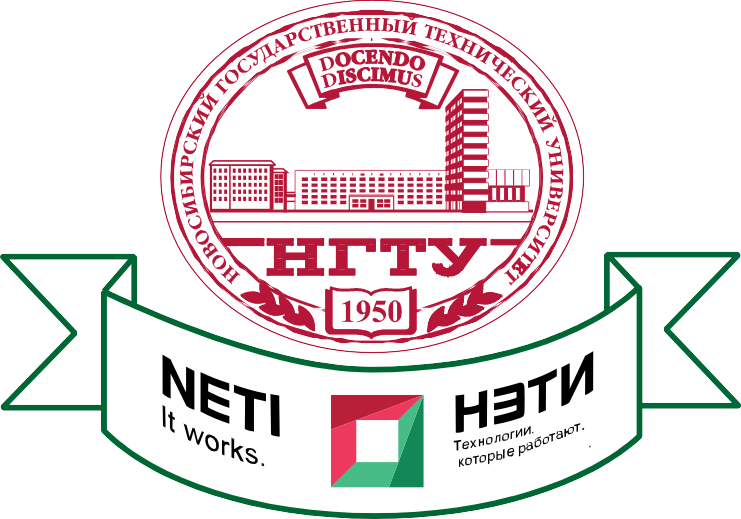
|  |  |
| --- | --- |
| Министерство науки и высшего образования  Российской Федерации | |
| Федеральное государственное бюджетное образовательное учреждение высшего образования | |
| «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» | |
| Кафедра прикладной математики | |
| Лабораторная работа № 2 | |
| по дисциплине «Методы оптимизации» | |
| Методы спуска | |
| Факультет: | ПМИ |
| Группа: | ПМ-81 |
| Бригада: | 8 |
| Студенты: | Редут Анатолий, |
|  | Нетягин Михаил, |
|  | Параев Павел |
| Преподаватели: | Чимитова Е.В. |
|  |  |
| Новосибирск | |
| 2021 |  |



# Цель работы

# 

# Постановка задачи

# 

# 

# Вариант

# 

# Функция Розенброка

# 

# Квадратичная функция

# 

1. **Исследования реализованных алгоритмов на квадратичной функции, функции Розенброка и на тестовой функции**

**Точка x = 0, y = 0**

# Квадратичная функция

# 

# E = 1E-4

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 257  Вычислений функций – 5140  Результат – 0.9999, 0.9999 | Итерация – 32  K = 22  Вычислений функций – 640  Результат – 1.00000, 1.00000 | Итерация – 1989  Вычислений функций – 39780  Результат – 0,99994, 0.99994 |

# E = 1E-5

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 1157  K = 2  Вычислений функций – 27768  Результат – 0.99999, 0.99999 | Итерация – 21  K = 22  Вычислений функций – 504  Результат – 1.00000, 0.99999 | Итерация – 2453  Вычислений функций – 588720  Результат – 0,99999, 0.99999 |

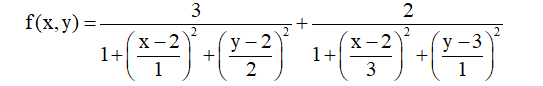
# E = 1E-6

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 4673  K = 2  Вычислений функций – 135527  Результат – 0.99999, 0.99999 | Итерация – 21  K = 22  Вычислений функций – 609  Результат – 0.99999, 0.99999 | Итерация – 2917  Вычислений функций – 84593  Результат – 0,99999, 0.99999 |

# E = 1E-7

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 6749  K = 2  Вычислений функций – 229466  Результат – 0.99999, 0.99999 | Итерация – 22  K = 22  Вычислений функций – 748  Результат – 0.99999, 0.99999 | Итерация – 3379  Вычислений функций – 114886  Результат – 0.9999, 0.9999 |

# Вариант



# E = 1E-4

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 11  K = 2  Вычислений функций – 220  Результат – 1,99999, 2.756 | Итерация – 17  K = 2  Вычислений функций – 340  Результат – 1,99999, 2.756 | Итерация – 14  Вычислений функций – 280  Результат – 2, 2.756 |

# E = 1E-5

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 13  K = 2  Вычислений функций – 312  Результат – 1,99999, 2.756 | Итерация – 18  K = 2  Вычислений функций – 432  Результат – 1,99999, 2.756 | Итерация – 15  Вычислений функций – 360  Результат – 1,99999, 2.756 |

