**Метод наискорейшего градиентного спуска:**

Постановка задачи: реализовать программу нахождения min функции с использованием метода Наискорейшего градиентного спуска. По условию:.

На Рисунке 1.1 представлено решение поставленной задачи, выполненное вручную. Вычисления проводились до момента, пока k не стало равно 3.

Изображение выглядит как текст, письмо

Автоматически созданное описание

Рисунок 1.1 – Решение задачи без программы

На Рисунке 1.2 представлена реализация поставленной задачи на языке программирования python.

import math  
def function(x,y):  
 return 3\*x\*x+4\*y\*y-2\*x\*y+x  
  
def const(a, b, e):  
 return (a+b+e)/2  
  
def norma(a):  
 return math.sqrt(a[0]\*a[0]+a[1]\*a[1])  
  
def grad\_func(a):  
 mas=[]  
 mas.append(6\*a[0]-2\*a[1]+1)  
 mas.append(8\*a[1] - 2\*a[0])  
 return mas  
  
def x\_koord(x, f, t):  
 return 3\*((x[0]-(t\*f[0]))\*\*2)+4\*((x[1]-(t\*f[1]))\*\*2)-2\*(x[0]-(t\*f[0]))\*(x[1]-(t\*f[1]))+(x[0]-(t\*f[0]))  
  
def xt\_koord(x,f,t,x\_new):  
 x\_new.append(x[0]-(t\*f[0]))  
 x\_new.append(x[1]-(t\*f[1]))  
 return x\_new  
  
def norm\_razn(x\_1,x\_0):  
 a=[]  
 a.append(x\_1[0]-x\_0[0])  
 a.append(x\_1[1]-x\_0[1])  
 return norma(a)  
  
def fun\_razn(X\_1,X\_0):  
 a1 = function(X\_1[0], X\_1[1])  
 a =function(X\_0[0], X\_0[1])  
 return abs(a1-a)  
  
E1 = 0.1  
E2 = 0.15  
M = 10  
E = 0.2  
l = 0.25  
a = -3  
b = 3  
N = 4  
k = 1  
kol = -1  
summa=0  
i=0  
X = []  
t = []  
#задание х0  
s = '2;1.5'  
X.append(list(map(float, (s.split(";")))))  
while (summa !=2):  
 kol += 1  
 f = grad\_func(X[i])  
 print("k=" + str(kol) + " f=" + str(f), end=" ")  
 if (norma(f) < E1):  
 print("Min" + str(X[i]))  
 break  
 if (kol >= M):  
 print("Min" + str(X[i]))  
 break  
 while(k > 0):  
 #print("k=" + str(k) + ", N=" + str(N), end=" ")  
 y = const(a, b, -E)  
 z = const(a, b, E)  
 f\_y = x\_koord(X[i],f,y)  
 f\_z = x\_koord(X[i],f,z)  
 #print("f(y)="+str(f\_y) + ", f(z)=" + str(f\_z), end=" ")  
 if (f\_y < f\_z):  
 b = z  
 else:  
 a = y  
 #print("L=["+str(a)+"," + str(b)+"]", "|L|=" + str(abs(b-a)), end=" ")  
 if (abs(b-a) <l):  
 x = (a+b)/2  
 t.append(x)  
 print("t = " + str(x), end=" ")  
 break  
 else:  
 k += 1  
 N = 2\*(k+1)  
 x\_new = []  
 x\_new = xt\_koord(X[i], f, t[i], x\_new)  
 print("x\_next= " + str(x\_new), end=" ")  
 X.append(x\_new)  
 on=norm\_razn(X[i+1], X[i])  
 tw=fun\_razn(X[i+1], X[i])  
 if ( on< E2 and tw < E2):  
 print(str(on)+"<"+str(E2), end=" ")  
 print(str(tw) + "<" + str(E2), end=" ")  
 summa += 1  
 else:  
 summa = 0  
 print()  
 i += 1  
if (summa==2):  
 print("Min= " + str(X[-1]))  
print("Otvet = [-0.18, -0.045]")

Рисунок 1.2 – Код программы

На Рисунке 1.3 показан вывод кода, изображенного на Рисунке 1.2, на консоли.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1.3 – Результат выполнения кода

**Вывод:**

Преимущества: легко реализуется, быстро сходится, хорош в обработке больших объемов данных, требует меньше памяти и вычислительной мощности, чем другие методы оптимизации.  
 Недостатки: может быть медленным или даже не соответствовать определенным моделям (особенно главным компонентам), может "застревать" в локальных минимумах, требует тщательной настройки параметров.

