МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ» (ФГБОУ ВО «КубГУ»)**

**Факультет математики и компьютерных наук**

 **Кафедра математических и компьютерных методов**

**КУРСОВАЯ РАБОТА**

**ПРОГРАММИРОВАНИЕ МЕТОДОВ ОДНОМЕРНОЙ
ОПТИМИЗАЦИИ СРЕДСТВАМИ PYTHON**

Работу выполнила                                                                           А. И. Новицкая

(подпись)

Направление подготовки 02.03.01 Математика и компьютерные науки курс 2

Научный руководитель

Кандидат педагогических наук, доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. Н. Царева

 (подпись, дата)

Нормоконтролер

преподаватель                                                                                    А. А. Лахтина

(подпись, дата)

Краснодар

2024

**Содержание**

[Введение 3](#_Toc166501531)

[1 Методы одномерной оптимизации 4](#_Toc166501532)

[1.1 Постановка задачи одномерной оптимизации 4](#_Toc166501533)

[1.2 Метод деления пополам 6](#_Toc166501534)

[1.3 Метод дихотомии 7](#_Toc166501535)

[1.4 Метод золотого сечения 8](#_Toc166501536)

[2 Алгоритм программы 10](#_Toc166501537)

[2.1 Ввод и вывод 10](#_Toc166501538)

[2.2 Метод деления пополам 12](#_Toc166501539)

[2.3 Метод дихотомии 13](#_Toc166501540)

[2.4 Метод золотого сечения 14](#_Toc166501541)

[3 Эффективность методов одномерной оптимизации 15](#_Toc166501542)

[3.1 Способы сравнения 15](#_Toc166501543)

[3.2 Результат работы программы 18](#_Toc166501544)

[3.3 Сравнение методов одномерной оптимизации 24](#_Toc166501545)

[Заключение 28](#_Toc166501546)

[Список использованных источников 30](#_Toc166501547)

Введение

В курсовой работе выбрано 3 вида итерационных методов одномерной оптимизации:

1) метод деления пополам;

2) метод дихотомии;

3) метод золотого сечения.

Основная задача курсовой работы – разработать программу с применением выбранных методов оптимизации на языке Python представить к сравнению по эффективности и быстродействию.

В ходе работы написаны программы, позволяющие вычислить единственный минимум или максимум унимодальных функций с помощью метода деления пополам, метода дихотомии и метода золотого сечения. Интервал поиска экстремума и точность вычислений пользователь вводит с клавиатуры.

Рассмотрена эффективность работы разных методов по таким показателям, как время, количество итераций, количество вычислений функций и асимптотической сложности.

1. Методы одномерной оптимизации
	1. Постановка задачи одномерной оптимизации

Оптимизация – это поиск наиболее подходящего (оптимального) решения для некоторой системы при наличии множества альтернативных решений. Если систему можно представить в виде функции, то задача оптимизации заключается в нахождении экстремума функции. В таком случае для решения строят математическую модель задачи оптимизации.

Таким образом, задача одномерной оптимизации состоит в минимизации или же максимизации функции *f(x)* одномерного аргумента *x*∈[*а, b*].

Необходимо ввести следующие определения:

Монотонность функции. Функция является монотонной на интервале, если для любых *x1* и *x2* из этого интервала, таких, что выполняется неравенство *f(x1) <= f(x2)*, если функция монотонно возрастающая или *f(x1) <= f(x2)*, если функция, монотонно убывающая [1].

 Унимодальность. Функция является унимодальной на отрезке, если она монотонна по обе стороны от единственной на отрезке точки *х0*, то есть при поиске минимума функция в полуинтервале [*а, х0*) убывает, а в полуинтервале (*х0,b*] возрастает, а при поиске максимума слева возрастает, а справа убывает. Примеры графиков унимодальных функций приведены на рис. 1.



Рисунок 1 – Примеры унимодальных функций с точкой минимума x0

Точка *х0* может быть внутренней точкой отрезка [*а, b*] или совпадать с одним из его концов. Унимодальная функция не обязательно непрерывна на отрезке [*а, b*].

Глобальный минимум. Функция, определённая на множестве D достигает глобального минимума в точке *x0*, если для всех *x* принадлежащих D будет верно неравенство *x> x0*.

