Лескевич Даниил Казимирович Б05-903

Предварительные сведения

Векторные нормы:

$||u||_\infty = \max_i |u_i|$

$||u||_1 = \sum_i |u_i|$

$||u||_2 = \left(\sum_i |u_i|^2\right)^{\frac{1}{2}}$

Матричные нормы:

$||A||_\infty = \max_i \sum_j |a_{ij}|$

$||A||_1 = \max_j \sum_i |a_{ij}|$

$||A||_2 = (\max_i \lambda_i(AA^*))^{\frac{1}{2}}$

Контрольный вопрос: какова будет вторая норма матрицы, если матрица самосопряженная?

Ваш ответ на контрольный вопрос: $||A||_2 = \max_i |\lambda_i|$

In [1]:
```
import numpy as np
import numpy.linalg as la

A = np.array([[1,2],[3,4]])
v = range(0,3)
Vander = np.vander(v)
print('norm_1 = ', la.norm(Vander, 1))
print('norm_2 = ', la.norm(Vander, 2))
print('norm_inf = ', la.norm(Vander, np.inf))
Vander
```

```
norm_1 =  5.0
norm_2 =  4.844958524498339
norm_inf =  7.0
```

Out[1]:
```
array([[0, 0, 1],
       [1, 1, 1],
       [4, 2, 1]])
```

Обусловленность:

$$(A + \delta A)u = f + \delta f$$

$$\frac{||\delta u||}{||u||} \leq \frac{\mu}{1 - \mu\frac{||\delta A||}{||A||}}\left(\frac{||\delta f||}{||f||} + \frac{||\delta A||}{||A||}\right)$$

μ - число обусловленности матрицы A, $\mu(A) = ||A^{-1}|| \cdot ||A||$, $\mu \geq 1$.

Пример проблемы использования метода Гаусса для решения СЛАУ

In [2]:
```
import numpy as np
import numpy.linalg as la

#функция меняет A и b
def gauss( A, b):
    n = b.size
    for k in range(0,n-1):
        for i in range(k+1,n):
            if A[i,k]!=0:
                c = A[i,k]/A[k,k]
                A[i,k+1:n] = A[i,k+1:n] - c*A[k,k+1:n]
                #было непонятно почему A не треугольнатая
                #A[i,k:n] = A[i,k:n] - c*A[k,k:n] так A привелась бы к треугольному
                b[i] = b[i] - c*b[k]

    # обратный ход
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(A[k,k+1:n],b[k+1:n]))/A[k,k];
    return b

#все числа в представлены как вещественные
A1 = np.array([[1e-16, 1., -1.], [-1., 2., -1.], [2., -1., 0.]]);
b1 = np.array([0., 0., 1.]);

A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])
```

In [3]:
```
print(A2)
print('u2 = ', gauss(A2, b2))#la.solve(A2, b2))
```

```
[[ 2.e+00 -1.e+00  0.e+00]
 [-1.e+00  2.e+00 -1.e+00]
 [ 1.e-16  1.e+00 -1.e+00]]
u2 =  [1. 1. 1.]
```

In [4]:
```
print(A2)
```

```
[[ 2.00000000e+00 -1.00000000e+00  0.00000000e+00]
 [-1.00000000e+00  1.50000000e+00 -1.00000000e+00]
 [ 1.00000000e-16  1.00000000e+00 -3.33333333e-01]]
```

In [5]:
```
A1 = np.array([[1e-16, 1., -1.], [-1., 2., -1.], [2., -1., 0.]]);
b1 = np.array([0., 0., 1.]);

A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])

print('mu1 = ', la.cond(A1))
print('mu2 = ', la.cond(A2))

print('u1 = ', gauss(A1, b1))
#print('u1 = ', la.solve(A1, b1))
print('u2 = ', gauss(A2, b2))#la.solve(A2, b2))
```

```
mu1 =  16.39373162228438
mu2 =  16.393731622284395
u1 =  [0.55511151 0.25       0.25      ]
u2 =  [1. 1. 1.]
```

Часть 1. LU разложение

Задание:

реализовать алгоритм решения предыдущей задачи с матрицей A2 с помощью LU-разложение В решении должна выводиться L, U и собственно решение системы.