# E = 1E-6

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 15  K = 2  Вычислений функций – 425  Результат – 1,99999, 2.756 | Итерация – 20  K = 2  Вычислений функций – 580  Результат – 1,99999, 2.756 | Итерация – 17  Вычислений функций – 496  Результат – 1,99999, 2.756 |

# E = 1E-7

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 17  K = 2  Вычислений функций – 518  Результат – 1,99999, 2.756 | Итерация – 21  K = 2  Вычислений функций – 714  Результат – 1,99999, 2.756 | Итерация – 18  Вычислений функций – 612  Результат – 2, 2.756 |

# Функция Розенброка

# 

# E = 1E-4

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 516  K = 15  Вычислений функций – 10320  Результат – 1, 1 | Итерация – 9416  Вычислений функций - 188320  Результат – 0,9999, 0.99999 |

# E = 1E-5

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 1722  K = 20  Вычислений функций – 41328  Результат – 1, 1 | Итерация – MaxIter  Результат – 0,9999, 0.99999 |

# E = 1E-6

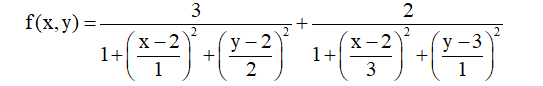
|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 2932  K = 22  Вычислений функций – 96280  Результат – 1, 1 | Итераций = MaxIter  Результат – 0,99999, 0.99999 |

# E = 1E-7

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Метод не сходится за 10000 итераций | Итерация = MaxIter  Результат – 0.9999, 0.99999 |

**Точка x = 10, y = 10**

# Вариант



# E = 1E-4

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 115  K = 2  Вычислений функций – 2300  Результат – 1,99999, 2.756 | Итерация – 52  K = 22  Вычислений функций – 1040  Результат – 1,99999, 2.756 | Итерация – 209  Вычислений функций – 4180  Результат – 2, 2.756 |

# E = 1E-5

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 115  K = 2  Вычислений функций – 2760  Результат – 1,99999, 2.756 | Итерация – 58  K = 22  Вычислений функций – 1392  Результат – 1,99999, 2.756 | Итерация – 211  Вычислений функций – 5064  Результат – 1,99999, 2.756 |

# E = 1E-6

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 115  K = 2  Вычислений функций – 3336  Результат – 1,99999, 2.756 | Итерация – 35  K = 22  Вычислений функций – 1015  Результат – 1,99999, 2.756 | Итерация – 212  Вычислений функций – 6148  Результат – 1,99999, 2.756 |

# E = 1E-7

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Итерация – 117  K = 2  Вычислений функций – 3978  Результат – 1,99999, 2.756 | Итерация – 54  K = 22  Вычислений функций –1836  Результат – 1,99999, 2.756 | Итерация – 214  Вычислений функций – 7276  Результат – 2, 2.756 |

# Функция Розенброка

# 

# E = 1E-4

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится при заданной точности | Метод не сходится при заданной точности | Метод не сходится за 10000 итераций |

# E = 1E-5

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 353  K = 100  Вычислений функций – 8472  Результат – 1, 1 | Метод не сходится за 10000 итераций |

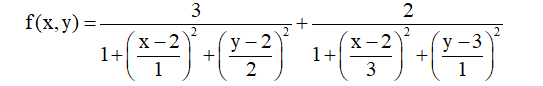
# E = 1E-6

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 638  K = 0  Вычислений функций – 206132  Результат – 1, 1 | Метод не сходится за 10000 итераций |

