Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Информационные системы и технологии

Алгоритмы одномерной минимизации

Отчёт по лабораторной работе №2

Работу выполнили:
Деревицкая П. К., студентка М32081
Фирсова Д. А., студентка М32081
Борздун А. В., студентка М32051

Преподаватель:
Свинцов М. В.

Санкт-Петербург,
2023

Оглавление

Цель работы: знакомство с одномерными алгоритмами минимизации функций без производной

Задачи, решаемые во время выполнения работы:

1. Реализация данных методов на Python 3.
2. Тестирование реализованных алгоритмов для задач минимизации многомодальных функций.
3. Сравнение методов по количеству итераций и количеству вычислений функции в зависимости от разной точности

Метод дихотомии.

Алгоритм метода половинного деления (метод дихотомии)

Метод дихотомии — это один из классических методов минимизации функций, основанный на принципе деления отрезка пополам. Алгоритм заключается в последовательном уменьшении интервала поиска значений минимума функции до достижения заранее заданной точности.

Метод половинного деления позволяет отсекать в точности половину интервала на каждой итерации. При использовании метода считается, что функция непрерывна и что в интервале есть только один минимум. Если эти предположения не выполняются, алгоритм может не сходиться к истинному минимуму или может сходиться к локальному минимуму вместо глобального минимума. После вычисления значения функции в середине интервала одна часть интервала отбрасывается так, чтобы функция имела разный знак на концах оставшейся части. Итерации метода деления пополам прекращаются, если достигается нужная точность (интервал становится достаточно малым).

Метод выбора точки деления - ключевой для скорости работы метода. Так как внутри функции могут быть тяжелые операции, то главным критерием оптимизации будет являться минимизация числа делений (без ухудшения точности метода).

Количество итераций: $K = \frac{\ln\left(\frac{b_0 - a_0}{\varepsilon}\right)}{\ln(2)}$

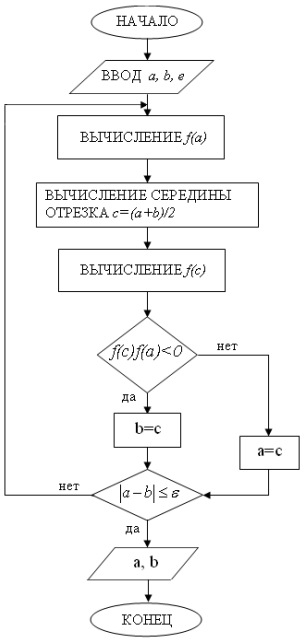
При этом на каждой итерации мы вычисляем два значения функции.

Преимущества: гарантированная сходимость (метод дихотомии гарантированно сходится к минимуму, если функция унимодальна) => это гарантирует, что алгоритм всегда даст правильный ответ; не зависит от начальных условий, поэтому может быть использован для оптимизации функций различной структуры.

Недостатки: неэффективен для функций, которые содержат несколько минимумов.

Шаги алгоритма

1. Выбираем начальный интервал $[a, b]$ и задаем погрешность.
2. Вычисляем значение функции f в точках a и b.
3. Пока длина текущего интервала больше погрешности:
    a. Отрезок $[a, b]$ делим пополам (вычисляется значение $x_{\text{ср}}$) и вычисляем значение функции в новой точке.
    b. Из двух отрезков: $[a, x_{\text{ср}}]$ и $[x_{\text{ср}}, b]$ выбираем тот, который содержит корень и рассматривается как новый отрезок [a, b] на следующей итерации.
4. Возвращаем полученный отрезок
5. Процесс деления отрезка пополам завершается при выполнении условий: $|b - a| \leq \varepsilon$ и $\frac{|f(b) - f(a)|}{2} \leq \delta$



Время работы алгоритма

Для оценки времени работы метода дихотомии можно использовать следующую формулу:

$$T = \left(log\left(\frac{b-a}{\varepsilon}\right) + 1\right) * k$$

где a и b - концы интервала, ε - требуемая точность, а k - время, необходимое для вычисления функции f(x).

Т. е. время работы линейно зависит от количества итераций, необходимых для достижения требуемой точности (количество итераций, в свою очередь, зависит от ширины интервала и требуемой точности) и уменьшается логарифмически при увеличении точности.

Практическое решение

Реализация алгоритма:

```
import math
def dichotomy_method(a, b, eps):
  function_call[0]=0
  iteration_count[0]=0
  delta=eps/3
  while abs(b - a) > eps:
    iteration_count[0]+=1
    x1 = (a + b) / 2  - delta
    x2 = (a + b) / 2  + delta
    if f(x1) < f(x2):
      function_call[0]+=2
      b = x2
    else:
      function_call[0]+=2
      a = x1
  return round((a + b) / 2, 4)
```

Протестируем заданную функцию $y(x) = e^{sin(x)} \cdot x^2$ на отрезке $[-5; 2]$. Полученные результаты:

Method	Accuracy	Calls	Iterations
Dichotomy	0.2	14	7
Dichotomy	0.1	16	8
Dichotomy	0.01	24	12
Dichotomy	0.001	30	15

Метод золотого сечения.

Алгоритм метода золотого сечения.

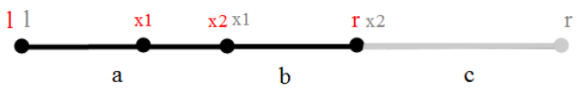
Метод золотого сечения — это также алгоритм, служащий для нахождения минимума/максимума функции. Метод золотого сечения требует вычисления лишь в одной точке (за исключением первой итерации).

Золотое сечение – такое деление отрезка [a, b], что отношение меньшей части к большей, такое же, как отношение большей части ко всему отрезку.

Константа золотого сечения – это значение, показывающее насколько уменьшается интервал неопределенности.

Точки внутри интервалов поиска решения выбираются исходя из следующих соображений: на текущей итерации оба потенциальных интервала сокращения равны между собой, и на каждой итерации интервал должен уменьшаться.

Обоснование: рассмотрим одну итерацию алгоритма троичного поиска. Попробуем подобрать такое разбиение отрезка на три части, чтобы на следующей итерации одна из точек нового разбиения совпала с одной из точек текущего разбиения. Тогда в следующий раз не придется считать функцию в двух точках, так как в одной она уже была посчитана.



Для этого нам потребуется, чтобы одновременно выполнялись равенства:

$$\frac{a+b}{c} = \frac{b+c}{a} = \varphi$$

Полученное расстояние:

- от l до x_1: $a + b - c = a'$,
- от x_2 до r: $b = c'$,
- от x_1 до x_2: $c - b = b'$.