**Метод Ньютона:**

Постановка задачи: реализовать программу нахождения min функции с использованием метода Ньютона. По условию:.

На Рисунке 2.1 представлено решение поставленной задачи, выполненное вручную. Вычисления проводились до момента, пока k не стало равно 3.

Изображение выглядит как текст, письмо

Автоматически созданное описание

Рисунок 2.1 – Решение задачи без программы

На Рисунке 2.2 представлена реализация поставленной задачи на языке программирования python.

from sympy import \*  
import numpy as np  
import math  
  
# основная ф-ия  
def function(x,y):  
 return 3\*x\*x+4\*y\*y-2\*x\*y+x

# ф-ии для Дихотомии  
def const(a, b, e):  
 return (a+b+e)/2  
  
def norma(a):  
 return math.sqrt(a[0]\*a[0]+a[1]\*a[1])  
# поиск градиента ф-ии при передаче ф-ии  
def grad\_func(a):  
 mas=[]  
 mas.append(6\*a[0]-2\*a[1]+1)  
 mas.append(8\*a[1] - 2\*a[0])  
 return mas  
  
def x\_koord(x, f, t):  
 return 3\*((x[0]-(t\*f[0]))\*\*2)+4\*((x[1]-(t\*f[1]))\*\*2)-2\*(x[0]-(t\*f[0]))\*(x[1]-(t\*f[1]))+(x[0]-(t\*f[0]))  
# шаг 10. Подстановка т для поиска след х  
def xt\_koord(x,d,t,):  
 xk= (np.add(x,np.dot(t,d)))  
 return xk  
# норма разности 2-х точек  
def norm\_razn(x\_1,x\_0):  
 a=[]  
 a.append(x\_1[0]-x\_0[0])  
 a.append(x\_1[1]-x\_0[1])  
 return norma(a)  
# модуль разности двух ф-ий  
def fun\_razn(X\_1,X\_0):  
 a1 = function(X\_1[0], X\_1[1])  
 a =function(X\_0[0], X\_0[1])  
 return abs(a1-a)  
# поиск производной и формирование матрицы  
def find\_H():  
 x = Symbol('x')  
 y = Symbol('y')  
 z1 = 6 \* x - 2\*y+1  
 z2 = 8\*y - 2\*x  
 H, h1, h2 = [], [], []  
  
 x\_one = z1.diff(x)  
 xy\_one = z1.diff(y)  
 h1.append(float(x\_one))  
 h1.append(float(xy\_one))  
 y\_one = z2.diff(y)  
 h2.append(float(xy\_one))  
 h2.append(float(y\_one))  
 H.append(h1)  
 H.append(h2)  
 H = np.array(H)  
 print(H)  
 return H  
  