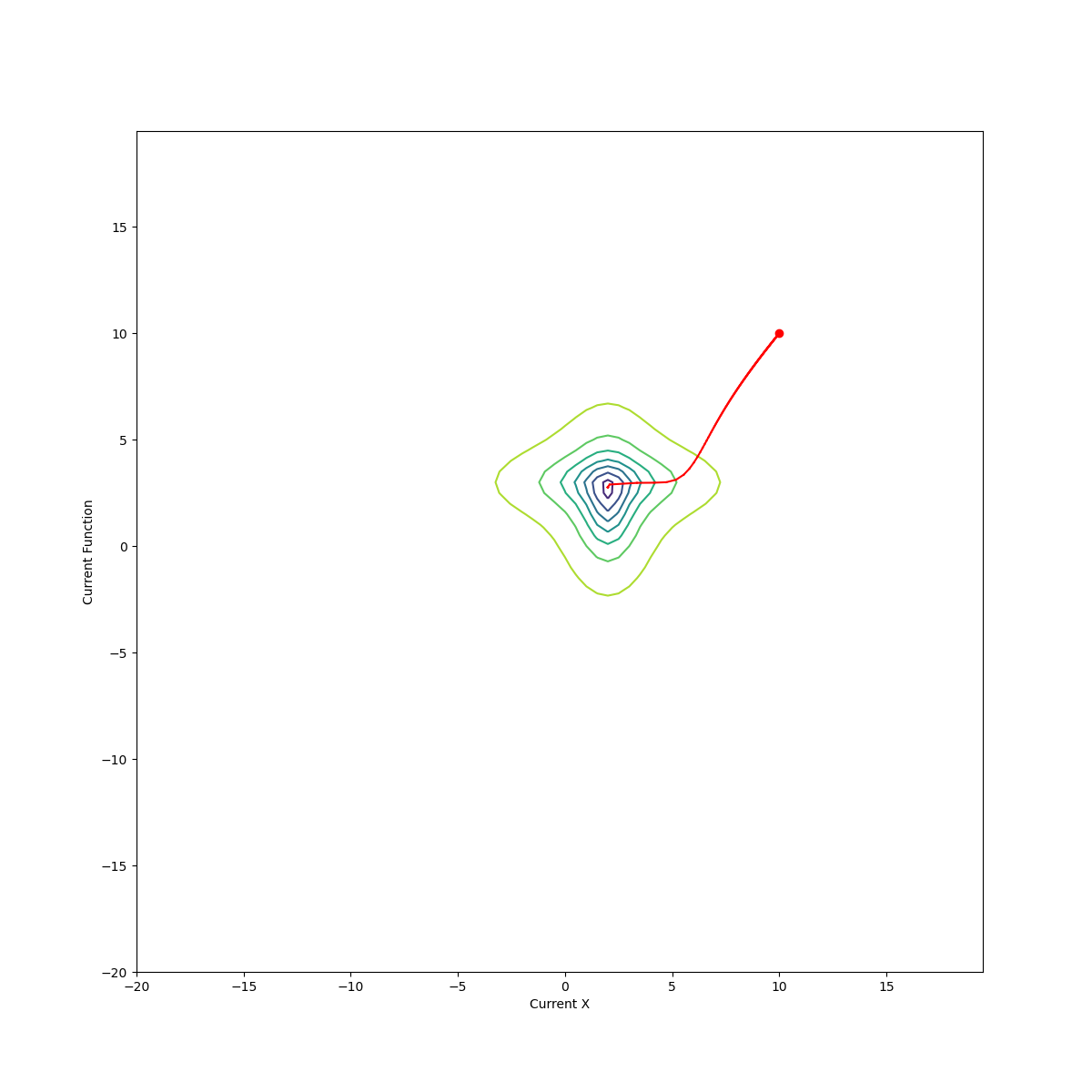
# E = 1E-7

|  |  |  |
| --- | --- | --- |
| **Второй Пирсон** | **Третий Пирсон** | **Наискорейший спуск** |
| Метод не сходится за 10000 итераций | Итерация – 4294  K = 0  Вычислений функций – 314092  Результат – 1, 1 | Метод не сходится за 10000 итераций |

