|  |  |  |
| --- | --- | --- |
| Методы оптимизации  Лабораторная работа № 1  Численные методы оптимизации  Одномерная и многомерная безусловная оптимизация | ФИО студента | Бородин Н.Ю. |
| Группа | ИВТ-363 |
| Преподаватель | Асанова Н.В. |
| Дата |  |
| Оценка |  |

Вариант:

1.1. Поиск отрезка локализации минимума [a,b]

1.2.1. Метод пассивного поиска

2.1.2. Метод конфигураций Хука-Дживса

2.2.3. Метод переменной метрики Дэвидона-Флетчера-Пауэлла

2.3.1. Метод Ньютона-Рафсона

Результаты:

**Метод Дэвиса-Свенна-Кэмпи**

**шаг** h =1.

*Функция (x-1)­2*

**Начальные** **данные**: и

**Результат**: [2;4]

*Функция 4x3-8x2-11x+5*

**Начальная** **точка**: 1

**Результат**: [1; 3]

*Функция x + 3/x2*

**Начальная** **точка**: 1

**Результат**: [1; 3]

*Функция (x+2.5)/(4-x2)*

Начальная точка: 0.1

Результат: [-1.9; 0.1]

*Функция –sin(x) – sin(3x)/3*

Начальная точка: 1

Шаг: 0.5

Результат: [0.5;1.5]

Начальная точка: 2.5

Шаг: 0.5

Результат: [2;3]

*Функция -2sin(x) – sin(2x) – 2sin(3x)/3*

Начальная точка: 0

Результат: [0;2]

**Метод параболической аппроксимации Пауэлла**

Функция (x-1)­2

Отрезок с минимумом: [-2; 4]

Точка минимума: 1.0058574218749998

Функция 4x3-8x2-11x+5

Отрезок с минимумом: [0; 10]

Точка минимума: 1.8326996731031098

Функция x + 3/x2

Отрезок с минимумом: [0; 3]

Точка минимума: 1.8173819863109173

Функция (x+2.5)/(4-x2)

Отрезок с минимумом: [0; 3]

Точка минимума: -0.9984766277128898

Функция –sin(x) – sin(3x)/3

Отрезок с минимумом: [-2;2]

Точка минимума: 0.7858148233705298

Функция -2sin(x) – sin(2x) – 2sin(3x)/3

Отрезок с минимумом: [0;3]

Точка минимума: 0.7852612983363028

**Метод Покоординатного спуска**

Функция Химмельблау 1

Начальная точка: [0.0, 0.0]

Точка минимума: [4.999986424870496, 6.00002546129407]

Значение функции в точке: 1.3854140599495976e-09

Функция Химмельблау 2

Начальная точка: [0.0, 0.0]

Точка минимума: [2.9999801462802074, 2.0000584207212215]

Значение функции в точке: 4.940900753574346e-08

Функция Вуда

Начальная точка: [2.0, -1.0, -3.0, -1.0]

Точка минимума: [0.994809399957602, 1.0003854114587936, 1.0006332832720233, 1.002769398904669]

Значение функции в точке: 0.01186464647122445

Функция Пауэлла

Начальная точка: [3.0, -1.0, 0.0, 1.0]

Точка минимума: [0.0030000000000029747, -0.0010000000000001889, -0.04299999999999844, -0.04299999999999933]

Значение функции в точке: 0.0001459752

**Метод Флетчера-Ривса**

Функция Химмельблау 1

Начальная точка: [0.0, 0.0]

Точка минимума: [ 5.00000016 6.00000005]

Значение функции в точке: 0.0000000000

Функция Химмельблау 2

Начальная точка: [0.0, 0.0]

Точка минимума: [ 3.00000011 2.00000015]

Значение функции в точке: 0.0000000000

Функция Вуда

Начальная точка: [2.0, -1.0, -3.0, -1.0]

Точка минимума: [ 0.9999668 0.99993354 1.00003269 1.00006554]

Значение функции в точке: 0.0000000039

Функция Пауэлла

Начальная точка: [3.0, -1.0, 0.0, 1.0]

Точка минимума: [ 0.01699837 -0.00169979 0.00781747 0.00782133]

Значение функции в точке: 0.0000001613

**Метод Ньютона-Рафсена**

Функция Химмельблау 1

Начальная точка: [0.0, 0.0]

