|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 4 | | |
| по дисциплине «Методы оптимизации» | | |
| **Статистические методы поиска** | | |
|  | | |
|  | Бригада 2 | Буданцев Дмитрий |
| Группа ПМ-13 | голубь Андрей |
| Вариант 2 |  |
|  |  |
|  |  |
|  |  |
| Преподаватель | Филиппова Елена Владимировна |
|  | Лемешко Борис Юрьевич |
| Новосибирск,2024 | | |

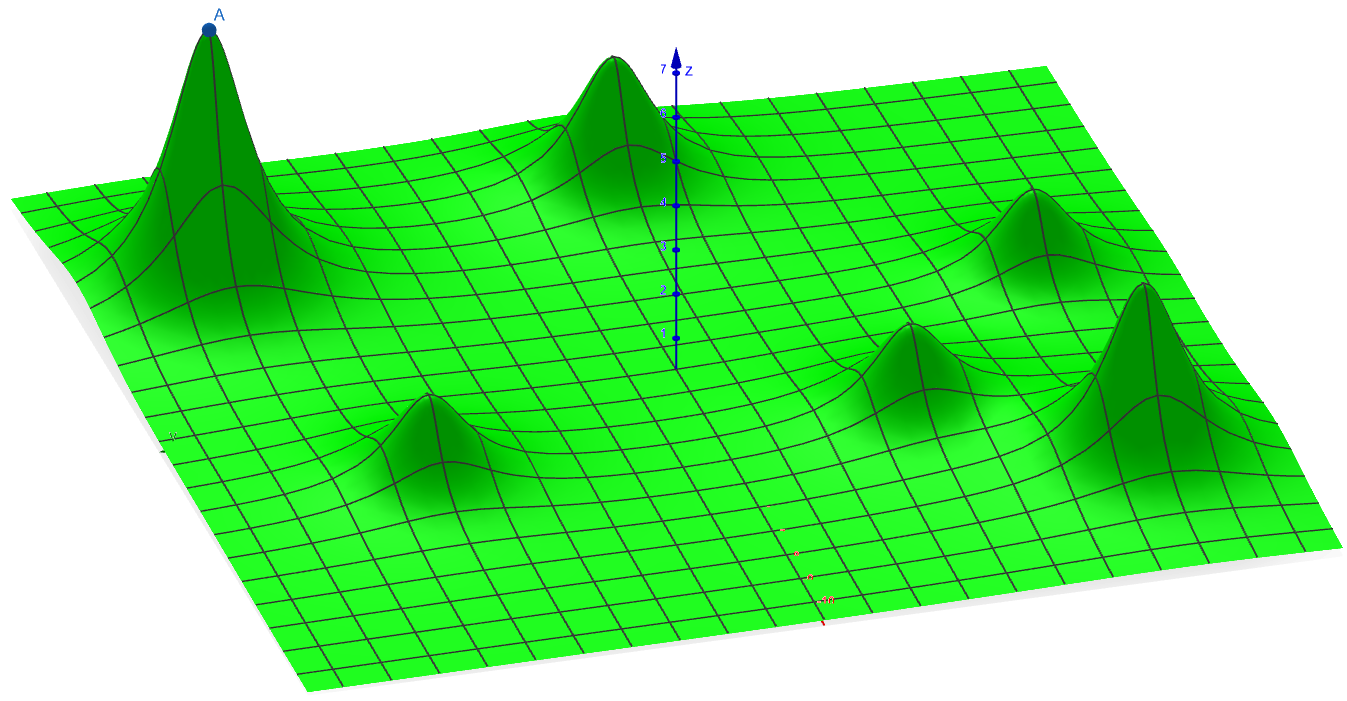
1. **Цель работы**:

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

1. **Задания:**
   1. Разработать программу для решения задачи поиска глобального экстремума с использованием **метода простого случайного поиска и трех алгоритмов глобального поиска.**
   2. Исследовать метод простого случайного поиска глобального экстремума при различных  и . Результат представить в таблице
   3. Исследовать алгоритмы поиска глобального экстремума. Сравнить результаты поиска по количеству вычислений функции и найденной точке экстремума. Исследование провести при различных значениях числа попыток .
2. П.1.3 повторить при пяти разных начальных значениях ГСЧ. Сделать выводы об устойчивости различных алгоритмов