E1 = 0.1  
E2 = 0.15  
M = 10  
E = 0.2  
l = 0.25  
a = -3  
b = 3  
N = 4  
k = 1  
kol = -1  
summa=0  
i, t = 0,0  
X, D, T = [],[],[]  
#задание х0  
s = '2;1.5'  
X.append(list(map(float, (s.split(";")))))  
H = find\_H()  
while (summa!=2):  
 kol += 1  
 f = grad\_func(X[i]) #Поиск градиента  
 f = np.array(f).T  
 print("k=" + str(kol) + " f=" + str(f))  
 if (norma(f) < E1):  
 summa+=1  
 print("Min" + str(X[i]))  
 break  
 if (kol >= M):  
 print("Min" + str(X[i]))  
 break  
 H = np.linalg.inv(H) #Поиск обратной матрицы  
 print("H`-1=")  
 print(H)  
 if (H[0][0]>0 and (H[0][0]\*H[1][1]-H[0][1]\*H[1][0])>0):#Проверка главных миноров  
 d = np.dot(H, f)\*(-1)  
 t = 1  
 else:  
 d = f \* (-1)  
 print("d = ")  
 print(d)  
 if (t==1):  
 T.append(t)  
 else:  
 while(k > 0):  
 #print("k=" + str(k) + ", N=" + str(N), end=" ")  
 y = const(a, b, -E)  
 z = const(a, b, E)  
 f\_y = x\_koord(X[i],f,y)  
 f\_z = x\_koord(X[i],f,z)  
 #print("f(y)="+str(f\_y) + ", f(z)=" + str(f\_z), end=" ")  
 if (f\_y < f\_z):  
 b = z  
 else:  
 a = y  
 #print("L=["+str(a)+"," + str(b)+"]", "|L|=" + str(abs(b-a)), end=" ")  
 if (abs(b-a) <l):  
 x = (a+b)/2  
 T.append(x)  
 print("t = " + str(x), end=" ")  
 break  
 else:  
 k += 1  
 N = 2\*(k+1)  
 x\_new = []  
 x\_new = xt\_koord(X[i], d, T[i])  
 print("x\_next= ")  
 print(x\_new)  
 X.append(x\_new)  
  
 on=norm\_razn(X[i+1], X[i])  
 tw=fun\_razn(X[i+1], X[i])  
 if ( on< E2 and tw < E2):  
 print(str(on)+"<"+str(E2), end=" ")  
 print(str(tw) + "<" + str(E2), end=" ")  
 summa += 1  
 else:  
 summa = 0  
 print()  
 i += 1  
print()  
if (summa==2):  
 print("Min= " + str(X[-1]))  
print("Otvet = [-0.18, -0.045]")

Рисунок 2.2 – Код программы

На Рисунке 2.3 показан вывод кода, изображенного на Рисунке 2.2, на консоли.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 2.3 – Результат выполнения кода

**Вывод:**

Преимущества: сходится быстрее, чем наискорейший градиентный спуск, эффективен для определения высокоуровневых градиентов и матрицы Гессе, обычно сходится к глобальному минимуму.  
Недостатки: вычислительно более сложен, чувствителен к результатам начальной инициализации, может потребовать значительное количество памяти.

**Метод Ньютона-Рафсона:**

Постановка задачи: реализовать программу нахождения min функции с использованием метода Ньютона-Рафсона. По условию:.

На Рисунке 3.1 представлено решение поставленной задачи, выполненное вручную.