Локальный минимум. Функция, определённая на множестве D имеет локальный минимум в точке *x0*, если существует такая *ε*-окрестность точки *x0*, что для всех *x* из этой *ε*-окрестности будет верно неравенство *x*> *x0*.

Для решения задач одномерной оптимизации разработаны специальные методы. Они подразделяются на методы локальной и глобальной оптимизации. Методы локальной оптимизации предназначены для поиска единственного точечного локального экстремума. К методам глобальной оптимизации относятся методы, предназначенные для решения многоэкстремальных задач. Основным и наиболее часто используемым методом является метод глобального поиска [3].

Существуют два подхода для решения задачи.

Первый метод: аналитический метод, использующий необходимое условие экстремума; Алгоритм аналитического метода представляет следующие этапы:

1) получение математического выражения для первой производной;

2) анализ уравнения;

3) нахождение аналитического решения *x0* или корней уравнения, называемых стационарными точками;

4) анализ полученных стационарных точек с целью выделения из них точек минимума, максимума, перегиба;

5) вычисление значения критерия в точках локального минимума, максимума и в граничных точках (если множество D закрытое). Сравнение значений критерия и выбор наибольшего и наименьшего значений.

Для множества оптимизационных задач нахождения аналитического решения уравнения вызывает определённые трудности. В таких случаях используют итерационные (приближенные) методы поиска решения *x\** оптимизационной задачи на открытом множестве D = {*x | x− < x < x+*} [4].

Итерация – многократное выполнение одного и того же действия. Итерационные методы решения оптимизационной задачи заключаются в многократном применении одной и той же математической операции к *xk* и получения последовательности точек *x1, x2,…, xk* (где *k* – номер итерации *k* = 0,1,2,….), сходящейся к точному решению *x0*.

* 1. Метод деления пополам

Для всех представленных ниже методов необходимо, чтобы для исследуемой части функции выполнялось условие унимодальности.

Пусть дан интервал [*а, b*]. Метод заключается в том, что исходный интервал [*а, b*] делится средней точкой *x0* = (*b + a*) / 2 на два подинтервала [*а, x0*] и [*x0, b*], в одном из которых лежит точка минимума .

Далее выбираются 2 точки, которые делят отрезки [*а, x0*] и [*x0, b*] пополам. Пусть точки будут названы *х1* и *х2* соответственно.

В точках *х1, x0* и *х2* вычисляются значения функции. Эти значения сравниваются (рис. 2).



Рисунок 2 - Иллюстрация метода деления пополам

Если *f(x1) < f(x0)*, то точка x0 становится точкой *b*, диапазон [*x0, b*] отсекается. Иначе если *f(x0) >f(x2)*, то точка x0 становится точкой a, диапазон [*a, x0*] отсекается.

Если это условие не выполняется и *f(x0)* меньше *f(x1)* и меньше *f(x2*), то точка *х1* становится точкой *a*, точка *х2* становится точкой *b*, а диапазоны [*a, x1*] и [*x2, b*] отсекаются.

Данный алгоритм повторяется до тех пор, пока полученный отрезок [*a, b*] больше заданной точности *e*. По окончании алгоритма точка *с* становится искомой точкой минимума [5].

* 1. Метод дихотомии

Метод похож на метод деления пополам. В отличии от прошлого способа точки *х1* и *х2* выбираются по-другому.

Пусть дан интервал [*а, b*]. Метод заключается в том, что исходный интервал [а, b] делится средней точкой *x0 = (b + a*) / 2 на два подинтервала [*а, x0*] и [*x0, b*] в одном из которых лежит точка минимума.

Для выбора подинтервала, для хорошо дифференцируемой функции анализируют знак производной *f* '(*x0*). Для этого можно выбрать 2 точки *x1* и *x2*, отдалённые с разных сторон от точки *c* на определённое расстояние *d. d* должно варьироваться в диапазоне (0, (*b - a*) / 2).

Сравниваются значение функции в точках *x1* и *x2*. Если *f(x1) >= f(x2)*, то точка *x1* становится точкой *a*, диапазон [*a, x1*] отсекается, если *f(x1) < f(x2)*, то точка *x2* становится точкой *b*, диапазон [*x2, b*] отсекается. Вычисление точек проиллюстрировано на рисунках 3.