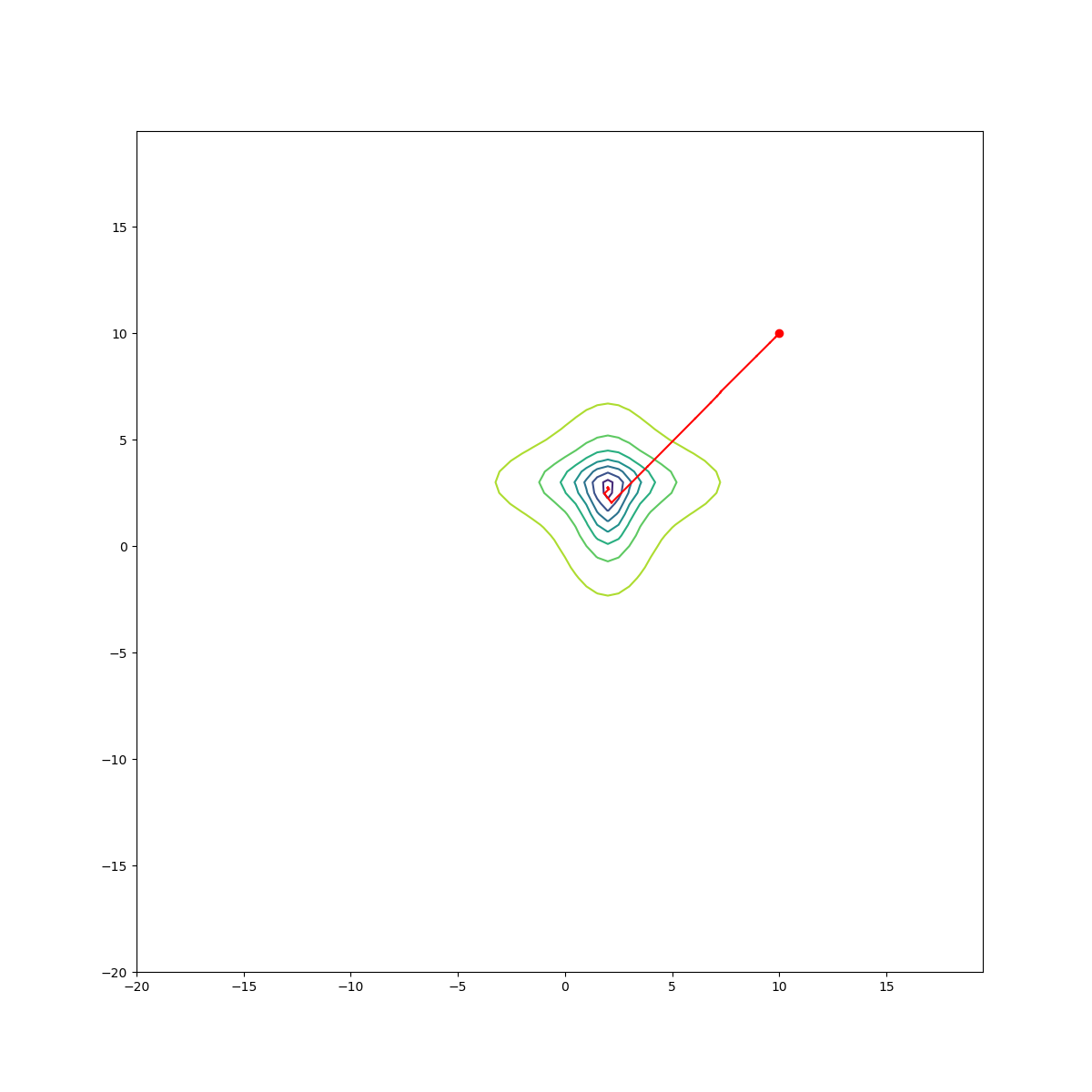
1. **Наглядное представление сходимости на тестовой функции**



