Task 1

I have created the following routine for generating an upper triangular matrix that satisfies the following definition for its entries:

$$a_{i,j} = \begin{cases} 0, & j < i \\ i + j - 1, & j \geqslant i \end{cases}$$

```python
import numpy as np

def matrixU(n, m):
    matrix = np.zeros([n, m])
    for i in range(0, n):
        for j in range(0, m):
            if j < i:
                matrix[i, j] = 0
            else:
                matrix[i, j] = i + j - 1
    print("The upper triangular matrix of size", n, "by", m, "is the following\n", matrix)
    return matrix
```

The routine can also be found here.
And I used the following code to find the solution to the upper triangular matrix A that satisfies the following equation

$$Ax = b$$

where A is created by user's choice, with $b_i = 1$.

```python
import numpy as np
from matrixU import matrixU

np.set_printoptions(precision=4)

# ask the user for the size of the matrix

n = int(input('Please give the rows of this upper triangular matrix: '))
m = int(input('Please give the columns of this upper triangular matrix: '))

# set up the matrix and what not call it inputMatrix?
inputMatrix = matrixU(n, m)

# to find the solution, probably want to check the determinant first just in case

# initialize the solution and the result vector

x = [1 for i in range(m)]
b = [1 for i in range(n)]

if np.linalg.det(inputMatrix) != 0:
    x = np.dot(np.linalg.inv(inputMatrix), b)
    print("The solution of the given upper triangular matrix with result b_i = 1 is,\n x =", x)
else:
    print("Please pick a better n and m next time!")
```

The routine can also be found here, with the following output:

```
xianggao@Xiangs-MacBook-Pro Tasksheet_07 % python Task_1.py
Please give the rows of this upper triangular matrix: 5
Please give the columns of this upper triangular matrix: 5
('The upper triangular matrix of size', 5, 'by', 5, 'is the following\n', array([[-1.,  0.,  1.,  2.,  3.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  0.,  3.,  4.,  5.],
       [ 0.,  0.,  0.,  5.,  6.],
       [ 0.,  0.,  0.,  0.,  7.]]))
('The solution of the given upper triangular matrix with result b_i = 1 is,\n x =', array([-0.4571,  0.2286,  0.0571,  0.0286,  0.1429]))
```

Figure 1. Running the Code From the Terminal.

However, since this looks a little bit ugly, I will from now on use the result from my IDE, which is the following

```
The upper triangular matrix of size 5 by 5 is the following
 [[-1.  0.  1.  2.  3.]
 [ 0.  1.  2.  3.  4.]
 [ 0.  0.  3.  4.  5.]
 [ 0.  0.  0.  5.  6.]
 [ 0.  0.  0.  0.  7.]]
The solution of the given upper triangular matrix with result b_i = 1 is,
 x = [-0.4571  0.2286  0.0571  0.0286  0.1429]
```

Figure 2. Result of Task 1 from the IDE(PyCharm).

Task 2

I have created the following routine for generating an lower triangular matrix, notice that there are two functions in this routine, I did this just to reminds myself of some linear algebra:

```python
import numpy as np
from matrixU import matrixU

def matrixLame(n, m):
    matrix = np.transpose(matrixU(n, m))
    return matrix

def matrixLool(n, m):
    matrix = np.zeros([n, m])
    for i in range(0, n):
        for j in range(0, m):
            if j > i:
                matrix[i, j] = 0
            else:
                matrix[i, j] = i + j - 1
    print("The lower triangular matrix of size", n, "by", m, "is the following\n", matrix)
    return matrix
```

The routine can also be found here.
And I used the following code to find the solution to the lower triangular matrix A that satisfies the following equation

$$Ax = b$$

where A is created by user's choice, with $b_i = 1$.

```python
import numpy as np
from matrixL import matrixLame

np.set_printoptions(precision=4)

# ask the user for the size of the matrix

n = int(input('Please give the rows of this lower triangular matrix: '))
m = int(input('Please give the columns of this lower triangular matrix: '))

# set up the matrix and what not call it inputMatrix?
inputMatrix = matrixLame(n, m)

# to find the solution, probably want to check the determinant first just in case

# initialize the solution and the result vector
```

```

x = [1 for i in range(m)]
b = [1 for i in range(n)]

if np.linalg.det(inputMatrix) != 0:
    x = np.dot(np.linalg.inv(inputMatrix), b)
    print("The solution of the given lower triangular matrix with result b_i = 1 is,\n x =",
     x)
else:
    print("Please pick a better n and m next time!")
```

The routine can also be found here, with the following output:

```
Please give the rows of this lower triangular matrix: 5
Please give the columns of this lower triangular matrix: 5
The upper triangular matrix of size 5 by 5 is the following
 [[-1.  0.  1.  2.  3.]
 [ 0.  1.  2.  3.  4.]
 [ 0.  0.  3.  4.  5.]
 [ 0.  0.  0.  5.  6.]
 [ 0.  0.  0.  0.  7.]]
The solution of the given lower triangular matrix with result b_i = 1 is,
 x = [-1.0000e+00  1.0000e+00 -1.1102e-16 -2.7756e-17  2.7756e-17]
```

Figure 3. Result of Task 2 from the IDE(PyCharm).

