|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра теоретической и прикладной информатики | | | |
|  | | | |
| Лабораторная работа № 4 | | | |
| по дисциплине «Методы оптимизации» | | | |
| **Статические методы поиска** | | | |
|  | | | |
|  | Бригада 8 | ПМ-81 Параев П.а. |
|  | Пм-81 Редут А.ю. |
|  |  |
|
|
|
| Преподаватель | Чимитова Е.В. |
|  | | | |
| Новосибирск | | | |

1. **Цель работы**

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

1. **Задание**

Разработать программу для решения задачи поиска глобального экстремума с использованием метода простого случайного поиска и трех алгоритмов глобального поиска.

Исследовать метод простого случайного поиска глобального экстремума при различных  и .

Исследовать алгоритмы поиска глобального экстремума. Сравнить результаты поиска по количеству вычислений функции и найденной точке экстремума. Исследование провести при различных значениях числа попыток . Повторить при 5 разных начальных значениях ГСЧ. Сделать выводы об устойчивости различных алгоритмов.

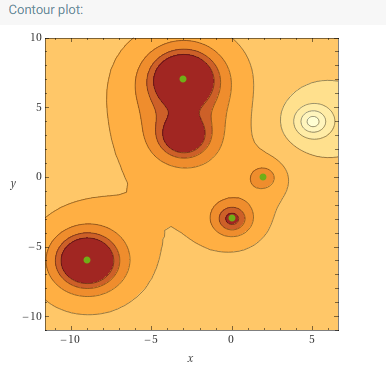
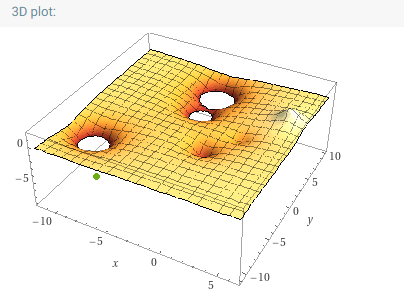
*Вариант 2:*

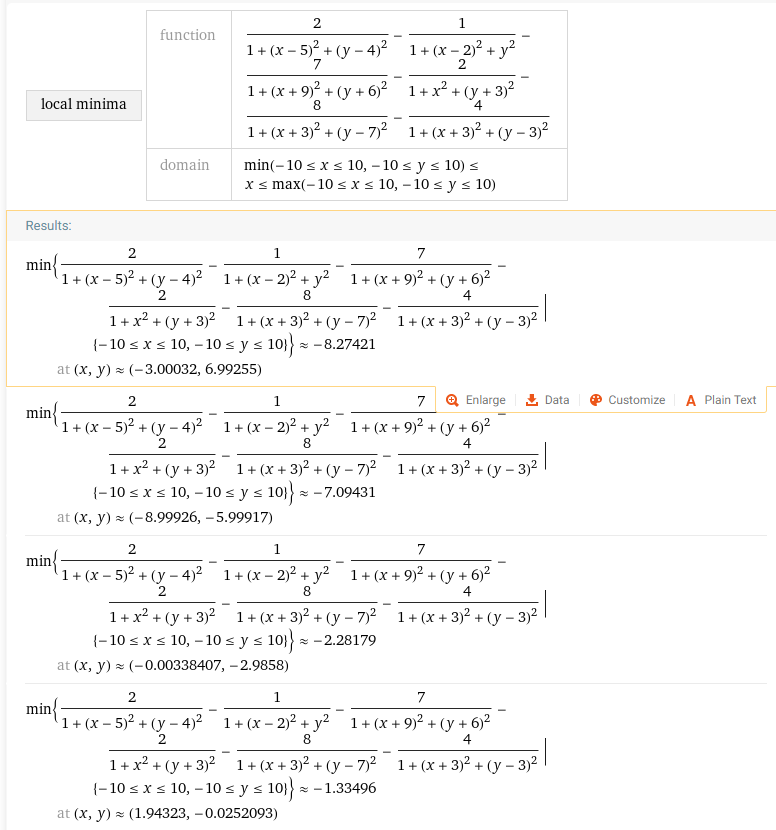
Найти максимум заданной функции:



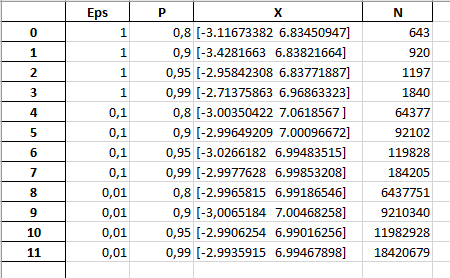
на области , .

