Национальный исследовательский университет ИТМО
Факультет информационных технологий и программирования
Прикладная математика и информатика

Методы одномерной оптимизации

Отчёт по лабораторной работе №2

Работу выполнили:
Гуревич Михаил
Трохан Александр

Преподаватель:
Москаленко Мария Александровна

Санкт-Петербург
2023

Лабораторная работа №2

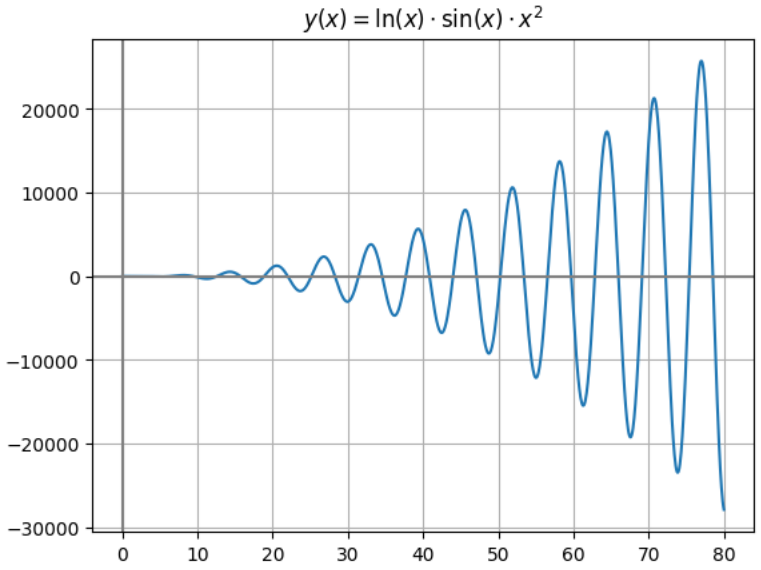
Выполнили: Гуревич Михаил и Трохан Александр, М32001

✅ Полный код лабораторной работы с комментариями можно найти на Github, а отчёт в Notion.

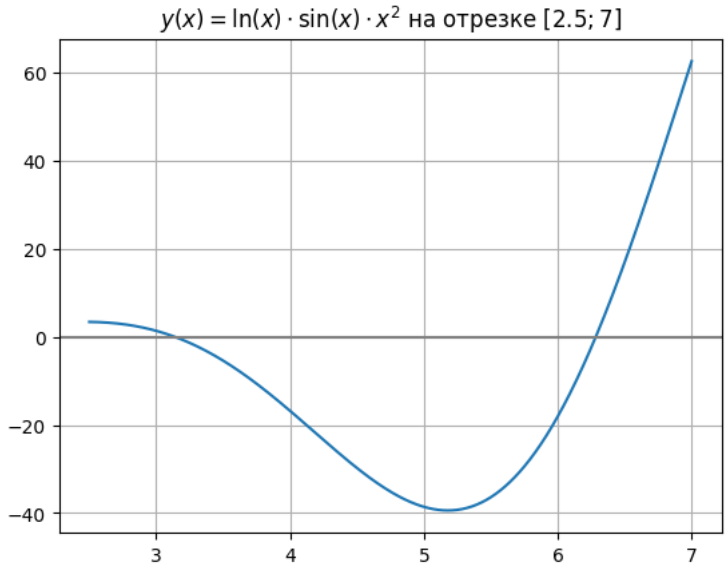
Дороги

Вариант 8

Так как освещение нужно установить в положении с наименьшей освещённостью, задача сводится к нахождению минимума функции $y(x) = \ln(x) \cdot \sin(x) \cdot x^2$. График этой функции:



Как видно из графика, функция обладает бесконечным числом локальных минимумов. Для корректной работы методов оптимизации функция должна быть **унимодальной** — то есть иметь единственный экстремум на заданном интервале (в нашем случае — локальный минимум). В противном же случае методы могут привести нас к нахождению локального, но не глобального минимума, или же к границам интервала (например, если будет присутствовать локальный максимум). Поэтому для решения задачи было решено рассматривать заданную функцию на отрезке $[2.5; 7]$. Её график:



Как можно заметить, на данном отрезке функция унимодальна.