То есть, если мы подставим a', b', c' в старое соотношение $\frac{a+b}{c}$ мы получим $\frac{a+b-c+c-b}{b} = \frac{a}{b}$. $\frac{a}{b} = \varphi, \frac{c}{b} = \varphi$, где φ –это некоторое отношение, в котором мы делим отрезок (точки x_1 и x_2 разбивают отрезок симметрично). Тогда:

$$a + b = \varphi c, a = \varphi b, c = \varphi b ,$$

откуда получаем $\varphi + 1 = \varphi^2$, $\varphi = \frac{1+\sqrt{5}}{2}$ (тот корень уравнения, который меньше нуля, по понятным причинам отбросили). Это число совпадает с золотым сечением. Отсюда название метода.

Преимущества: в сравнение с методом дихотомии, функция вычисляется один раз, так как она разбиваются каждый раз на золотое сечение;

Недостатки: интервал уменьшается не так быстро, как в методе дихотомии.

Шаги алгоритма

1. Определяем границы поиска l и r, затем устанавливаем текущее разбиение:

$$x_1 = l + \frac{r - l}{\varphi + 1}$$
$$x_2 = r - \frac{r - l}{\varphi + 1}$$

   и вычислим функцию на них:

$$f_1 = f(x_1),\ f_2 = f(x_2)$$

2. Если $f_1 < f_2$, тогда:

$$r = x_2, \quad x_2 = x_1, \quad f_2 = f_1$$
$$x_1 = l + \frac{r - l}{\varphi + 1}, \quad f_1 = f(x_1)$$

   Иначе:

$$l = x_1, \quad x_1 = x_2, \quad f_1 = f_2$$
$$x_2 = r - \frac{r - l}{\varphi + 1}, \quad f_2 = f(x_2)$$

3. Если точность $|r - l| < \varepsilon$ нас устраивает, тогда останавливаемся и искомая точка $x = \frac{l + r}{2}$, иначе возвращаемся к шагу 2.

Время работы алгоритма

В общем случае время работы алгоритма может быть оценено как $O(log(\varepsilon))$, где ε - желаемая точность. Так как на каждой итерации мы считаем одно значение функции и уменьшаем область поиска в φ раз, пока $r - l > \varepsilon$, то время работы алгоритма составит $O\left(\log_{\varphi}\left(\frac{r-l}{\varepsilon}\right)\right)$.

Количество итераций: $\frac{log\left(\frac{1}{\varepsilon}\right)}{log(\varphi)}$, где φ – константа золотого сечения.

Если мы начинаем поиск на очень широком интервале, то алгоритм может потребовать значительно больше итераций, чтобы достичь нужной точности. А если функция очень сложная, то каждая итерация может занимать много времени.

Практическое решение

Программное решение:

```
def golden_section(a, b, eps):
  function_call[1]=0
  iteration_count[1]=0
  phi = 1 - (math.sqrt(5) - 1) / 2
  x1 = a + phi * (b - a)
  x2 = b - phi * (b - a)
  f1, f2 = f(x1), f(x2)
  function_call[1]+=2
  while abs(b - a) > eps:
    iteration_count[1]+=1
    if f1 < f2:
        b, x2, f2 = x2, x1, f1
        x1 = a + phi * (b - a)
        f1 = f(x1)
        function_call[1]+=1
    else:
        a, x1, f1 = x1, x2, f2
        x2 = b - phi * (b - a)
        f2 = f(x2)
        function_call[1]+=1
  return (a + b) / 2
```

Полученные результаты тестирования:

Method	Accuracy	Calls	Iterations
Golden Section	0.2	10	8
Golden Section	0.1	11	9
Golden Section	0.01	16	14
Golden Section	0.001	21	19

Метод Фибоначчи.

Алгоритм метода Фибоначчи

Метод Фибоначчи— это улучшение реализации поиска с помощью золотого сечения, служащего для нахождения минимума/максимума функции. Алгоритм заключается в делении исследуемого отрезка на два подотрезка, с использованием соотношения золотого сечения, и далее выборе одного из этих подотрезков для продолжения поиска. Каждый раз размер отрезка уменьшается пропорционально числу Фибоначчи, что позволяет достигать быстрой сходимости к минимуму функции.

Подобно методу золотого сечения, он требует двух вычислений функции на первой итерации, а на каждой последующей только по одному. Однако этот метод отличается от метода золотого сечения тем, что коэффициент сокращения интервала неопределенности меняется от итерации к итерации.

Метод основан на последовательности чисел Фибоначчи $F_v = F_{v-1} + F_{v-2}$, $F_0 = F_1 = 1$.

Предположим, что на $k-$й итерации интервал неопределенности равен $[a_k, b_k]$. Рассмотрим две точки λ_k и μ_k, определяемые следующим образом:

$$\lambda_k = a_k + \frac{F_{n-k-1}}{F_{n-k+1}} * (b_k - a_k)$$

$$\mu_k = a_k + \frac{F_{n-k}}{F_{n-k+1}} * (b_k - a_k),$$

где k = 1, 2, … , n-1 и n – заданное общее число вычислений функции.

Новый интервал неопределенности $[a_{k+1}, b_{k+1}]$ будет равен $[\lambda_k, b_k]$, если $f(\lambda_k) > f(\mu_k)$ и $[a_k, \mu_k]$, если $f(\lambda_k) \leq f(\mu_k)$.

В первом случае, учитывая λ_k и полагая $v = n - k$, получим

$$b_{k+1} - a_{k+1} = \ b_k - a_k - \frac{F_{n-k-1}}{F_{n-k+1}} * (b_k - a_k) = \frac{F_{n-k}}{F_{n-k+1}} * (b_k - a_k).$$

Во втором случае, учитывая μ_k, получаем

$$b_{k+1} - a_{k+1} = \ \mu_k - a_k = \frac{F_{n-k}}{F_{n-k+1}} * (b_k - a_k).$$

Таким образом, в обоих случаях длина интервала неопределенности сжимается с коэффициентом $\frac{F_{n-k}}{F_{n-k+1}}$.

На k-й итерации либо $\lambda_k = \mu_k$, либо $\lambda_k = \mu_{k+1}$, так что требуется только одно новое вычисление функции.