Рисунок 3 - Иллюстрация метода дихотомии

Процедура продолжается до тех пор, пока не будет достигнута заданная точность e [3].

* 1. Метод золотого сечения

В основе метода золотого сечения лежит принцип деления отрезка в пропорциях золотого сечения.

Отрезок [*а, b*] делится точкой *x2* на два внутренних отрезка так, что
[*a, b*] / [*a, x2*] = [*a, x2*] / [*x2, b*].

Вторая точка *x1* ставится симметрично *x2* относительно центра отрезка [*а, b*]. Сравниваются значение функции в точках *x1* и *x2*(рис. 4).



Рисунок 4 - Иллюстрация метода золотого сечения

Если *f(x1) >= f(x2)*, то точка x1 становится точкой a, диапазон [*a, x1*] отсекается, а точка x2 становится точкой x1. Новая точка x2 ставится симметрично x1 относительно центра нового отрезка [*a, b*].

Если *f(x1) < f(x2)*, то точка x2 становится точкой b, диапазон [*x2, b*] отсекается, а точка x1 становится точкой *x2*. Новая точка *x1* ставится симметрично *x2* относительно центра нового отрезка [*a, b*]. На рисунке 4.3 представлен выбор нового диапазона и контрольной точки *x2*.

На следующих итерациях в силу показанного выше свойства золотого сечения уже надо искать всего одну новую точку.

Процедура продолжается до тех пор, пока не будет достигнута заданная точность *e* [3].

Иллюстрации алгоритма вычисления экстремума составлены средствами Freeform.

1. Алгоритм программы
	1. Ввод и вывод

Поиск экстремума был реализован на языке Python. Для подсчёта времени, за которое алгоритм будет исполнен, использовалась библиотека time. Методы оптимизации представлены в виде функций. Для проверки вводных данных от пользователя был написан вспомогательный модуль mymodule.

from time import time

from mymodule import inputing\_float, diap\_float

Пользователь вводит начальную точку *a* и конечную точку *b*. Так задаётся диапазон поиска. Далее выбор: оставить текущую точность вычислений или вписать свою. После этого пользователь может выбрать, минимум или максимум он хочет посчитать.

Функция ввода данных от пользователя f выглядит следующим образом:

# Выбранный пример функции f(x)

def f(x):

 y = x\*\*2 - 12\*x +5

 #y = math.atan(x)\*6 + 12

 #y = (-(abs(4\*x + 50))+300)/5

 return y

Данный алгоритм вычисляет значение функции *y = x2 - 12x +5.* Вместо неё можно вписать любую другую функцию. На вход функция принимает аргумент *x –* точку, значение в которой необходимо вычислить. Результат работы функции – значение *y* в этой точке.

Функция ввода значений inputting не имеет аргументов. Во время выполнения этой функции пользователь должен указать диапазон поиска, точность вычислений, а также указать, минимум вводится или максимум. Для проверки ввода значений нужного формата использованы функции модуля mymodule.py.

def inputting():

 a,b = diap\_float ('Укажите начальную и конечную точки через пробел: ')

 print('По умолчанию точность вычислений = 0.000000001\nХотите изменить?\n 1. да\n 2. нет')

 while True:

 ans = input('Ответ: ')

 if ans == '1':

 while True:

 e = inputting\_float ('Укажите точность вычислений: ')

 if e > 0:

 break

 else:

 print('Число должно быть >0')

 elif ans == '2':

 e = 0.000000001

 else:

 print('Укажите нужную цифру!')

 continue

 break

 print('Необходимо посчитать минимум или максимум?\n 1. мин\n 2. макс')

 while True:

 ans = input('Ответ: ')

 if ans == '1':

 targ = 0

 elif ans == '2':

 targ = 1

 else:

 print('Укажите нужную цифру!')

 continue

 break

 return a, b, e, targ

После исполнения кода программы по поиску оптимального решения на консоль выводятся следующие значения для каждого из методов:

1) method: Название метода;

2) x: Точка минимума (максимума);

3) y: Значение функции в точке минимума (максимума);

4) kol: Количество выполненных итераций;

5) fnt: Время поиска.