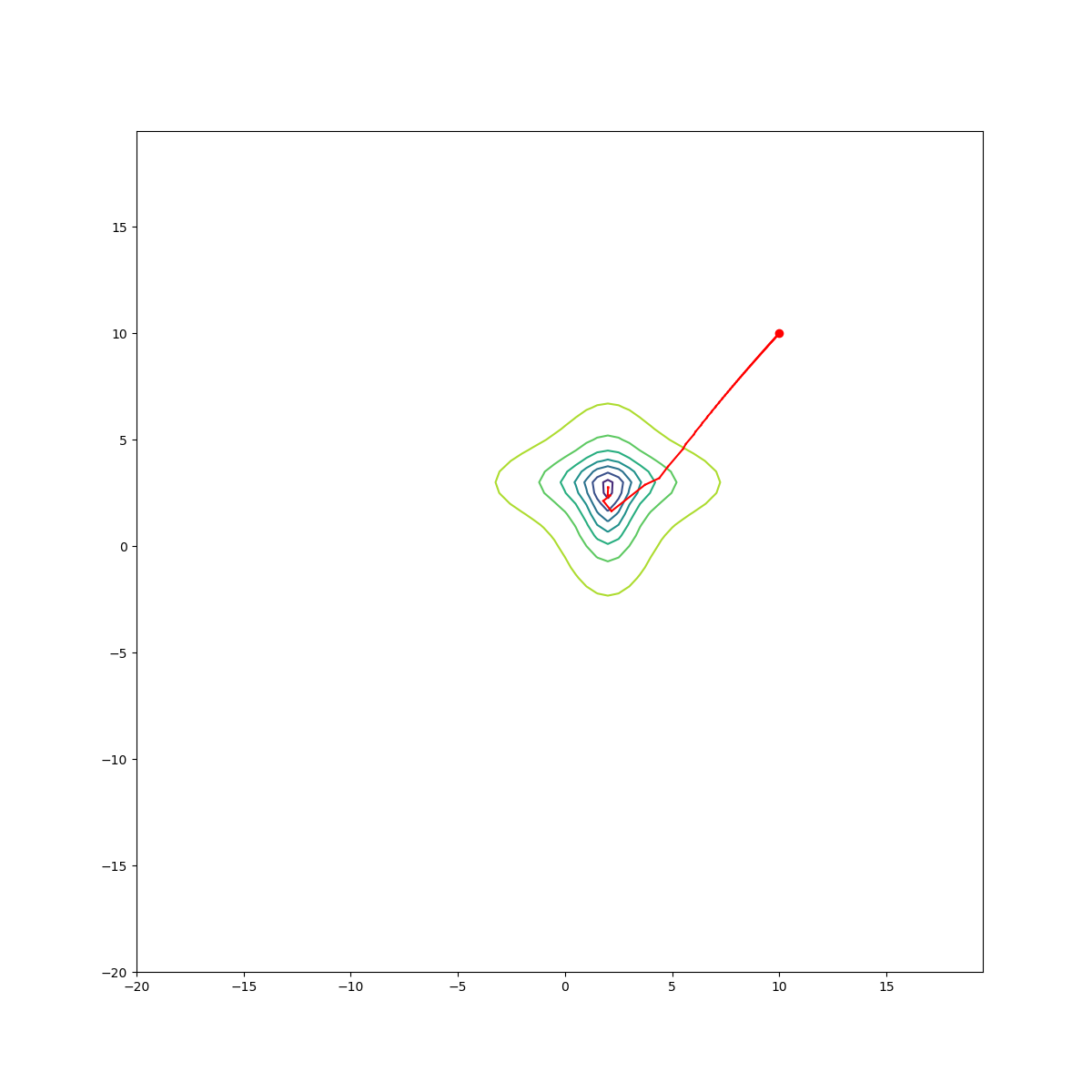
**Fast gradient**

****

**Third Pearson**

****

**Second Pearson**

****

1. **Исследование на сходимость**

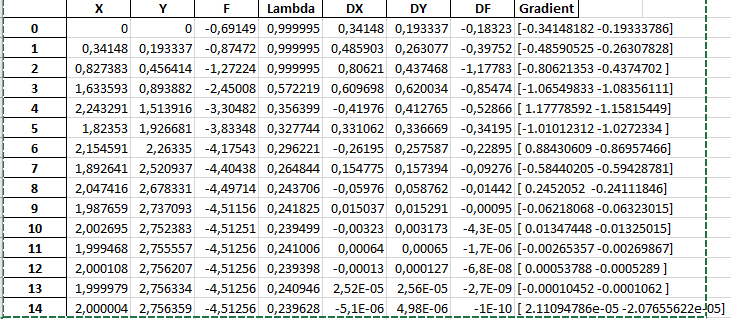
**Eps = 1e-5**

**Начальное приближение: x = 0, y = 0**

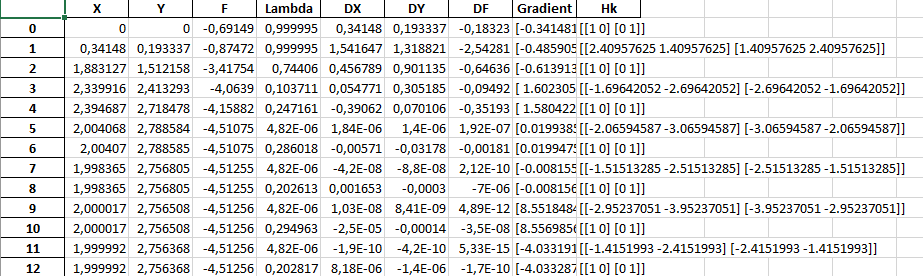
**K = 2**

**Тестовая функция**

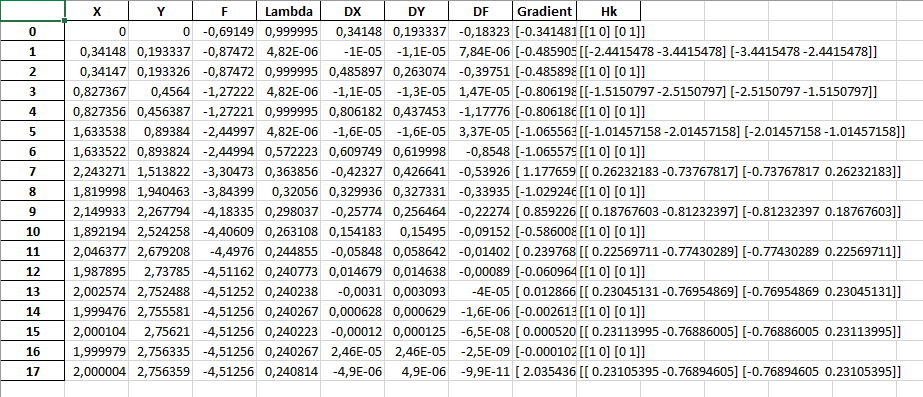
**Fast Gradient**

****

**Second Pearson**

****

**Third Pearson**

****

1. **Выводы**

Исследование показали, что Третий алгоритм Пирсона в целом эффективнее для вычисления минимума(при выборе наилучшего параметра k).

Исследования выбора параметра k показали, что для второго метода Пирсона оптимальным и единственным является k = 2(иначе алгоритм расходится).

Для Третьего алгоритма Пирсона выбор параметра зависит от начального приближения и от точности(при маленькой точности параметре k лучше всего выставить в 0(не обновлять матрицу до единичной вообще)).

Метод градиентного спуска при приближении к минимум начинает сходиться медленно по причине очень маленького градиента функции. Соответственно, метод был неэффективен для функции Розенброка и квадратичной функции.

Второй метод Пирсона оказался эффективнее всех на тестовой функции.

1. **Код программы**

**Файл main.py**

import numpy as np  
import numpy.linalg as ln  
from method import golden\_ratio\_2**,** golden\_ratio  
import numdifftools as nd  
import pandas as pd  
import matplotlib.pyplot as plt  
  
  