В отличие от метода золотого сечения в методе Фибоначчи требуется, чтобы общее число вычислений n (или коэффициент сокращения исходного интервала) было задано заранее. Это объясняется тем, что точки, в которых производятся вычисления, зависят от n. Длина интервала неопределенности на k-той итерации сжимается с коэффициентом $\frac{F_{n-k}}{F_{n-k+1}}$. Следовательно, после (n−1) итерации, где n — заданное общее число вычислений функции f(x), длина интервала неопределенности сократится от $(b_1 - a_1)$ до $\frac{(b_1-a_1)}{F_n}$.

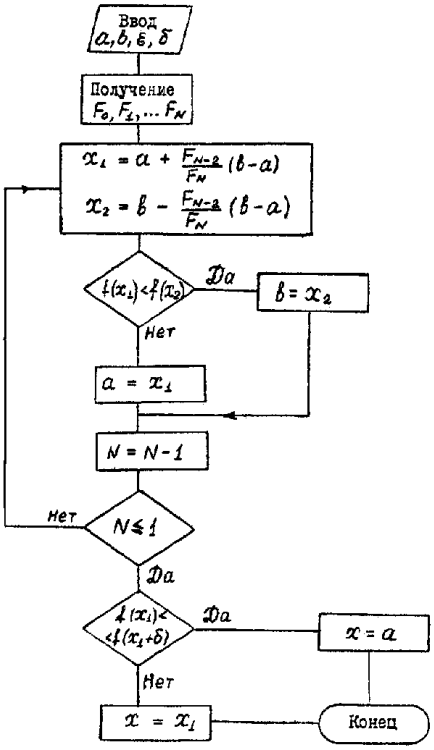
Преимущества: требует меньше итераций, чем метод золотого сечения, для той же точности.

Недостатки: нужно хранить избыточный набор чисел Фибоначчи либо многократно генерировать числа по мере необходимости; Фибоначчи нелегко приспособить к часто используемому критерию остановки, требующему, чтобы значения функции на окончательном интервале неопределенности разнились на величину меньше заданной погрешности.

Шаги алгоритма

1. Выбрать начальный интервал $[a, b]$, точность ε и рассчитать количество шагов n.
2. Вычислить первые два числа Фибоначчи $F(k-1),\ F(k)$ такие, что $F(k) > \frac{b-a}{\varepsilon}$.
3. Установить значения $x_1 = a + \left(\frac{F(k-2)}{F(k)}\right) * (b-a),\ x_2 = a + \left(\frac{F(k-1)}{F(k)}\right) * (b-a)$ и вычислить значения функции f(x1) и f(x2).
4. Пока число шагов n не равно 1, повторять следующее:
   a. Если $f(x_1) < f(x_2)$, заменить b на x_2 и установить $x_2 = x_1$.
      Вычислить новое значение x_1 как $a + \left(\frac{F(k-n+1)}{F(k-n+2)}\right) * (b-a)$ и вычислить значение функции $f(x_1)$.
   b. Иначе, заменить a на x_1 и установить $x_1 = x_2$.
      Вычислить новое значение x_2 как $a + \left(\frac{F(k-n+2)}{F(k-n+2)}\right) * (b-a)$ и вычислить значение функции $f(x_2)$.
   c. Уменьшить n на 1.
5. Возвращать значение $\frac{a+b}{2}$ в качестве оптимальной точки.

Эти шаги повторяются до тех пор, пока не будет достигнута нужная точность или ограничение на количество итераций.



Время работы алгоритма

Теоретическая сложность метода Фибоначчи равна $O\left(log\left(\frac{1}{\varepsilon}\right)\right)$.

Количество итераций: $K = log\left(\frac{1}{\varepsilon}\right) + \frac{log(\sqrt{5})}{2\log(\varphi)}$, где φ - золотое сечение.

Практическое решение

Программное решение:

```
def fibonacci_search(a, b, eps):
  function_call[2]=0
  iteration_count[2]=0

  Fn = (b-a)/eps
  n=1
  while(Fn>fibonacci(n)):
    n+=1

  x1 = a + (fibonacci(n-2)/fibonacci(n))*(b-a)
  x2 = a + (fibonacci(n-1)/fibonacci(n))*(b-a)

  f1, f2 = f(x1), f(x2)
  function_call[2]+=2
  for i in range(n-1):
    iteration_count[2]+=1
    if f1 < f2:
      b = x2
      x2 = x1
      f2 = f1
      x1 = a + (fibonacci(n-i-3)/fibonacci(n-i-1))*(b-a)
      f1 = f(x1)
      function_call[2]+=1
    else:
      a = x1
      x1 = x2
      f1 = f2
      x2 = a + (fibonacci(n-i-2)/fibonacci(n-i-1))*(b-a)
      f2 = f(x2)
      function_call[2]+=1
  return (a+b)/2
```

Полученные результаты:

Method	Accuracy	Calls	Iterations
Fibonacci	0.2	11	9
Fibonacci	0.1	12	10
Fibonacci	0.01	17	15
Fibonacci	0.001	22	20

Метод парабол.