Вывод значений реализован с помощью функции outputting.

def outputting( a, b, e, targ):

 #method, x, y, fnt, kol = bisection (a, b, e, targ)

 un = 1

 #un = unimodal(a, b, e, targ, x, y)

 if un:

 ans = 'минимума' if targ == 0 else 'максимума'

 print(f'Функция < y = x\*\*2 - 12\*x +5 > в диапазоне от {a} до {b} с точностью {e}')

 lst = [bisection(a, b, e, targ), dichotomy(a, b, e, targ), golden\_ratio (a, b, e, targ)]

 for i in lst:

 method, x, y, fnt, kol = i

 print(f'\n\nМетод:\t{method}\nТочка {ans}:\t{x}\nЗнач. в точке:\t{y}\nКоличество шагов: {kol}\nВремя работы алгоритма: \t{fnt}')

 else:

 print('Условие унимодальности не выполнено!')

#код

a, b, e , targ = inputting()

outputting(a, b, e, targ)

Функция имеет аргументы, которые пользователь указал при вводе, а именно границы диапазона a и b, точность вычислений e и флаг, указывающий, будут искать минимум или максимум.

Каждый метод, представленный отдельными функциями в программе, принимает в качестве аргументов те же самые значения. Функции являются рекурсивными [2].

* 1. Метод деления пополам

# Метод деления пополам

def bisection(a, b, e, targ):

 stt = time()

 kol = 0

 while (b-a > e):

 l = b - a

 x0 = (a + b) / 2

 x1, x2 = a + l/4, b - l/4

 fx0, fx1, fx2 = f(x0), f(x1), f(x2)

 if not targ:

 if (fx0 > fx1):

 b = x0

 elif (fx0 > fx2):

 a = x0

 else:

 a, b = x1, x2

 else:

 if (fx0 < fx1):

 b = x0

 elif (fx0 < fx2):

 a = x0

 else:

 a, b = x1, x2

 kol += 1

 fnt = time() - stt

 x=round(x0,4)

 y=round(f(x),4)

 method = 'деление пополам'

 return method, x, y, fnt, kol

* 1. Метод дихотомии

# Метод дихотомии

def dichotomy(a, b, e, targ):

 stt = time()

 d = 0.000000001

 kol = 0

 while (b-a > e) and ((b-a)/2 > d):

 x1, x2 = (a+b-d)/2, (a+b+d)/2

 fx1, fx2 = f(x1), f(x2)

 if not targ:

 if (fx1 >= fx2):

 a = x1

 else:

 b = x2

 else:

 if (fx1 >= fx2):

 b = x2

 else:

 a = x1

 kol += 1

 fnt = time() - stt

 x=round((a+b)/2,4)

 y=round(f(x),4)

 method = 'дихотомия'

 return method, x, y, fnt, kol

* 1. Метод золотого сечения

# Метод золотого сечения

def golden\_ratio( a, b, e, targ):

 stt = time()

 x1 = a + ((3-(5\*\*0.5))/2)\*(b-a)

 x2 = a + (((5\*\*0.5)-1)/2)\*(b-a)

 fx1, fx2 = f(x1), f(x2)

 kol = 0

 while (b-a > e):

 if not targ:

 if fx1 >= fx2:

 a = x1

 x1 = x2

 fx1 = fx2

 x2 = a + (((5\*\*0.5)-1)/2)\*(b-a)

 fx2 = f(x2)

 else:

 b = x2

 x2 = x1

 fx2 = fx1

 x1 = a + ((3-(5\*\*0.5))/2)\*(b-a)

 fx1 = f(x1)

 else:

 if fx1 < fx2:

 a = x1

 x1 = x2

 fx1 = fx2

 x2 = a + (((5\*\*0.5)-1)/2)\*(b-a)

 fx2 = f(x2)

 else:

 b = x2

 x2 = x1

 fx2 = fx1

 x1 = a + ((3-(5\*\*0.5))/2)\*(b-a)

 fx1 = f(x1)

 kol += 1

 fnt = time() - stt

 x=round((a+b)/2,4)

 y=round(f(x),4)

 method = 'золотое сечение'

 return method, x, y, fnt, kol

1. Эффективность методов одномерной оптимизации
	1. Способы сравнения

Быстродействие метода оптимизации играет важную роль в обеспечении успешного и эффективного решения задач оптимизации. Существуют разные способы сравнить быстродействие методов одномерной оптимизации, они представлены ниже.

