Question 1

```
# Gauss elimination method

import numpy as np

def gauss_elimination():
  # number of variables
  n = int(input(''))

  # initializing X to zeros
  X = np.zeros(n)

  #initializing the matrix augmented A with zeros
  A = np.zeros((n,n+1))

  # A matrix inputs
  print('Enter A matrix coefficients:')
  for i in range(n):
      for j in range(n+1):
          A[i][j] = float(input("A[{}][{}]".format(i, j)))

  for i in range(n):
      # code for pivoting
      if (i != n-1) and abs(A[i][0]) < abs(A[i+1][0]):
          for j in range(n+1):
            temp = A[i][j]
            A[i][j] = A[i+1][j]
            A[i+1][j] = temp
      for j in range(i+1, n):
          E = A[j][i]/A[i][i]
          for k in range(n+1):
              A[j][k] = A[j][k] - E * A[i][k]

  X[n-1] = A[n-1][n]/A[n-1][n-1]

  for i in range(n-2,-1,-1):
      X[i] = A[i][n]

      for j in range(i+1,n):
          X[i] = X[i] - A[i][j]*X[j]

      X[i] = X[i]/A[i][i]
  # Output
  print()
  print('Solution is: ')
  for i in range(n):
      print('X{} = {}'.format(i+1,X[i]))

gauss_elimination()
# output below
3
Enter A matrix coefficients:
A[0][0]0
A[0][1]3
A[0][2]5
A[0][3]1.20736
A[1][0]3
A[1][1]-4
A[1][2]0
A[1][3]-2.34066
A[2][0]5
A[2][1]0
A[2][2]6
A[2][3]-0.329193

Solution is:
X1 = 0.14285608695652194
X2 = 0.6923070652173915
X3 = -0.17391223913043488

# experimenting with different systems
# system whose coefficient determinant is small
gauss_elimination()
# output
3
Enter A matrix coefficients:
A[0][0]0
A[0][1]0.03
A[0][2]0.05
A[0][3]0.0120736
A[1][0]0.03
A[1][1]-0.04
A[1][2]0
A[1][3]-0.0234066
A[2][0]0.05
A[2][1]0
A[2][2]0.06
A[2][3]-0.00329193
```

```
Solution is:
X1 = 0.14285608695652185
X2 = 0.6923070652173914
X3 = -0.17391223913043488
```

```
# experimenting with system with large systems
gauss_elimination()
# output
```

```
4
Enter A matrix coefficients:
A[0][0]0
A[0][1]300
A[0][2]500
A[0][3]600
A[0][4]120
A[1][0]300
A[1][1]-400
A[1][2]0
A[1][3]100
A[1][4]-234
A[2][0]500
A[2][1]0
A[2][2]600
A[2][3]700
A[2][4]1000
A[3][0]100
A[3][1]200
A[3][2]300
A[3][3]400
A[3][4]500

Solution is:
X1 = 4.077826086956522
X2 = 3.350869565217392
X3 = -0.3665217391304383
X4 = -1.169999999999997
```