Изображение выглядит как текст, письмо

Автоматически созданное описание

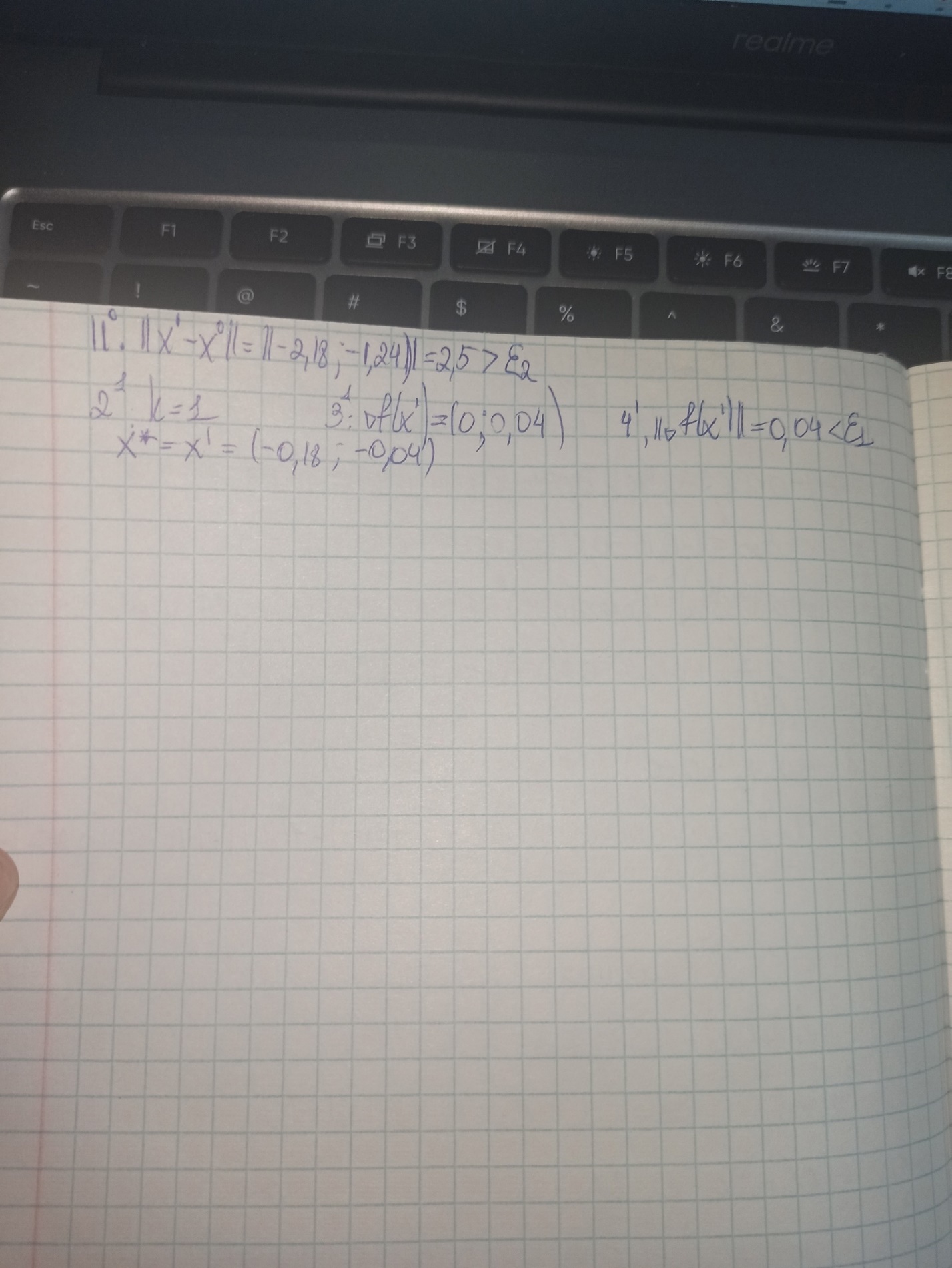


Рисунок 3.1 – Решение задачи без программы

На Рисунке 3.2 представлена реализация поставленной задачи на языке программирования python.

from sympy import \*  
import numpy as np  
import math  
  
# основная ф-ия  
def function(x,y):  
 return 3\*x\*x+4\*y\*y-2\*x\*y+x

# ф-ии для Дихотомии  
def const(a, b, e):  
 return (a+b+e)/2  
# поиск нормы точки  
def norma(a):  
 return math.sqrt(a[0]\*a[0]+a[1]\*a[1])  
# поиск градиента ф-ии при передаче ф-ии  
def grad\_func(a):  
 mas=[]  
 mas.append(6\*a[0]-2\*a[1]+1)  
 mas.append(8\*a[1] - 2\*a[0])  
 return mas  
# ф-ия для поиска t  
def x\_koord(x, f, t):  
 return 3\*((x[0]+(t\*f[0]))\*\*2)+4\*((x[1]+(t\*f[1]))\*\*2)-2\*(x[0]+(t\*f[0]))\*(x[1]+(t\*f[1]))+(x[0]+(t\*f[0]))  
  
# шаг 10. Подстановка т для поиска след х  
def xt\_koord(x,d,t,):  
 xk= (np.add(x,np.dot(t,d)))  
 return xk  
  
# норма разности 2-х точек  
def norm\_razn(x\_1,x\_0):  
 a=[]  
 a.append(x\_1[0]-x\_0[0])  
 a.append(x\_1[1]-x\_0[1])  
 return norma(a)  
  