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№ варианта** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **8** | 2 | 1 | 7 | 2 | 8 | 4 | 5 | 2 | -9 | 0 | -3 | -3 | 4 | 0 | -6 | -3 | 7 | 3 |



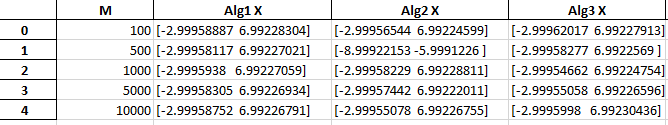


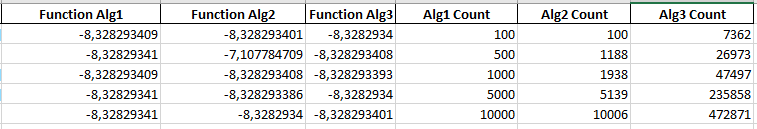
1. **Исследование метода простого случайного поиска глобального экстремума при различных ξ и Р.**



При увеличении Р - вероятности того, что мы найдем решение за N испытаний, и при увеличении требуемой точности решения ξ увеличивается число итераций N и, соответственно, число вычислений функции.

1. **Исследование алгоритмов поиска глобального экстремума**



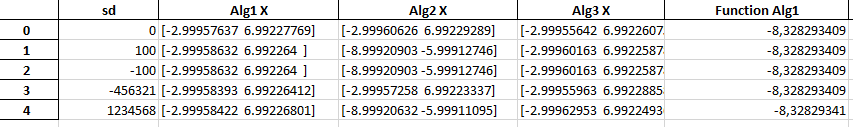


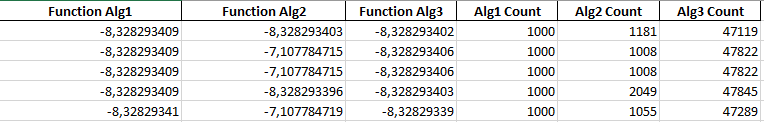
При маленьком значении m методы могут сходиться в локальный минимум.

В общем случае можно выбрать значение M 1000.

Также видно, что первый алгоритм эффективнее остальных, т.к. он имеет меньшее число вычислений функции.

1. **Исследование алгоритмов поиска глобального экстремума при 5 различных начальных значениях ГСЧ**





Наблюдаем, что второй алгоритм совсем не устойчив к выбору ГСЧ.

Первый и третий устойчивы.

1. **Программа**

def simple\_search(function**,** P**,** eps):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 V = (V\_x[1] - V\_x[0]) \* (V\_y[1] - V\_y[0])  
  
 V\_eps = eps \* eps  
 P\_eps = V\_eps / V  
 N = round(np.log(1 - P) / np.log(1 - P\_eps))  
 min\_x = [0**,** 0]  
 global\_min = float('inf')  
  
 iter\_counter = 0  
 while iter\_counter < N:  
 first\_x = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 second\_x = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
  
 if function(first\_x) < function(second\_x) and function(first\_x) < global\_min:  
 min\_x = first\_x  
 global\_min = function(min\_x)  
 elif function(first\_x) > function(second\_x) and function(second\_x) < global\_min:  
 min\_x = second\_x  
 global\_min = function(min\_x)  
 iter\_counter += 1  
 return min\_x**,** N  
  
  
def StohatisticGradientMethod(function**,** eps**,** g**,** m**,** maxiter=10000):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 iter\_counter: int = 0  
 alpha = 1  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
  
 while iter\_counter < maxiter:  
 delta\_f = calculate\_vector\_sum(function**,** xk**,** m**,** g**,** function(xk)**,** V\_x**,** V\_y)  
  
 q = lambda alpha: function(xk - alpha \* delta\_f)  
 coeff = golden(q)  
 #golden\_ratio(q, -100, 100, eps)  
 xNext = xk + coeff \* delta\_f / ln.norm(delta\_f)  
  
 if ln.norm(delta\_f) < eps:  
 return xNext  
  
 xk = xNext  
 iter\_counter += 1  
  
  
  
def pair\_method(function**,**eps**,** g**,** maxiter=100000):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 alpha = 10  
 iter\_counter: int = 0  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
  
 while iter\_counter < maxiter:  
 first\_x = calculate\_x(xk**,** V\_x**,** V\_y)  
  
 first\_func = function(xk + g \* first\_x)  
 second\_func = function(xk - g \* first\_x)  
  
 if first\_func < second\_func:  
 xk = xk + alpha \* first\_x \* first\_func  
 elif second\_func < first\_func:  
 xk = xk + alpha \* first\_x \* second\_func  
  
 iter\_counter += 1  
  
 return xk  
  
  
def Alg1(function**,** M**,** sd):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 random.seed(sd)  
 iter\_counter: int = 0  
 func\_counter = 0  
 m = 0  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 current\_value = 0  
 fmin = float('inf')  
 xmin = 0  
  
 while m < M:  
 iter\_counter += 1  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
  
 xNext = minimize(function**,** xk**,** method='Nelder-Mead').x  
 current\_func = function(xNext)  
 func\_counter += 1  
  
 if current\_func < fmin:  
 fmin = current\_func  
 xmin = xNext  
 m += 1  
  
 return xmin**,** fmin**,** func\_counter  
  
  
def Alg2(function**,** M**,** sd):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 random.seed(sd)  
 iter\_counter: int = 0  
 func\_counter = 0  
 m = 0  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 xmin = minimize(function**,** xk**,** method='Nelder-Mead').x  
 fmin = function(xmin)  
  