Точка минимума: [ 5.00000013 5.99999958]

Значение функции в точке: 0.0000000000

Функция Химмельблау 2

Начальная точка: [0.0, 0.0]

Точка минимума: [ 3.0000072 1.99996596]

Значение функции в точке: 0.0000000167

Функция Вуда

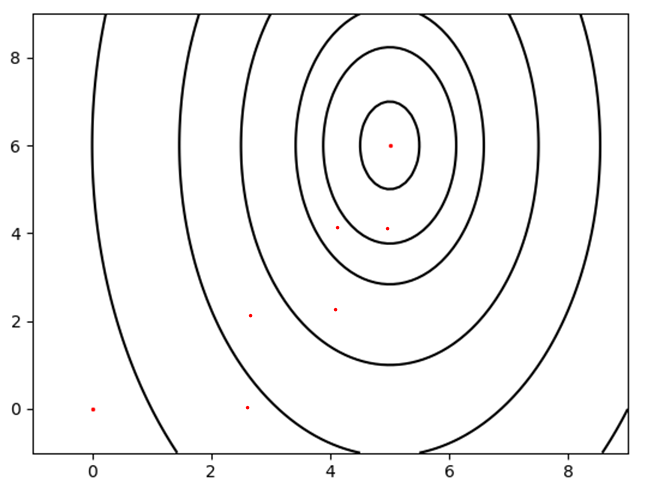
Начальная точка: [2.0, -1.0, -3.0, -1.0]

Точка минимума: [ 1.00249064 1.00503246 0.99687456 0.99382859]

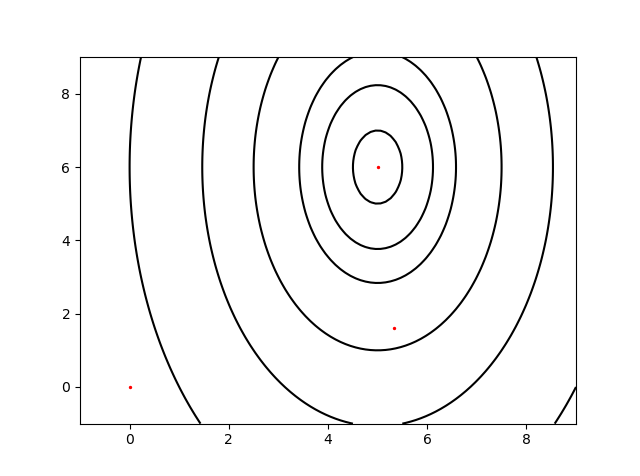
Значение функции в точке: 0.0000421358

Графики поиска минимума:

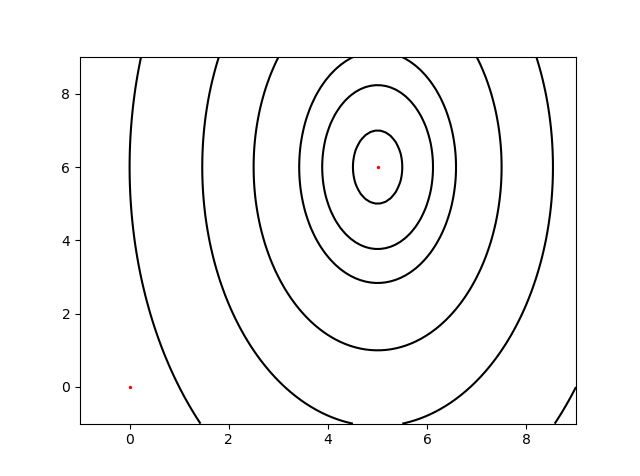
Алгоритм покоординатного спуска.



Алгоритм Флетчера-Ривса.



