МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**ОТЧЁТ**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

(подпись)

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность Методы оптимизации

Руководитель А.С. Чёрная

(подпись)

Краснодар

2024

**Постановка задачи**

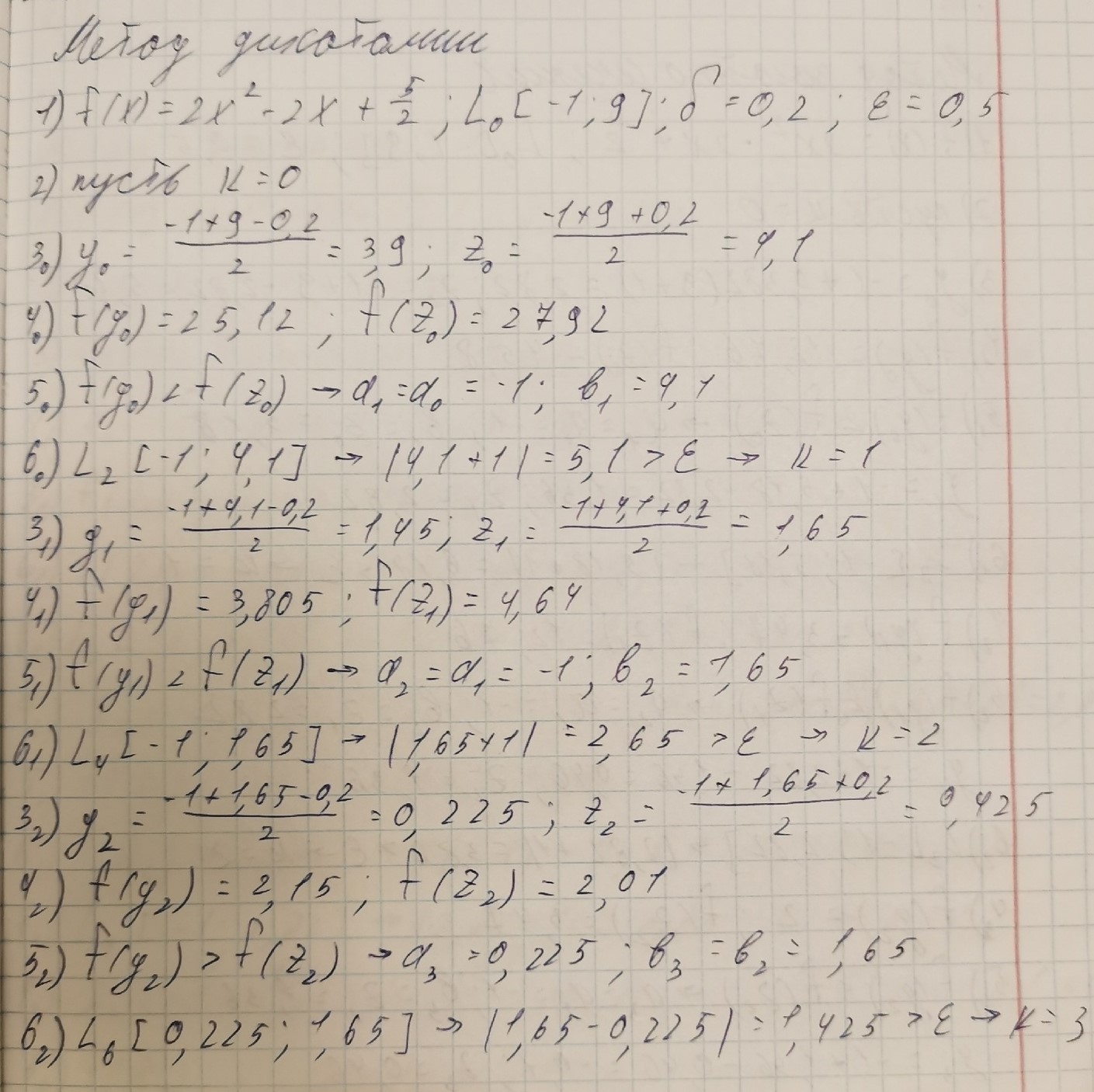
Требуется для функции  и начального интервала неопределённости  с заданной точностью  найти точку минимума и значение в этой точке несколькими методами (методом дихотомии, методом золотого сечения, методом Фибоначчи). Также нужно провести сравнение методов оптимизации.