Реализация алгоритмов одномерной минимизации

Для решения задачи реализуем следующие методы одномерной минимизации функции без производной:

> ⚠️ Для выполнения пункта 2 лабораторной работы методы должны уметь подсчитывать количество итераций и изменение исследуемого отрезка. Для простоты понимания эти части были опущены в реализациях, приведённых в отчёте.

Метод дихотомии

Описание метода: Пусть функция $f(x)$ задана на отрезке $[a; b]$. Возьмём две точки x_1 и x_2, симметричные относительно центра отрезка, которые принадлежат этому отрезку. Тогда $x_1 = \frac{a+b}{2} - \delta$, $x_2 = \frac{a+b}{2} + \delta$, δ можно выбрать любой из интервала $\left(0; \frac{b-a}{2}\right)$. Сравним значение функции в этих точках:

- Если $f(x_1) > f(x_2)$, то новым отрезком исследования становится $[x_1, b]$, и мы возвращаемся к первому шагу.
- Иначе новым отрезком исследования становится $[a, x_2]$, и мы также возвращаемся к первому шагу.

Данная процедура повторяется, пока не будет достигнута заданная точность, к примеру, пока длина отрезка не достигнет удвоенного значения заданной погрешности. Затем можно будет выбрать середину полученного отрезка как результат нахождения минимума.

Реализация метода:

```
def dichotomy(func, left, right, epsilon=1e-6, delta_func=lambda x, y: (y - x) / 4):
    while right - left > epsilon:
        middle = (left + right) / 2
        delta = delta_func(left, right)  # в нашей реализации delta = 1/4 отрезка
        x_1 = middle - delta
        x_2 = middle + delta
        if func(x_1) < func(x_2):
            right = x_2
        else:
            left = x_1
    return (left + right) / 2
```

Метод золотого сечения

Описание метода: Данный метод является оптимизацией метода дихотомии. В предыдущем методе нам приходилось вычислять значение функции дважды на каждой итерации. Теперь вместо случайного δ будем выбирать такие точки, что отрезок будет делиться в отношении золотого сечения, то есть:

$$\frac{b-a}{b-x_1} = \frac{b-a}{x_2-a} = \Phi$$

Таким образом, что $x_1 = b - \frac{b-a}{\Phi}$, $x_2 = a + \frac{b-a}{\Phi}$. В силу свойства золотого сечения, после выбора новых границ отрезка при вычислении новых значений для x_1 или x_2 одно из них совпадёт со значением, вычисленным на предыдущем шаге. Это позволяет снизить количество вычислений значение функции.

Реализация метода: Так как запросы к функциям в Python не кэшируются по умолчанию, мы можем написать для этого отдельный класс и в дальнейшем использовать его:

```
class FunctionCache:
    def __init__(self, func):
        self.func = func
        self.cache = {}

    def __call__(self, x):
        if x not in self.cache:
            self.cache[x] = self.func(x)
        return self.cache[x]
```

```
def golden_section(func, left, right, epsilon=1e-6):
    while right - left > epsilon:
        delta = (right - left) / PHI
        x_1 = right - delta
        x_2 = left + delta

        if func(x_1) < func(x_2):
            right = x_2
        else:
            left = x_1
    return (left + right) / 2
```

Метод Фибоначчи

Описание метода: Данный метод является оптимизацией метода золотого сечения и применяется, когда требуется выполнить не более n итераций. Так как $\Phi = \lim\limits_{n\to\infty} \frac{F_{n+1}}{F_n}$, то $x_1 = a + (b-a)\frac{F_{n-2}}{F_n}$, $x_2 = a + (b-a)\frac{F_{n-1}}{F_n}$, при этом после каждой итерации следует уменьшать n на 1. Алгоритм завершится за n итераций, и длина конечного интервала составит $\frac{b-a}{F_n}$.