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 | 4 | 2 | 6 | 2 | 3 | -3 | -6 | 2 | 6 | -3 | 8 | 6 | -8 | -8 | 8 | -4 | -1 |



**Алгоритм направленного случайного поиска:** Метод статического градиента

**Алгоритм ненаправленного случайного поиска:** Простой случайный поиск

1. **Проведённые исследования**
   1. Исследование метода ПСП при разных параметрах  и 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0.1 | 0.8 | 16 | (-3.119, -5.139) | 1.21117 |
| 0.1 | 0.9 | 22 | (5.225, 6.744) | 1.99088 |
| 0.1 | 0.95 | 29 | (-5.195, -6.396) | 1.20197 |
| 0.1 | 0.99 | 44 | (1.663, -8.599) | 1.5291 |
| 0.1 | 0.999 | 66 | (7.361, 8.566) | 1.96283 |
| 0.025 | 0.8 | 64 | (6.684, 8.917) | 2.67175 |
| 0.025 | 0.9 | 91 | (5.704, 8.022) | 5.60053 |
| 0.025 | 0.95 | 119 | (6.845, 7.545) | 3.20837 |
| 0.025 | 0.99 | 182 | (5.885, 8.262) | 5.62771 |
| 0.025 | 0.999 | 273 | (5.838, 7.950) | 5.91872 |
| 0.01 | 0.8 | 161 | (6.367, 7.491) | 4.39555 |
| 0.01 | 0.9 | 230 | (6.482, 8.157) | 4.85314 |
| 0.01 | 0.95 | 299 | (8.075, -1.023) | 3.11691 |
| 0.01 | 0.99 | 459 | (6.249, 7.907) | 5.68805 |
| 0.01 | 0.999 | 688 | (5.977, 7.749) | 5.72863 |
| 0.005 | 0.8 | 322 | (6.471, 7.775) | 4.79857 |
| 0.005 | 0.9 | 460 | (6.621, 7.851) | 4.34343 |
| 0.005 | 0.95 | 598 | (-6.254, -8.129) | 3.83457 |
| 0.005 | 0.99 | 919 | (5.643, 8.274) | 5.07057 |
| 0.005 | 0.999 | 1379 | (5.622, 8.327) | 4.88182 |
| 0.001 | 0.8 | 1609 | (6.021, 7.955) | 6.06977 |
| 0.001 | 0.9 | 2302 | (6.062, 8.064) | 6.03633 |
| 0.001 | 0.95 | 2995 | (5.991, 7.977) | 6.08097 |
| 0.001 | 0.99 | 4603 | (6.220, 7.883) | 5.73375 |
| 0.001 | 0.999 | 6905 | (5.972, 8.041) | 6.06904 |

Вывод: Исходя из таблицы, данный алгоритм введёт себя так, как задуман. С увеличением параметров алгоритма становится более стабильным к нахождению глобального минимума. При более скромных параметрах имеет маленькую вероятность нахождения глобального минимума. Серьёзным минусом алгоритма является его не стабильность.

* 1. Исследование методов глобального поиска
     1. Алгоритм №1

|  |  |  |  |
| --- | --- | --- | --- |
|  | Count Calc Func |  |  |
| 4 | 168286 | (5.799, 7.957) | 5.84243 |
| 8 | 147242 | (5.844, 8.185) | 5.75162 |
| 10 | 84352 | (7.903, -0.953) | 3.10352 |
| 20 | 189610 | (5.986, 8.193) | 5.86491 |
| 30 | 526309 | (5.995, 7.990) | 6.08364 |
| 40 | 526328 | (5.998, 7.997) | 6.08426 |