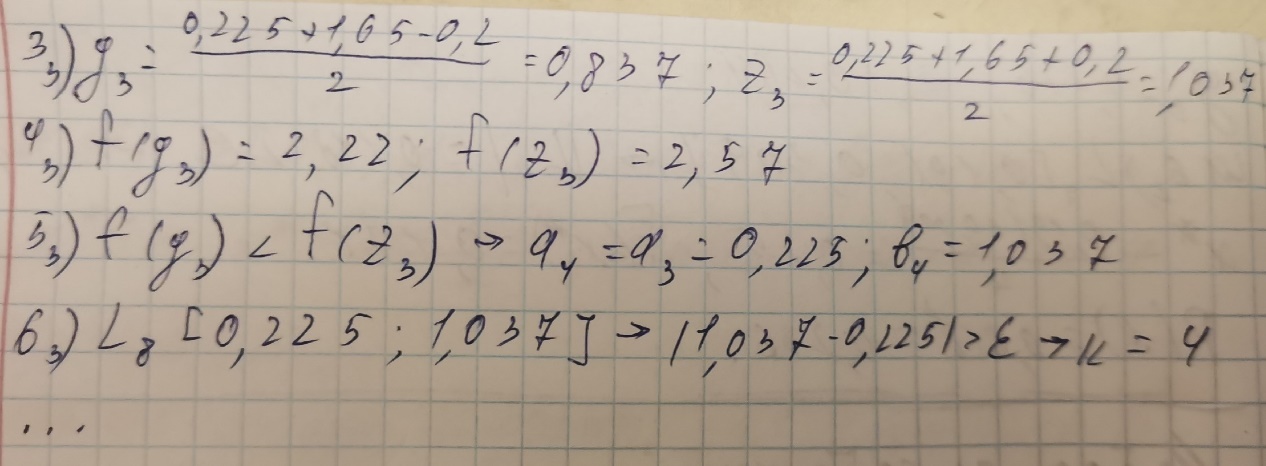
**Метод дихотомии**

**Алгоритм метода**

Метод дихотомии – это итеративный метод поиска минимума функции, который использует деление отрезка пополам и последующее сужение интервала до заданной точности. Каждый шаг включает оценку значения функции в двух точках, выбор нового интервала наименьшей ширины, содержащего минимум, и повторение процесса до достижения требуемой точности.

**Пример работы алгоритма**





**Код программы**

import math

import matplotlib.pyplot as plt

import numpy as np

def function(*x*):

    """Функция, которая считает значение основной функции из условия"""

    res = []

    for i in range(len(*x*)):

        res.append((2 \* math.pow(*x*[i], 2)) - (2 \* *x*[i]) + (5 / 2))

    return res

def y\_z(*a*, *b*):

    """Функция, которая находит новые значения y и z"""

    y\_new = (*a* + *b* - betta) / 2

    z\_new = (*a* + *b* + betta) / 2

    return y\_new, z\_new

def fy\_fz(*y*, *z*):

    """Функция, которая считает значение основной функции в точках y и z"""

    f\_y = function([*y*])

    f\_z = function([*z*])

    if f\_y <= f\_z:

        l[1] = *z*

    else:

        l[0] = *y*

    return 2 \* (k + 1)

def accuracy(*a*, *b*):

    """Функция, которая проверяет точность"""

    f = False

    if math.fabs(*b* - *a*) < epsilon:

        f = True

    return f

l = [-1, 9]

betta = 0.2

epsilon = 0.5

k = 0

n = 0

print()

print(f"""Входные данные:

Интервал неопределённости: {l}

Шаг betta: {betta}

Точность epsilon: {epsilon}

Сама функция: 2x^2 - 2x + 5/2

Начальное k: {k}

      """)

while not accuracy(l[0], l[1]):

    y, z = y\_z(l[0], l[1])

    n = fy\_fz(y, z)

    accuracy(l[0], l[1])

    k += 1

print(f"""Выходные данные:

Исходная функция: 2x^2 - 2x + 5/2

Точка минимума: {(l[1] + l[0]) / 2}

Конечный интервал неопределённости: {l}

Значение функции в точке минимума: {function([(l[1] + l[0]) / 2])}

Индекс конечного интервала неопределённости: {n}

Сходимость: {1 / (math.pow(2, n))}

Конечное к: {k}

Точность (по отношению к epsilon): {math.fabs(l[1] - l[0])}

      """)

x\_range = np.linspace(-10, 10 ,1000)

plt.plot(x\_range, function(x\_range), *label*='f(x)')

plt.scatter((l[1] + l[0]) / 2, function([(l[1] + l[0]) / 2]), *color*='blue', *s*=20, *label*='Минимум')