Реализация метода:

```
def fibonacci(func, left, right, epsilon=1e-6):
    max_ = int((right - left) / epsilon)  # наибольшее требуемое число
    fib = [1, 1]
    while fib[-1] < max_:
        fib.append(fib[-1] + fib[-2])

    # Завершение должно произойти после корректного числа итераций
    n = len(fib) - 1
    if n <= 2:
        return (left + right) / 2

    while n > 2:
        x_1 = left + fib[-3] / fib[-1] * (right - left)
        x_2 = left + fib[-2] / fib[-1] * (right - left)

        if func(x_1) < func(x_2):
            right = x_2
```

```
        else:
            left = x_1
        fib.pop()
        n -= 1
    return (left + right) / 2
```

Метод парабол

Описание метода: На каждой итерации метода парабол строится парабола, которая проходит через три выбранные точки графика функции $f(x)$. При этом точка минимума параболы является приближением точки минимума исследуемой функции.

Выберем 3 точки из отрезка, на котором определена функция, таких, что $x_1 < x_2 < x_3$ и $f(x_1) > f(x_2) < f(x_3)$, и при этом минимум f(x) лежит внутри $[x_1; x_3]$. Пусть наша парабола задана уравнением $p(x) = ax^2 + bx + c$ и проходить через точки x_1, x_2, x_3. Тогда коэффициенты параболы можно найти из системы линейных уравнений:

$$ax_i^2 + bx_i + c_i = f(x_i), i \in \{1, 2, 3\}$$

Решая систему получим, что минимум этой параболы равен:

$$u = -\frac{b}{2a} = x_2 - \frac{(x_2 - x_1)^2(f(x_2) - f(x_3)) - (x_2 - x_3)^2(f(x_2) - f(x_1))}{2((x_2 - x_1)(f(x_2) - f(x_3)) - (x_2 - x_3)(f(x_2) - f(x_1)))}$$

В силу условия $f(x_1) > f(x_2) < f(x_3)$, точка u точно находится внутри отрезка $[x_1, x_3]$. Затем аналогично другим методам, сравнивая значения функции в точках x_2 и u определяем новые границы интервала (x_1и x_3).

На первой итерации можно взять x_1 и x_3 равными границам исходного интервала, так как в силу унимодальности функции любая точка между ними будет удовлетворять нужным условиям.

Реализация метода:

```
def parabola_minimum(f, x_1, x_2, x_3):
    return x_2 - ((x_2 - x_1) ** 2 * (f(x_2) - f(x_3)) - (x_2 - x_3) ** 2 * (f(x_2) - f(x_1))) / (2 * (x_2 - x_1) * (f(x_2) - f(x_3)) - (x_

def parabolic(func, left, right, epsilon=1e-6):
    x_2 = (left + right) / 2

    while right - left > epsilon:
        u = parabola_minimum(func, left, x_2, right)
        if func(x_2) < func(u):
            if x_2 < u:
                right = u
            else:
                left = u

        else:
            if x_2 < u:
                left = x_2
            else:
                right = x_2
            x_2 = u
    return (left + right) / 2
```

Комбинированный метод Брента

Описание метода: Данный метод объединяет в себе методы парабол и золотого сечения. Пусть a, c — границы отрезка, x — точка текущего наименьшего значения функции. На каждой итерации сначала используется метод парабол, и если:

- u попадает внутрь интервала и отстоит от границ интервала не менее, чем на заданную заранее величину допустимого отклонения;
- u отстоит от точки x не более, чем на половину от длины [пред]предыдущего шага.

то значение метода принимается, в противном случае новые значения для границ отрезка вычисляются с помощью метода золотого сечения большего из интервалов $[a, x]$ и $[x, c]$.