# модуль разности двух ф-ий  
def fun\_razn(X\_1,X\_0):  
 a1 = function(X\_1[0], X\_1[1])  
 a =function(X\_0[0], X\_0[1])  
 return abs(a1-a)  
  
# поиск производной и формирование матрицы  
def find\_H():  
 x = Symbol('x')  
 y = Symbol('y')  
 z1 = 6 \* x - 2\*y+1  
 z2 = 8\*y - 2\*x  
 H, h1, h2 = [], [], []  
  
 x\_one = z1.diff(x)  
 xy\_one = z1.diff(y)  
 h1.append(float(x\_one))  
 h1.append(float(xy\_one))  
 y\_one = z2.diff(y)  
 h2.append(float(xy\_one))  
 h2.append(float(y\_one))  
 H.append(h1)  
 H.append(h2)  
 H = np.array(H)  
 print(H)  
 return H  
def sven(a, b, Y, f):  
 X = []  
 x=-2  
 t=1  
 i=0  
 X.append(x)  
 f\_1 = x\_koord(Y, f, (X[i]-t))  
 f\_2 = x\_koord(Y, f,X[i])  
 f\_3 = x\_koord(Y, f,X[i]+t)  
 if (f\_1 >= f\_2 and f\_2 <= f\_3):  
 print("[" + str(X[i]-t) + ";" + str(X[i]+t)+ "]")  
 elif (f\_1 <= f\_2 and f\_2 >= f\_3):  
 print("задайте другой x0")  
 else:  
 if (f\_1 >= f\_2 >= f\_3):  
 d = t  
 a = X[0]  
 x\_next=X[0]+t  
 i = 1  
 else:  
 d = -t  
 b = X[0]  
 x\_next = X[0] - t  
 i = 1  
 X.append(x\_next)  
 while(1>0):  
 x\_next = X[i] + pow(2,i) \* d  
 X.append(x\_next)  
 if (x\_koord(Y, f,X[i+1])< x\_koord(Y, f,X[i]) and d == t):  
 a = X[i]  
 elif(x\_koord(Y, f,X[i+1])< x\_koord(Y, f,X[i]) and d == t):  
 b = X[i]  
 else:  
 if (d == t):  
 b = X[i+1]  
 else:  
 a = X[i+1]  
 break  
 i += 1  
 print("["+ str(a)+ ";" + str(b))  
  
E1 = 0.1  
E2 = 0.15  
M = 10  
E = 0.2  
l = 0.25  
a = -5  
b = 3  
N = 4  
k = 1  
kol = -1  
summa=0  
i, t = 0,0  
X, D, T = [], [], []  
  
#задание х0  
s = '2;1.5'  
X.append(list(map(float, (s.split(";")))))  
  
H = find\_H()  
while (summa!=2):  
 kol += 1  
 f = grad\_func(X[i]) #Поиск градиента  
 f = np.array(f).T  
 print("k=" + str(kol) + " f=" + str(f))  
 if (norma(f) < E1):  
 summa+=1  
 print("Min" + str(X[i]))  
 break  
 if (kol >= M):  
 print("Min" + str(X[i]))  
 break  
 H = np.linalg.inv(H) #Поиск обратной матрицы  
 print("H`-1=")  
 print(H)  
 if (H[0][0]>0 and (H[0][0]\*H[1][1]-H[0][1]\*H[1][0])>0):#Проверка главных миноров  
 d = np.dot(H, f)\*(-1)  
 else:  
 d = f \* (-1)  
 print("d = ")  
 print(d)  
 sven(a,b,X[i], d)  
#Метод Дихотомии  
 while (k > 0):  
 # print("k=" + str(k) + ", N=" + str(N), end=" ")  
 y = const(a, b, -E)  
 z = const(a, b, E)  
 f\_y = x\_koord(X[i], d, y)  
 f\_z = x\_koord(X[i], d, z)  
 # print("f(y)="+str(f\_y) + ", f(z)=" + str(f\_z), end=" ")  
 if (f\_y < f\_z):  
 b = z  
 else:  
 a = y  
 # print("L=["+str(a)+"," + str(b)+"]", "|L|=" + str(abs(b-a)), end=" ")  
 if (abs(b - a) < l):  
 x = (a + b) / 2  
 T.append(x)  
 print("t = " + str(x), end=" ")  
 break  
 else:  
 k += 1  
 N = 2 \* (k + 1)  
 x\_new = []  
 x\_new = xt\_koord(X[i], d, T[i])  
 print("x\_next= ")  
 print(x\_new)  
 X.append(x\_new)  
  
