Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра автоматизированных систем (АСУ)

**Оптимизация функции двух переменных**

Отчет по лабораторной работе № 3 по дисциплине

«Методы оптимизации»

|  |
| --- |
| Выполнили: |
| Студенты гр. 439-1 |
| Зозуля Е.Д.  Рахматуллин Т.Т. |
| |  | | --- | | Руководитель: | | А.А. Шелестов | | « »\_\_\_\_\_\_\_\_\_\_2021 г. |   16.12.2021 г. |
|  |

1. **Задание**

Найти минимум функции двух переменных. Использовать следующие методы:

А) два прямых метода (симплексный метод*,* метод Хука-Дживса);

Точность .

Вариант задания:

9) .

График функции представлен на рисунке 1.1

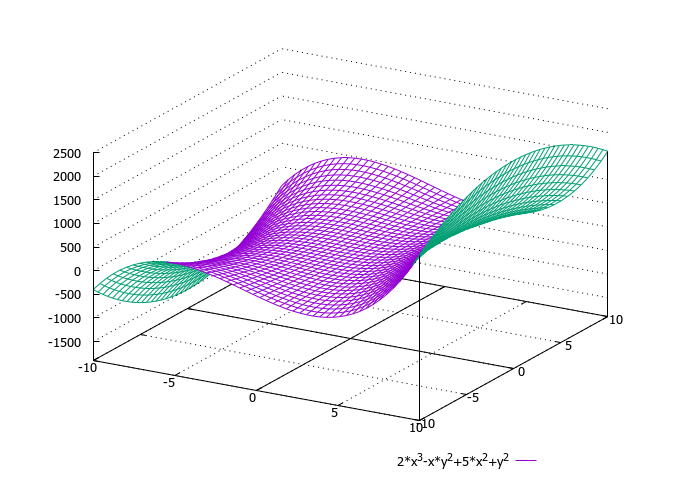


Рисунок 1.1 – График функции

**Симплексный метод**

Метод поиска по симплексу был предложен в 1962 г. Спендли, Хекстом и Химсвортом. Этот метод называют последовательным симплекс-методом (ПСМ). Следует отметить, что указанный метод и другие подобные методы не имеют отношения к симплекс-методу линейного программирования, а сходство названий носит чисто случайный характер.

Определение: В -мерном эвклидовом пространстве -мерный симплекс представляет собой фигуру, образованную  точками (вершинами), не принадлежащими одновременно ни одному пространству меньшей размерности.

В одномерном пространстве симплекс есть отрезок прямой; в двумерном – треугольник; в трехмерном – треугольная пирамида (тетраэдр) и т.д.

Симплекс называется регулярным, если расстояния между вершинами равны. В ПСМ используются регулярные симплекс-планы.

Из любого симплекса, отбросив одну его вершину, можно получить новый симплекс, если к оставшимся добавить всего лишь одну точку.

*Алгоритм:*

Шаг 1. Задается точность . Определяются координаты исходного симплекса с помощью матрицы:

где - номер вершины, -номер координаты, ; - размерность вектора ;

; .

Шаг 2. Вычислить значение функции в каждой вершине. Определить «наихудшую» вершину , в которой значение функции максимально и «наилучшую» вершину , в которой значение функции минимально

,

.

Оставшуюся вершину обозначим.

Шаг 3. Найти центр тяжести всех вершин за исключением наихудшей:

.

Шаг 4. Выполнить операцию отражения наихудшей вершины через центр тяжести ( - коэффициент отражения)

.

Шаг 5. Сравниваются значения и:

1. Если <, то получено наименьшее значение функции и выполняется растяжение с коэффициентом :
2. Если <, то заменяем точку на и переходим к шагу 8
3. Если>, то заменяем точку на (точку отбрасываем) и переходим к шагу 8
4. Если>, но <, то заменяем точку на и переходим к шагу 8
5. Если>,>, то переходим на шаг 6.

Шаг 6. Сравниваются значения и:

Если >, то выполняем сжатие

Если <, то выполняем сжатие.

Шаг 7. Сравниваются значения и :

1. Если <, то заменяем точку на и переходим к шагу 8
2. Если >, то уменьшаем размерность симплекса делением пополам расстояния от каждой точки симплекса до . Т.е. точка заменяется на точку . Переходим к шагу 8.

Шаг 8. Проверить условие окончания

а) если , где , то процесс поиска можно завершить и в качестве приближенного решения взять наилучшую точку текущего многогранника .

б) если переходим к шагу 2.

**Метод Хука–Дживса**

Процедура Хука–Дживса представляет собой комбинацию двух поисков:

а) **"исследующий" поиск** (для выявления характера локального поведения ЦФ и определения направления движения вдоль "оврагов") с циклическим изменением переменных;

б) **ускоряющий поиск** по образцу с использованием определен­ных эвристических правил.

**Исследующий поиск**. Выбирается некоторая исходная точка . Задается величина шага , которая может быть различной для разных координатных направлений и изменяться в процессе поиска.

Если значение ЦФ в пробной точке меньше значения ЦФ в исходной точке, то шаг поиска успешный. В противном случае из исходной точки делается шаг в противоположном направлении. После перебора всех координат исследующий поиск завершается. Полученная точка называется базовой.

**Поиск по образцу**. Осуществляется шаг из полученной базовой точки вдоль прямой, соединяющей эту точку с предыдущей базовой. Новая точка образца определяется по формуле:

.

Как только движение по образцу не приводит к уменьшению ЦФ, точка фиксируется в качестве временной базовой точки и вновь проводится исследующий поиск. Если в результате получается точка с меньшим значением ЦФ, чем в точке , то она рассматривается как новая базовая точка . Но если исследующий поиск неудачен, то следует вернуться в точку и провести исследующий поиск с целью выявления нового направления минимизации. В конечном итоге возникает ситуация, когда такой поиск не приводит к успеху. В этом случае уменьшается шаг путем введения коэффициента и возобновляется исследующий поиск.