* + 1. Алгоритм №2

|  |  |  |  |
| --- | --- | --- | --- |
|  | Count Calc Func |  |  |
| 4 | 191598 | (5.741, 8.278) | 5.32661 |
| 8 | 341174 | (6.119, 8.163) | 5.84725 |
| 10 | 298530 | (5.956, 7.876) | 5.98322 |
| 20 | 469148 | (6.000, 7.939) | 6.0628 |
| 30 | 1087444 | (5.981, 7.959) | 6.07271 |
| 40 | 1556206 | (5.979, 7.956) | 6.07058 |

* + 1. Алгоритм №3

|  |  |  |  |
| --- | --- | --- | --- |
|  | Count Calc Func |  |  |
| 4 | 21216 | (-2.939, 6.202) | 2.04998 |
| 8 | 42211 | (1.982, -7.793) | 2.09624 |
| 10 | 63407 | (-5.908, -7.971) | 4.11088 |
| 20 | 147787 | (-5.806, -8.137) | 3.93322 |
| 30 | 128115 | (7.763, -1.063) | 2.9686 |
| 40 | 21762 | (5.943, 7.943) | 6.04697 |

Вывод: Каждый из этих алгоритмов имеет свои плюсы и минусы. Алгоритм №1 имеет риск не дойти до глобального минимума из-за того, что выбираемые точки зависят от ГСЧ и вероятность не нулевая того, что ГСЧ будет генерировать точки возле одного локального минимума. Но не смотря на это Алгоритм №1 имеет среднее кол-во подсчёта функции. Алгоритм №2 в отличие от Алгоритма №1 является более стабильным к нахождению глобального минимума из-за того, что точки выбираются благодаря ПСП, в котором можно выбирать допустимые для задачи  и . Но в свою очередь имеет большое кол-во вычислений функций, которое можно снизить путём редактирования параметров ПСП. Алгоритм №3 является самым часто не доходящим до глобального минимума. В отличие от остальных алгоритмов не использует случайный выбор точки, а использует случайное направление. Причиной из-за которого алгоритм часто не доходит до глобального минимума, является сильная зависимость от настраемых параметров. В нашей реализации алгоритма, он сильно зависит от величины шага, от задаваемой точности, от числа . Если параметры подобраны оптимально, то алгоритм сходится стабильно. Весомым плюсом является то, что этот алгоритм имеет меньше всего вычислений функции.

* 1. Исследование алгоритмов при зафиксированном начальном значении ГСЧ

Начальное значение ГСЧ: 12

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм 1 | | | | Алгоритм 2 | | | |
|  | Count Calc Func |  |  |  | Count Calc Func |  |  |
| 4 | 21242 | (5.983, 7.966) | 6.0763 | 4 | 149276 | (5.916, 7.912) | 5.99764 |
| 8 | 105330 | (5.983, 7.966) | 6.0763 | 8 | 234564 | (5.916, 7.912) | 5.99764 |
| 10 | 105374 | (5.983, 7.966) | 6.0763 | 10 | 277208 | (5.916, 7.912) | 5.99764 |
| 20 | 231594 | (5.983, 7.966) | 6.0763 | 20 | 1215376 | (5.985, 7.968) | 6.07745 |
| 30 | 273982 | (5.983, 7.966) | 6.0763 | 30 | 1428596 | (5.985, 7.968) | 6.07745 |
| 40 | 400202 | (5.983, 7.966) | 6.0763 | 40 | 1641816 | (5.985, 7.968) | 6.07745 |

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм 3 | | | |
|  | Count Calc Func |  |  |
| 4 | 505730 | (-2.948, 5.971) | 2.12958 |
| 8 | 549043 | (-2.952, 5.973) | 2.13042 |
| 10 | 828443 | (-2.968, 5.983) | 2.1335 |
| 20 | 1494351 | (-2.951, 5.973) | 2.13041 |
| 30 | 181587 | (5.991, 7.982) | 6.08206 |
| 40 | 312538 | (5.992, 7.984) | 6.08249 |