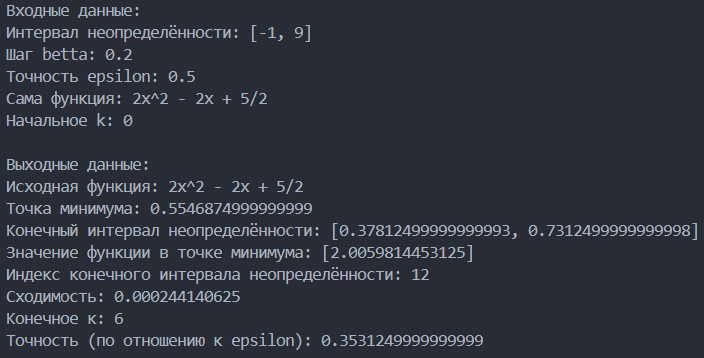
plt.xlabel('X')

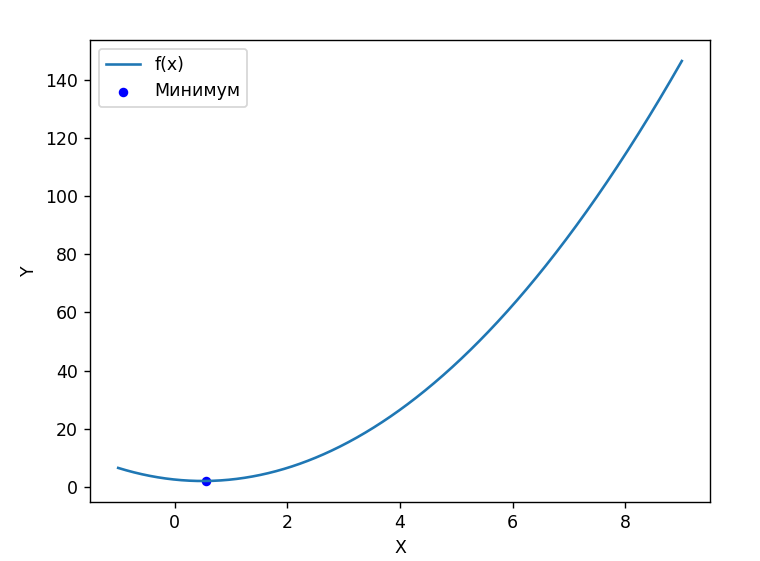
plt.ylabel('Y')

plt.legend()

plt.show()

