```
# QUESTION 1
import numpy as np

# Coefficients matrix A and constants vector B
A = np.array([[2, -6, -1],
              [-3, -1, 7],
              [-8, 1, -2]], dtype=float)
B = np.array([-38, -34, -20], 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):
    # Partial pivoting: find the row with the largest absolute value in the current
    max_row = np.argmax(np.abs(augmented_matrix[i:, i])) + i

    # Swap the current row with the row containing the pivot element
    augmented_matrix[[i, max_row]] = augmented_matrix[[max_row, i]]

    # Divide the current row by the pivot element to make the diagonal 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, :]

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

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

```
Initial Augmented Matrix:
[[  2.  -6.  -1. -38.]
 [ -3.  -1.   7. -34.]
 [ -8.   1.  -2. -20.]]

Solutions:
x_1 = 4.0000
x_2 = 8.0000
x_3 = -2.0000
```

```
#QUESTION 2A
import numpy as np

# Coefficients matrix A and constants vector B
A = np.array([[1, 1, -1],
              [6, 2, 2],
              [-3, 4, 1]], dtype=float)
B = np.array([-3, 2, 1], 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
for i in range(n):
    # Divide the current row by the pivot element to make the diagonal 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, :]

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

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

```
 Initial Augmented Matrix:
 [[ 1.  1. -1. -3.]
  [ 6.  2.  2.  2.]
  [-3.  4.  1.  1.]]

 Solutions:
 x_1 = -0.2500
 x_2 = -0.5000
 x_3 = 2.2500
```

```
#QUESTION 2B
import numpy as np

# Coefficients matrix A and constants vector B
A = np.array([[1, 1, -1],
              [6, 2, 2],
              [-3, 4, 1]], dtype=float)
B = np.array([-3, 2, 1], 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 row with the largest absolute value in the current column (partial pivoting)
    max_row = np.argmax(np.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 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, :]

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

# Print the solutions
print("Solutions:")
for i in range(n):
```

```
    print(f"x_{i + 1} = {solutions[i]:.4f}")
```

```
Initial Augmented Matrix:
[[ 1.  1. -1. -3.]
 [ 6.  2.  2.  2.]
 [-3.  4.  1.  1.]]

Solutions:
x_1 = -0.2500
x_2 = -0.5000
x_3 = 2.2500
```

```
#QUESTION 2C

import numpy as np

# Coefficients matrix A and constants vector B
A = np.array([[1, 1, -1],
              [6, 2, 2],
              [-3, 4, 1]], dtype=float)
B = np.array([-3, 2, 1], 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 partial pivoting
for i in range(n):
    # Divide the current row by the pivot element to make the diagonal 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, :]

# Extract the solutions from the last column of the augmented matrix
solutions = augmented_matrix[:, -1]

# Print the solutions
print("Solutions:")
for i, solution in enumerate(solutions):
    print(f"x_{i + 1} = {solution:.4f}")
```

```
Initial Augmented Matrix:
[[ 1.  1. -1. -3.]
 [ 6.  2.  2.  2.]
 [-3.  4.  1.  1.]]

Solutions:
x_1 = -0.2500
x_2 = -0.5000
x_3 = 2.2500
```