Начальное значение ГСЧ: 24

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм 1 | | | | Алгоритм 2 | | | |
|  | Count Calc Func |  |  |  | Count Calc Func |  |  |
| 4 | 105242 | (6.174, 7.915) | 5.8679 | 4 | 319852 | (5.920, 7.904) | 5.99399 |
| 8 | 147330 | (6.174, 7.915) | 5.8679 | 8 | 405140 | (5.920, 7.904) | 5.99399 |
| 10 | 189374 | (6.174, 7.915) | 5.8679 | 10 | 447784 | (5.920, 7.904) | 5.99399 |
| 20 | 273720 | (6.174, 7.915) | 5.8679 | 20 | 661004 | (5.920, 7.904) | 5.99399 |
| 30 | 358108 | (6.174, 7.915) | 5.8679 | 30 | 1641816 | (5.973, 7.920) | 6.04321 |
| 40 | 653304 | (5.850, 7.946) | 5.9371 | 40 | 1855036 | (5.973, 7.920) | 6.04321 |

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм 3 | | | |
|  | Count Calc Func |  |  |
| 4 | 105651 | (5.984, 7.968) | 6.07693 |
| 8 | 169338 | (5.981, 7.952) | 6.06887 |
| 10 | 190702 | (5.989, 7.978) | 6.08089 |
| 20 | 340389 | (5.979, 7.953) | 6.06904 |
| 30 | 322110 | (5.978, 7.958) | 6.07146 |
| 40 | 496400 | (5.989, 7.978) | 6.0811 |

Начальное значение ГСЧ: 909

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм 1 | | | | Алгоритм 2 | | | |
|  | Count Calc Func |  |  |  | Count Calc Func |  |  |
| 4 | 63326 | (-6.354, -7.975) | 3.68702 | 4 | 170598 | (5.979, 7.959) | 6.07208 |
| 8 | 84414 | (-6.354, -7.975) | 3.68702 | 8 | 255886 | (5.979, 7.959) | 6.07208 |
| 10 | 84458 | (-6.354, -7.975) | 3.68702 | 10 | 298530 | (5.979, 7.959) | 6.07208 |
| 20 | 336942 | (5.923, 7.909) | 6.00176 | 20 | 511750 | (5.979, 7.959) | 6.07208 |
| 30 | 421162 | (5.923, 7.909) | 6.00176 | 30 | 724970 | (5.979, 7.959) | 6.07208 |
| 40 | 526403 | (5.923, 7.909) | 6.00176 | 40 | 938190 | (5.979, 7.959) | 6.07208 |

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм 3 | | | |
|  | Count Calc Func |  |  |
| 4 | 64186 | (-2.997, -4.051) | 2.26789 |
| 8 | 633498 | (-5.956, -7.973) | 4.13597 |
| 10 | 106820 | (-2.995, -4.055) | 2.26724 |
| 20 | 255924 | (-6.005, -8.006) | 4.14403 |
| 30 | 405775 | (-5.968, -7.978) | 4.14 |
| 40 | 1130964 | (-5.953, -7.971) | 4.1344 |

Начальное значение ГСЧ: 69

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм 1 | | | | Алгоритм 2 | | | |
|  | Count Calc Func |  |  |  | Count Calc Func |  |  |
| 4 | 105220 | (6.186, 8.337) | 5.304 | 4 | 191598 | (5.719, 7.945) | 5.63199 |
| 8 | 147308 | (6.186, 8.337) | 5.304 | 8 | 276886 | (5.719, 7.945) | 5.63199 |
| 10 | 168352 | (6.186, 8.337) | 5.304 | 10 | 319530 | (5.719, 7.945) | 5.63199 |
| 20 | 273656 | (6.186, 8.337) | 5.304 | 20 | 809936 | (5.993, 7.860) | 5.97039 |
| 30 | 589382 | (5.946, 7.921) | 6.03135 | 30 | 1620172 | (5.982, 7.964) | 6.07511 |
| 40 | 652602 | (5.946, 7.921) | 6.03135 | 40 | 1833392 | (5.982, 7.964) | 6.07511 |