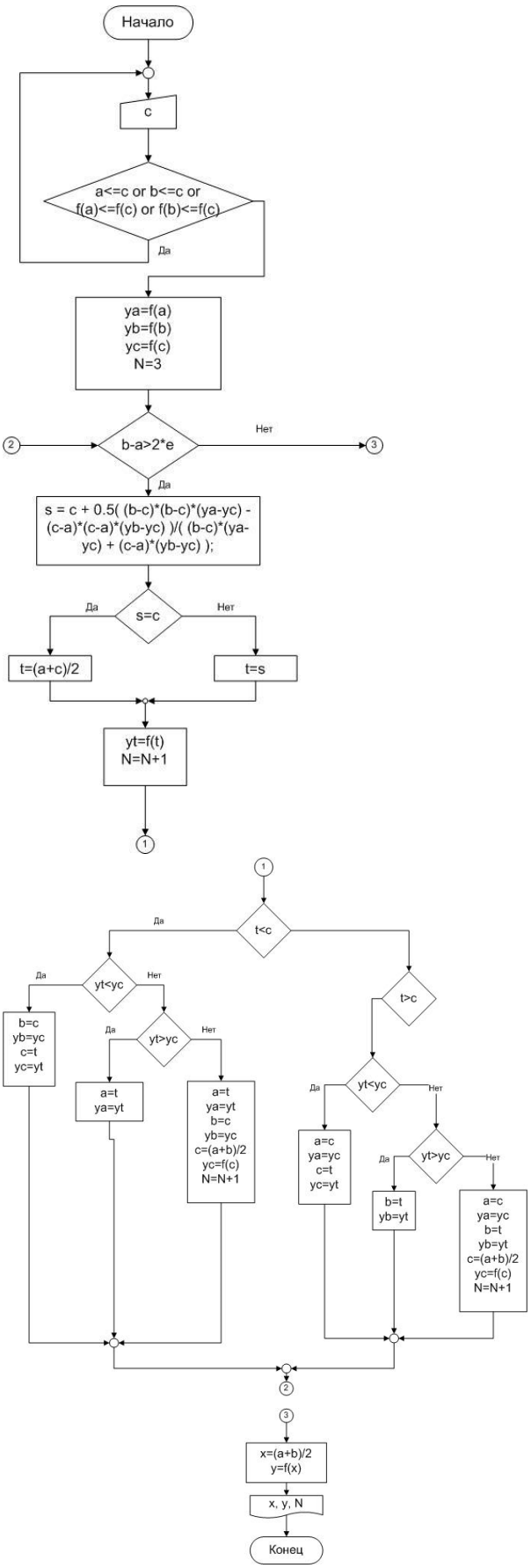
Алгоритм метода парабол

Метод парабол — это метод минимизации одномерной функции на основе интерполяции параболой. Он использует только значения функции в трех точках, чтобы аппроксимировать локальную форму функции и найти ее минимум.

Преимущества: может быть очень быстрым для функций с квадратичной зависимостью, ищет не только минимумы, но и максимумы.
Недостатки: не гарантирует сходимость, если функция не является квадратичной.

Шаги алгоритма

1. Выбор начального интервала поиска.
2. Определение трех точек на этом интервале.
3. Вычисление значений функции в этих точках.
4. Построение параболы через эти три точки.
5. Вычисление координаты вершины параболы.
6. Вычисление значения функции в новой точке.
7. Если значение функции в новой точке меньше, чем значение в одной из старых точек, замена этой старой точки новой.
8. Проверка условия окончания алгоритма. Если оно не выполнено, повторение шагов 3-7.
9. Возвращение точки с наименьшим значением функции в качестве результата.



Время работы

Теоретическая сложность метода Брента составляет $O\left(log\left(\frac{1}{\varepsilon}\right)\right)$. Однако на практике, метод Брента часто работает быстрее, чем методы золотого сечения и Фибоначчи. Он требует меньшее количество итераций.

Количество итераций: $K = log\left(\frac{1}{\varepsilon}\right) * 2 * \frac{\left(log\left(\frac{1}{\varepsilon}\right)+\frac{log(\sqrt{5})}{2}\right)}{(log(\varphi)+1)}$, где φ - золотое сечение (примерно 1,618)

Практическое решение

Программное решение:

```
def parabolic_minimization(a, b, eps):
  function_call[3]=0
  iteration_count[3]=0
  x1, x2, x3 = a, (a+b)/2, b
  f1, f2, f3 = f(x1), f(x2), f(x3)
  function_call[3]+=3
  while abs(x3 - x1) > eps:
    iteration_count[3]+=1
    A = ((x2 - x3)*(f1 - f2) - (x1 - x2)*(f2 - f3)) / ((x1 - x2)*(x3 - x1)*(x3 - x2))
    B = ((x2**2 - x3**2)*(f1 - f2) - (x1**2 - x2**2)*(f2 - f3)) / ((x1 - x2)*(x3 - x1)*(x3 - x2))
    x_min = -B / (2*A)
    if x_min < x2:
      x1, x2, x3 = x1, x_min, x2
      f1, f2, f3 = f1, f(x_min), f2
      function_call[3]+=1
    else:
      x1, x2, x3 = x2, x_min, x3
      f1, f2, f3 = f2, f(x_min), f3
      function_call[3]+=1
  return (x1 + x3) / 2
```

Полученные результаты:

Method	Accuracy	Calls	Iterations
Parabolic	0.2	9	6
Parabolic	0.1	9	6
Parabolic	0.01	11	8
Parabolic	0.001	137	134

Комбинированный метод Брента.

Алгоритм комбинированного метода Брента.

Комбинированный метод Брента - это метод минимизации одномерной функции, который использует сочетание методов золотого сечения, секущих и парабол для ускорения сходимости и устранения их недостатков.

Преимущества: быстрый и надежный для большинства функций, эффективен для поиска минимума в функциях, содержащих несколько минимумов.

Недостатки: сложная реализация, может быть медленным.

Шаги алгоритма

1. Выбрать начальный интервал, содержащий минимум функции.
2. Вычислить значения функции в двух концах интервала и его середине.
3. Если значение функции в середине интервала достаточно мало, то закончить поиск и вернуть середину.
4. Иначе, если один из концов интервала и середина имеют равные значения функции, то выбрать дихотомический шаг путем деления интервала пополам.
5. В противном случае вычислить положение точки минимума параболы, проходящей через три точки: концы интервала и середину.
6. Если точка минимума попадает в интервал, отличный от текущего, то выбрать золотое сечение, чтобы переместиться к новому интервалу.
7. Иначе, выполнить шаг интерполяции квадратичной функции.
8. Если точность достигнута, то завершить процесс и вернуть текущую точку минимума.
9. Иначе, повторить шаги 2-8

Время работы

Количество итераций: $K = \frac{log\left(\frac{1}{\varepsilon}\right)}{log(\varphi)}$, где ε - требуемая точность результата, а φ - золотое сечение.

Отличие комбинированного метода Брента от базового заключается в том, что он использует эвристики для ускорения второй итерации базового метода Брента, если она не сходится быстро или требует много итераций.

Таким образом, время работы = время одной итерации * K * F, где F - коэффициент ускорения, который зависит от функции и начального интервала.