1) время работы: можно измерить время, затраченное на выполнение каждого метода для оптимизации заданной функции. Этот метод может быть не всегда достаточно точным или полным, так как он зависит от аппаратного обеспечения, на котором она выполняется. Для задач, представленных в курсовой работе, каждый метод вычисляется очень быстро, из-за данного фактора сравнение невозможно.

2) количество итераций: можно сравнить количество итераций, необходимых для сходимости каждого метода к оптимальному решению. Использование количества итераций как способа сравнения быстродействия программы может быть более информативным и объективным, чем простое сравнение времени работы. Данный метод используется для сравнения трёх представленных методов оптимизации.

3) скорость сходимости: определяет, насколько быстро он приближается к оптимальному решению или к заданной точности. Это важный параметр, который позволяет оценить эффективность и производительность алгоритма.

4) эффективность: можно сравнивать эффективность методов по различным критериям, таким как точность нахождения оптимального значения и т. д. В данной курсовой работе точность задаётся с консоли, поэтому этот критерий не будет рассмотрен.

Нотация O («О-большое») является одним из способов описания асимптотического поведения функции или алгоритма. Она используется для оценки верхней границы роста функции или времени выполнения алгоритма в зависимости от размера входных данных.

Значение сложности алгоритма не показывает точное время выполнения алгоритма. Сложность показывает динамику увеличения скорости в зависимости от количества данных, с которыми работаем алгоритм. Начиная с определённого количества вводимых элементов, функция растёт не быстрее, чем асимптотическая сложность алгоритма. По-другому эту зависимость можно представить с помощью формулы 1:

(1)

$$∃\left(C>0\right),∃ n\_{0}: ∀\left(n>n\_{0}\right)f\left(n\right)\leq Cg(n)$$

Где:

1) *g(n)* – асимптотическая сложность алгоритма;

2) *f(n)* – точная сложность алгоритма;

3) *С* – константа.

Для примера в качестве рассматриваемых функций выбраны 3 унимодальные функции:

1) квадратичная функция *y = x2 -* 12*x +* 5 (рис. 5.1);

2) обратная тригонометрическая функция 6 *arctan(x*) + 12 (рис. 5.2);

3) функция (- | 4*x* + 50 | + 300) / 5(рис. 5.3).

Графики функций с точками единственного минимума / максимума проиллюстрированы на рисунках 5.1, 5.2 и 5.3 с помощью графического калькулятора Desmos.



Рисунок 5.1 – График функции 1



Рисунок 5.2 – График функции 2

 

Рисунок 5.3 – График функции 3

* 1. Результат работы программы

Ввод значений с консоли представлен на рисунке 6.



Рисунок 6 – Пример ввода значений с консоли

Функция *y = x2 -* 12*x +* 5 возрастает в полуинтервале (-∞, 6) убывает, а в полуинтервале (6, +∞) возрастает. Точка минимума имеет координаты *x* = 6, *y* = -31.

Результаты работы программы с разным диапазоном поиска и разной точностью представлен на рисунках 7.1, 7.2 и 7.3.



Рисунок 7.1 – Результат работы программы в диапазоне (0, 100) с точностью 1е-09



Рисунок 7.2 – Результат работы программы в диапазоне (0, 1000) с точностью 1е-09



Рисунок 7.3 – Результат работы программы в диапазоне (0, 1000000) с точностью 1е-14

Функция 6 *arctan(x*) + 12 возрастает на (-∞, +∞), то есть функция – монотонно возрастающая. Точка минимума будет располагаться на левой границе поиска, а точка максимума – на правой границе. Результаты работы программы с разным диапазоном поиска и разной точностью представлен на рисунках 8.1, 8.2, 8.3 и 8.4.