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм 3 | | | |
|  | Count Calc Func |  |  |
| 4 | 42460 | (-2.954, 5.975) | 2.13109 |
| 8 | 43033 | (-2.954, 5.975) | 2.13109 |
| 10 | 85704 | (-2.967, 5.983) | 2.13345 |
| 20 | 130427 | (-2.952, 5.973) | 2.13042 |
| 30 | 489537 | (-2.952, 5.974) | 2.13056 |
| 40 | 498571 | (-2.951, 5.973) | 2.13041 |

Начальное значение ГСЧ: 599

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Алгоритм 1 | | | | Алгоритм 2 | | | |
|  | Count Calc Func |  |  |  | Count Calc Func |  |  |
| 4 | 21176 | (6.009, 7.835) | 5.92662 | 4 | 170598 | (5.865, 7.811) | 5.77954 |
| 8 | 42264 | (6.009, 7.835) | 5.92662 | 8 | 255886 | (5.865, 7.811) | 5.77954 |
| 10 | 84308 | (6.009, 7.835) | 5.92662 | 10 | 298530 | (5.865, 7.811) | 5.77954 |
| 20 | 357858 | (5.963, 7.934) | 6.05081 | 20 | 1130088 | (5.982, 7.942) | 6.06271 |
| 30 | 442078 | (5.963, 7.934) | 6.05081 | 30 | 1961646 | (5.981, 7.961) | 6.07334 |
| 40 | 547298 | (5.963, 7.934) | 6.05081 | 40 | 2174866 | (5.981, 7.961) | 6.07334 |

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм 3 | | | |
|  | Count Calc Func |  |  |
| 4 | 568987 | (7.958, -0.976) | 3.12994 |
| 8 | 653987 | (7.958, -0.976) | 3.12999 |
| 10 | 717871 | (7.969, -0.983) | 3.13309 |
| 20 | 257083 | (5.990, 7.979) | 6.08131 |
| 30 | 303991 | (5.981, 7.961) | 6.0738 |
| 40 | 174650 | (5.993, 7.985) | 6.0828 |

Вывод: Из таблиц видно, что самым устойчивым алгоритмом, из исследуемых, является Алгоритм №2. При каждом значении начального значения ГСЧ он показал близкий результат. Было сказано ранее, что Алгоритм №1 зависит от значений ГСЧ, но в данных тестах результаты, как видно, тоже довольно близкие к истинному. Алгоритм №3, как и было сказано ранее, сильно зависит от параметров метода. При данном исследовании они не подбирались из-за, чего в некоторых тестах он не сошёлся. Но алгоритм, по-прежнему, требует меньше вычислений функций, чем остальные.

1. Программа

import pandas as pd

import numpy as np

from math import log, ceil, sin, cos, radians, sqrt

Name\_OutFile = "OutputFile"

var\_c = np.array([2, 4, 2, 6, 2, 3], dtype=np.int8)

var\_a = np.array([-3, -6, 2, 6, -3, 8], dtype=np.int8)

var\_b = np.array([6, -8, -8, 8, -4, -1], dtype=np.int8)

def func(x):

    if len(x) != 2:

        return 0

    subfunc = lambda x, y, a, b, c: c/(1 + (x - a)\*\*2 + (y - b)\*\*2)

    res = 0.0

    for i in range(6):

        res += subfunc(x[0], x[1], var\_a[i], var\_b[i], var\_c[i])

    return -res

def find\_N(P, Peps):

        if P > 1 or Peps > 1:

            return 0

        return ceil(log(1-P)/log(1-Peps))

def out\_exel(df, name : str):

    if Name\_OutFile != "":

        with pd.ExcelWriter(Name\_OutFile + ".xlsx") as writer:

            df.to\_excel(writer, sheet\_name=name, float\_format="%.8f", index=False)

def simple\_random\_search(p : float, pe : float, xmin = None):

