|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
|  | | |
| Лабораторная работа № 4 | | |
| по дисциплине «Методы оптимизации» | | |
|  | | |
| **Статистические методы поиска** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМИ-72 |
| Бригада: | 2 |
| Студенты: | Антонов С. |
|  | Кайль Д.  Арнольд Э. |
|  |  |
| Преподаватель: | Постовалов С.Н. |
|  |  |
|  | | |
| Новосибирск | | |
| 2020 | | |

1. **Цель работы**

Ознакомиться со статистическими методами поиска при реше­нии задач нелинейного программирования. Изучить методы случайного поиска при определении глобального экстремума функции.

1. **Задание**
2. Разработать программу для решения задачи поиска глобального экстремума с использованием **метода простого случайного поиска и трех алгоритмов глобального поиска**.
3. Исследовать метод простого случайного поиска глобального экстремума при различных  и . Результат представить в таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |

1. Исследовать алгоритмы поиска глобального экстремума. Сравнить результаты поиска по количеству вычислений функции и найденной точке экстремума. Исследование провести при различных значениях числа попыток .
2. **Условие задачи**

Условие задачи:

Найти **максимум** заданной функции:



на области , .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |  | |
| **2** | **2** | | **4** | | **2** | | **6** | | **2** | | **3** | | **-3** | | **-6** | | **2** | | **6** | | **-3** | | **8** | | **6** | | **-8** | | **-8** | | **8** | | **-4** | | **-1** | |

1. **Результат работы**
2. **Простой случайный поиск**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EPS | P\_EPS = EPS\*EPS/400 | P | N | x | y | f(x,y) |
| 1,00E+01 | 2,50E-01 | 0,8 | 5 | -6.456801e+00 | 4.932096e+00 | 6.993042e-01 |
| 1,00E+01 | 2,50E-01 | 0,9 | 8 | -6.456801e+00 | -4.932096e+00 | 6.993042e-01 |
| 1,00E+01 | 2,50E-01 | 0,999 | 24 | -5.666372e+00 | -6.052126e+00 | 1.335475e+00 |
| 1,00E-01 | 2,50E-03 | 0,8 | 64376 | -5.963622e+00 | -8.012024e+00 | 6.075537e+00 |
| 1,00E-01 | 2,50E-03 | 0,9 | 92102 | -6.022828e+00 | -8.019349e+00 | 6.078602e+00 |
| 1,00E-01 | 2,50E-03 | 0,999 | 276306 | -6.022828e+00 | -8.019349e+00 | 6.078602e+00 |
| 1,00E-02 | 1,00E-06 | 0,8 | 6437750 | -5.999023e+00 | -7.997375e+00 | 6.084277e+00 |
| 1,00E-02 | 1,00E-06 | 0,9 | 9210339 | -5.999023e+00 | -7.997375e+00 | 6.084277e+00 |
| 1,00E-02 | 1,00E-06 | 0,999 | 27631017 | -5.999023e+00 | -7.997375e+00 | 6.084277e+00 |

*Алгоритмы глобального поиска*

Для направленного поиска (локального экстремума), используемого в 3х алгоритмах, реализован алгоритм наилучшей пробы с направляющим гиперквадратом. Параметры метода: α=2. Критерии выхода: изменение функции на шаге меньше задаваемой точности ε или число итераций больше максимального.

В качестве ненаправленного поиска используется алгоритм простого случайного поиска.

