= factor * augmented_matrix[i, :]

    # Print the current state of the augmented matrix
    print(f"Step {i + 1}:")
    print(augmented_matrix)
    print()

# Extract the solutions
solutions = augmented_matrix[:, -1]

# Print the solutions
print("Solutions:")
for i in range(n):
    print(f"x_{i + 1} = {solutions[i]:.4f}")
```

```
Initial Augmented Matrix:
[[ 2.  1. -1.  2.]
 [ 5.  2.  2.  9.]
 [ 3.  1.  1.  5.]]

Step 1:
[[ 1.   0.5 -0.5  1. ]
 [ 0.  -0.5  4.5  4. ]
 [ 0.  -0.5  2.5  2. ]]

Step 2:
[[ 1.  0.  4.  5.]
 [-0.  1. -9. -8.]
 [ 0.  0. -2. -2.]]

Step 3:
[[ 1.  0.  0.  1.]
 [-0.  1.  0.  1.]
 [-0. -0.  1.  1.]]

Solutions:
x_1 = 1.0000
x_2 = 1.0000
x_3 = 1.0000
```

#Question 4

```
#Question 4

import numpy as np

# Coefficients matrix
A = np.array([[15, -3, -1],
              [-3, 18, -6],
              [-4, -1, 12]], dtype=float)

# Constants vector
B = np.array([3300, 1200, 2400], dtype=float)

# Number of equations
n = len(B)

# Initialize the solution vector
solution = np.zeros(n, dtype=float)

# Tolerance (ε_s)
epsilon_s = 0.05  # 5%

# Perform Gauss-Seidel iterations
max_iterations = 100  # Set a maximum number of iterations to prevent infinite loops
for iteration in range(max_iterations):
    previous_solution = solution.copy()
    for i in range(n):
        summation = 0
        for j in range(n):
            if i != j:
                summation += A[i, j] * solution[j]
        solution[i] = (B[i] - summation) / A[i, i]

    # Calculate the approximate relative error
    approximate_relative_error = np.max(np.abs(solution - previous_solution) / np.abs(solution + np.finfo(float).eps))

    print(f"Iteration {iteration + 1}: {solution}, Approx. Relative Error = {approximate_relative_error:.6f}")

    # Check for convergence
    if approximate_relative_error < epsilon_s:
        print("Converged.")
        break
else:
    print("Did not converge within the maximum number of iterations.")
```

```
Iteration 1: [220.         103.33333333 281.94444444], Approx. Relative Error = 1.000000
Iteration 2: [259.46296296 203.89197531 303.47865226], Approx. Relative Error = 0.493196
Iteration 3: [281.01030521 214.66126829 311.55854076], Approx. Relative Error = 0.076678
Iteration 4: [283.70282304 217.80331743 312.71788413], Approx. Relative Error = 0.014426
Converged.
```