Численные методы решения СЛАУ

Юлия Прохорова

0.0.1 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: вторая норма будет равна $max_i|\lambda_i|$

[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
```

[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, $\mu(A) = ||A^{-1}|| \cdot ||A||$, $\mu \geq 1$.

0.1 Пример проблемы использования метода Гаусса для решения СЛАУ

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

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:
```

```python
                c = A[i,k]/A[k,k]
                A[i,k+1:n] = A[i,k+1:n] - c*A[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(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.])
```

[3]:
```python
print(A1)
```

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

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

```
[[ 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]]
```

[5]:
```python
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))
#print('u2= ', la.solve(A2, b2))
```

```
mu1 =  16.393731622284385
mu2 =  16.393731622284392
u1 =  [0.55511151 0.25       0.25      ]
u2 =  [1. 1. 1.]
```

0.2 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 пользоваться пакетными методами (в частности, la.solve)

0.2.1 LU - разложение с помощью пакета sympy

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

[81]:
```
A2 = np.array([[2., -1., 0.], [-1., 2., -1.], [1e-16, 1., -1.]])
b2 = np.array([1., 0., 0.])

def LU(A):
    n = len(A[0])
    U = np.zeros((n,n))
    U = A
    L = np.zeros((n,n))

    for i in range(n):
       for j in range(i+1):
           L[i][j]=U[i][j]/U[j][j]

    for k in range(1, n):
        for i in range(k-1, n):
            for j in range(i, n):
                L[j][i] = U[j][i]/U[i][i]
        for i in range(k, n):
            for j in range(k-1, n):
                U[i][j] -=L[i][k-1]*U[k-1][j]
    return (L, U)

def Ly(L, b):

    #Ly = b
    n = b.size
    y = b.copy()
    for i in range(n):
        for j in range(i):
            y[i] = y[i] -  y[j]*L[i][j]
        y[i] /= L[i][i]
    return y

def Ux(U, y):
    #Ux = y
    n = y.size
    x = y.copy()
    x[n-1] /= U[n-1][n-1]
    for i in np.arange(n-2, -1, -1):
        for j in range(i+1, n):
            x[i] -= x[j]*U[i][j]
        x[i] /= U[i][i]
    return x

result = LU(A2)
print('L = \n', result[0])
print('U = \n', result[1])
y = Ly(result[0], b2)
x = Ux(result[1], y)
print('x = ', x)
```

```
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]]
x =  [1. 1. 1.]
```

[10]:
```python
import sympy as sp

A = sp.Matrix([[2, 3], [5, 4]])
L, U, _ = A.LUdecomposition()
U
```

[10]:
$$\begin{bmatrix} 2 & 3 \\ 0 & -\frac{7}{2} \end{bmatrix}$$

0.3 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.

[80]:
```python
A = np.array([[1,1,1], [0,1,2], [7,1,4]])
result = LU(A)
L = result[0]
U = result[1]
Y = np.zeros(L.shape)
invA = np.zeros(L.shape)
n = len(L[0])

for i in range(n):
    ei = np.eye(n)[i]
    Y[i] = Ly(L, ei)

for i in range(n):
    invA.T[i] = Ux(U, Y[i])

print('invA = \n', invA)
```

```
invA =
 [[ 0.22222222 -0.33333333  0.11111111]
 [ 1.55555556 -0.33333333 -0.22222222]
 [-0.77777778  0.66666667  0.11111111]]
```

Полученная матрица линейными преобразованиям приводится, к полученной с помощью функции numpy.inv:

$$A^{-1} = \begin{pmatrix} 0.(2) & -0.(3) & 0.(1) \\ 1.(5) & -0.(3) & -0.(2) \\ -0.(7) & 0.(6) & 0.(1) \end{pmatrix} \sim \begin{pmatrix} 0.(2) & -0.(3) & 0.(1) \\ 0 & 0.(9) & 0 \\ -0.(7) & 0.(6) & 0.(1) \end{pmatrix} \sim \begin{pmatrix} 0.(9) & -0.(9) & 0 \\ 0 & 0.(9) & 0 \\ -0.(7) & 0.(6) & 0.(1) \end{pmatrix}$$

$$\sim \begin{pmatrix} 0.(9) & 0 & 0 \\ 0 & 0.(9) & 0 \\ -0.(7) & 0.(6) & 0.(1) \end{pmatrix} \sim \begin{pmatrix} 0.(9) & 0 & 0 \\ 0 & 0.(9) & 0 \\ 0 & 0 & 0.(1) \end{pmatrix} \sim \begin{pmatrix} 0.(9) & -1 & 0.(1) \\ 0 & 0.(9) & -0.(2) \\ 0 & 0 & 0.(1) \end{pmatrix}$$

Таким образом, получена обратная матрица A^{-1} с необходимой точностью.

[77]:
```
invA_pr = np.array([[0.99999999, -1, 0.11111111], [0, 0.99999999, -0.22222222], [0,␣
↪0, 0.11111111]])
print('||invA||1 = ', la.norm(invA_pr, ord = 1))
```

```
||invA||1 =  1.99999999
```

[79]:
```
print('np.linalg.inv(A) =\n', np.linalg.inv(A))
print('||inv(A)||1 = ', la.norm(np.linalg.inv(A), ord = 1))
```

```
np.linalg.inv(A) =
 [[ 1.          -1.          0.11111111]
 [ 0.          1.         -0.22222222]
 [ 0.          0.          0.11111111]]
||inv(A)||1 =  2.0
```