 on=norm\_razn(X[i+1], X[i])  
 tw=fun\_razn(X[i+1], X[i])  
 if ( on< E2 and tw < E2):  
 print(str(on)+"<"+str(E2), end=" ")  
 print(str(tw) + "<" + str(E2), end=" ")  
 summa += 1  
 else:  
 summa = 0  
 print()  
 i += 1  
print()  
if (summa==2):  
 print("Min= " + str(X[-1]))  
print("Otvet = [-0.18, -0.045]")

Рисунок 3.2 – Код программы

На Рисунке 3.3 показан вывод кода, изображенного на Рисунке 3.2, на консоли.



Рисунок 3.3 – Результат выполнения кода

**Вывод:**

Преимущества: эффективен для определения высокоуровневых градиентов и матрицы Гессе, сходится быстрее, чем Ньютона.  
 Недостатки: может потребовать значительное количество памяти, могут возникнуть проблемы с неустойчивостью, необоснованными обратными матрицами.

**Метод Флетчера-Ривса:**

Постановка задачи: реализовать программу нахождения min функции с использованием метода Флетчера-Ривса. По условию:.

На Рисунке 4.1 представлено решение поставленной задачи, выполненное вручную.

Изображение выглядит как текст, письмо

Автоматически созданное описание

Изображение выглядит как текст, письмо

Автоматически созданное описание

Рисунок 4.1 – Решение задачи без программы

На Рисунке 4.2 представлена реализация поставленной задачи на языке программирования python.

import numpy as np  
import math  
  
# основная ф-ия  
def function(x,y):  
 return 3\*x\*x+4\*y\*y-2\*x\*y+x  
# ф-ии для Дихотомии  
def const(a, b, e):  
 return (a+b+e)/2  
# поиск нормы точки  
def norma(a):  
 return math.sqrt(a[0]\*a[0]+a[1]\*a[1])  
# поиск градиента ф-ии при передаче ф-ии  
def grad\_func(a):  
 mas=[]  
 mas.append(6\*a[0]-2\*a[1]+1)  
 mas.append(8\*a[1] - 2\*a[0])  
 return mas  
# ф-ия для поиска t  
def x\_koord(x, f, t):  
 return 3\*((x[0]+(t\*f[0]))\*\*2)+4\*((x[1]+(t\*f[1]))\*\*2)-2\*(x[0]+(t\*f[0]))\*(x[1]+(t\*f[1]))+(x[0]+(t\*f[0]))  
  
# шаг 10. Подстановка т для поиска след х  
def xt\_koord(x,d,t,):  
 xk= (np.add(x,np.dot(t,d)))  
 return xk  
  
# норма разности 2-х точек  
def norm\_razn(x\_1,x\_0):  
 a=[]  
 a.append(x\_1[0]-x\_0[0])  
 a.append(x\_1[1]-x\_0[1])  
 return norma(a)  
  