def fast\_gradient\_method(f**,** fprime**,** x0**,** maxiter=10000**,** epsi=1e-6):  
 df = pd.DataFrame()  
  
 fig\_size = plt.rcParams["figure.figsize"]  
 fig\_size[0] = 12  
 fig\_size[1] = 12  
 plt.rcParams["figure.figsize"] = fig\_size  
  
 X = np.arange(-20**,** 20**,** 0.5)  
 Y = np.arange(-20**,** 20**,** 0.5)  
 X**,** Y = np.meshgrid(X**,** Y)  
 plt.contour(X**,** Y**,** f([X**,** Y]))  
 plt.xlabel('Current X')  
 plt.ylabel('Current Function')  
  
 plt.plot(x0[0]**,** x0[1]**,** 'ro')  
  
 iter\_counter: int = 0  
 xk = x0  
 current\_gradient = fprime(x0)  
 iter\_counter\_func = 0  
  
 while iter\_counter < maxiter:  
 q = lambda alpha: f(xk - alpha \* current\_gradient)  
 coeff**,** iter\_func = golden\_ratio(q**,** 0**,** 1**,** epsi)  
 iter\_counter\_func += iter\_func  
 xNext = xk - coeff \* current\_gradient  
  
 current\_iter = [{'X': xk[0]**,** 'Y': xk[1]**,** 'F': f(xk)**,** 'Lambda': coeff**,** 'DX': xNext[0] - xk[0]**,** 'DY': xNext[1] - xk[1]**,** 'DF': f(xNext) - f(xk)**,** 'Gradient': current\_gradient}]  
 df = df.append(current\_iter**,** ignore\_index=True)  
  
 plt.plot([xk[0]**,** xNext[0]]**,** [xk[1]**,** xNext[1]]**,** '-r')  
  
 xk = xNext  
 current\_gradient = fprime(xk)  
 iter\_counter += 1  
  
 if np.linalg.norm(current\_gradient) < epsi:  
 plt.show()  
 df.to\_excel('Report.xlsx')  
 return xk**,** iter\_counter**,** iter\_counter\_func  
  
 plt.show()  
 df.to\_excel('Report.xlsx')  
 return xk**,** iter\_counter**,** iter\_counter\_func  
  