Реализация метода:

```
def brent(func, left, right, epsilon=1e-6):
    x = (left + right) / 2
    w = x  # предыдущее значение x или u
    v = x  # предпредыдущее значение x или u
    e = d = right - left  # удаление u от x

    while (right - left) > epsilon:
        g = e
        e = d
        accept = False
        if x != w and w != v and x != v and func(x) != func(w) and func(x) != func(v) and func(w) != func(v):
            u = parabola_minimum(func, x, w, v)
            if left + epsilon <= u <= right - epsilon and abs(u - x) < g / 2:
                accept = True
                d = abs(u - x)

        if not accept:
            if x < (left + right) / 2:
                delta = (right - x) / PHI
                u = right - delta  # золотое сечение [x, c]
                d = right - x
            else:
                delta = (x - left) / PHI
                u = left + delta  # золотое сечение [a, x]
                d = x - left

        if func(u) <= func(x):
            if u >= x:
                left = x
            else:
                right = x
            v = w
            w = x
            x = u
        else:
            if u >= x:
                right = u
            else:
                left = u
            if func(u) <= func(w):
                v = w
                w = u
            elif func(u) <= func(v) or v == x or v == w:
                v = u
    return (left + right) / 2
```

Решение задачи

Помимо реализации методов, требуется также решить саму задачу. После вычисления результата с точностью $\varepsilon = 0.01$, получим следующие результаты:

1. Метод дихотомии: $5.178...$
2. Метод золотого сечения: $5.181...$
3. Метод Фибоначчи: $5.182...$
4. Метод парабол: $5.179...$
5. Метод Брента: $5.179...$

Приближенное к реальному значение: 5.1789

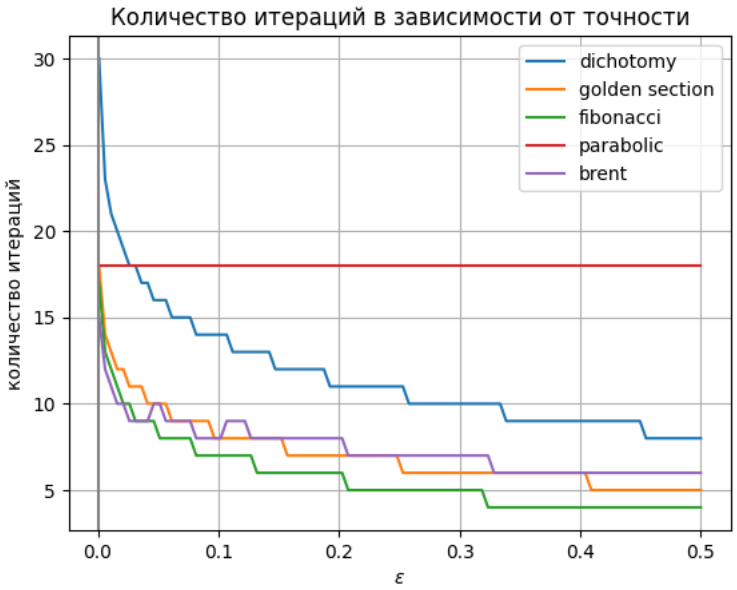
Как и ожидалось, методы дают результат, отличающийся от реального значения менее, чем на ε.

Ответ на задачу, посчитанный с помощью реализованных методов (с точностью до 10^{-10}): 5.17890565239635.

Сравнение методов

Зависимость числа итераций от точности

После реализации методов построим для каждого их них зависимость числа итераций от ε:



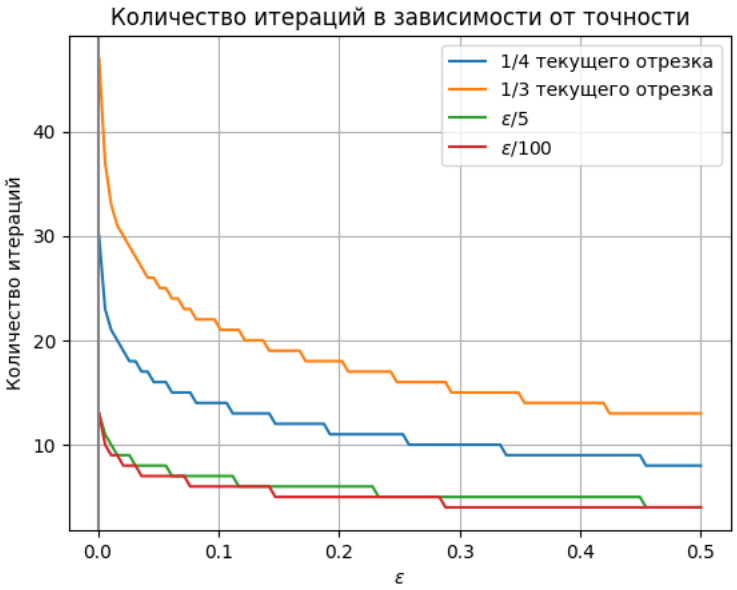
Выводы:

- В целом, для всех методов справедливо, что чем выше точность, тем больше итераций потребуется для вычисления значения. Это легко объяснить, так как количество итераций (условие остановки цикла) напрямую зависит от ε, и чем ε меньше, тем позднее остановится цикл.
- Метод Фибоначчи оказывается самым эффективным методом по числу итераций на заданной функции, однако при малых ε (близких к 0) метод Брента становится эффективнее.
- Метод парабол всегда даёт результат за одно и то же количество итераций для заданной функцию.
- Метод Брента — единственный метод, которому иногда может потребоваться больше итераций при большем ε. Это можно объяснить тем, что метод парабол в какой то момент вернул точку, которая не была принята, и следующая точка была вычислена методом золотого сечения, что увеличило число итераций по сравнению с меньшим ε.

Далее рассмотрим несколько интересных особенностей, вывод о которых нельзя сделать из данного графика.

Дополнительное сравнение: методы дихотомии

В результатах выше метод дихотомии показал себя хуже других методов. Проверим, есть ли зависимость между числом итераций метода и значения δ в нём. Для этого построим график зависимости числа итераций от ε для методов с разными δ :

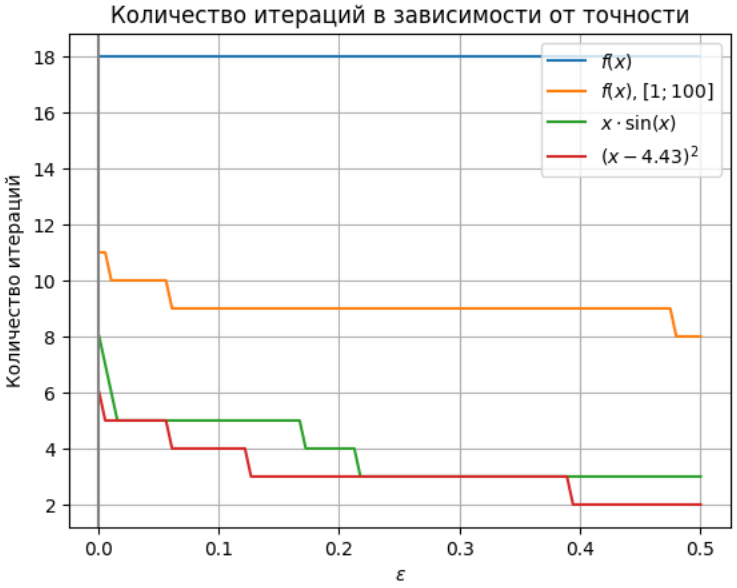


Вывод:

- Эффективность метода дихотомии действительно зависит от выбранного δ. Как можно судить по графику, чем меньше δ , тем более эффективно работает метод. Это можно объяснить тем, что отрезок при малых δ будет сходиться быстрее, так как будет мало отставать от середины отрезка, и за одну итерацию получится сократить отрезок почти вдвое.