Рисунок 8.1 - Результат работы программы в диапазоне (-100, 100) с точностью 1е-9 вычисление минимума



Рисунок 8.2 - Результат работы программы в диапазоне (-100, 100) с точностью 1е-9, вычисление максимума



Рисунок 8.3 - Результат работы программы в диапазоне (-10000, 10000) с точностью 1е-9 вычисление минимума



Рисунок 8.4 - Результат работы программы в диапазоне (-10000, 10000) с точностью 1е-9, вычисление максимума

Функция (-|4\*x + 50|+300) /5 возрастает на промежутке (-∞, -12.5), убывает на промежутке (-12.5, +∞). Точка максимума имеет координаты *x* = 6,
*y* = -31. Результаты работы программы с разным диапазоном поиска и разной точностью представлен на рисунках 9.1, 9.2 и 9.3.



Рисунок 9.1 - Результат работы программы в диапазоне (-100, 100) с точностью 1е-9



Рисунок 9.2 - Результат работы программы в диапазоне (-10000, 10000) с точностью 1е-9



Рисунок 9.3 - Результат работы программы в диапазоне (-10000, 10000) с точностью 1е-11

* 1. Сравнение методов одномерной оптимизации

Для оценки сложности методов одномерной оптимизации, таких как метод деления пополам, метод дихотомии и метод золотого сечения, использован подсчёт времени, количества итераций, найдена асимптотическая сложность.

Время на всех представленных примерах вне зависимости от величины выбранного участка поиска и заданной точности равно 0.0. Данный способ сравнения не подходит для выбранных задач.

В ходе проверки результатов возникли неточности с методом дихотомии. При больших диапазонах или высокой точности, метод принимает за минимум крайнюю правую точку диапазона. Такая ситуация наблюдается на рисунках 7.3, 8.3 и 8.4.

С помощью подсчёта количества итераций методы сравнить можно. Из представленных на рисунках 7.1 – 7.2 и 9.1 – 9.3 результатов можно сделать вывод, что метод деления пополам и метод дихотомии затрачивают одинаковое количество итераций, если диапазон поиска и точность равны. Метод золотого сечения же использует больше проходов цикла (Табл. 1).

Таблица 1 – Количество итераций в разных методах

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № рисунка | Количество итераций (метод деления пополам) | Количество итераций (метод дихотомии) | Количество итераций (метод золотого сечения) | Соотношение итераций в 1 методе и в 3 методе |
| 7.1 | 37 | 37 | 53 | 1,4324 |
| 7.2 | 40 | 40 | 58 | 1,45 |
| 9.1 | 38 | 38 | 55 | 1,4473 |
| 9.2 | 45 | 45 | 64 | 1,4222 |
| 9.3 | 51 | 51 | 74 | 1,4509 |

Среднее соотношение между количеством итераций – 1,4405. Можно заметить, что в методе золотого сечения итераций больше в среднем на 44% по сравнению с другими методами. Это связано с тем, что первые 2 метода на каждой итерации отсекают 1/2 отрезка, а 3 метод отсекает одну сторон, каждая из которых меньше.

Кроме количества итераций важную роль играет количество точек, для которых необходимо вычислить функцию. Таблица 2 демонстрирует количество вычисляемых точек.

Таблица 2 – Количество вычислений функции в точке

|  |  |  |  |
| --- | --- | --- | --- |
|  | Метод деления пополам | Методдихотомии | Метод золотого сечения |
| Количество вычислений точек на каждой итерации | 3 | 2 | 2 |
| Количество вычислений точек на каждой итерации, кроме 1 в модернизированном алгоритме | 2 | 2 | 1 |

Из таблицы можно понять, что метод делений пополам больше раз вычисляет значение функций в точках. В случае со сложными функциями, вычисление значений для которых может занимать много времени, этот метод может существенно замедлить программу.

Чтобы избежать задержки программы, можно учитывать значения функции в некоторых точках, которые были вычислены на прошлой итерации. Данное улучшение позволит вычислять на каждой итерации, кроме первой, на 1 значение меньше в методах деления пополам и золотого сечения. Таким образом, метод золотого сечения может позволить производить меньше вычислений значений функции в точках.

Сложность методов можно вычислить, проверяя разные вводные данные. Были выбраны контрольные значения границ функции *y = x2 -* 12*x +* 5, точность осталась выбранная по умолчанию - 0.000000001. Результаты представлены в таблице 3.