Практическая часть

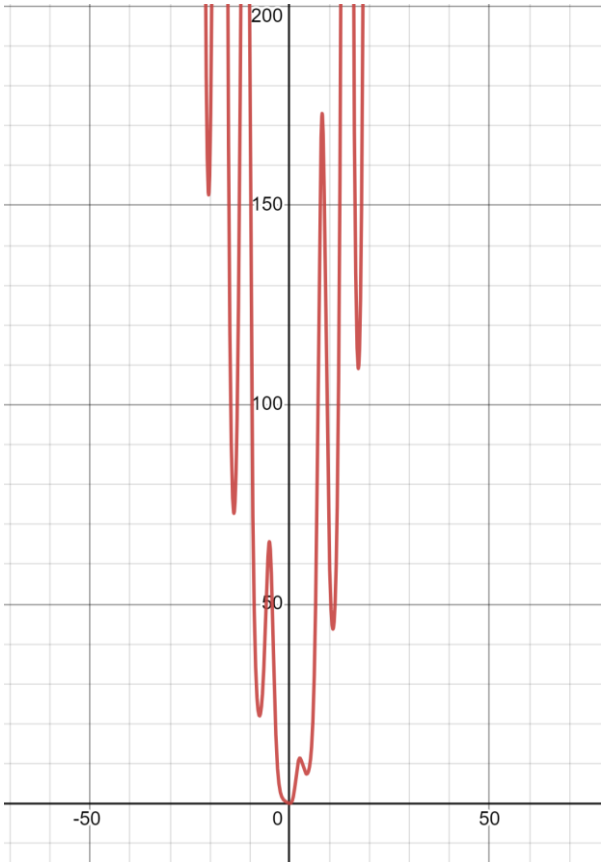
Полученные результаты:

Method	Accuracy	Calls	Iterations
Brent	0.2	9	9
Brent	0.1	9	9
Brent	0.01	12	12
Brent	0.001	12	12

Тестирование методов

Сравнение работы методов исходной функции

Приведём график нашей функции $y = e^{sin(x)} \cdot x^2$

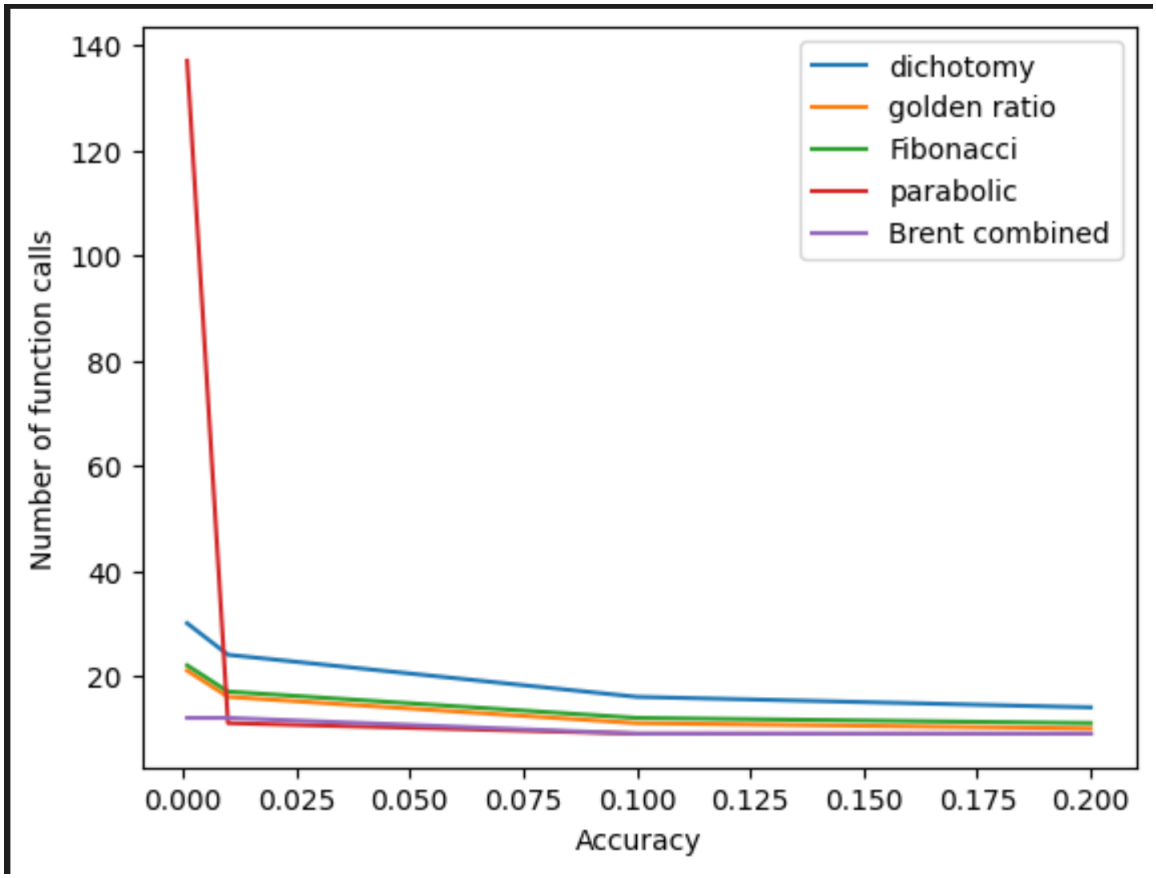


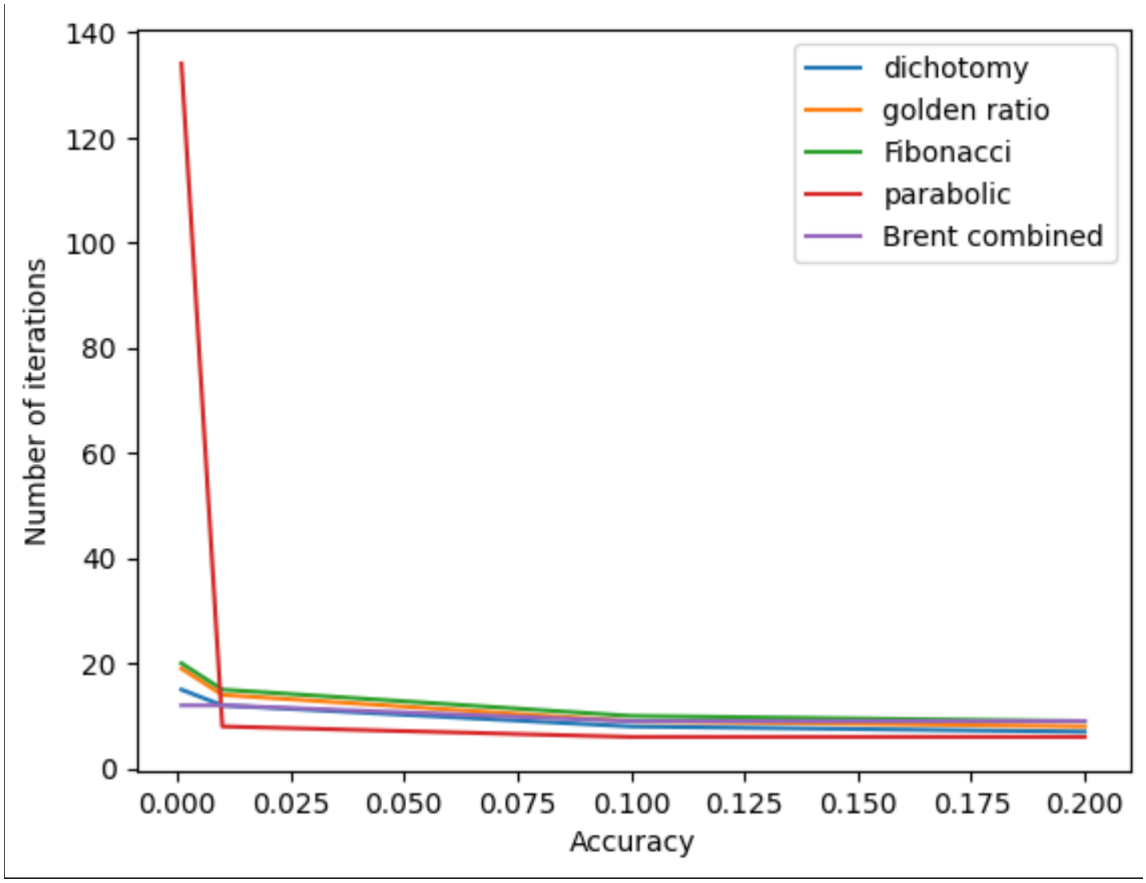
Функция имеет один глобальный минимум в точке (0;0).