    N = find\_N(p, pe)

    if N == 0:

        return 0

    if xmin is None:

        xmin = np.random.uniform(-10, 10, 2)

    ymin = func(xmin)

    for i in range(N):

        xi = np.random.uniform(-10, 10, 2)

        yi = func(xi)

        if (yi < ymin):

            xmin, ymin = xi, yi

    return xmin, N + 1

def static\_grad\_medhod(xk, m : int, h : float = 0.5, eps : float = 0.1, maxiter : int = 1000):

    if m < 2:

        return 0

    gk = np.array([[0.0, 1.0]])

    for i in range(1, m):

        psi = radians(360\*i/m)

        gk = np.concatenate((gk, np.array([[sin(psi), cos(psi)]])), axis=0)

    def calc\_dgrad(xi):

        dgrad=np.array([0.0, 0.0])

        fxi = func(xi)

        for i in range(m):

            dgrad += gk[i] \* (func(xk + eps\*gk[i]) - fxi)

        return dgrad

    def norm(vector):

        calc = 0

        for i in vector:

            calc += i\*\*2

        return sqrt(calc)

    dgrad, k = calc\_dgrad(xk), 0

    norm\_dgrad = norm(dgrad)

    count\_calc\_func = m + 1

    while norm\_dgrad > eps and k < maxiter:

        xkp1 = xk - h\*dgrad/norm\_dgrad

        if xkp1[0] > 10:

            xkp1[0] = 10

        if xkp1[0] < -10:

            xkp1[0] = -10

        if xkp1[1] > 10:

            xkp1[1] = 10

        if xkp1[1] < -10:

            xkp1[1] = -10

        xk = xkp1

        if (xkp1[0] == 10 or xkp1[0] == -10) and (xkp1[1] == 10 or xkp1[1] == -10):

            break

        dgrad = calc\_dgrad(xk)

        count\_calc\_func += m + 1

        norm\_dgrad = norm(dgrad)

        k += 1

    return xk, count\_calc\_func

def global\_search\_1(eps : float = 0.1, l : int = 3):

    xmin = np.random.uniform(-10, 10, 2)

    xmin, count\_calc\_func = static\_grad\_medhod(xmin, 20, eps=eps)

    flag = 0

    fmin = func(xmin)

    count\_calc\_func += 1

    fk = fmin + 1

    while flag < l:

        xk = np.random.uniform(-10, 10, 2)

        xk, temp\_calc\_func = static\_grad\_medhod(xk, 20, eps=eps)

        fk = func(xk)

        count\_calc\_func += 1 + temp\_calc\_func

        if fk < fmin:

            fmin, xmin = fk, xk

            flag = 0

        else:

            flag += 1

    return xmin, count\_calc\_func

def global\_search\_2(eps : float = 0.1, l : int = 3):

    xmin, count\_calc\_func = static\_grad\_medhod(np.random.uniform(-10, 10, 2), 20, eps=eps)

    fmin = func(xmin)

    count\_calc\_func += 1

    flag = 0

    xk = xmin

    while flag < l:

        xk, temp\_calc\_func = simple\_random\_search(p=0.95, pe=0.01, xmin=xk)

        count\_calc\_func += temp\_calc\_func

        xk, temp\_calc\_func = static\_grad\_medhod(xk, 20, eps=eps)

        fk = func(xk)

        count\_calc\_func += temp\_calc\_func + 1

        if fk < fmin:

            fmin, xmin = fk, xk

            flag = 0

        else:

            flag += 1

    return xmin, count\_calc\_func

def global\_search\_3(x0 = None, eps : float = 0.1, m : int = 10, h : float = 0.1):

    if x0 is None:

        x0 = np.random.uniform(-10, 10, 2)

    elif len(x0) != 2 or m < 2:

        return 0

    gk = np.array([[0.0, 1.0]])

    for i in range(1, m):

        psi = radians(360\*i/m)

        gk = np.concatenate((gk, np.array([[sin(psi), cos(psi)]])), axis=0)

    find\_point = 0

    xmin, count\_calc\_func = static\_grad\_medhod(x0, 20, eps=eps)

    fmin = func(xmin)

    count\_calc\_func += 1

    while find\_point < m:

        for i in range(m):

            xk, xkp1 = xmin, xmin + h \* gk[i]