Таблица 3 – Количество итераций на разных областях поиска

|  |  |  |  |
| --- | --- | --- | --- |
| Диапазон | Метод деления пополам | Метод дихотомии | Метод золотого сечения |
| (-100; 100) | 38 | 38 | 55 |
| (-200; 200) | 39 | 39 | 56 |
| (-400; 400) | 40 | 40 | 57 |

Можно заметить, что с увеличением выбранной области поиска в два раза, количество итераций увеличивается на 1. Это означает, что сложность всех представленных методов составляет:

(2)

$log\_{2}\frac{b-a}{e}$ или $log\_{2}N$

В формуле 2 (b – a) расстояние между левой и правой границей, а e – заданная точность. N – соотношение двух величин. Это связано с тем, что на каждой итерации интервал поиска уменьшается вдвое (или в пропорции золотого сечения) до тех пор, пока не будет достигнута требуемая точность. График роста количества итераций в зависимости от N представлен на рисунке 10.



Рисунок 10 – График роста количества итераций в зависимости от N

Заключение

В ходе исследования одномерной оптимизации было проведено подробное изучение и сравнение трёх методов: метод деления пополам, метод дихотомии и метод золотого сечения. Для каждого метода был представлен и проиллюстрирован пошаговый алгоритм поиска минимума/максимума функции.

Для программной реализации методов поиска экстремума был выбран язык Python. Разработаны функции для каждого метода, а также вспомогательные функции ввода условий с консоли и вывода результатов поиска на консоль.

Было проведено сравнение эффективности методов по количеству итераций, вычислений функции в точке и сложности алгоритмов. Это позволило сделать следующие выводы:

1) Время выполнения на всех примерах составляло 0.0, независимо от размера выбранного интервала поиска и заданной точности, что не позволяет использовать данную метрику для сравнения эффективности методов. Кроме того, при проверке результатов были выявлены неточности с методом дихотомии, особенно при больших диапазонах или высокой точности.

2) Подсчет количества итераций позволяет сравнить эффективность методов. В случаях, когда все алгоритмы работают верно, метод деления пополам и метод дихотомии требуют одинаковое количество итераций. В то же время метод золотого сечения в среднем использует на 44% больше итераций по сравнению с другими методами.

3) Важную роль играет количество точек, для которых необходимо вычислить функцию. Метод делений пополам вычисляет значение 3 функций в точках на каждой итерации, другие методы вычисляют по 2 точки. В случае со сложными функциями этот метод может существенно замедлить программу. Чтобы избежать задержки программы, можно учитывать значения функции в некоторых точках, которые были вычислены на прошлой итерации. Данное улучшение позволит вычислять на каждой итерации, кроме первой, на 1 значение меньше в методах деления пополам и золотого сечения.

4) Асимптотическая сложность представленных методов – ($log\_{2}\frac{(b-a)}{e}$). Это связано с тем, что на каждой итерации интервал поиска (b – a) уменьшается вдвое (или в пропорции золотого сечения) до тех пор, пока не будет достигнута требуемая точность e. Это позволяет оценить сложность всех этих методов как логарифмическую относительно размера области поиска и требуемой точности.

Таким образом, при выборе метода одномерной оптимизации для конкретной задачи необходимо учитывать, как требуемую точность результата, так и вычислительные затраты. Каждый из рассмотренных методов имеет свои преимущества и может быть эффективным в зависимости от поставленных задач.

Список использованных источников

1. Аттетков, А. В. Методы оптимизации / А. В. Аттетков, С. В. Галкин, В. С. Зарубин – Изд. МГТУ им. Н. Э. Баумана, 2001. ISBN, 5703817706
2. Вирт, Никлаус. Алгоритмы и структуры данных / Никлаус Вирт. - М.: ДМК Пресс, 2016. - 889 c. – ISBN: 978-5-97060-011-5.
3. Методы одномерной оптимизации / сост. Т. М. Попова. – Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2011. – 26 с
4. Файловый архив студентов // Оптимизация технологических процессов, Ангарская Государственная Техническая Академия: [сайт].– 2023.– URL: <https://studfile.net/preview/1467149/> (дата обращения: 02.12.2023).
5. Левитин А. В. Глава 11. Преодоление ограничений: Метод деления пополам // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 476—480. — 576 с. — ISBN 978-5-8459-0987-9