**Вывод программы**



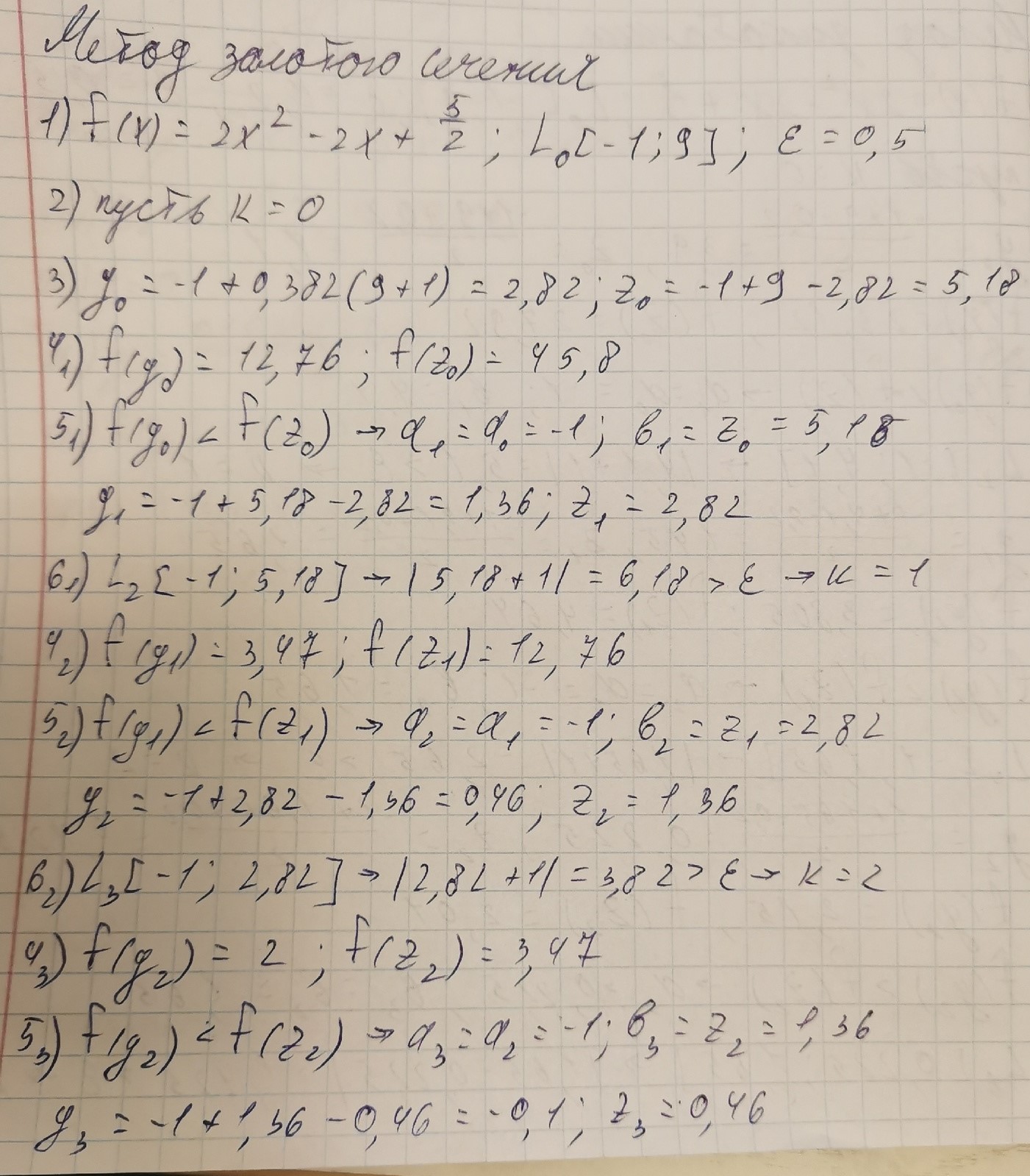


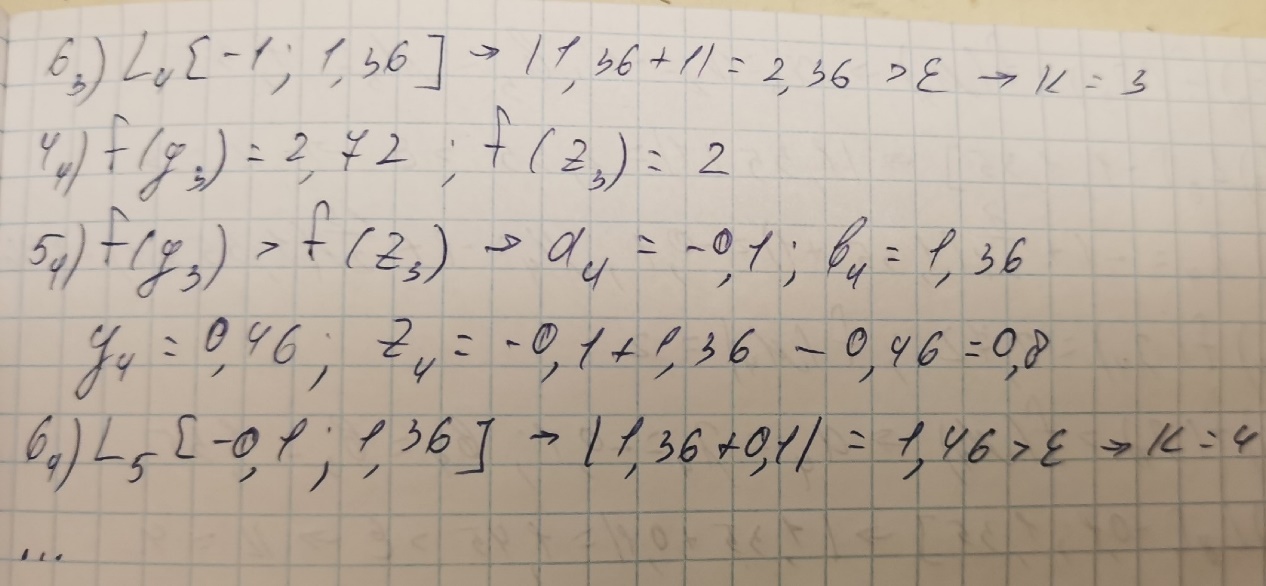
**Метод золотого сечения**

**Алгоритм метода**

Метод золотого сечения заключается в нахождении минимума функции путем деления отрезка на две части в "золотом" соотношении (примерно 0.618) и выбора подотрезка с меньшим значением функции в качестве нового отрезка для деления. Процесс повторяется до достижения заданной точности.