 while True:  
 iter\_counter += 1  
 m = 0  
 while m < M:  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
  
 current\_func = function(xk)  
 func\_counter += 1  
  
 if current\_func < fmin:  
 fmin = current\_func  
 xmin = xk  
 break  
 m += 1  
 if m == M:  
 return xmin**,** fmin**,** func\_counter  
 else:  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 xmin = minimize(function**,** xk**,** method='Nelder-Mead').x  
 fmin = function(xmin)  
 func\_counter += 1  
  
  
def Alg3(function**,** M**,** sd):  
 V\_x = [-10**,** 10]  
 V\_y = V\_x  
  
 random.seed(sd)  
 iter\_counter: int = 0  
 func\_counter = 0  
 m = 0  
 xk = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 xmin = minimize(function**,** xk**,** method='Nelder-Mead').x  
 fmin = function(xmin)  
 delta = 0.5  
  
 while True:  
 iter\_counter += 1  
 m = 0  
 while m < M:  
 m += 1  
 gen\_x = np.array([random.uniform(V\_x[0]**,** V\_x[1])**,** random.uniform(V\_y[0]**,** V\_y[1])])  
 point = [0**,** 0]  
 while True:  
 point[0] += delta \* (gen\_x[0] / math.sqrt(gen\_x[0]\*\*2 + gen\_x[1]\*\*2))  
 point[1] += delta \* (gen\_x[1] / math.sqrt(gen\_x[0] \*\* 2 + gen\_x[1] \*\* 2))  
  
 func\_counter += 2  
 if function(point) < function(xmin) or (point[0] < V\_x[0] or point[0] > V\_x[1] or point[1] < V\_y[0] or point[1] > V\_y[1]):  
 break  
  
 if point[0] >= V\_x[0] and point[0] <= V\_x[1] and point[1] >= V\_y[0] and point[1] <= V\_y[1] :  
 break  
 if m == M:  
 return xmin**,** fmin**,** func\_counter  
 else:  
 current\_x = minimize(function**,** point**,** method='Nelder-Mead').x  
  
 if function(current\_x) < function(xmin):  
 xmin = current\_x  
 func\_counter += 1  
 fmin = function(xmin)  
  
  
#xk, counter, iter\_counter = fast\_gradient\_method(func\_for\_simple, gradient, np.array([5, 5]), epsi=1e-5)  
#result, k, third\_Pearson\_iter = third\_Pearson(func\_for\_simple, gradient, np.array([10, 10]), epsi=1e-3)  
#result1, k2, second\_Pearson\_iter = second\_Pearson(func\_for\_simple, gradient, np.array([10, 10]), epsi=1e-5)  
#xk, iter\_counter = barrier\_method(function, gradient, barrier\_function, g, np.array([5, 5]), eps=1e-5)  
#xk1, iter\_counter1 = penalty\_method(function, gradient, penalty\_function, g, np.array([1, 1]), eps=1e-5)  
test\_func = func\_for\_simple([-3**,** 7])  
P = [0.8**,** 0.9**,** 0.95**,** 0.99]  
eps = [1**,** 1e-1]  
"""  
df = pd.DataFrame()  
for i in range(len(eps)):  
 for j in range(len(P)):  
 min\_x, N = simple\_search(func\_for\_simple, P[j], eps[i])  
 current\_iter = [{'Eps': eps[i], 'P': P[j], 'X': min\_x, 'N': N}]  
 df = df.append(current\_iter, ignore\_index=True)  
df.to\_excel("simple\_search.xlsx")  
"""  
df = pd.DataFrame()  
m = [100**,** 500**,** 1000**,** 5000**,** 10000]  
sd = [0**,** 100**,** -100**,** -456321**,** 1234568]  
for i in range(len(sd)):  
 min\_x1**,** fmin1**,** func\_counter1 = Alg1(func\_for\_simple**,** 1000**,** sd[i])  
 min\_x2**,** fmin2**,** func\_counter2 = Alg2(func\_for\_simple**,** 1000**,** sd[i])  
 min\_x3**,** fmin3**,** func\_counter3= Alg3(func\_for\_simple**,** 1000**,** sd[i])  
 current\_iter = [{'sd': sd[i]**,** 'Alg1 X': min\_x1**,** 'Alg2 X': min\_x2**,** 'Alg3 X': min\_x3**,** 'Function Alg1': fmin1**,** 'Function Alg2': fmin2**,** 'Function Alg3': fmin3**,** 'Alg1 Count': func\_counter1**,** 'Alg2 Count': func\_counter2**,** 'Alg3 Count': func\_counter3}]  
 df = df.append(current\_iter**,** ignore\_index=True)  
df.to\_excel("alg.xlsx")  
  
  
  
#global\_min, min\_x = simple\_search(func\_for\_simple, 0.95, 1e-1)  
#global\_min, min\_x = pair\_method(func\_for\_simple, 1e-1, 0.001)  
x\_min**,** f\_min = Alg3(func\_for\_simple**,** 1e-3)  
#result = Alg1(func\_for\_simple, 1, 0.1)  
print(min\_x)