  
  
  
def function(x):  
 #return x[0]\*\*2 + x[1]\*\*2  
 #x, y = X  
 #return 10 \* x\*\*2 + y\*\*2  
 #return x\*\*2-x\*y+y\*\*2+9\*x-6\*y+20  
 #return (1 - x) \*\* 2 + 100 \* (y - x\*\*2) \*\* 2  
 return -3 / (1 + (x[0] - 2)\*\*2 + (x[1] - 2)\*\*2 / 4) - 2 / (1 + (x[0] - 2)\*\*2 / 9 + (x[1] - 3)\*\*2)  
  
  
def gradient(x):  
 grad = nd.Gradient(function)  
 dx**,** dy = grad([x[0]**,** x[1]])  
 return np.array([dx**,** dy])  
  
  
def second\_Pearson(f**,** fprime**,** x0**,** maxiter=10000**,** epsi=1e-4):  
 df = pd.DataFrame()  
  
 fig\_size = plt.rcParams["figure.figsize"]  
 fig\_size[0] = 12  
 fig\_size[1] = 12  
 plt.rcParams["figure.figsize"] = fig\_size  
  
 X = np.arange(-20**,** 20**,** 0.5)  
 Y = np.arange(-20**,** 20**,** 0.5)  
 X**,** Y = np.meshgrid(X**,** Y)  
 plt.contour(X**,** Y**,** f([X**,** Y]))  
  
 plt.plot(x0[0]**,** x0[1]**,** 'ro')  
  
 k = 0  
 current\_gradient = fprime(x0)  
 N = len(x0)  
  
 I = np.eye(N**,** dtype=int)  
 Hk = I  
 xk = x0  
 alpha\_k = 0.005  
 iter\_counter\_func = 0  
  
 while k < maxiter:  
 pk = np.dot(Hk**,** current\_gradient)  
 q = lambda alpha: f(xk - alpha \* pk)  
 alpha\_k**,** iter\_func = golden\_ratio\_2(q**,** 0**,** 1**,** epsi)  
 iter\_counter\_func += iter\_func  
  
 xkp1 = xk - alpha\_k \* pk  
  
 current\_iter = [{'X': xk[0]**,** 'Y': xk[1]**,** 'F': f(xk)**,** 'Lambda': alpha\_k**,** 'DX': xkp1[0] - xk[0]**,** 'DY': xkp1[1] - xk[1]**,** 'DF': f(xkp1) - f(xk)**,** 'Gradient': current\_gradient**,** 'Hk': Hk}]  
 df = df.append(current\_iter**,** ignore\_index=True)  
 plt.plot([xk[0]**,** xkp1[0]]**,** [xk[1]**,** xkp1[1]]**,** '-r')  
  
 sk = xkp1 - xk  
 xk = xkp1  
  
 gfkp1 = fprime(xkp1)  
 yk = gfkp1 - current\_gradient  
 current\_gradient = gfkp1  
 k += 1  
  
 temp = 1 / np.dot(yk[np.newaxis**,** :]**,** sk)  
  
 if k % 2 == 0:  
 Hk = np.eye(N**,** dtype=int)  
 else:  
 Hk = Hk + ((np.dot(sk - np.dot(Hk**,** yk)**,** np.dot(yk**,** Hk).transpose())) \* temp)  
  
 if ln.norm(current\_gradient) < epsi:  
 df.to\_excel("Report\_Second\_Pearson.xlsx")  
 plt.show()  
 return xk**,** k**,** iter\_counter\_func  
  
 df.to\_excel("Report\_Second\_Pearson.xlsx")  
 plt.show()  
 return xk**,** k**,** iter\_counter\_func  
  
  
def third\_Pearson(f**,** fprime**,** x0**,** maxiter=10000**,** epsi=1e-4):  
 df = pd.DataFrame()  
  
 fig\_size = plt.rcParams["figure.figsize"]  
 fig\_size[0] = 12  
 fig\_size[1] = 12  
 plt.rcParams["figure.figsize"] = fig\_size  
  
 X = np.arange(-20**,** 20**,** 0.5)  
 Y = np.arange(-20**,** 20**,** 0.5)  
 X**,** Y = np.meshgrid(X**,** Y)  
 plt.contour(X**,** Y**,** f([X**,** Y]))  
  
 plt.plot(x0[0]**,** x0[1]**,** 'ro')  
  
 k = 0  
 current\_gradient = fprime(x0)  
 N = len(x0)  
  