**Пример работы алгоритма**





**Код программы**

import math as mt

import matplotlib.pyplot as plt

import numpy as np

def function(*x*):

    """Функция, которая считает значение основной функции"""

    res = []

    for i in range(len(*x*)):

        res.append((2 \* mt.pow(*x*[i], 2)) - (2 \* *x*[i]) + (5 / 2))

    return res

def y\_z():

    """Функция, которая находит новые значения y и z"""

    y = l[0] + num\_1 \* (l[1] - l[0])

    z = l[0] + l[1] - y

    return y, z

def fy\_fz(*y*, *z*, *count\_n\_for*, *n*):

    """Функция, которая считает значение функции в точках y и z"""

    fy = function([*y*])

    fz = function([*z*])

    if fy <= fz:

        l[1] = *z*

        y\_new = l[0] + l[1] - *y*

        z\_new = *y*

    else:

        l[0] = *y*

        y\_new = *z*

        z\_new = l[0] + l[1] - *z*

    if *count\_n\_for* == 0:

*n* = 2 \* (k + 1)

*count\_n\_for* += 1

    else:

*n* += 1

*count\_n\_for* += 1

    return y\_new, z\_new, *count\_n\_for*, *n*

def accuracy():

    """Функция, которая проверяет точность"""

    f = False

    if mt.fabs(l[1] - l[0]) < epsilon:

        f = True

    return f

num\_1 = (3 - mt.sqrt(5)) / 2

num\_2 = 1 - num\_1

l = [-1, 9]

epsilon = 0.5

k = 0

count\_n\_for\_l = 0

n = 0

print()

print(f"""

Входные данные:

Интервал неопределённости l = {l}

Точность epsilon = {epsilon}

Сама функция: 2x^2 - 2x + 5/2

Начальное k: {k}

""")

y, z = y\_z()

while not accuracy():

    y, z, count\_n\_for\_l, n = fy\_fz(y, z, count\_n\_for\_l, n)

    k += 1

print()

print(f"""

Выходные данные:

Исходная функция: 2x^2 - 2x + 5/2

Точка min = {(l[1] + l[0]) / 2}

Значение функции в точке min = {function([(l[1] + l[0]) / 2])}

Интервал, в котором находится точка min: {l}

Индекс конечного интервала n = {n}

Сходимость = {mt.pow(num\_2, n - 1)}

Конечное k = {k}

Точность (по отношению к epsilon) = {mt.fabs(l[1] - l[0])}

""")

x\_range = np.linspace(-1, 9 ,1000)

plt.plot(x\_range, function(x\_range), *label*='f(x)')

plt.scatter((l[1] + l[0]) / 2, function([(l[1] + l[0]) / 2]), *color*='blue', *s*=20, *label*='Минимум')

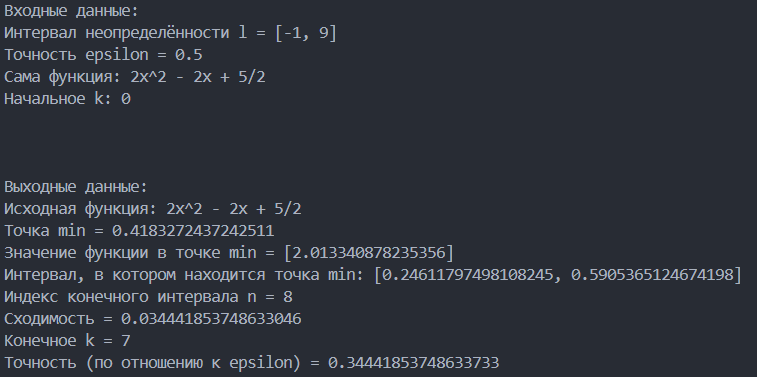
plt.xlabel('X')

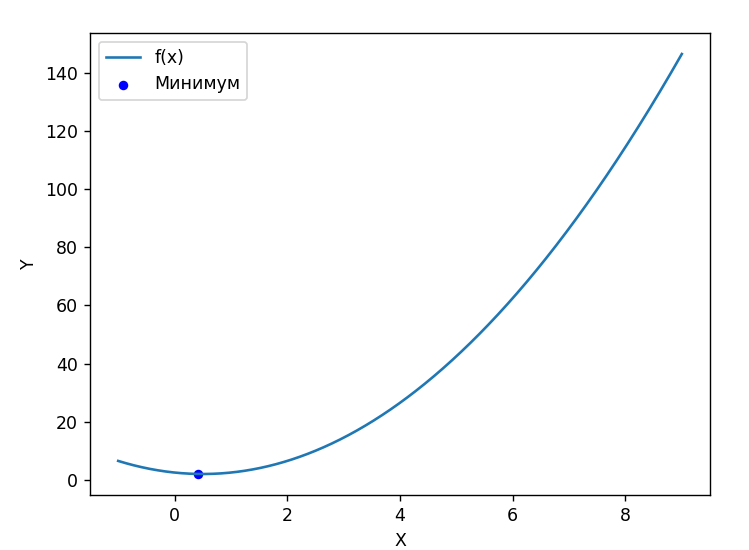
plt.ylabel('Y')

plt.legend()

plt.show()

