Решение систем линейных уравнений

Метод LU разложения

Классический алгоритм исключения неизвестных (Метод Гаусса) связан с представлением исходной матрицы A в виде произведения двух треугольных.

Если выполнены условия

$$a_{11} \neq 0;\ \det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \neq 0;\ \det |A| \neq 0$$

тогда матрица A представима в виде $A = LU$, где L - нижняя треугольная, U - верхняя треугольная.

Приведем рекуррентные формулы для определения треугольных матриц L и U:

$$u_{11} = a_{11},$$

$$u_{1j} = a_{1j},\ l_{j1} = \frac{a_{j1}}{u_{11}},\ j = 2, 3, ..., n$$

$$u_{ii} = a_{ii} - \sum_{k=1}^{i-1} l_{ik} u_{ki}$$

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj},\ l_{ji} = \frac{1}{u_{ii}} \left(a_{ji} - \sum_{k=1}^{j-1} l_{jk} u_{ki} \right),\ i = 2, 3, ..., n,\ j = i+1, i+2, ..., n$$

После разложения решение сводится к последовательному решениюдвух систем уравненийс треугольными матрицами:

$$Ly = f;\ Ux = y$$

$$y_k = f_k - \sum_{j=1}^{k-1} l_{kj} y_j,\ x_k = \frac{1}{u_{kk}} \left(y_k - \sum_{j=k+1}^{n} u_{kj} x_j \right)$$

Метод Холецкого

Матрица A преобразуется как $A = LL^T$

Находим решение системы $Lv = f$

Затем $L^T u = v$

Формулы расчеты коэффициентов векторов решения рассчитываются по формулам:

Для первой системы:

$$v_i = l_{ii}^{-1}\left(f_i - \sum_{k=1}^{i} l_{ki} v_k\right)$$

Для второй системы:

$$u_k = l_{kk}^{-1}\left(v_k - \sum_{j=k+1}^{n} l_{kj} u_j\right)$$

Пример с семинара

$$A = \begin{pmatrix} 0,78 & 0,563 \\ 0,457 & 0,33 \end{pmatrix},\ f = \begin{pmatrix} 0,217 \\ 0,127 \end{pmatrix}$$

Решение данной системы $x_1 = 1,\ x_2 = -1$

Получем решение данной системы с помощью метода Гаусса. Проверим условия:

$$a_{11} = 0,78 \neq 0$$

$$\det\begin{pmatrix} 0,78 & 0,563 \\ 0,457 & 0,33 \end{pmatrix} = 0,78 * 0,33 - 0,563 * 0,457 = 1,09 \times 10^{-4} \neq 0$$

$$\det|A| = |1,09 \times 10^{-4}| \neq 0$$

```
import numpy as np
import matplotlib.pyplot as plt
from numba import jit
```

```
A = np.asarray( [ [ 0.78, 0.563 ], [ 0.457, 0.33 ] ] )

f = np.asarray( [ 0.217, 0.127 + 0.0005 ] )
```

```python
def decLU( A ):

    size = A.shape[ 0 ]

    U = np.zeros( ( size, size ) )

    L = np.zeros( ( size, size ) )

    for i in np.arange( 0, size, 1 ):

        for j in np.arange( 0, size, 1 ):

            if i == 0:

                U[ i ][ j ] = A[ i ][ j ]

                L[ j ][ i ] = A[ j ][ i ] / U[ 0 ][ 0 ]


            else:

                S = 0.0

                for k in np.arange( 0, i - 1, 1 ):

                    S += L[ i ][ k ] * U[ k ][ i ]

                U[ i ][ i ] = A[ i ][ i ] - S

                S = 0.0

                for k in np.arange( 0, i, 1 ):

                    S += L[ i ][ k ] * U[ k ][ j ]

                U[ i ][ j ] = A[ i ][ j ] - S

                S = 0.0

                for k in np.arange( 0, i, 1 ):

                    S += L[ j ][ k ] * U[ k ][ i ]

                L[ j ][ i ] = ( A[ j ][ i ] - S ) / U[ i ][ i ]

    return U, L
```

```python
def Solve( A, f ):

    global decLU

    size = A.shape[ 0 ]

    y = np.zeros( size )

    x = np.zeros( size )

    U, L = decLU( A )

    for k in np.arange( 0, size, 1 ):

        #S = 0.0

        #for j in np.arange( 0, k-1, 1 ):

            #S += L[ k ][ j ] * u[ j ]

        y[ k ] = f[ k ] - np.dot( L[ k ][ 0:k ], y[ 0:k ] )

    for k in np.arange( size - 1, -1, -1 ):

        x[ k ] = ( y[ k ] - np.dot( U[ k ][ k+1:size ], x[ k+1:size ] ) ) / U[ k ][ k ]

    return x
```

```python
x = Solve( A, f )
```

```python
x
```

```
array([-1.58256881,  2.57798165])
```