# модуль разности двух ф-ий  
def fun\_razn(X\_1,X\_0):  
 a1 = function(X\_1[0], X\_1[1])  
 a =function(X\_0[0], X\_0[1])  
 return abs(a1-a)  
  
def sven(a, b, Y, f):  
 X = []  
 x=-2  
 t=1  
 i=0  
 X.append(x)  
 f\_1 = x\_koord(Y, f, (X[i]-t))  
 f\_2 = x\_koord(Y, f,X[i])  
 f\_3 = x\_koord(Y, f,X[i]+t)  
 if (f\_1 >= f\_2 and f\_2 <= f\_3):  
 print("[" + str(X[i]-t) + ";" + str(X[i]+t)+ "]")  
 elif (f\_1 <= f\_2 and f\_2 >= f\_3):  
 print("задайте другой x0")  
 else:  
 if (f\_1 >= f\_2 >= f\_3):  
 d = t  
 a = X[0]  
 x\_next=X[0]+t  
 i = 1  
 else:  
 d = -t  
 b = X[0]  
 x\_next = X[0] - t  
 i = 1  
 X.append(x\_next)  
 while(1>0):  
 x\_next = X[i] + pow(2,i) \* d  
 X.append(x\_next)  
 if (x\_koord(Y, f,X[i+1])< x\_koord(Y, f,X[i]) and d == t):  
 a = X[i]  
 elif(x\_koord(Y, f,X[i+1])< x\_koord(Y, f,X[i]) and d == t):  
 b = X[i]  
 else:  
 if (d == t):  
 b = X[i+1]  
 else:  
 a = X[i+1]  
 break  
 i += 1  
  
E1 = 0.1  
E2 = 0.15  
M = 10  
E = 0.2  
l = 0.25  
a = -5  
b = 3  
N = 4  
k = 1  
kol = -1  
summa=0  
i, t, tmp = 0,0, 0  
X, D, T = [], [], []  
  
#задание х0  
s = '2;1.5'  
X.append(list(map(float, (s.split(";")))))  
  
  
while (summa!=2):  
 kol += 1  
 f = grad\_func(X[i]) #Поиск градиента  
 f = np.array(f).T  
 print("k=" + str(kol) + " f=" + str(f))  
 if (norma(f) < E1):  
 summa+=1  
 print("Min" + str(X[i]))  
 break  
 if (kol >= M):  
 print("Min" + str(X[i]))  
 break  
 if (tmp == 0):  
 d = f \* (-1)

tmp += 1  
 else:  
 f\_pred = grad\_func(X[i - 1])  
 bet = (norma(f) \* norma(f)) / (norma(f\_pred) \* norma(f\_pred))  
 print("betta = " + str(bet))  
 d = f \* (-1) + np.dot(bet, d)  
 print("d=")  
 print(d)  
 sven(a,b,X[i], d)  
#Метод Дихотомии  
 while (k > 0):  
 y = const(a, b, -E)  
 z = const(a, b, E)  
 f\_y = x\_koord(X[i], d, y)  
 f\_z = x\_koord(X[i], d, z)  
 if (f\_y < f\_z):  
 b = z  
 else:  
 a = y  
 if (abs(b - a) < l):  
 x = (a + b) / 2  
 T.append(x)  
 print("t = " + str(x), end=" ")  
 break  
 else:  
 k += 1  
 N = 2 \* (k + 1)  
 x\_new = []  
 x\_new = xt\_koord(X[i], d, T[i])  
 print("x\_next= ")  
 print(x\_new)  
 X.append(x\_new)  
  
 on=norm\_razn(X[i+1], X[i])  
 tw=fun\_razn(X[i+1], X[i])  
 if ( on< E2 and tw < E2):  
 print(str(on)+"<"+str(E2), end=" ")  
 print(str(tw) + "<" + str(E2), end=" ")  
 summa += 1  
 else:  
 summa = 0  
 print()  
 i += 1  
print()  
if (summa==2):  
 print("Min= " + str(X[-1]))  
print("Otvet = [-0.18, -0.045]")

Рисунок 3.2 – Код программы

На Рисунке 4.3 показан вывод кода, изображенного на Рисунке 4.2, на консоли.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.3 – Результат выполнения кода

**Вывод:**

Преимущества: эффективный и универсальный метод оптимизации, сходится быстрее, чем Ньютона-Рафсона.  
 Недостатки: может потребовать значительное количество памяти, могут возникнуть проблемы с неустойчивостью, необоснованными обратными матрицами.