Answer to Problem Set 12

- Student Name: Gaole Dai
- Student ID: S01435587

Problem 1

$$
\begin{aligned}
x_1 &= b_1 \\
a_{2,1}x_1 + x_2 &= b_2 \\
a_{3,1}x_1 + a_{3,2}x_2 + x_3 &= b_3 \\
&\cdots
\end{aligned}
$$

Start with x_1, and read solution;

Given x_1, compute $a_{2,1}x_1$, then $x_2 = b_2 - a_{2,1}x_1$, and so on up to x_n.

Algorithm for solving the lower triangular system is shown below:

```
import numpy as np
def l_solver(L, b):
    # Obtain the size of lower triangular system
    n = L.shape[0]
    # Init the result matrix
    x = np.zeros(n)
    # Forward iteration
    for i in range(n):
        tmp = b[i]
        for j in range(i):
            tmp -= L[i,j] * x[j]
        x[i] = tmp / L[i,i]
    return x
```

The algorithm runs two for loops, each loop it runs i times, total cost is
$T = 0 + 1 + 2 + \cdots + N - 1 = \frac{(N-1)*(N-1)}{2} = \mathcal{O}(n^2)$.

Problem 2

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 &= b_3 \\
a_{2,2}x_2 + a_{2,3}x_3 &= b_2 \\
a_{3,3}x_3 &= b_3
\end{aligned}
$$

Start with x_3, $\frac{b_3}{a_{3,3}}$

Compute $b_3 - a_{2,3}x_3$ and so on up to x_n.

Algorithm for solving the upper triangular system is shown below:

```python
import numpy as np
def u_solver(U, b):
    # Obtain the size of lower triangular system
    n = U.shape[0]
    # Init the result matrix
    x = np.zeros(n)
    # Backward iteration
    for i in range(n-1, -1, -1):
        tmp = b[i]
        for j in range(i+1, n):
            tmp -= U[i,j] * x[j]
        x[i] = tmp / U[i,i]
    return x
```

Similar to problem 1, the algorithm runs two for loops, each loop it runs i times, total cost is also $T = 0 + 1 + 2 + \cdots + N - 1 = \frac{(N-1)*(N-1)}{2} = \mathcal{O}(n^2)$.

Problem 3

$x_1 = 3, x_2 = 2, x_3 = 1$

Run the python code in Problem 1

```python
L = np.array([[1, 0, 0], [4, 1, 0], [-6, 5, 1]])
b = np.array([3, 14, -7])
x = l_solver(L, b)
print(x)
```

```
==> [3   2   1]
```

Start with x_1, and the solution could be read as $x_1 = 3$;

Then compute the x_2, $4 \times x_1 + x_2 = 14 \implies x_2 = \frac{14-4\times3}{1} = 2$

After that, compute the x_3,
$-6 \times x_1 + 5 \times x_2 + x_3 = -7 \implies x_3 = -7 + 6 \times 3 - 5 \times 2 = 1$

Problem 4

`L =`

`[[ 1.  0.  0.]`

`[ 2.  1.  0.]`

`[ 3.  2.  1.]]`

`U =`

`[[ 4. -5.  6.]`

`[ 0.  4. -5.]`

`[ 0.  0.  4.]]`

```python
import numpy as np
def lu_decomp(mat):

    # Get the number of rows
```

```python
    n = mat.shape[0]
    # Copy the original array
    U = mat.copy()
    # Generate two-dimensional array with 1 on the diagonal and 0 elsewhere
    L = np.eye(n, dtype=np.double)

    # Loop over rows
    for i in range(n-1):

        # Eliminate entries below i with row operations
        # on U and reverse the row operations to
        # manipulate L
        factor = U[i+1:, i] / U[i, i]
        L[i+1:, i] = factor
        U[i+1:] -= factor[:, np.newaxis] * U[i]

    return L, U
```

```python
# Run the algirithm
mat = np.array([[4.0, -5.0, 6.0], [8.0, -6.0, 7.0], [12.0, -7.0, 12.0]])
L, U = lu_decomp(mat)
print(L)
print(U)
=>
[[ 1.  0.  0.]
 [ 2.  1.  0.]
 [ 3.  2.  1.]]
[[ 4. -5.  6.]
 [ 0.  4. -5.]
 [ 0.  0.  4.]]
```

First Step:

Calculate the k_1 by, $4 \times k_1 + 8 = 0$, then we extract $k_1 = -2$.

The U could be processed by $R_1 \times k_1 + R_2$,

$L[0,1] = -k_1 = 2, U[1,1] = -5 \times (-2) + (-6) = 4, U[2,1] = 6 \times (-2) + 7 = -5$

The current L and U is:

`L_cur =`

`[[ 1.  0.  0.]`

`[ 2.  1.  0.]`

`[ 0.  0.  0.]]`

`U_cur =`

`[[ 4. -5.  6.]`

`[ 0.  4.  -5.]`

`[ 12. -7. 12.]]`

Second Step:

Calculate the k_2 by $R_1 \times k_2 + R_2 = 0, 4 \times k_2 + 12 = 0$, then we extract $k_2 = -3$.

The U could be processed $R_1 \times k_2 + R3$,

$L[0,2] = -k_2 = 3, U[1,2] = -5 \times (-3) + (-7) = 8, U[2,2] = 6 \times (-3) + 12 = -6$

The current L and U is:

`L_cur =`

```
[[ 1.   0.   0.]
 [ 2.   1.   0.]
 [ 3.   0.   0.]]
```

`U_cur =`

```
[[ 4. -5.   6.]
 [ 0.   4.  -5.]
 [ 0.   8. -6.]]
```

Third Step:

Calculate the k_3 by $4 \times k_3 + 8 = 0$, then we extract $k_3 = -2$.

The U could be processed $R_2 \times k_3 + R3$,

$L[1,2] = -k_3 = 2, U[2,2] = -5 \times (-2) + (-6) = 4$

The current L and U is:

`L_cur =`

```
[[ 1.   0.   0.]
 [ 2.   1.   0.]
 [ 3.   2.   0.]]
```

`U_cur =`

```
[[ 4. -5.   6.]
 [ 0.   4.  -5.]
 [ 0.   0.   4.]]
```

Problem 5

With the instruction on the lecture, the Koratsuba-based algorithm is shown below:

```
def karatsuba(x,y):
    # Extract a, b, c and d
    a = x // 100
    b = x % 100
    c = y // 100
    d = y % 100
    # Compute a*c
    t1 = a * c
    # Compute b*d
    t2 = b * d
    # Compute (a+b)*(c+d) = ac + ad + bc + bd
    t3 = (a+b) * (c+d)
    t4 = t3 - t1 - t2
    production = t1 * 10000 + t2 + (t4 * 100)
    return production

print(karatsuba(5822, 4104))
```

1. Extract 2-decimal digit a, b, c and d - $a = 58, b = 22, c = 41, d = 04$
2. Compute $a * c = 58 * 41 = 2378$
3. Compute $b * d = 88$
4. Compute $(a + b) * (c + d) = (58 + 22) * (41 + 4) = 3600$
5. Subtract step 4 from step 3 and step 2 $3600 - 2378 - 88 = 1134$
6. Pad with zeros and obtain the final result $2378 * 10000 + 88 + 1134 * 100 = 23893488$

The production is then 23893488.