ЛР 6. Решение систем линейных уравнений

ПИН-21 Чендемеров Алексей

June 19, 2021

1 ЛР 6. Решение систем линейных уравнений.

[1]:
```
import numpy as np
import numpy.linalg as lin
import sympy as sp
import math
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams["figure.figsize"] = (15,10)
plt.rcParams['lines.linewidth'] = 2

from time import time


def timer_func(func):
    # This function shows the execution time of
    # the function object passed
    def wrap_func(*args, **kwargs):
        t1 = time()
        result = func(*args, **kwargs)
        t2 = time()
        print(f'Function {func.__name__!r} executed in {(t2-t1):.4f}s')
        return result
    return wrap_func
```

1.1 Задание 1

Задайте матрицу A и вектор-столбец f системы линейных уравнений $AX = f$, используя генератор случайных чисел. Очевидно, можно получить решение таким образом: $X = A^{-1}f$ (предварительно проверив, что матрица A не вырожденная) или по правилу Крамера ($x_i = \frac{detA_i}{detA}$, где A_i — матрица, получающаяся из матрицы A заменой i-го столбца на столбец правой части f). Реализуйте и проверьте работоспособность этих методов. Несмотря на простоту использования в Matlab, эти варианты чрезвычайно неэкономичны по числу операций

[2]:
```
N = 5

@timer_func
def default_solution(A, f):
```

```
    X = lin.lstsq(A, f, rcond=None)[0]
    return np.reshape(X, len(X))

@timer_func
def kramer(A, f):
    n = np.size(A, 1)
    X = np.zeros((n, 1))
    det_A = lin.det(A)
    if det_A == 0:
        print("Singular matrix given!")
        return []
    for i in range(n):
        tmp = np.copy(A)
        tmp[:,i] = f.T
        X[i] = lin.det(tmp)/det_A
    return np.reshape(X, len(X))

A = np.random.randint(1, 10, size=(N, N)).astype("float")
f = np.random.randint(-5, 5, size=(N, 1)).astype("float")
print("A =\n", A)
print("f = \n", f)
X1 = default_solution(A, f)
X2 = kramer(A, f)
print("X1 =\n", np.reshape(X1, len(X1)))
print("X2 =\n", X2)
```

```
A =
 [[4. 8. 3. 8. 5.]
 [4. 9. 3. 5. 2.]
 [2. 8. 9. 6. 3.]
 [8. 8. 1. 5. 3.]
 [5. 5. 1. 9. 1.]]
f =
 [[-1.]
 [ 2.]
 [-4.]
 [ 4.]
 [-2.]]
Function 'default_solution' executed in 0.0003s
Function 'kramer' executed in 0.0003s
X1 =
 [ 0.36392811  0.68741977 -0.63564399 -0.73127942 -0.03958066]
X2 =
 [ 0.36392811  0.68741977 -0.63564399 -0.73127942 -0.03958066]
```

Всё работает, как и ожидается, причём левое деление работает очень быстро (что ожидаемо).

1.2 Задание 2

Напишите программу нахождения решения системы линейных урав- нений методом Гаусса с выбором главного элемента.

[3]:
```
@timer_func
def gauss(A_, f_):
    A = A_.copy()
    f = f_.copy()
    B = np.c_[A,f]
    n = np.size(A, 1)
    max_elems = np.zeros(n-1)
    k = 0
    idx = 0
    for i in range(n-1):
        max_elems[i] = np.amax(abs(A[i:,k]))
        idx = np.argmax(abs(A[i:,k]))
        idx = idx + i
        tmp = A[k,:].copy()
        A[k,:] = A[idx,:].copy()
        A[idx,:] = tmp.copy()
        tmp = f[k].copy()
        f[k] = f[idx].copy()
        f[idx] = tmp.copy()
        for j in range(i+1, n):
            coef = A[j,k] / A[i,k]
            A[j,:] = A[j,:] - A[i,:] * coef
            f[j] = f[j] - f[i] * coef
        k += 1
    matr = A.copy()
    b = f.copy()
    X = np.zeros(n)
    X[n-1] = b[n-1] / matr[n-1][n-1]
    for i in range(n-2, -1, -1):
        sum_ = 0
        for j in range(n-1, i-1, -1):
            sum_ += X[j] * matr[i,j]
        X[i] = (b[i] - sum_) / matr[i,i]
    return X

X3 = gauss(A, f)
print("X2 =\n", X2)
print("X3 =\n", X3)
```

```
Function 'gauss' executed in 0.0005s
X2 =
 [ 0.36392811  0.68741977 -0.63564399 -0.73127942 -0.03958066]
X3 =
 [ 0.36392811  0.68741977 -0.63564399 -0.73127942 -0.03958066]
```

