2019 год

Практикум по численным методам: минимизация квадратичной функции

Мусаева Аида, группа 301

1 Постановка задачи

Пусть X – евклидово n-мерное пространство, обозначаемое далее $\mathbb{R}^n$. В пространстве $\mathbb{R}^n$ рассмотрим квадратичную функцию:
$f(x) = \frac{1}{2}x^T Ax + x^T b$,
где A – положительно определенная матрица, т.е. $A = A^T$ и имеют место неравенства:
$m\|x\|^2 \leq (x, Ax) \leq M\|x\|^2, m \geq 0$

Для такой функции существует единственная точка минимума $\overline{x}$, удовлетворяющая СЛАУ $Ax + b = 0$.
Задание:

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 1 \\ 1 & 8,2 & -1 \\ 1 & -1 & 10.2 \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} 1 \\ -2 \\ 3 \end{pmatrix}$$

2 Методы наискорейшего спуска

Поставим следующую задачу: имея точку $x_k \in \mathbb{R}^n$ построить точку $x_{k+1} \in \mathbb{R}^n$ такую, чтобы выполнялось соотношение:
$f(x_{k+1}) < f(x_k)$ Будем искать точку x_{k+1} в следующем виде:

$$x_{k+1} = x_k + \mu_k q, \tag{1}$$

$$\mu_k = -\frac{q^T(Ax_k + b)}{q^T * Aq}, \tag{2}$$

где q - заданный вектор из $\mathbb{R}^n$, называемый направлением спуска, а μ_k – искомый параметр, называемый шагом метода в направлении спуска. Продолжая указанные построения, получим последовательность x_k, которую естественно назвать последовательностью убывания для функции f.

2.1 Метод наискорейшего градиентного спуска

Если в формуле (1) считать, что $q = grad\, f(x_k) = Ax_k + b$, то соответствующий метод построения последовательности x_k называют *градиентным методом.* Если к тому же шаг метода μ_k выбирается по формуле (2), то такой метод называют (одношаговым) *методом наискорейшего градиентного спуска* (МНГС). В этом случае формула (2) принимает вид: $\mu_k = -\frac{\|Ax_k+b\|^2}{(Ax_k+b)^T A(Ax_k+b)}$. Метод наискорейшего градиентного спуска сходится для любого начального вектора x_0

2.1.1 Код программы

```
import numpy as np
import math
from pylab import *
from sympy import *
x, y, z  = symbols('x y z')
f = 2*x**2 + 4.1*y**2 + 5.1*z**2 + x*y - y*z + x*z + x - 2*y + 3*z + 11
```

```
def grad(f,dot):
    x, y, z = symbols('x y z')
    a = np.array([x, y, z])
    grad0 = np.array([f.diff(i) for i in a])
    return np.array([grad0[i].subs([(x,dot[0]), (y,dot[1]), (z,dot[2])])
           for i in range(3)])
def norm(a):
    return math.sqrt(a[0] ** 2 + a[1] ** 2 + a[2] ** 2)
def steepestGradDesc(f, x_prev=np.array([0,0,0]), eps = 0.000001):
    A = np.array([[4, 1, 1], [1, 8.2, -1], [1, -1, 10.2]])
    x, y, z  = symbols('x y z')
    a = np.array([x, y, z])
    phi_prev = - ((grad(f,x_prev).transpose()).dot(grad (f,x_prev)))
            /((grad(f,x_prev).transpose()).dot(A.dot((grad(f,x_prev)))))
    x_next = x_prev + phi_prev * grad(f,x_prev)
    i = 1;
    while (norm(x_next-x_prev)>eps):
        x_prev = x_next
        phi_prev = - ((grad(f, x_prev).transpose()).dot(grad(f, x_prev)))
                / ((grad(f, x_prev).transpose()).dot(A.dot((grad(f, x_prev)))))
        x_next = x_prev + phi_prev * grad(f, x_next)
        print(x_next)
        i+=1
    return(x_next,i)
xOpt, iter  = steepestGradDesc(f)
print(xOpt, iter)
```

2.1.2 Результат работы программы

$$\mathbf{X} = \begin{pmatrix} -0.249677203320831 \\ 0.244389825968740 \\ -0.245679518667231 \end{pmatrix}$$

16 итераций

2.2 Метод наискорейшего покоординатного спуска

В случае выбора направлений спуска q в формуле (1) на каждом шаге в виде $q = e^i = (\underbrace{0, \cdots, 0, 1}_{i}, 0, \cdots, 0)^T$, где $e^i - i$-ый орт пространства $\mathbb{R}^n$, метод носит название *метода покоординатного спуска*. При выборе шага метода μ_k по формуле (2) его называют *методом наискорейшего покоординатного спуска*(МНПС). В этом случае формула (2) принимает вид:
$\mu_k = -\frac{e^i(Ax_k+b)}{e^i * Ae^i}$

Метод наискорейшего покоординатного спуска сходится для любого начального вектора x_0.

2.2.1 Код программы

```
import numpy as np
import math
import pylab
```

```
from mpl_toolkits.mplot3d import axes3d, Axes3D
from sympy import *
x, y, z  = symbols('x y z')
f = 2*x**2 + 4.1*y**2 + 5.1*z**2 + x*y - y*z + x*z + x - 2*y + 3*z + 11


def grad(f,dot):
    x, y, z = symbols('x y z')
    a = np.array([x, y, z])
    grad0 = np.array([f.diff(i) for i in a])
    return np.array([grad0[i].subs([(x,dot[0]), (y,dot[1]), (z,dot[2])])
            for i in range(3)])


def norm(a):
    return math.sqrt(a[0] ** 2 + a[1] ** 2 + a[2] ** 2)


def steepestCoordinateDesc(f, x_prev=np.array([0,0,0]), eps = 0.000001):
    A = np.array([[4, 1, 1], [1, 8.2, -1], [1, -1, 10.2]])
    x, y, z  = symbols('x y z')
    a = np.array([x, y, z])
    def e(i):
        return np.array([int(j == i) for j in range(3)]).transpose()
    phi_prev = - (e(0).dot(grad(f,x_prev)))/
            (e(0).transpose().dot(A.dot(e(0).transpose())))
    x_next = x_prev + phi_prev * e(0).transpose()
    iter = 1
    i = 0
    while (norm(x_next-x_prev)>eps):
        x_prev = x_next
        i+=1
        i%=3
        phi_prev = - (e(i).dot(grad(f,x_prev)))/
                (e(i).transpose().dot(A.dot(e(i).transpose())))
        x_next = x_prev + phi_prev * e(i).transpose()
        print(x_next)
        iter+=1
    return(x_next, iter)

xOpt, it = steepestCoordinateDesc(f)
print(xOpt, it)
```

2.2.2 Результат работы программы

$$\mathbf{X} = \begin{pmatrix} -0.249678219015254 \\ 0.244390165391223 \\ -0.245679570156228 \end{pmatrix}$$

18 итераций