1. **Алгоритм 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPS | m | N | x | y | f(x,y) |
| 1,00E-01 | 1 | 6 | 7.989290e+00 | -9.793771e-01 | 3.134971e+00 |
| 1,00E-01 | 5 | 15 | 7.989290e+00 | -9.793771e-01 | 3.134971e+00 |
| 1,00E-01 | 15 | 63 | 6.010277e+00 | 7.993675e+00 | 6.083437e+00 |
| 1,00E-01 | 40 | 101 | 6.010277e+00 | 7.993675e+00 | 6.083437e+00 |
| 1,00E-03 | 1 | 10 | -3.010822e+00 | -4.008539e+00 | 2.270952e+00 |
| 1,00E-03 | 5 | 57 | -5.996680e+00 | -7.998538e+00 | 4.144638e+00 |
| 1,00E-03 | 15 | 144 | 5.996522e+00 | 7.998236e+00 | 6.084235e+00 |
| 1,00E-03 | 40 | 326 | 6.000202e+00 | 7.999627e+00 | 6.084292e+00 |
| 1,00E-05 | 1 | 9 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-05 | 5 | 56 | -5.996578e+00 | -7.996930e+00 | 4.144652e+00 |
| 1,00E-05 | 15 | 204 | 5.999762e+00 | 7.999210e+00 | 6.084296e+00 |
| 1,00E-05 | 40 | 401 | 5.999762e+00 | 7.999210e+00 | 6.084296e+00 |
| 1,00E-07 | 1 | 11 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 5 | 180 | 5.999571e+00 | 7.999120e+00 | 6.084296e+00 |
| 1,00E-07 | 15 | 442 | 5.999585e+00 | 7.999076e+00 | 6.084296e+00 |
| 1,00E-07 | 40 | 975 | 5.999597e+00 | 7.999102e+00 | 6.084296e+00 |

1. **Алгоритм 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPS | m | N | x | y | f(x,y) |
| 1,00E-01 | 1 | 4 | -3.172368e+00 | -4.058161e+00 | 2.212964e+00 |
| 1,00E-01 | 5 | 8 | -3.172368e+00 | -4.058161e+00 | 2.212964e+00 |
| 1,00E-01 | 15 | 18 | -3.172368e+00 | -4.058161e+00 | 2.212964e+00 |
| 1,00E-01 | 40 | 43 | -3.172368e+00 | -4.058161e+00 | 2.212964e+00 |
| 1,00E-03 | 1 | 5 | -3.028951e+00 | -4.013508e+00 | 2.269874e+00 |
| 1,00E-03 | 5 | 9 | -3.028951e+00 | -4.013508e+00 | 2.269874e+00 |
| 1,00E-03 | 15 | 19 | -3.028951e+00 | -4.013508e+00 | 2.269874e+00 |
| 1,00E-03 | 40 | 73 | 6.000177e+00 | 8.000805e+00 | 6.084276e+00 |
| 1,00E-05 | 1 | 11 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-05 | 5 | 15 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-05 | 15 | 25 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-05 | 40 | 50 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 1 | 11 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 5 | 15 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 15 | 25 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 40 | 50 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |

1. **Алгоритм 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| EPS | m | N | x | y | f(x,y) |
| 1,00E-01 | 1 | 4 | -3.172368e+00 | -4.058161e+00 | 2.212964e+00 |
| 1,00E-01 | 5 | 36 | 5.977533e+00 | 7.940403e+00 | 6.060814e+00 |
| 1,00E-01 | 15 | 56 | 5.977533e+00 | 7.940403e+00 | 6.060814e+00 |
| 1,00E-01 | 40 | 109 | 5.977533e+00 | 7.940403e+00 | 6.060814e+00 |
| 1,00E-03 | 1 | 7 | -3.028951e+00 | -4.013508e+00 | 2.269874e+00 |
| 1,00E-03 | 5 | 39 | 6.003579e+00 | 7.997924e+00 | 6.084193e+00 |
| 1,00E-03 | 15 | 80 | 6.003579e+00 | 7.997924e+00 | 6.084193e+00 |
| 1,00E-03 | 40 | 173 | 6.003579e+00 | 7.997924e+00 | 6.084193e+00 |
| 1,00E-05 | 1 | 3 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-05 | 5 | 55 | -5.996505e+00 | -7.996560e+00 | 4.144653e+00 |
| 1,00E-05 | 15 | 227 | 5.999540e+00 | 7.999025e+00 | 6.084296e+00 |
| 1,00E-05 | 40 | 357 | 5.999540e+00 | 7.999025e+00 | 6.084296e+00 |
| 1,00E-07 | 1 | 3 | -3.004407e+00 | -4.012366e+00 | 2.271056e+00 |
| 1,00E-07 | 5 | 82 | -5.996467e+00 | -7.996649e+00 | 4.144653e+00 |
| 1,00E-07 | 15 | 145 | -5.996467e+00 | -7.996649e+00 | 4.144653e+00 |
| 1,00E-07 | 40 | 547 | 5.999505e+00 | 7.998997e+00 | 6.084296e+00 |