 I = np.eye(N**,** dtype=int)  
 Hk = I  
 xk = x0  
 alpha\_k = 0.005  
 iter\_counter\_func = 0  
  
 while k < maxiter:  
 pk = np.dot(Hk**,** current\_gradient)  
 q = lambda alpha: f(xk - alpha \* pk)  
 alpha\_k**,** iter\_func = golden\_ratio\_2(q**,** 0**,** 1**,** epsi)  
 iter\_counter\_func += iter\_func  
 xkp1 = xk - alpha\_k \* pk  
  
 current\_iter = [{'X': xk[0]**,** 'Y': xk[1]**,** 'F': f(xk)**,** 'Lambda': alpha\_k**,** 'DX': xkp1[0] - xk[0]**,** 'DY': xkp1[1] - xk[1]**,** 'DF': f(xkp1) - f(xk)**,** 'Gradient': current\_gradient**,** 'Hk': Hk}]  
 df = df.append(current\_iter**,** ignore\_index=True)  
  
 plt.plot([xk[0]**,** xkp1[0]]**,** [xk[1]**,** xkp1[1]]**,** '-r')  
  
 sk = xkp1 - xk  
 xk = xkp1  
  
 gfkp1 = fprime(xkp1)  
 yk = gfkp1 - current\_gradient  
 current\_gradient = gfkp1  
 k += 1  
  
 temp = 1 / np.dot(np.dot(yk[np.newaxis**,** :]**,** Hk)**,** yk)  
  
 if k % 1 == 0:  
 Hk = np.eye(N**,** dtype=int)  
 else:  
 Hk = Hk + ((np.dot(sk - np.dot(Hk**,** yk)**,** np.dot(yk**,** Hk).transpose())) \* temp)  
  
 if ln.norm(current\_gradient) < epsi:  
 df.to\_excel("Report\_Third\_Pearson.xlsx")  
 plt.show()  
 return xk**,** k**,** iter\_counter\_func  
  
 df.to\_excel("Report\_Third\_Pearson.xlsx")  
 plt.show()  
 return xk**,** k**,** iter\_counter\_func  
  
  
xk**,** counter**,** iter\_counter = fast\_gradient\_method(function**,** gradient**,** np.array([10**,** 10])**,** epsi=1e-7)  
result**,** k**,** third\_Pearson\_iter = third\_Pearson(function**,** gradient**,** np.array([0**,** 0])**,** epsi=1e-7)  
result1**,** k2**,** second\_Pearson\_iter = second\_Pearson(function**,** gradient**,** np.array([10**,** 10])**,** epsi=1e-7)  
print("Stop")

**Файл methods.py**

from math import sqrt  
  
  
def golden\_ratio(function**,** a**,** b**,** eps):  
 SQRT5 = sqrt(5)  
  
 x1 = a + (3 - SQRT5) / 2 \* (b - a)  
 x2 = a + (SQRT5 - 1) / 2 \* (b - a)  
  
 f1 = function(x1)  
 f2 = function(x2)  
 i: int = 0  
  
 while abs(a - b) > eps:  
 if f1 < f2:  
 b = x2  
 x2 = x1  
 x1 = a + (3 - SQRT5) / 2 \* (b - a)  
 f2 = f1  
 f1 = function(x1)  
 else:  
 a = x1  
 x1 = x2  
 x2 = a + (SQRT5 - 1) / 2 \* (b - a)  
 f1 = f2  
 f2 = function(x2)  
  
 i += 1  
 return (a + b) / 2**,** i  
  
  
def golden\_ratio\_2(function**,** a**,** b**,** eps):  
 SQRT5 = sqrt(5)  
 x1 = a + (3 - SQRT5) / 2 \* (b - a)  
 x2 = a + (SQRT5 - 1) / 2 \* (b - a)  
  
 f1 = function(x1)  
 f2 = function(x2)  
 iter\_counter: int = 0  
  
 while abs(a - b) > eps:  
 if f1 < f2:  
 b = x2  
 x2 = x1  
 x1 = a + (3 - SQRT5) / 2 \* (b - a)  
 f2 = f1  
 f1 = function(x1)  
 else:  
 a = x1  
 x1 = x2  
 x2 = a + (SQRT5 - 1) / 2 \* (b - a)  
 f1 = f2  
 f2 = function(x2)  
  
 iter\_counter += 1  
 return (a + b) / 2**,** iter\_counter