ВАЖНО: реализация метода LU должна быть получена путем небольшой модификации метода gauss! При это саму реализацию можно разделить на два метода: один метод собственно находит LU разложение (можно сделать переделкой цикла для матрицы A метода gauss), второй метод - непосредственное решение системы с помощью прямого и обратного хода. Ни в каком виде нельзя ползоваться пакетными методами (в частности, la.solve)

LU - разложение с помощью пакета sympy

Чтобы убедиться, что разложение получено верно, можно воспользоваться скриптом ниже

In [9]:

```
import sympy as sp
import numpy as np

#метод гаусса, который сохранит наши переменные в первозданном виде
def gauss1( A, b ):
    x, y = A.copy(), b.copy()
    return gauss(x, y)

#вычисление элемента для U
def elem_U(x, y, L, U, A):
    U[x,y] = A[x, y]
    for i in range(x):
        U[x,y] -= L[x, i]*U[i,y]
    return U[x,y]
#вычисление элемента для L
def elem_L(x, y, L, U, A):
    if U[y,y] != 0:
        L[x,y] = A[x, y]
        for i in range(y):
            L[x,y] -= L[x, i]*U[i,y]
        L[x,y] /= U[y,y]
    return L[x,y]

#LU разложение
def LU(A, n):
    L = np.eye(n)
    U = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if i <= j:
                U[i,j] =  elem_U(i,j, L, U, A)
            else:
                L[i,j] = elem_L(i,j, L, U, A)
    print("L = \n", L)
    print("U = \n", U)
    return L, U

# решение методом LU разложения
def LUsolve(A, b):
    n = b.size
    L, U  = LU(A, n)
    #далее код взят из функции gauss где A заменено на L в прямом  и на U в обратном
    #прямой ход
    for k in range(0,n-1):
        for i in range(k+1,n):
            if L[i,k]!=0:
```

```
                c = L[i,k]/L[k,k]
                L[i,k+1:n] = L[i,k+1:n] - c*L[k,k+1:n]
                b[i] = b[i] - c*b[k]

    # обратный ход
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(U[k,k+1:n],b[k+1:n]))/U[k,k];
    return b

#решение методом LU разложения, который сохранит наши переменные в первозданном виде
def LUsolve1(A, b):
    x, y = A.copy(), b.copy()
    return LUsolve(x, y)
b = np.array([-1., 1.])
#LUsolve(A, b)
A = sp.Matrix([[2, 3], [5, 4]])

L, U, _ = A.LUdecomposition()
L
```

Out[9]:

$$\begin{bmatrix} 1 & 0 \\ \frac{5}{2} & 1 \end{bmatrix}$$

In [10]:

```
LU(A, 2)
```

```
L =
 [[1.  0. ]
 [2.5 1. ]]
U =
 [[ 2.   3. ]
 [ 0.  -3.5]]
```

Out[10]:

```
(array([[1. , 0. ],
        [2.5, 1. ]]),
 array([[ 2. ,  3. ],
        [ 0. , -3.5]]))
```

In [13]:

```
#некрасиво, но быстро, т.к. мы используем функции, которые меняют матрицы A и b
A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])
print('u2 = ', gauss(A2, b2))
A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])
print('lu2 = ', LUsolve(A2,b2))
```

```
u2 =  [1. 1. 1.]
L =
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-5.00000000e-01  1.00000000e+00  0.00000000e+00]
 [ 5.00000000e-17  6.66666667e-01  1.00000000e+00]]
U =
 [[ 2.         -1.          0.        ]
 [ 0.          1.5        -1.        ]
 [ 0.          0.         -0.33333333]]
lu2 =  [1. 1. 1.]
```

In [14]:

```
#красиво, но медленно,тратим время на копирование исходных матриц, но зато сохраняем
A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])
print('lu2 = ', LUsolve1(A2,b2))
print('u2 = ', gauss1(A2, b2))
```

```
L =
```

```
 [[ 1.00000000e+00  0.00000000e+00  0.00000000e+00]
 [-5.00000000e-01  1.00000000e+00  0.00000000e+00]
 [ 5.00000000e-17  6.66666667e-01  1.00000000e+00]]
U =
 [[ 2.         -1.          0.        ]
 [ 0.          1.5        -1.        ]
 [ 0.          0.         -0.33333333]]
lu2 =  [1. 1. 1.]
u2 =  [1. 1. 1.]
```

Часть 2. Нахождение обратной матрицы с помощью LU разложения

Задание:

Предложить алгоритм с использованием LU-разложения и найти обратную матрицу с точностью $\epsilon = 10^{-3}$:

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 7 & 1 & 4 \end{pmatrix}$$

Для необходимых оценок использовать первую норму. Сравнить результат со значением, найденным с помощью функции numpy.inv.