Question 2

```
# code for gauss elimination without pivoting

def gauss_elimination_without_pivot():
  # number of variables
  n = int(input(''))
```

```
 # initializing X to zeros
 X = np.zeros(n)

 #initializing the matrix augmented A with zeros
 A = np.zeros((n,n+1))

 # A matrix inputs
 print('Enter A matrix coefficients:')
 for i in range(n):
     for j in range(n+1):
         A[i][j] = float(input("A[{}][{}]".format(i, j)))

 for i in range(n):
     for j in range(i+1, n):
         E = A[j][i]/A[i][i]
         for k in range(n+1):
             A[j][k] = A[j][k] - E * A[i][k]

 X[n-1] = A[n-1][n]/A[n-1][n-1]

 for i in range(n-2,-1,-1):
     X[i] = A[i][n]

     for j in range(i+1,n):
         X[i] = X[i] - A[i][j]*X[j]

     X[i] = X[i]/A[i][i]

 print()
 print('Solution is: ')
 for i in range(n):
     print('X{} = {}'.format(i+1,X[i]))

gauss_elimination_without_pivot()
# output
```

```
2
Enter A matrix coefficients:
A[0][0]1e-20
A[0][1]1
A[0][2]1
A[1][0]1
A[1][1]1
A[1][2]2
```

```
Solution is:
X1 = 0.0
X2 = 1.0
```

```
# running the same code with epsilon as 1e-60

gauss_elimination_without_pivot()
# output
```

```
2
Enter A matrix coefficients:
A[0][0]1e-60
A[0][1]1
A[0][2]1
A[1][0]1
A[1][1]1
A[1][2]2

Solution is:
X1 = 0.0
X2 = 1.0
```

```
# gauss elimination with pivoting

def gauss_elimination():
  # number of variables
  n = int(input(''))

  # initializing X to zeros
  X = np.zeros(n)

  #initializing the matrix augmented A with zeros
  A = np.zeros((n,n+1))

  # A matrix inputs
  print('Enter A matrix coefficients:')
  for i in range(n):
      for j in range(n+1):
          A[i][j] = float(input("A[{}][{}]".format(i, j)))

  for i in range(n):
      # code for pivoting
      if (i != n-1) and abs(A[i][0]) < abs(A[i+1][0]):
          for j in range(n+1):
            temp = A[i][j]
            A[i][j] = A[i+1][j]
            A[i+1][j] = temp
      for j in range(i+1, n):
          E = A[j][i]/A[i][i]
          for k in range(n+1):
              A[j][k] = A[j][k] - E * A[i][k]

  X[n-1] = A[n-1][n]/A[n-1][n-1]

  for i in range(n-2,-1,-1):
      X[i] = A[i][n]

      for j in range(i+1,n):
          X[i] = X[i] - A[i][j]*X[j]

      X[i] = X[i]/A[i][i]

  print()
  print('Solution is: ')
  for i in range(n):
      print('X{} = {}'.format(i+1,X[i]))

gauss_elimination()
# Output
```

```
2
Enter A matrix coefficients:
A[0][0]1e-20
A[0][1]1
A[0][2]1
A[1][0]1
A[1][1]1
A[1][2]2

Solution is:
X1 = 1.0
X2 = 1.0
```

So we see that with pivoting, the solution is (1,1) which is the actual solution. But without pivoting, the solution came out to be (0,1) which is incorrect. This happened because the first coefficient value in matrix A is very small (10^-20) as compared to the value of the constant.

Question 3

```
# gauss seidel method

# part a: with inputs 0,0,0

# these functions return the updated x1, x2 and x3 values
def f1(x1, x2, x3):
    return (6 - x2 - x3)/10

def f2(x1, x2, x3):
    return (6 - x1 - x3)/10

def f3(x1, x2, x3):
    return (6 - x1 - x2)/10

def gauss_seidel(x1_0, x2_0, x3_0):
    count = 1
    tol = 1
    while tol > 0.0001:
        # the upper triangular matrix part in the coefficients passed in the functions below
        # has the previous  values, whereas lower triangular matrix has the new values
        x1 = f1(x1_0, x2_0, x3_0)
        x2 = f2(x1, x2_0, x3_0)
        x3 = f3(x1, x2, x3_0)
        tol = abs((x1 - x1_0)**2 + (x2 - x2_0)**2 + (x3 - x3_0)**2)
        print("iter{}: x1={}, x2={}, x3={}".format(count, x1, x2, x3))
        count += 1
        x1_0 = x1
        x2_0 = x2
        x3_0 = x3

    print()
    print("Final solution:  x1={},  x2={},  x3={}".format(x1, x2, x3))

# input
x1_0, x2_0, x3_0 = 0, 0, 0
gauss_seidel(x1_0, x2_0, x3_0)
# output below
```

```
iter1: x1=0.6, x2=0.54, x3=0.48600000000000004
iter2: x1=0.4974, x2=0.50166, x3=0.500094
iter3: x1=0.4998246, x2=0.50000814, x3=0.5000167259999999

Final solution:  x1=0.4998246,  x2=0.50000814,  x3=0.5000167259999999
```

```
# part b: with inputs 10, 10, 10
# input
x1_0, x2_0, x3_0 = 10, 10, 10
gauss_seidel(x1_0, x2_0, x3_0)
# output below
iter1: x1=-1.4, x2=-0.25999999999999995, x3=0.766
iter2: x1=0.5494, x2=0.46846, x3=0.49821399999999993
iter3: x1=0.5033326, x2=0.49984533999999997, x3=0.49968220599999996
iter4: x1=0.5000472454, x2=0.50002705486, x3=0.49999256997399993

Final solution:  x1=0.5000472454,  x2=0.50002705486,
x3=0.49999256997399993
```

So both the inputs lead to the approximate solutions which were very close to the actual solutions(0.5, 0.5, 0.5). The second input took one step iteration longer than the first input as the first input was much closer to the actual solution than the second input