1. **Вывод**
2. В алгоритме случайного ненаправленного поиска число итераций напрямую зависит от заданной точности **ε** и вероятности **Р**. Точность решения повышается при повышении этих величин. При небольшой точности метод может попасть в локальный максимум, не являющийся наибольшим, но при увеличении ε находит глобальный максимум с заданной точностью.
3. Алгоритм 1 глобального поиска предполагает спуск к локальному экстремуму из каждой новой случайной точки, поэтому увеличение числа **m** (выброшенных случайных точек) увеличивает число вычислений функции. При недостаточном числе случайных точек алгоритм попадает в другой локальный экстремум, причем необходимое число случайных проб растет с ростом заданной точности **ε**. Однако при **ε=1е-6** метод сходится к глобальному максимуму и при **m=1**, так как спуск направленного случайного поиска методом наилучшей пробы с гиперквадратом находит точное решение для любой точки.
4. В алгоритме 2 в каждой случайной точке вычисляется значение функции, но поиск локального максимума осуществляется далеко не из каждой такой точки. Поэтому число вычислений функций в данном алгоритме ниже, чем в предыдущем. Но ввиду высоких требований выбора случайной точки текущей, необходимое для достижения глобального экстремума число генерируемых случайных проб значительно выше, чем в алгоритме 1. Зависимость **m** от **ε** для нахождения глобального экстремума аналогична предыдущему алгоритму.
5. Число вычислений функции в алгоритме 3 больше, чем во втором, но меньше, чем в первом алгоритмах. Функция вычисляется в нескольких точках на сгенерированном случайно направлении. Число вычислений может зависеть от частоты проверяемых точек на таком направлении. Так как точка, которая может стать начальной для поиска локального экстремума, должна удовлетворять строгим условиям, такие точки изначально выбираются хорошо, поэтому для высокого значения точности достаточно малого числа допустимых случайных проб(направлений).
6. **Код программы**

import numpy as np

import pandas as pd

import random

import math

from tqdm import tqdm\_notebook

def function(point) :

global number

number += 1

x, y = point

s = [0] \* 6

s[0] = 2 / (1 + (x - 3) \* \*2 + (y + 4) \* \*2)

s[1] = 3 / (1 + (x + 5) \* \*2 + (y + 6) \* \*2)

s[2] = 8 / (1 + x \* \*2 + (y + 1) \* \*2)

s[3] = 3 / (1 + (x - 3) \* \*2 + (y - 7) \* \*2)

s[4] = 2 / (1 + (x + 4) \* \*2 + y \* \*2)

s[5] = 8 / (1 + (x - 6) \* \*2 + (y - 5) \* \*2)

return sum(s)

# Обычный поиск

# bounds - граница по x, y

def search(bounds, epsilon = list((1e-2, 1e-2)), p = 0.9) :

bound\_x, bound\_y = bounds

eps\_x, eps\_y = epsilon

eps\_p = eps\_x \* eps\_y / (bound\_x[1] - bound\_x[0]) / (bound\_y[1] - bound\_y[0])

n = math.log2(1 - p) / math.log2(1 - eps\_p)

xk = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

fk = function(xk)

# т.к.по началу n - вещественное

n = int(n) + 1

for i in range(n) :

x\_next = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

f\_next = function(x\_next)

if fk < f\_next:

xk, fk = x\_next, f\_next

# n - кол - во итераций

# xk - лучшая точка

# fk - значение f(x, y) в лучшей точке

return n, xk, fk

# Обычный поиск

# x0 - обычный поиск

# bounds - граница по x, y

# m - количество вбросов случайных точек

def search2(x0, bounds, m = 1000) :

bound\_x, bound\_y = bounds

xk, fk = x0, function(x0)

for i in range(m) :

x\_next = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

f\_next = function(x\_next)

if fk < f\_next:

xk, fk = x\_next, f\_next

# xk - лучшая точка

# fk - значение f(x, y) в лучшей точке

return xk, fk

# Метод наилучшей пробы(с гиперквадратом)

# x0 - начальная точка

# side\_k - начальная сторона гиперквадрата(лучше покрыть всю область)

# number\_sq - общее количество гиперквадратов

# m - количество вбросов случайных точек внутрь гиперквадрата

# bounds - граница по x, y

# alpha - параметр для гиперквадрата(по умолчанию - 1)

def best\_probe(x0, bounds, side\_k, number\_sq = 10, m = 1000, alpha = 1) :