Скорее всего, не самая эффективная реализация метода Гаусса.

1.3 Задание 3

Задайте случайным образом матрицу A размерности 20×20 и вектор X. Определите число обусловленности матрицы A с помощью функции cond. Изменяя значения некоторых элементов матрицы A, добейтесь, чтобы её число обусловленности стало больше 10^3 . Используя A и X, найдите вектор $f = AX$. Полагая вектор X неизвестным, решите систему линейных уравнений всеми предложенными выше методами и сравните найденные решения с уже известным. Какой из методов дал более точный результат? Обратите внимание на решения, полученные обычным методом Гаусса и методом с выбором главного элемента.

[4]:
```
N = 20
A = np.random.randint(1, 100, size=(N, N)).astype("float")
X = np.random.randint(-50, 50, size=(N, 1)).astype("float")
while lin.cond(A) <= 10**3:
    A += 100
f = np.matmul(A, X)
X2 = default_solution(A, f)
X3 = kramer(A, f)
X4 = gauss(A, f)
print("X =\n", np.reshape(X, len(X)))
print("X2 =\n", X2)
print("X3 =\n", X3)
print("X4 = \n", X4)
```

```
Function 'default_solution' executed in 0.0004s
Function 'kramer' executed in 0.0011s
Function 'gauss' executed in 0.0022s
X =
 [ 23.   9. -35.   9.  42. -32.  -2. -33. -34. -13.  41.  -4. -14.  30.
 -43.  12. -20. -25.  41.  -7.]
X2 =
 [ 23.   9. -35.   9.  42. -32.  -2. -33. -34. -13.  41.  -4. -14.  30.
 -43.  12. -20. -25.  41.  -7.]
X3 =
 [ 23.   9. -35.   9.  42. -32.  -2. -33. -34. -13.  41.  -4. -14.  30.
 -43.  12. -20. -25.  41.  -7.]
X4 =
 [ 23.   9. -35.   9.  42. -32.  -2. -33. -34. -13.  41.  -4. -14.  30.
 -43.  12. -20. -25.  41.  -7.]
```

Комментировать, вроде как, нечего.

1.4 Дополнительно

Засечем время работы для матрицы большой размерности

[5]:
```
N = 200
A = np.random.randint(1, 100, size=(N, N)).astype("float")
X = np.random.randint(-50, 50, size=(N, 1)).astype("float")
while lin.cond(A) <= 10**3:
    A += 100
f = np.matmul(A, X)
X2 = default_solution(A, f)
X3 = kramer(A, f)
X4 = gauss(A, f)
```

```
Function 'default_solution' executed in 0.0196s
/usr/lib/python3.9/site-packages/numpy/linalg/linalg.py:2158: RuntimeWarning:
overflow encountered in det
  r = _umath_linalg.det(a, signature=signature)
<ipython-input-2-7f42471d9aab>:19: RuntimeWarning: invalid value encountered in
double_scalars
  X[i] = lin.det(tmp)/det_A
Function 'kramer' executed in 0.4923s
Function 'gauss' executed in 0.1451s
```

Видим, что нативная реализация (левое матричное деление) работает быстрее всего. Потом по скорости идёт метод Гаусса. Самым медленным является метод Крамера. Возможно, можно сделать метод Гаусса ещё быстрее, улучшив реализацию.