            fk, fkp1 = fmin, func(xkp1)

            count\_calc\_func += 1

            while xkp1[0] <= 10.0 and xkp1[0] >= -10.0 and xkp1[1] >= -10.0 and xkp1[1] <= 10.0 and fk < fkp1:

                xk = xkp1

                xkp1 += h \* gk[i]

                fk, fkp1 = fkp1, func(xk)

                count\_calc\_func += 1

            if xkp1[0] <= 10.0 and xkp1[0] >= -10.0 and xkp1[1] >= -10.0 and xkp1[1] <= 10.0:

                xk, temp\_calc\_func = static\_grad\_medhod(xkp1, 20, eps=eps)

                fk = func(xk)

                count\_calc\_func += 1 + temp\_calc\_func

                if fk < fmin:

                    find\_point = 0

                    xmin, fmin = xk, fk

                    break

                else:

                    find\_point += 1

            else:

                find\_point += 1

    return xmin, count\_calc\_func

def research\_PSP():

    df = pd.DataFrame(columns=np.array(["Peps", "P","N", "(x, y)", "f(x, y)"]))

    P = [0.8, 0.9, 0.95, 0.99, 0.999]

    Pe = [0.1, 0.025, 0.01, 0.005, 0.001]

    for pe in Pe:

        for p in P:

            xmin = simple\_random\_search(p=p, pe=pe)

            df.loc[len(df) - 1] = np.array([f"{pe}", f"{p}", f"{find\_N(p, pe)}", "({0:.3f}, {1:.3f})".format(xmin[0], xmin[1]), "{0:.5f}".format(-func(xmin))])

    out\_exel(df, "PSP")

def research\_global\_search\_1(M, seed=None):

    df = pd.DataFrame(columns=np.array(["m", "Count Calc Func", "(x, y)", "f(x, y)"]))

    for m in M:

        if not seed is None:

            np.random.seed(seed=seed)

        xmin, count\_calc\_func = global\_search\_1(eps=1e-3, l=m)

        df.loc[len(df) - 1] = np.array([str(m), str(count\_calc\_func), "({0:.3f}, {1:.3f})".format(\*xmin), "{0:.5f}".format(-func(xmin))])

    return df

def research\_global\_search\_2(M, seed=None):

    df = pd.DataFrame(columns=np.array(["m", "Count Calc Func", "(x, y)", "f(x, y)"]))

    for m in M:

        if not seed is None:

            np.random.seed(seed=seed)

        xmin, count\_calc\_func = global\_search\_2(eps=1e-3, l=m)

        df.loc[len(df) - 1] = np.array([str(m), str(count\_calc\_func), "({0:.3f}, {1:.3f})".format(\*xmin), "{0:.5f}".format(-func(xmin))])

    return df

def research\_global\_search\_3(M,seed=None):

    df = pd.DataFrame(columns=np.array(["m", "Count Calc Func", "(x, y)", "f(x, y)"]))

    for m in M:

        if not seed is None:

            np.random.seed(seed=seed)

        xmin, count\_calc\_func = global\_search\_3(eps=1e-6, m=m)

        df.loc[len(df) - 1] = np.array([str(m), str(count\_calc\_func), "({0:.3f}, {1:.3f})".format(\*xmin), "{0:.5f}".format(-func(xmin))])

    return df

temp\_M = np.array([4, 8, 10, 20, 30, 40], dtype=np.int8)

df1 = research\_global\_search\_1(temp\_M, seed=599)

df2 = research\_global\_search\_2(temp\_M, seed=599)

df3 = research\_global\_search\_3(temp\_M, seed=599)

if Name\_OutFile != "":

        with pd.ExcelWriter(Name\_OutFile + ".xlsx") as writer:

            df1.to\_excel(writer, sheet\_name="Algorithm 1", float\_format="%.8f", index=False)

            df2.to\_excel(writer, sheet\_name="Algorithm 2", float\_format="%.8f", index=False)

            df3.to\_excel(writer, sheet\_name="Algorithm 3", float\_format="%.8f", index=False)