**Вывод программы**

****

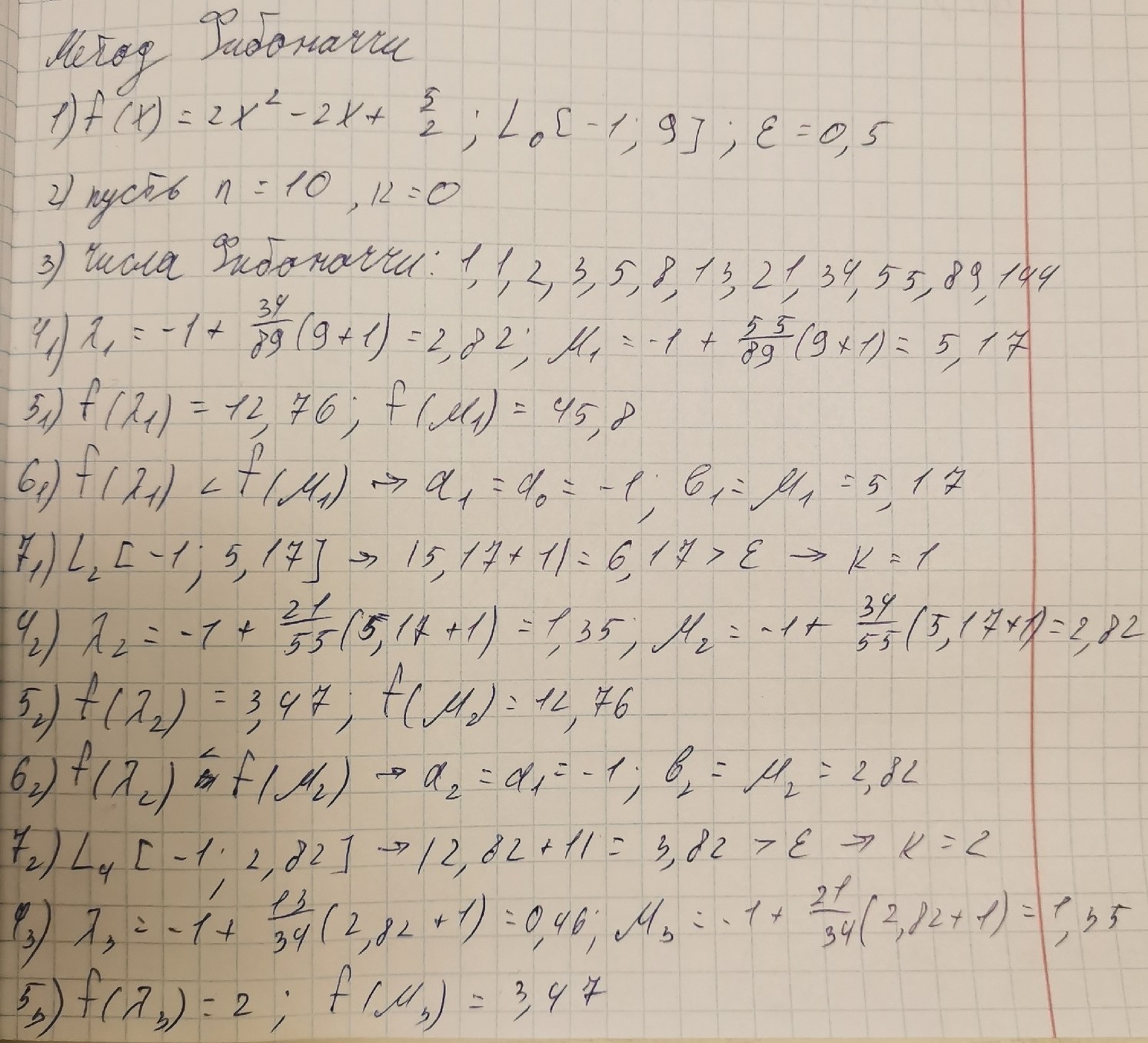
****

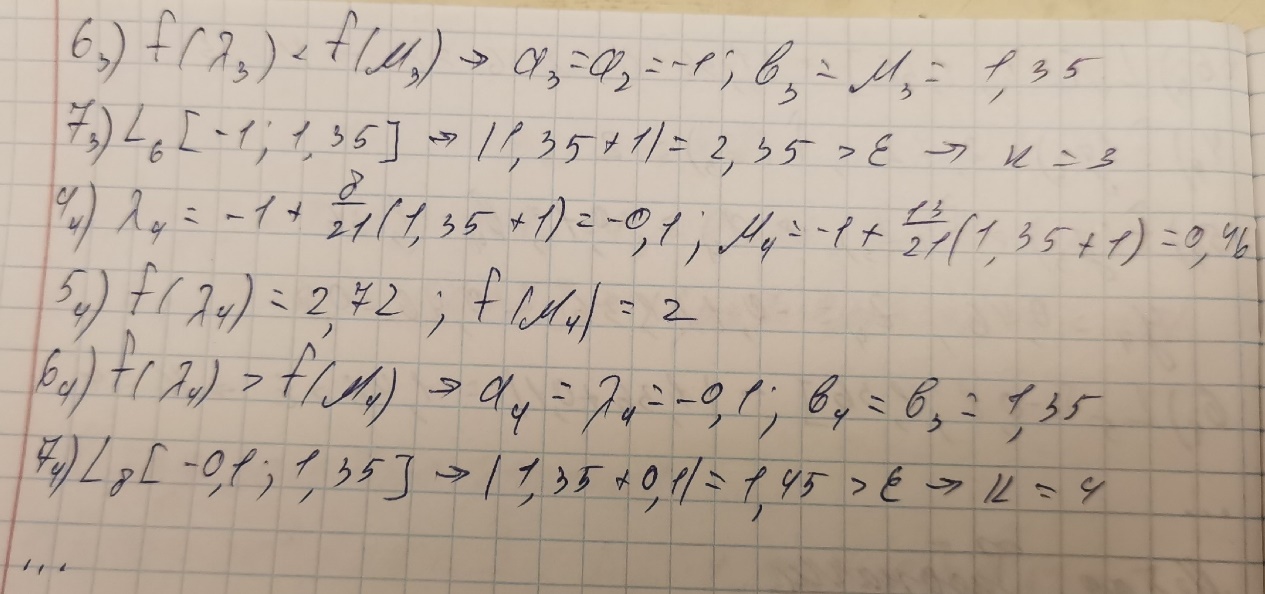
**Метод Фибоначчи**

**Алгоритм метода**

Метод Фибоначчи для поиска минимума заключается в уменьшении интервала, в котором предполагается нахождение минимума, путем использования последовательности чисел Фибоначчи. Каждый раз интервал делится на две части, и значения функции оцениваются только в определенных точках, определяемых числами Фибоначчи.

**Пример работы алгоритма**

****



**Код программы**

import math as mt

import matplotlib.pyplot as plt

import numpy as np

def base\_func(*x\_list*):

    """

    Базовая функция из условия

    Args: x\_list - список входных значений

    Return: список результатов функции

    """

    result = []

    for i in range(len(*x\_list*)):

        result.append((2 \* mt.pow(*x\_list*[i], 2)) - (2 \* *x\_list*[i]) + (5 / 2))

    return result

def fibonachchi\_func(*quantity*):

    """

    Функция, которая считает числа Фибоначчи

    Args: quantity - количество чисел, которое нужно найти

    Return: список чисел Фибоначчи

    """

    if *quantity* == 1:

        return [1]

    elif *quantity* == 2:

        return [1, 1]

    else:

        result = [1, 1]

        for i in range(*quantity*):

            if i < 2:

                pass

            else:

                result.append(result[i - 1] + result[i - 2])

        return result

def calculate\_lambda\_i\_mu\_i(*l*, *fib\_list*, *i*):

    """

    Функция, которая рассчитывает значения lambda\_i и mu\_i

    Args: l - интервал неопределённости; fib\_list - список чисел Фибоначчи; i - индекс итерации

    Return: lambda\_i и mu\_i

    """

    n = len(*fib\_list*)

    lambda\_i = *l*[0] + ((*fib\_list*[(n - 1) - *i* - 1] / *fib\_list*[(n - 1) - *i* + 1]) \* (*l*[1] - *l*[0]))

    mu\_i = *l*[0] + ((*fib\_list*[(n - 1) - *i*] / *fib\_list*[(n - 1) - *i* + 1]) \* (*l*[1] - *l*[0]))

    return lambda\_i, mu\_i

def calculate\_new\_l(*lambda\_i*, *mu\_i*, *l*):

    """

    Функция, которая переопределяет интервал неопределённости

    Args: lambda\_i - текущая lambda\_i; mu\_i - текущее mu\_i; l - интервал неопределённости

    Return: новый интервал неопределённости

    """

    new\_l = [0, 0]

    f\_lambda\_i = base\_func([*lambda\_i*])[0]

    f\_mu\_i = base\_func([*mu\_i*])[0]

    if f\_lambda\_i > f\_mu\_i:

        new\_l[0] = *lambda\_i*

        new\_l[1] = *l*[1]

    else:

        new\_l[0] = *l*[0]

        new\_l[1] - *mu\_i*

    return new\_l

def calculate\_accuracy(*l*, *epsilon*):

    """