In [18]:

```
from numpy.linalg import inv
A = np.array([[1., 1., 1.], [0., 1., 2.], [7, 1., 4.]])

#решаем для L
def L_solve(L, b):
    n = b.size
    x = b.copy()
    #прямой ход
    for k in range(0,n-1):
        for i in range(k+1,n):
            if L[i,k]!=0:
                c = L[i,k]/L[k,k]
                #L[i,k+1:n] = L[i,k+1:n] - c*L[k,k+1:n]
                x[i] = x[i] - c*x[k]
    return x

#решаем для U
def U_solve(U, b):
    n = b.size
    for k in range(n-1,-1,-1):
        b[k] = (b[k] - np.dot(U[k,k+1:n],b[k+1:n]))/U[k,k];
    return b

#
def my_inv(A, n):
    C = A.copy()
    L, U = LU(C, n)#получаем LU разложение
    E = np.eye(n)#создаём единичную матрицу
    L_1, U_1 = np.zeros((n,n)),np.zeros((n,n))
    L_c, U_c = L.copy(), U.copy()
    # вычисляем i столбец обратных матриц
    for i in range(n):
        L_1[i] = L_solve(L_c, E[i])
        U_1[i] = U_solve(U_c, E[i])
    L_1 = L_1.T
    U_1 = U_1.T
    print("L^{-1} = \n", L_1)
```

```
    print("U^{-1} = \n", U_1)
    print("A^{-1} = \n", np.dot(U_1, L_1))
    print("A.inv = \n", inv(A))
    return np.dot(U_1, L_1)

print("A*my_inv(A) = ", np.dot(A, my_inv(A, 3)))
A = np.array([[1., 1., 1.], [0., 1., 2.], [7, 1., 4.]])
print("A*inv(A) = ", np.dot(A, inv(A)))
```

```
L =
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 7. -6.  1.]]
U =
 [[1. 1. 1.]
 [0. 1. 2.]
 [0. 0. 9.]]
L^{-1} =
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [-7.  6.  1.]]
U^{-1} =
 [[ 1.         -1.          0.11111111]
 [ 0.          1.         -0.22222222]
 [ 0.          0.          0.11111111]]
A^{-1} =
 [[ 0.22222222 -0.33333333  0.11111111]
 [ 1.55555556 -0.33333333 -0.22222222]
 [-0.77777778  0.66666667  0.11111111]]
A.inv =
 [[ 0.22222222 -0.33333333  0.11111111]
 [ 1.55555556 -0.33333333 -0.22222222]
 [-0.77777778  0.66666667  0.11111111]]
A*my_inv(A) =  [[ 1.0000000e+00  0.0000000e+00  0.0000000e+00]
 [ 0.0000000e+00  1.0000000e+00  0.0000000e+00]
 [ 4.4408921e-16 -4.4408921e-16  1.0000000e+00]]
A*inv(A) =  [[1.00000000e+00 1.11022302e-16 2.77555756e-17]
 [0.00000000e+00 1.00000000e+00 0.00000000e+00]
 [0.00000000e+00 4.44089210e-16 1.00000000e+00]]
```

In [19]:

```
A = np.array([[1., 1., 1.], [0., 1., 2.], [7, 1., 4.]])
A_1 = my_inv(A, 3)
print("Ошибка в первой норме", la.norm(inv(A) - A_1, ord=1))
```

```
L =
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 7. -6.  1.]]
U =
 [[1. 1. 1.]
 [0. 1. 2.]
 [0. 0. 9.]]
L^{-1} =
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [-7.  6.  1.]]
U^{-1} =
 [[ 1.         -1.          0.11111111]
 [ 0.          1.         -0.22222222]
 [ 0.          0.          0.11111111]]
A^{-1} =
 [[ 0.22222222 -0.33333333  0.11111111]
 [ 1.55555556 -0.33333333 -0.22222222]
 [-0.77777778  0.66666667  0.11111111]]
A.inv =
```

```
 [[ 0.22222222 -0.33333333  0.11111111]
 [ 1.55555556 -0.33333333 -0.22222222]
 [-0.77777778  0.66666667  0.11111111]]
Ошибка в первой норме 8.326672684688674e-17
```

In [ ]: