Нахождение собственных чисел матрицы с помощью степенного метода и классического метода Якоби

Курдоякова Марина

Задание: Требуется найти максимальное собственное число заданной матрицы при помощи степенного методом, обратным степенным - наименьшее, и отдельным методом найти все собственные числа, в мое случае - классическим методом Якоби.

Матрица задается как (8 вариант):

$$a_{i,i} = a$$

$$a_{i,i+1} = \frac{i(i+1)}{s(s+1)}b = a_{i+1,i}$$

$$a_{i,k} = \frac{2}{i^2+k^2}, |i-k| > 1$$

где s - порядок матрицы, a,b некоторые константы
В результате решения должны быть получены значения λ_{max}, λ_{min} и λ - все собственные числа матрицы

Расчетные формулы:

1. Степеной метод:
Пусть A - данная матрица порядка s, X - произвольный вектор, $\{\lambda_i\}_{i=1}^s$ - собственные числа матрицы. Занумеруем собственные числа в порядке убывания их модулей в предположении, что существует наибольшее по модулю собственное число: $|\lambda_1| > |\lambda_2| \leq ...$ Предположим, что матрица A обладает полным набором собственных векторов:

$$u_i, i = 1, ..., s : Au_i = \lambda_i u_i$$

Будем рассматривать следующую последовательность векторов:

$$X, AX, A(AX) = A^2X, ..., A^nX = A(A^{n-1}X), ...$$

Разложим по базису $\{u_i\}_{i+1}^s$ каждый вектор из послежовательности:

$$X = c_1u_1 + c_2u_2 + ... + c_su_s,$$

$$AX = c_1\lambda_1u_1 + c_2\lambda_2u_2 + ... + c_s\lambda_su_s,$$

$$A^nX = c_1\lambda_1^nu_1 + c_2\lambda_2^2u_2 + ... + c_s\lambda_s^nu_s.$$

При достаточно больших n первое слагаемое в разложении векторов будет преобладать над остальными, тогда $A^nX \approx c_1\lambda_1^nu_1$. При вычислении каждого следующего приближения X_{n+1} будем выполнять нормировку на первую его компоненту:

$$\hat{X}_{n+1} = AX_n, X_{n+1} = \frac{\hat{X}_{n+1}}{\{\hat{X}_{n+1}\}_1}$$

Таким образом, если $X_n \approx u_1$, то $AX_n \approx \lambda_1u_1$ и $\{\hat{X}_{n+1}\}_1 \approx \lambda_1$.

Отсюда следует алгорит поиска наибольшего собственного числа:

(a) Вычисление элементов матрицы A, задаваемое в формульном виде.
(b) Задание начального начального значения вектора X (вектор из единиц).
(c) Вычисление нового приближения $X_{n+1} = AX_n$ пока разность между нормами двух последовательных приближений не окажется меньше заданного eps
(d) Вычисление $\lambda_1 = \{\hat{X}_{n+1}\}_1$

Алгоритм нахождения наименьшего собственного числа:

(a) Нахождения матрицы A^{-1}.

(b) Задание начального начального значения вектора X (вектор из единиц).

(c) Вычисление нового приближения $X_{n+1} = AX_n$ пока разность между нормами двух последовательных приближений не окажется меньше заданного eps

(d) Вычисление $\lambda_1 = (\{\hat{X}_{n+1}\}_1)^{-1}$

2. Классический метод Якоби:
Суть алгоритма заключается в том, чтобы для заданной симметрической матрице $A = A^{(0)}$ построить последовательность ортогонально подобных матриц $A^{(1)}, A^{(2)}, \ldots, A^{(m)}$, сходящуюся к диагональной матрице, на диагонали которой стоят собственные значения A. Для построения этой последовательности применяется специально подобранная матрица вращения J_i, такая что норма наддиагональной части $||A^{(i)}||_{off} = \sqrt{\sum_{1\le j\le k\le n}(a_{jk}^{(i))2}}$ уменьшается при каждом двустороннем вращении матрицы $A^{(i+1)} = J_i^T A^{(i)} J_i$. Это достигается выбором максимального по абсолютной величине элемента матрицы $A^{(i)}$ и его обнулением в матрице $A^{(i+1)}$. Если он расположен в j-й строке и k-м столбце, то

$$
J_i = \begin{array}{c c}
 & \begin{array}{ccccccc} & & j & & k & & \end{array} \\
\begin{array}{c} \\ \\ \\ j \\ \\ k \\ \\ \\ \\ \end{array} &
\begin{pmatrix}
1 & & & \cdots & & & \\
 & 1 & & \cdots & & & \\
 & & & \cdots & & & \\
 & \cdots & \cos(\theta) & \cdots & -\sin(\theta) & \cdots & \\
 & & & \cdots & & & \\
 & \cdots & \sin(\theta) & \cdots & \cos(\theta) & \cdots & \\
 & & & \cdots & & & \\
 & & & \cdots & & 1 & \\
 & & & \cdots & & & 1
\end{pmatrix}
\end{array}
$$

Если обозначить $s = \sin(\theta)$ и $c = \cos(\theta)$, то матрица $A^{(i+1)}$ состоит из следующих элементов, отличающихся от элементов $A^{(i)}$:

$$
\begin{aligned}
a_{jj}^{(i+1)} &= c^2 a_{jj}^{(i)} - 2sc a_{jk}^{(i)} + s^2 a_{kk}^{(i)} \\
a_{kk}^{(i+1)} &= s^2 a_{jj}^{(i)} + 2sc a_{jk}^{(i)} + c^2 a_{kk}^{(i)} \\
a_{jk}^{(i+1)} = a_{kj}^{(i+1)} &= (c^2 - s^2) a_{jk}^{(i)} + sc(a_{kk}^{(i)} - a_{jj}^{(i)}) \\
a_{jm}^{(i+1)} = a_{mj}^{(i+1)} &= c a_{jm}^{(i)} - s a_{km}^{(i)} \qquad m \ne j, k \\
a_{km}^{(i+1)} = a_{mk}^{(i+1)} &= s a_{jm}^{(i)} + c a_{km}^{(i)} \qquad m \ne j, k \\
a_{ml}^{(i+1)} &= a_{ml}^{(i)} \qquad m, l \ne j, k
\end{aligned}
$$

Можно выбрать θ так, чтобы $a_{jk}^{(i+1)} = 0$ и $a_{kj}^{(i+1)} = 0$. Отсюда следует равенство:

$$
\frac{a_{jj}^{(i)} - a_{kk}^{(i)}}{2a_{jk}^{(i)}} = \frac{c^2 - s^2}{2sc} = \frac{\cos(2\theta)}{\sin(2\theta)} = \mathrm{ctg}(2\theta) = \tau
$$

Если $a_{jj}^{(i)} = a_{kk}^{(i)}$, то выбирается $\theta = \frac{\pi}{4}$, в противном случае вводится $t = \frac{s}{c} = \mathrm{tg}(\theta)$ и тогда $t^2 - 2t\tau + 1 = 0$. Решение квадратного уравнения даёт

$$
t = \frac{sign(\tau)}{|\tau| + \sqrt{1 + \tau^2}}, \qquad c = \frac{1}{\sqrt{1 + t^2}}, \qquad s = tc
$$

Вычисление останавливается, когда выполняются критерии близости к диагональной матрице. Это малость суммы квадратов внедиагональрных элемпентов:

$$t^2(A) = \sum_{k \neq i} a_{ik}^2$$

Будем считать, что условие выполняется когда $t < eps$, где $eps = 0.0000001$

Код программы: .

```
import math
import numpy as np
from numpy.linalg import norm, inv
from numpy import array,identity,diagonal

print('Enter the size of matrix')
s = int(input())

a_const = 3.0
b_const = 1.0

A = np.zeros((s,s))

for i in range(s):
    A[i,i] = a_const

for i in range(s-1):
    A[i,i+1] = ((i+1)*(i+2)/(s*(s+1))) * b_const

for i in range(s-1):
    A[i+1,i] = A[i,i+1]

for i in range(s):
    for k in range(s):
        if abs(i-k) > 1:
            A[i,k] = 2/((i+1)**2+(k+1)**2)
print (A)

def firstmetod(A):

    X=np.array([1]*s)
    X.resize(s,1)

    x=np.dot(A,X)
    x1=float(x[0][0])
    d = norm(x,ord=2) - norm(X,ord=2)
    n=0

    while abs(d) > 0.00000001:   # 10^8
        X=x
        x=np.dot(A,X)
        x1=float(x[0][0])
        n=n+1
        res = x
        x=x/x1
```

```
        d = norm(X,ord=2) - norm(x,ord=2)
    print ('num of itt',n)
    return res

w , v = np.linalg.eig(A)

lambdamax = firstmetod(A)
maxl = lambdamax[0][0]
print ('calcul lambdamax: ', maxl)

maxW = max(w,key=abs)
print ('theory lambdamax: ', maxW)

print('----------------------------')

ainv = inv(A)
lambdamin = firstmetod(ainv)
lambdamin = lambdamin**(-1)
minl = lambdamin[0][0]
print ('calcul lambdamin: ', minl)

minW = min(w,key=abs)
print ('theory lambdamin:  ', minW)

B=array([[0,2,0], [2,3,0], [0,0,16]])
# print(B)

def maxElem(a): # Find largest off-diag. element a[k,l]
    n = len(a)
    aMax = 0.0
    for i in range(n-1):
        for j in range(i+1,n):
            if abs(a[i,j]) >= aMax:
                aMax = abs(a[i,j])
                k = i
                l = j
    return aMax, k,l

def Matrix(A,c,s):
    n=len(A)
    C = np.zeros((n,n))
    aMax, j,k = maxElem(A)
    C = A.copy()

    C[j,j] = c*c*A[j,j] + s*c*A[j,k] + s*c*A[k,j] + s*s*A[k,k]
    C[k,k] = s*s*A[j,j] - s*c*A[j,k] - s*c*A[k,j] + c*c*A[k,k]

    C[j,k] = 0
    C[k,j] = 0

    for m in range(n):
        if (m != j) and (m != k):
            C[j,m] = c*A[j,m] + s*A[k,m]
            C[m,j] = c*A[j,m] + s*A[k,m]
```

```
            C[k,m] = - s*A[j,m] + c*A[k,m]
            C[m,k] = - s*A[j,m] + c*A[k,m]
    return C

def diag(A):
    tA = 0
    n = len(A)
    for i in range(n):
        for j in range(n):
            if i != j:
                tA = tA + A[i,j]**2
    tA=math.sqrt(abs(tA))
    return tA

el = s*(s-1)/2

def Jacobi(A):
    n = len(A)
    eps = 10**(-12)
    h = 0
    tA=1
    aMax, j,k = maxElem(A)

    while (tA>eps):
        aMax, j,k = maxElem(A)

        if A[j,j] == A[k,k]:
            tet = math.pi/4
            s = math.sin(tet)
            c = math.cos(tet)
        else:
            tau = (A[j,j]-A[k,k])/(2*A[j,k])
            t = np.sign(tau)*1/(abs(tau)+math.sqrt(1+tau**2))
            c = 1/math.sqrt(1+t**2)
            s = c*t

        A = Matrix(A,c,s)
        tA = diag(A)
        if h % el == 0:
            print('tA = ', tA)
            print('eps = ', eps)
        h = h +1
    return A, h

A, h = Jacobi(A)

I = np.zeros(s)
print('Все собстченные числа: ')

for i in range(s):
    I[i] = A[i,i]
    print(I[i])

print('Число итераций = ', h)
```

Результаты: Были выбраны значения констант а = 3, и в = 1. Значения вычисленных lambdamax и lambdamin были получены степеным методом, теоретические lambdamax и lambdamin были получены с помощь встроеных функций. Все собственные числа получены классическим методом Якоби.

```
Размер матрицы =  3
----------------------------
Число итераций =  77
Вычисленная lambdamax:    3.610246274827007
Теоретическая lambdamax:  3.6102462883978355
----------------------------
Число итераций =  169
Вычисленная lambdamin:     2.498720341951175
Теоретическая lambdamin:   2.498720340741603
tA =  0.36817870057290875
eps =  1e-12
tA =  0.0007220715749897634
eps =  1e-12
tA =  4.574193113217442e-14
eps =  1e-12
Все собстченные числа:
2.8910333708605647
2.4987203407416025
3.610246288397832
Число итераций =  7
```