Task 3

This is the routine I've created for generating a $n \times n$ random square matrix. It can be found here.

```
import numpy as np
import random as rand

def matrixS(n):
    matrix = np.zeros([n, n])
    for i in range(0, n):
        for j in range(0, n):
            matrix[i, j] = rand.randint(1, 11)
    print("The random square matrix of size", n, "is the following\n", matrix)
    return matrix
```

This is the routine I've created for generating a $n \times n$ diagonal square matrix. It can be found here.

```
import numpy as np
import random as rand

def matrixD(n):
    matrix = np.zeros([n, n])
    a = rand.randint(1, 27)
    for i in range(0, n):
        for j in range(0, n):
            if j != i:
                matrix[i, j] = 0
            else:
                matrix[i, j] = a
    print("The random diagonal matrix of size", n, "is the following\n", matrix)
    return matrix
```

Task 4

Use the previous routine, I've created the following code to solve the systems of equations as mentioned in **Task 1** and **Task 2**:

```python
import numpy as np
from matrixD import matrixD

np.set_printoptions(precision=4)

# ask the user for the size of the matrix

n = int(input('Please give the size of this diagonal matrix: '))

# set up the matrix and what not call it inputMatrix?
inputMatrix = matrixD(n)

# to find the solution, probably want to check the determinant first just in case

# initialize the solution and the result vector

x = [1 for i in range(n)]
b = [1 for i in range(n)]

if np.linalg.det(inputMatrix) != 0:
    x = np.dot(np.linalg.inv(inputMatrix), b)
    print("The solution of the given diagonal matrix with result b_i = 1 is, x = ", x)
else:
    print("Please pick a better n next time!")
```

The routine can also be found here, with the following output:

```
Please give the size of this diagonal matrix: 5
The random diagonal matrix of size 5 is the following
 [[5. 0. 0. 0. 0.]
 [0. 5. 0. 0. 0.]
 [0. 0. 5. 0. 0.]
 [0. 0. 0. 5. 0.]
 [0. 0. 0. 0. 5.]]
The solution of the given diagonal matrix with result b_i = 1 is, x =  [0.2 0.2 0.2 0.2 0.2]
```

Figure 4. Result of Task 4 from the IDE(PyCharm).

Task 5

I have written the following code to get the row-reduce-echelon form of any random square matrix generated by the routine mentioned in **Task 4** with a given size $n \times n$:

```python
import numpy as np
from matrixS import matrixS

np.set_printoptions(precision=4)

# ask the user for the size of the matrix

n = int(input('Please give the size of this square matrix: '))

# set up the matrix and what not call it inputMatrix?
inputMatrix = matrixS(n)

# to find the reduced row echelon form of inputMatrix without testing bad pivots, we can just do the following

```

```
for k in range(n):
    if inputMatrix[k][k] == 0:
        exit('Division by zero detected!')

    for i in range(n):
        if i != k:
            ratio = inputMatrix[i][k] / inputMatrix[k][k]

            for j in range(n):
                inputMatrix[i][j] = inputMatrix[i][j] - ratio * inputMatrix[k][j]

print("The row echelon form of the given square matrix is \n", inputMatrix)
```

The routine can also be found here, with the following output:

```
Please give the size of this square matrix: 5
The random square matrix of size 5 is the following
 [[ 3.  8.  8.  7.  2.]
 [ 5.  7.  5.  6.  2.]
 [ 8.  9.  5.  5.  1.]
 [ 1. 11.  5.  4.  8.]
 [ 1.  4.  2.  7. 11.]]
The row echelon form of the given square matrix is
 [[ 3.0000e+00  0.0000e+00  0.0000e+00  7.1054e-15  0.0000e+00]
 [ 0.0000e+00 -6.3333e+00  0.0000e+00  0.0000e+00  0.0000e+00]
 [ 0.0000e+00  0.0000e+00 -1.0526e-01  0.0000e+00  0.0000e+00]
 [ 0.0000e+00  0.0000e+00  0.0000e+00  2.1000e+02  0.0000e+00]
 [ 0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  4.8952e+00]]
```

Figure 5. Result of Task 5 from the IDE(PyCharm).

Task 6

From this online pdf[1] I found, when it comes to the conditions on matrices that ensure we will be able to compute the solution of a linear system of equations. The following has to be mentioned:

- The system has to be well-posed, that is: n equations in n unknowns;
- The inverse of the matrix K, K^{-1} must exist.

For the second item of the list, without the condition K^{-1} exist, back substitution is required rather then just Gaussian elimination.

[1] https://ocw.mit.edu/ans7870/2/2.086/F12/MIT2086F12notesunit5.pdf