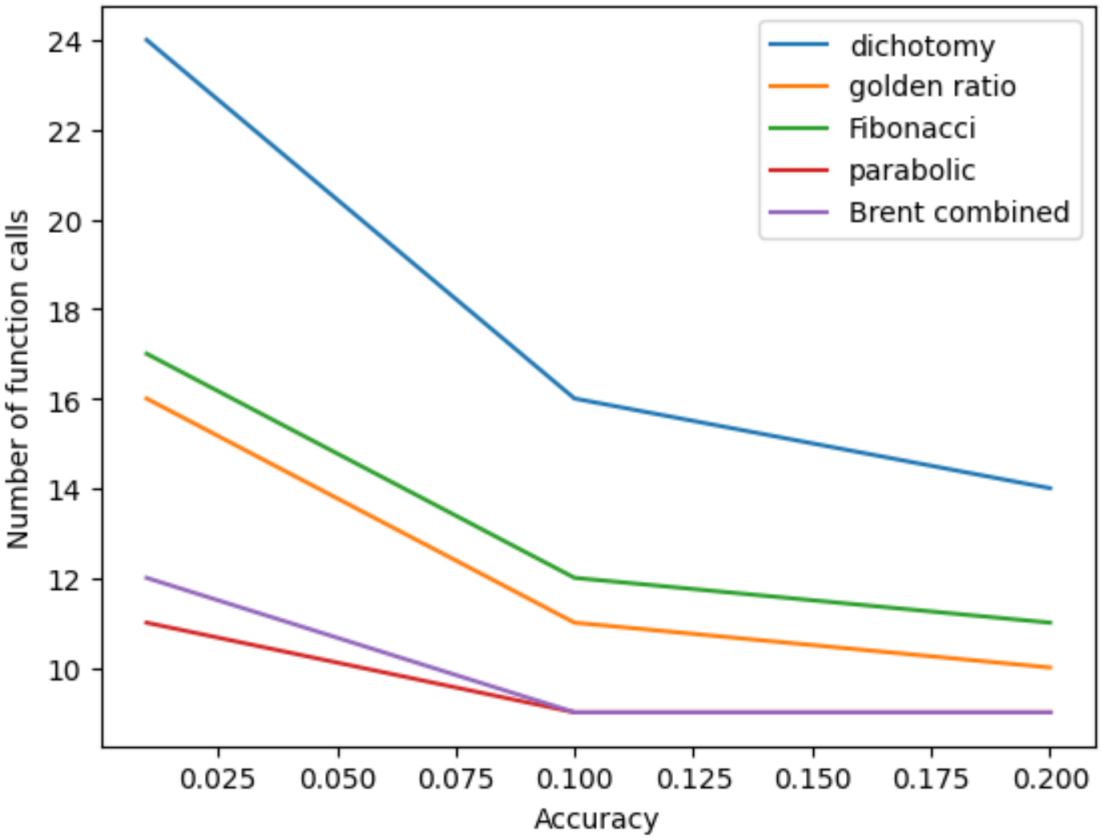
Сравним методы по количеству итераций и количеству вычислений функции в зависимости от разной точности:

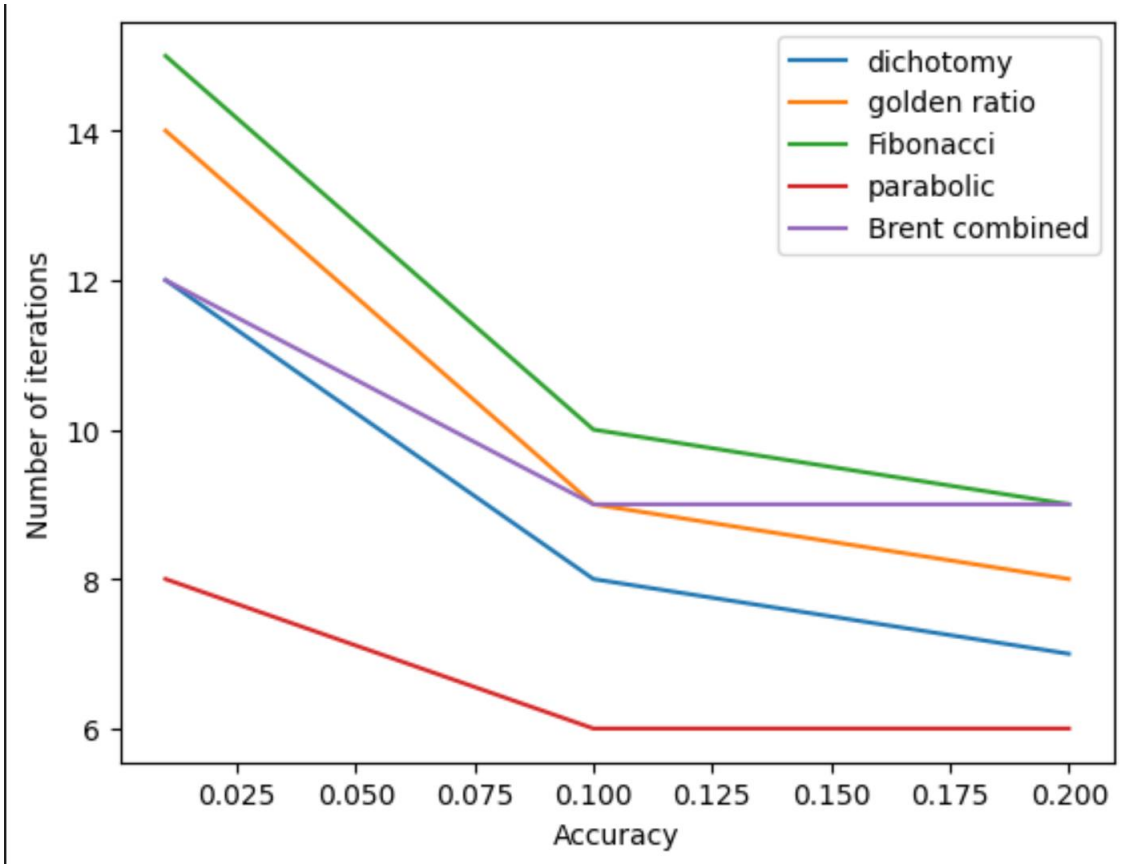
Method	Accuracy	Calls	Iterations
Dichotomy	0.2	14	7
Dichotomy	0.1	16	8
Dichotomy	0.01	24	12
Dichotomy	0.001	30	15
Golden Section	0.2	10	8
Golden Section	0.1	11	9
Golden Section	0.01	16	14
Golden Section	0.001	21	19
Fibonacci	0.2	11	9
Fibonacci	0.1	12	10
Fibonacci	0.01	17	15
Fibonacci	0.001	22	20
Parabolic	0.2	9	6
Parabolic	0.1	9	6
Parabolic	0.01	11	8
Parabolic	0.001	137	134
Brent	0.2	9	9
Brent	0.1	9	9
Brent	0.01	12	12
Brent	0.001	12	12

Чтобы лучше увидеть разницу показателей методов, построим графики по привидённым выше значениям:









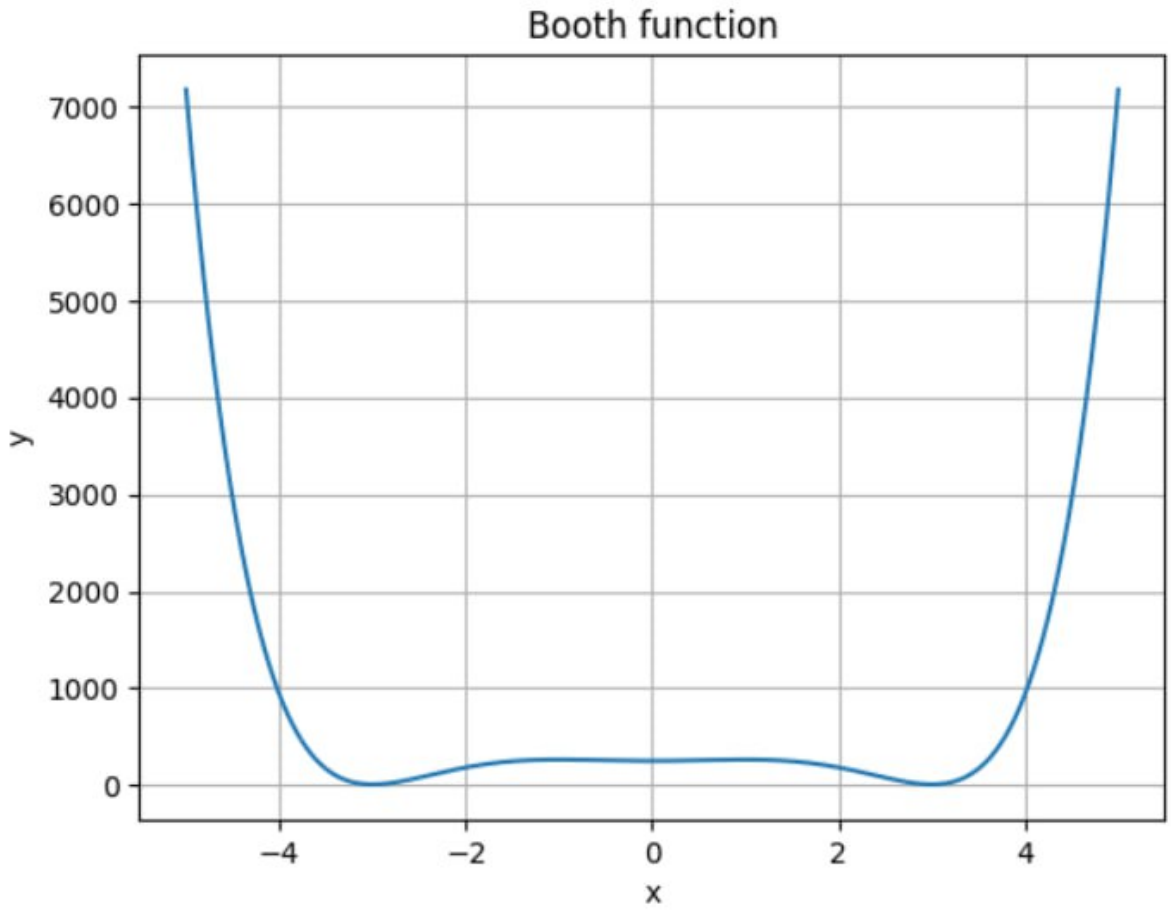
Из приведенных выше результатов, можно сделать вывод, что меньше всего итераций и вызовов занимает параболический метод. Это объясняется тем, что наша исходная функция приближена к параболе, соответственно, точность данного метода повышена. Метод