Алгоритм Ньютона-Рафсона:



Код:

Метод Дэвиса-Свенна-Кэмпи:

def DSK(x0, h, f):

    xcur = x0

    xnext = x0+h

    cur = f(xcur)

    next = f(xnext)

    if (cur>next):

        a = x0

        k = 2

    else:

        xcur = x0

        xnext = x0-h

        next = f(xnext)

        if (next>=cur):

            return [x0-h, x0+h]

        else:

            b = x0

            h = -h

            k = 2

    while (True):

        xcur = xnext

        xnext = x0 + h\*pow(2,(k-1))

        cur = next

        next = f(xnext)

        if (cur<next):

            if (h>0):

                b = xnext

            else:

                a = xnext

            return [a,b]

        else:

            if (h>0):

                a = xcur

            else:

                b = xcur

            k=k+1

Метод параболической аппроксимации Пауэлла:

def powell(func, a: float, b: float, eps =0.01):

# step 1

x1 = a

x2 = (a + b) / 2

x3 = b

X = [x1, x2, x3]

# step 2

while (True):

min\_x = X[0]

for i in X:

if (func(i)<func(min\_x)):

min\_x = i

# step 3

num = (X[1] \*\* 2 - X[2] \*\* 2) \* func(X[0]) + (X[2] \*\* 2 - X[0] \*\* 2) \* func(X[1]) + (X[0] \*\* 2 - X[1] \*\* 2)\* func(X[2])

denum = (X[1] - X[2]) \* func(X[0]) + (X[2] - X[0]) \* func(X[1]) + (X[0] - X[1]) \* func(X[2])

X.append(0.5 \* (num / denum))

# step 4

if(abs(X[3] - min\_x) <= eps):

break

#step 5

X.sort()

#step 6

max\_x = 0

for i in range(4):

if (func(X[i])>func(X[max\_x])):

max\_x = i

X.pop(max\_x)

#step 7

return X[-1]

Метод покоординатного спуска (метод Гаусса – Зейделя):

def coordinate\_descent(func: Callable[..., float], N: int, odm: Callable[[Callable[[float], float], float, float], float], eps: float = 0.0001, step\_crushing\_ratio: float = 0.1):

k = 0

h = np.array([1.0]\*N)

x\_points = [ [0]\*N ]

def euclidean\_norm(h: np.array):

return np.sqrt((h\*\*2).sum())

while euclidean\_norm(h) > eps:

x\_points.append([0]\*N)

for i in range(N):

args = x\_points[k].copy()

def odm\_func(x):

nonlocal i, func, args

args[i] = x

return func(\*args)

ak = odm(odm\_func, args[i], h[i])

x\_points[k+1][i] = ak

if any([abs(x\_points[k+1][i] - x\_points[k][i]) > eps for i in range(N)]):

k+=1

continue

h \*= step\_crushing\_ratio

return x\_points[k+1]

Метод Флетчера-Ривса:

machineAcc = 0.000000001

import sys

import numpy as np

from scipy import optimize

Path = []

def FR(x0, h, e, f):

    xcur = np.array(x0)

    Path.append(xcur)

    h = np.array(h)

    n = len(x0)

    k = 0 # step1

    grad = optimize.approx\_fprime(xcur, f, e\*\*4) # step2

    prevgrad = 1

    pk = -1\*grad

    while (any([abs(grad[i]) > e\*\*2 for i in range(n)])): # step3

        if (k%n==0): # step4

            pk = -1\*grad

        else:

            bk = (np.linalg.norm(grad)\*\*2)/(np.linalg.norm(prevgrad)\*\*2) # step5

            prevpk = pk

            pk = -1\*grad + bk\*prevpk # step6

        a = (optimize.minimize\_scalar(lambda x: f(xcur+pk\*x), bounds=(0,)).x)

        xcur = xcur + a\*pk #step8

        Path.append(xcur)

        k=k+1 #step8

        prevgrad=grad

        grad=optimize.approx\_fprime(xcur, f, e\*\*4) #step2

    return xcur #step10

Метод Ньютона-Рафсона:

import sys

import numpy as np

from scipy import optimize

def NR(x0, h, e, f, hess\_f):

xcur = np.array(x0)

n = len(x0)

grad = optimize.approx\_fprime(xcur, f, e\*\*4) # step2

while (any([pow(abs(grad[i]),1.5) > e for i in range(n)])): # step3

h = np.linalg.inv(hess\_f(xcur)) # step 4 & 5

pk = (-1\*h).dot(grad) # step 6

a = (optimize.minimize\_scalar(lambda a: f(xcur+pk\*a), bounds=(0,)).x) #step7

xcur = xcur + a\*pk #step8

grad=optimize.approx\_fprime(xcur, f, e\*e) #step2

return xcur #step10