    Функция, которая рассчитывает точность текущего интервала неопределённости

    Args: l - интервал неопределённости; epsilon - точность

    Return: True (точность достигнута) / False (точность не достигнута)

    """

    f = False

    if mt.fabs(*l*[1] - *l*[0]) < *epsilon*:

        f = True

    return f

l = [-1, 9]  *#интервал неопределённости*

epsilon = 0.5  *#точность*

i = 3  *#счётчик итераций*

n = 10  *#кол-во чисел Фибоначчи*

alpha = 0.1  *#константа*

print(f"""

Входные данные:

Функция: 2x^2 - 2x + 5/2

Интервал неопределённости: {l}

Точность: {epsilon}

Кол-во чисел Фибоначчи: {n}

""")

fib\_list = fibonachchi\_func(n)

while not calculate\_accuracy(l, epsilon):

    lambda\_i, mu\_i = calculate\_lambda\_i\_mu\_i(l, fib\_list, i)

    l = calculate\_new\_l(lambda\_i, mu\_i, l)

    i += 1

print(f"""

Выходные значения:

Функция: 2x^2 - 2x + 5/2

Интервал неопределённости: {l}

Точка минимума: {(l[1] + l[0]) / 2}

Значение функции в точке минимума: {base\_func([(l[1] + l[0]) / 2])[0]}

Конечное k: {i}

Конечный индекс интервала неопределённости: {i\*2 + 1}

Точность (по отношению к epsilon): {mt.fabs(l[1] - l[0])}

Сходимость: {1 / fib\_list[len(fib\_list) - 1]}

""")

x\_range = np.linspace(-1, 9, 1000)

plt.plot(x\_range, base\_func(x\_range), *label*="f(x)")

plt.scatter((l[1] - l[0]) / 2, base\_func([(l[1] - l[0]) / 2]), *color*="blue", *s*=20, *label*="min")

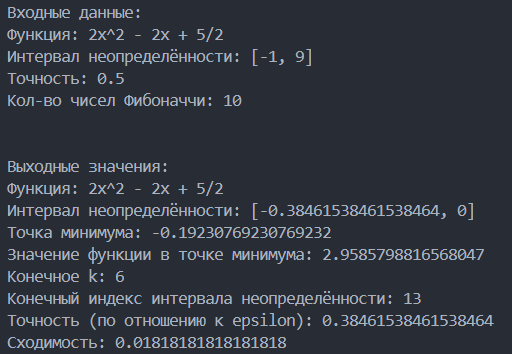
plt.xlabel("X")

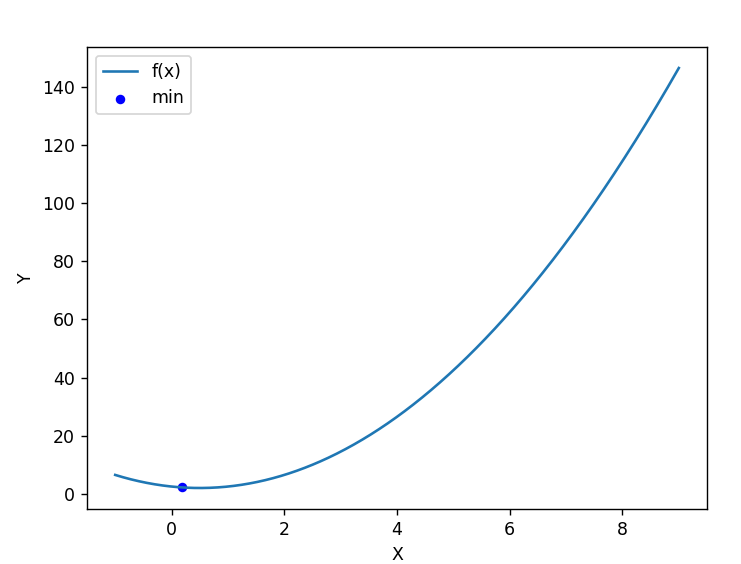
plt.ylabel("Y")

plt.legend()

plt.show()

**Вывод программы**





**Сравнение методов**

Исходя из результатов программ можно сделать следующие выводы по методам:  
Метод дихотомии: имеет самую простую реализацию; является относительно медленным методом (в сравнении с методом золотого сечения и методом Фибоначчи) и не всегда эффективен, если функция не является унимодальной (имеет несколько минимумов).

Метод золотого сечения: более эффективен, чем метод дихотомии, из-за более удачного выбора точек, однако его реализация сложнее, чем тот-же метод дихотомии.

Метод Фибоначчи: является ещё более эффективным методом, чем метод золотого сечения, из-за работы с числами Фибоначчи; требует больше вычислительных ресурсов из-за расчётов чисел Фибоначчи и в целом имеет более сложную структуру.