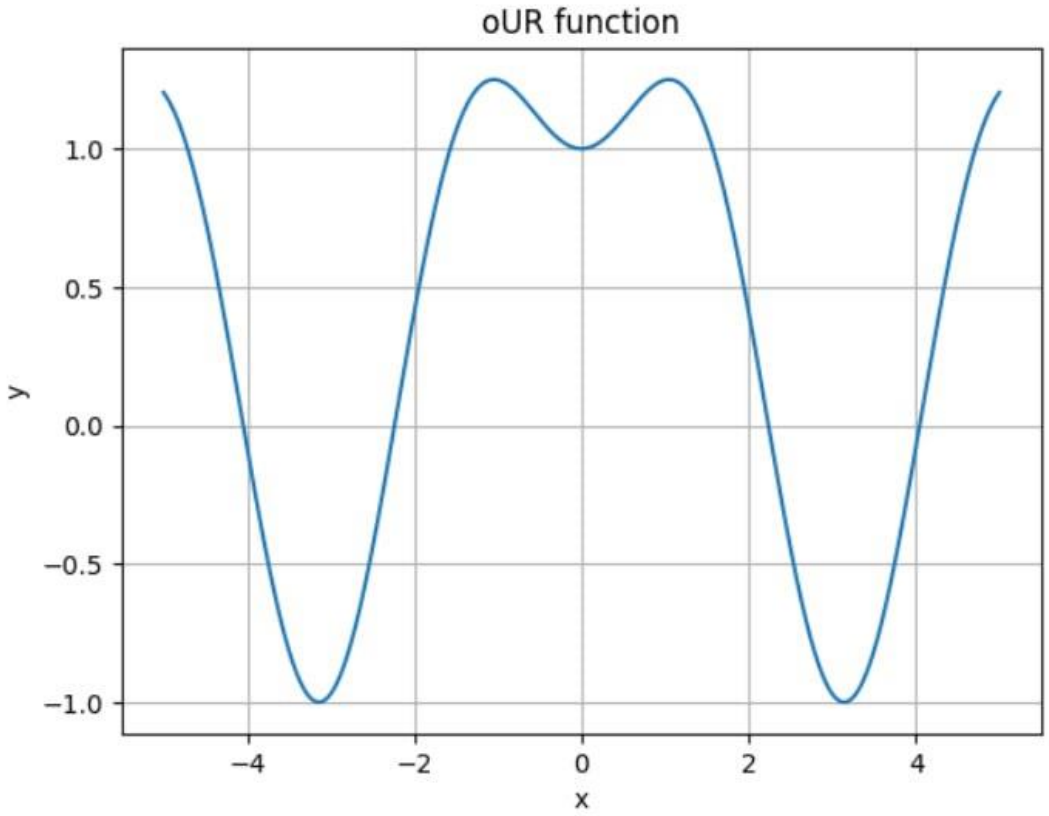
дихотомии показал наихудший результат в параметре «calls», потому что на каждом шаге метода необходимо вычислять значение функции в двух точках, чтобы определить, в какой половине интервала находится корень, а это приводит к увеличению времени работы алгоритма. Кроме того, если функция имеет много экстремумов, то при каждом шаге может потребоваться проверять несколько различных интервалов, что также увеличивает количество вызовов функции.
В параметре итераций метод Фибоначчи показал себя хуже всего, потому что последовательность имеет очень быстрый рост, что приводит к тому, что метод делает слишком маленькие шаги и требуется большое количество итераций, чтобы достичь заданной точности.

Сравнение работы методов на различных полиномах

Для тестирования данных методов мы решили использовать две функции: $f(x) = (x^6 - 15 * x^4 + 27 * x^2 + 250)$ (функция Бута) и $y = sin(x)^2 + cos(x)$.

Графики данных функций:





Функция Бутта отличается тем, что на всей своей области определения имеет два глобальных минимума, у нашей же они отсутсвуют.
Протестируем обе функции на алгоритмах:

```
print(golden_section(l,r,eps,himmelblau))
print(dichotomy_method(l,r,eps,himmelblau))
print(fibonacci_search(l,r,eps,himmelblau))
print(parabolic_minimization(l,r,eps,himmelblau))
print(Brent(l,r,eps,himmelblau))
print(function_call)
print(iteration_count)
```

```
3.140626054908277
3.1416
3.1453900709219855
0.0014270052382262422
-0.0016988171655038643
[24, 16, 17, 15, 17]
[12, 14, 15, 12, 17]
```

```
print(golden_section(l,r,eps,booth))
print(dichotomy_method(l,r,eps,booth))
print(fibonacci_search(l,r,eps,booth))
print(parabolic_minimization(l,r,eps,booth))
print(Brent(l,r,eps,booth))
print(function_call)
print(iteration_count)
```

```
3.001886325244824
2.9999
3.00354609929078
-6.53814195636451e-10
0.0011605522261880145
[24, 16, 17, 29, 10]
[12, 14, 15, 26, 10]
```

Заметим, что первые три метода (золотого сечения, дихотомии и Фибоначчи) исправно находят локальные минимумы на заданных промежутках. Метод Брента и параболической минимизации находят не глобальные минимумы, а локальные, это связано с тем, что они основываются на поиске минимума путем последовательного приближения к оптимальному значению, и если начальное приближение выбрано неправильно или функция имеет несколько локальных минимумов, то метод может сойтись к неправильному минимуму.
Также стоит отметить высокую эффективность нахождения приближенного значения минимума на промежутке методом золотого сечения, но, заметим, что этот метод (и некоторые другие) дают точные значения только на однопараметрических функциях.

Вывод

Метод дихотомии — это один из наиболее простых методов поиска корня уравнения. Он основан на принципе деления отрезка пополам до тех пор, пока не будет найден корень. Однако этот метод является не самым эффективным, так как он требует множества итераций для нахождения корня.

Метод золотого сечения — это более эффективный метод поиска корня уравнения, который основан на принципе деления отрезка в пропорции золотого сечения. Этот метод требует меньше итераций, чем метод дихотомии, но он также не является самым эффективным.

Метод Фибоначчи — это еще более эффективный метод поиска корня уравнения, который использует последовательность чисел Фибоначчи для нахождения корня. Этот метод требует меньше итераций, чем метод золотого сечения, но он также может быть медленным для некоторых функций.

Метод парабол — это метод, который использует квадратичную интерполяцию для нахождения корня. Этот метод может быть очень эффективным, но он также может быть неустойчивым для некоторых функций.

Метод комбинированного Брента — это метод, который комбинирует несколько методов поиска корня, включая метод дихотомии, метод золотого сечения и метод парабол. Этот метод является наиболее эффективным из всех рассмотренных методов.

Подводя итог, можно сказать, что для простых функций можно использовать метод дихотомии или золотого сечения, для более сложных функций лучше использовать метод Фибоначчи или комбинированный Брента. Если известно, что функция имеет локальный минимум или максимум, то лучше использовать метод парабол.