# т.к.на каждой итерации / (2 \* alpha)

alpha \*= 2

(bound\_x, bound\_y), xk = bounds, x0

side\_k /= 2

for k in range(number\_sq) :

# границы гиперквадрата

a1, b1 = max(xk[0] - side\_k, bound\_x[0]), min(xk[0] + side\_k, bound\_x[1])

a2, b2 = max(xk[1] - side\_k, bound\_y[0]), min(xk[1] + side\_k, bound\_y[1])

# вернёт либо xk, либо лучше

xk, fk = search2(xk, ((a1, b1), (a2, b2)), m = m)

side\_k /= alpha

# xk - лучшая точка

# fk - значение f(x, y) в лучшей точке

return xk, fk

# Алгоритм 1

# bounds - граница по x, y

# m - количество неудач в поиске лучшей точки

# number\_sq - общее количество гиперквадратов для направленного поиска

# number\_comp - количество пробных вычислений функции для направленного поиска

# alpha - параметр для гиперквадрата

def alg1(bounds, m = 1000, number\_sq = 10, number\_comp = 100, alpha = 1) :

# счётчик до m и количество итераций

j = t = 0

bound\_x, bound\_y = bounds

side\_k = max(bound\_x[1] - bound\_x[0], bound\_y[1] - bound\_y[0])

xk = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

x\_best, f\_best = best\_probe(x0 = xk, bounds = bounds, side\_k = side\_k,

number\_sq = number\_sq, m = number\_comp, alpha = alpha)

while j < m:

xk, fk = best\_probe(x0 = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1])),

bounds = bounds, side\_k = side\_k,

number\_sq = number\_sq, m = number\_comp, alpha = alpha)

if fk > f\_best:

x\_best, f\_best = xk, fk

j = 0

else:

j += 1

t += 1

# t - количество итераций

# x\_best - лучшая точка

# f\_best - значение f(x, y) в x\_best

return t, x\_best, f\_best

# Алгоритм 2

# bounds - граница по x, y

# m - количество неудач в поиске лучшей точки

# number\_sq - общее количество гиперквадратов для направленного поиска

# number\_comp - количество пробных вычислений функции для направленного поиска

# alpha - параметр для гиперквадрата

def alg2(bounds, m = 1000, number\_sq = 10, number\_comp = 100, alpha = 1) :

# счётчик до m и количество итераций

j = t = 0

bound\_x, bound\_y = bounds

side\_k = max(bound\_x[1] - bound\_x[0], bound\_y[1] - bound\_y[0])

xk = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

x\_best, f\_best = best\_probe(x0 = xk, bounds = bounds, side\_k = side\_k,

number\_sq = number\_sq, m = number\_comp, alpha = alpha)

while j < m:

xk = (np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1]))

fk = function(xk)

# значение лучше->запустить поиск с этой точки

if fk > f\_best:

x\_best, f\_best = best\_probe(x0 = xk, bounds = bounds, side\_k = side\_k,

number\_sq = number\_sq, m = number\_comp, alpha = alpha)

j = 0

else:

j += 1

t += 1

# t - количество итераций

# x\_best - лучшая точка

# f\_best - значение f(x, y) в x\_best

return t, x\_best, f\_best

# функция, к которой сводится минимизация исходной задачи

# для нахождения lambda\_k - оптимального для поиска

# на k - ой итерации Алгоритма 3

def function\_min(lambda\_k) :

return -function(xk\_best + lambda\_k \* dx)

# поиск интервала

def find\_interval(x\_beg, delta = 1e-2) :

x, h = [x\_beg, None, None], 0

if function\_min(x[0]) > function\_min(x[0] + delta) :

x[1] = x[0] + delta

h = delta

else:

x[1] = x[0] - delta

h = -delta

h \*= 2

x[2] = x[1] + h

while function\_min(x[1]) > function\_min(x[2]) :

h \*= 2

x[0], x[1] = x[1], x[2]

x[2] = x[1] + h

return (x[2], x[0]) if x[0] > x[2] else (x[0], x[2])