Потихоньку изменяем размер матрицы:

```
Размер матрицы =  6
----------------------------
Число итераций =  98
Вычисленная lambdamax:    3.975608807623351
Теоретическая lambdamax:  3.9756088139950205
----------------------------
Число итераций =  113
Вычисленная lambdamin:     2.1480529630353296
Теоретическая lambdamin:   2.14805296306643
tA =  0.9142391802735635
eps =  1e-12
tA =  0.04375021095425662
eps =  1e-12
tA =  0.00015827340940790315
eps =  1e-12
tA =  7.6223448844537e-10
eps =  1e-12
Все собстченные числа:
2.9029312593671204
2.9553625147854428
3.298018317591153
2.720026131194832
2.14805296306643
3.9756088139950196
Число итераций =  50
```

```
Размер матрицы =  10
----------------------------
Число итераций =  139
Вычисленная lambdamax:    4.154570597502463
Теоретическая lambdamax:  4.154570600498099
----------------------------
Число итераций =  91
Вычисленная lambdamin:     1.8984228248713049
Теоретическая lambdamin:   1.8984228248442836
tA =  1.454658713314922
eps =  1e-12
tA =  0.04894497679232096
eps =  1e-12
tA =  0.0002568830818286617
eps =  1e-12
tA =  1.7363381207371348e-10
eps =  1e-12
Все собстченные числа:
2.794079379789876
2.997799495697104
3.3371945113233723
2.9051976473956067
2.7535259540815633
3.1017308323698236
2.449721625393398
3.6077571286068877
1.8984228248442911
4.154570600498099
Число итераций =  149



Размер матрицы =  20
----------------------------
Число итераций =  194
Вычисленная lambdamax:    4.389358843766026
Теоретическая lambdamax:  4.389358844528605
----------------------------
Число итераций =  109
Вычисленная lambdamin:     1.6242624143749624
Теоретическая lambdamin:   1.6242624143715791
tA =  2.410007557889636
eps =  1e-12
tA =  0.050436538137009344
eps =  1e-12
tA =  0.00016605491542858578
eps =  1e-12
tA =  1.229878271233252e-09
eps =  1e-12
Все собстченные числа:
2.7716055885408326
3.0828367135571293
3.494574752970583
2.897876763021174
```

```
2.9483655053663553
2.98854150167272
2.808502972595166
3.0471880725884675
2.6946127146001717
3.261581801226806
2.541883865250226
3.3810226845575193
2.336314659519744
3.681961956482401
2.0519007954555595
3.9537912221104357
2.8901076035112188
3.153709568073381
1.6242624143715794
4.389358844528624
Число итераций =  636



Размер матрицы =  50
----------------------------
Число итераций =  346
Вычисленная lambdamax:    4.632813943663192
Теоретическая lambdamax:  4.6328139437731135
----------------------------
Число итераций =  152
Вычисленная lambdamin:     1.3698757644767705
Теоретическая lambdamin:   1.3698757644764807
tA =  4.193863268549275
eps =  1e-12
tA =  0.02641324021726503
eps =  1e-12
tA =  7.648123316459451e-05
eps =  1e-12
tA =  2.822242325706817e-10
eps =  1e-12
Все собстченные числа:
2.7594271267650363
2.877079947350648
3.3803182832644856
3.1024996618304774
3.0006550245824535
2.9688708580667
2.9794867120960093
2.990063324167661
2.961447297502711
3.02468186283919
2.9232098977442247
3.007830600548875
2.944073344639982
3.0877203359913694
2.870393124995586
3.15985415860905
```

```
2.837812200828587
3.044843632237744
2.7583032634723232
3.239021894669235
2.710383442197392
3.285936537722593
2.6562125736115165
3.1274218226020274
2.9757979541858153
3.4782403207501167
2.5259228596578374
3.066987766734394
2.447718140584405
3.554310834643262
2.8006017980644042
3.336896927704877
2.257562053425766
3.7434813622191956
2.595034403586842
3.415427878160481
2.0046814294405446
3.9960354407765863
2.3589274524470984
3.642956489766025
1.8423794308375694
4.159139662922146
2.898743603186198
3.1968930204494947
1.6413682594710428
4.359202428088218
2.140850163153898
3.8606036831589003
1.3698757644764912
4.632813943773142
Число итераций =  4019
```