*Алгоритм:*

Шаг 1. Задать начальную точку , число > 0 для остановки алгоритма, начальные величины шагов по координатным направлениям , коэффициент уменьшения шага > 1. Положить , .

Шаг 2. Осуществить исследующий поиск по выбранному координатному направлению:

а) если , т.е. ,

шаг считается удачным. В этом случае следует положить и пе­рейти к шагу 3;

б) если в п. "а" шаг неудачен, то делается шаг в противоположном направлении. Если , т.е. ,

шаг считается удачным. В этом случае следует положить и пе­рейти к шагу 3;

в) если в пп. "а" и "б" шаги неудачны, положить .

Шаг 3. Проверить условия:

а) если , то положить и перейти к шагу 2 (продолжить иссле­дующий поиск по оставшимся направлениям);

б) если , проверить успешность исследующего поиска:

- если , перейти к шагу 4;

- если , перейти к шагу 5.

Шаг 4. Провести поиск по образцу. Положить ,

, = 1, и перейти к шагу 2.

Шаг 5. Проверить условие окончания:

а) если все , то поиск закончить ;

б) для тех , для которых , уменьшить величину шага: . Положить

и перейти к шагу 2.

Листинг с реализацией методов приведен ниже.

Листинг 1

**import math**

import numpy as np

from copy import copy

def Jeeves(f, x0, e):

x = copy(x0)

increment = [ 1, 1 ]

y = x

xtemp = x

a = 2

success = 0

while True:

func = f(y[0], y[1])

if f(y[0] + increment[0], y[1]) < func:

y[0] += increment[0]

success += 1

if f(y[0], y[1] + increment[1]) < func:

y[1] += increment[1]

success += 1

if success == 0:

if f(y[0] - increment[0], y[1]) < func:

y[0] -= increment[0]

success += 1

if f(y[0], y[1] - increment[1]) < func:

y[1] -= increment[1]

success += 1

if success != 0:

while True:

success = 0

xtemp[0] = y[0] + (y[0] - x[0])

xtemp[1] = y[1] + (y[1] - x[1])

if f(xtemp[0], xtemp[1]) < f(y[0], y[1]):

x = y

y = xtemp

else:

break

else:

if math.sqrt(increment[0] \*\* 2 + increment[1] \*\* 2) > e:

increment[0] /= a

increment[1] /= a

else:

break

return x

def simplex(f, x0, e):

x = copy(x0)

xtemp = [ 0, 0 ]

vmirror = [ 0, 0 ]

V = [ [ x[0], x[1] ], [ 0, 0 ], [ 0, 0 ] ]

k = min = max = indexm = indexp = 0

n = 2

delta = 1

y = 0.5

for i in range(1, n + 1):

for j in range(0, n):

if (i - j) == 1:

V[i][j] = x[j] + delta \* (math.sqrt(n + 1) + n - 1) / (n \* math.sqrt(2))

else:

V[i][j] = x[j] + delta \* (math.sqrt(n + 1) - 1) / (n \* math.sqrt(2))

for i in range(0, n + 1):

for j in range(0, n):

x[j] += (1 / (n + 1)) \* V[i][j]

while True:

for i in range(0, n + 1):

func = f(V[i][0], V[i][1])

if i == 0:

max = func

if func>= max:

max = func

indexp = i

for i in range(0, n + 1):

for j in range(0, n):

if indexp != i:

vmirror[j] += (2 / n) \* V[i][j]

if i == n:

vmirror[j] -= V[indexp][j]

if f(vmirror[0], vmirror[1]) <= max:

for j in range(0, n):

V[indexp][j] = vmirror[j]

else:

delta \*= y

for i in range(0, n + 1):

func = f(V[i][0], V[i][1])

if i == 0:

min = func

if func <= min:

min = func

indexm = i

for i in range(0, n + 1):

for j in range(0, n):

V[i][j] = y \* V[i][j] + (1 - y) \* V[indexm][j]

xtemp = copy(x)

x[0] = 0

x[1] = 0

for i in range(0, n + 1):

for j in range(0, n):

x[j] += (1 / (n + 1)) \* V[i][j]

if abs(x[0] - xtemp[0]) <= e and abs(x[1] - xtemp[1]) <= e and abs(f(x[0], x[1]) - f(xtemp[0], xtemp[1])) <= e:

break

else:

k += 1

return x

if \_\_name\_\_ == '\_\_main\_\_':

f = lambda x1,x2: 2\*pow(x1,3) + x1\*pow(x2,2) - 5\*pow(x1,2)+pow(x2,2)

df = lambda x: pow(x[0],3) + pow(x[0],3) - 3\*x[0]\*x[1] 2\*pow(x[0]],3) + x[0]\*pow(x[1],2) - 5\*pow(x[0],2)+pow(x[1],2)

x = [1.5,.7]

e = 10e-4

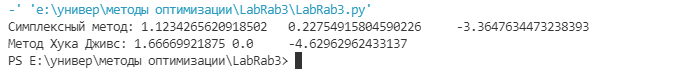
ans = simplex(f, x, e)

print("Симплексный метод: {0}\t{1}\t{2}".format(ans[0], ans[1], f(ans[0], ans[1])))

ans = Jeeves(f, x, e)

print("Метод Хука Дживс: {0}\t{1}\t{2}".format(ans[0], ans[1], f(ans[0],ans[1]) ))

1. **Результат работы**



1. **Выводы**

В результате лабораторной работы были изучены и реализованы на языке python два прямых метода: симплексный метод и метод Хука-Дживса для нахождения минимума функции с двумя переменными.