# число Фибоначчи - итерации

def fib\_number(n) :

fib1, fib2, fib\_sum, i = 1, 1, 1, 2

while i < n :

fib\_sum = fib2 + fib1

fib1, fib2 = fib2, fib\_sum

i += 1

return fib\_sum

# метод Фибоначчи

def fibonachi(a, b, eps = 1e-8) :

ak, bk = a, b

# длина отрезка < нужной точности

if bk - ak < eps:

return (ak + bk) / 2

# определение n

d = (bk - ak) / eps

n = 1

f = fib\_number(n)

list\_Fib = [None]

while f < d:

n += 1

list\_Fib.append(f)

f = fib\_number(n)

list\_Fib.append(f)

n -= 2

yk = ak + list\_Fib[n] / list\_Fib[n + 2] \* (bk - ak)

zk = ak + list\_Fib[n + 1] / list\_Fib[n + 2] \* (bk - ak)

f1, f2 = function\_min(yk), function\_min(zk)

k = 1

while k < n:

k += 1

if f1 <= f2 :

bk = zk

zk, f2 = yk, f1

yk = ak + list\_Fib[n - k + 1] / list\_Fib[n - k + 3] \* (bk - ak)

f1 = function\_min(yk)

else:

ak = yk

yk, f1 = zk, f2

zk = ak + list\_Fib[n - k + 2] / list\_Fib[n - k + 3] \* (bk - ak)

f2 = function\_min(zk)

# yk == zk == (ak + bk) / 2

return yk

def condition(bounds, interval, xk\_best) :

dx\_x, dx\_y = dx

bound\_x, bound\_y = bounds

a, b = interval

# следующая точка не выходит за интервал < ->bound[0] < (xk + 1 = xk + lambda\_k \* dx) < bound[1]

# выразив lambda\_k : (bound[0] - xk) / dx < lambda\_k < (bound[1] - xk) / dx

# направление в сторону увеличения координаты x

if dx\_x > 0 :

p = (bound\_x[1] - xk\_best[0]) / dx\_x

else:

p = (bound\_x[0] - xk\_best[0]) / dx\_x

if dx\_y > 0:

p2 = (bound\_y[1] - xk\_best[1]) / dx\_y

else:

p2 = (bound\_y[0] - xk\_best[1]) / dx\_y

return 0, min(p, p2)

# Алгоритм 3

# x0 - начальная точка

# bounds - граница по x, y

# m - количество неудач в поиске лучшей точки

# number\_sq - общее количество гиперквадратов для направленного поиска

# number\_comp - количество пробных вычислений функции для направленного поиска

# alpha - параметр для гиперквадрата

def alg3(x0, bounds, m = 1000, number\_sq = 10, number\_comp = 100, alpha = 1) :

global dx, xk\_best

j, t = 0, 0

bound\_x, bound\_y = bounds

side\_k = max(bound\_x[1] - bound\_x[0], bound\_y[1] - bound\_y[0])

xk, fk = np.array(x0), function(x0)

# будем считать лучшими

x\_max, f\_max = xk, fk

while j < m:

# локальный экстремум

xk\_best, fk\_best = best\_probe(x0 = xk, bounds = bounds, side\_k = side\_k,

number\_sq = number\_sq, m = number\_comp, alpha = alpha)

xk\_best = np.array(xk\_best)

if fk\_best > f\_max:

x\_max, f\_max = xk\_best, fk\_best

# вектор, в сторону которого будет следующий шаг

dx = xk\_best - xk

# вектор dx - нулевой

# пришли в лучшую точку->случайно выберем направление

if np.any(dx) == False:

dx = np.array((np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1])))

j = 0

# значение меньше->случайно выберем направление

else:

dx = np.array((np.random.uniform(low = bound\_x[0], high = bound\_x[1]),

np.random.uniform(low = bound\_y[0], high = bound\_y[1])))

j += 1

# оптимальные a, b

(a, b) = find\_interval(0)

# уточнить границу(чтобы не выйти за область)

a, b = condition(bounds, (a, b), xk\_best)

# лучшее значение

lambda\_k = fibonachi(0, b)

# следующее

xk = xk\_best + lambda\_k \* dx

t += 1

return t, x\_max, f\_max