Дополнительное сравнение: методы парабол

В отличие от других методов, метод парабол в исходном эксперименте всегда выполнялся за 18 итераций и не имел зависимость числа итераций от точности. Проверим, выполняется ли это для других функций и других вариантах исходного отрезка. Для этого построим график зависимости числа итераций от ε для различных функций и различных отрезков:



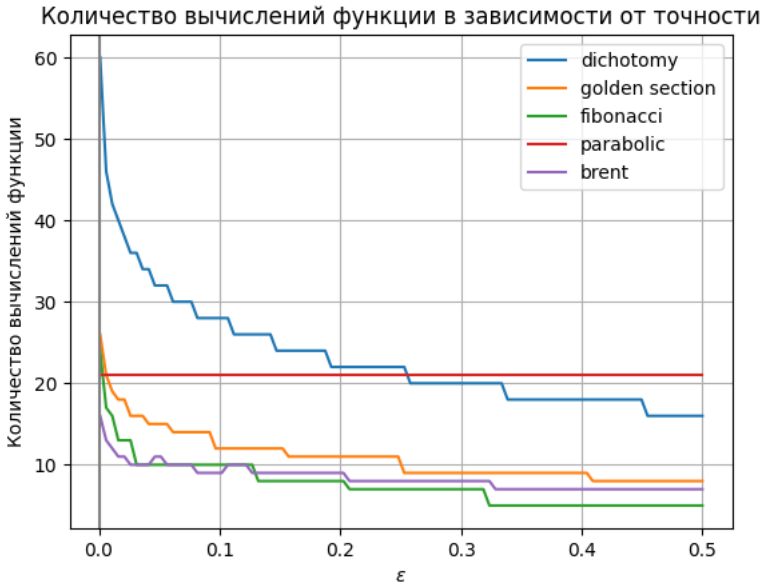
Вывод:

- Количество итераций метода парабол всё таки зависит от точности, однако это наблюдается не для всех функций и не для всех начальных отрезков. Так, использовав ту же функцию, но изменив отрезок (причём сделав его больше), можно

заметно сократить число итераций метода парабол. Для других функций число итераций и вовсе может свестись к двум. Получается, чем более “удачно” работает приближение метода парабол, тем меньше итераций он совершит.

Зависимость числа вычислений функции от точности

Построим для каждого их методов зависимость числа вычислений функции от ε:

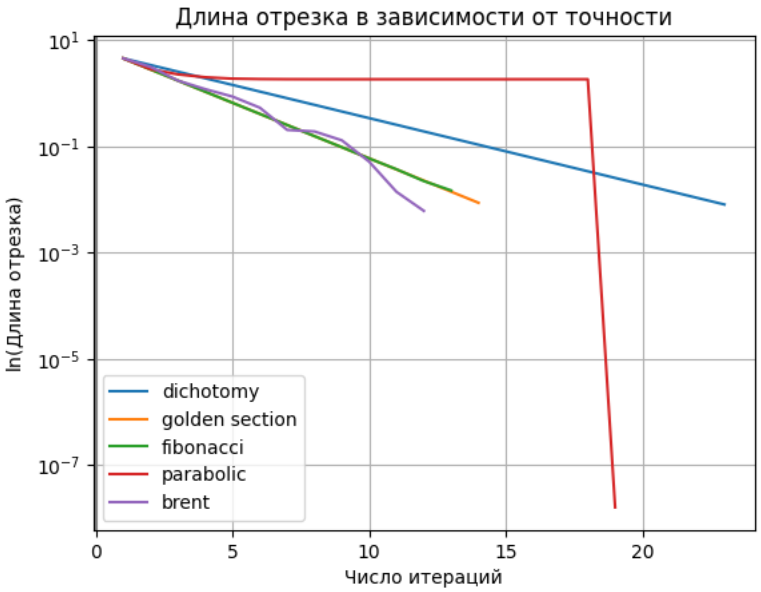


Выводы:

- Поведение данного графика повторяет поведение графика зависимости числа итераций от ε. Это можно объяснить тем, что все представленные методы имеют прямую зависимость количества вычислений функции (F) от числа итераций (I) — для метода дихотомии $F = 2I$, а для всех остальных $F = I + C, C \in \mathbb{Z}$, так как на каждой итерации возникает только одна новая точка.
- Наиболее эффективным по данному параметру также является метод Фибоначчи, однако для высокой точности ($\varepsilon < 0.1$) можно сказать, что более эффективным является метод Брента.

Скорость сходимости методов

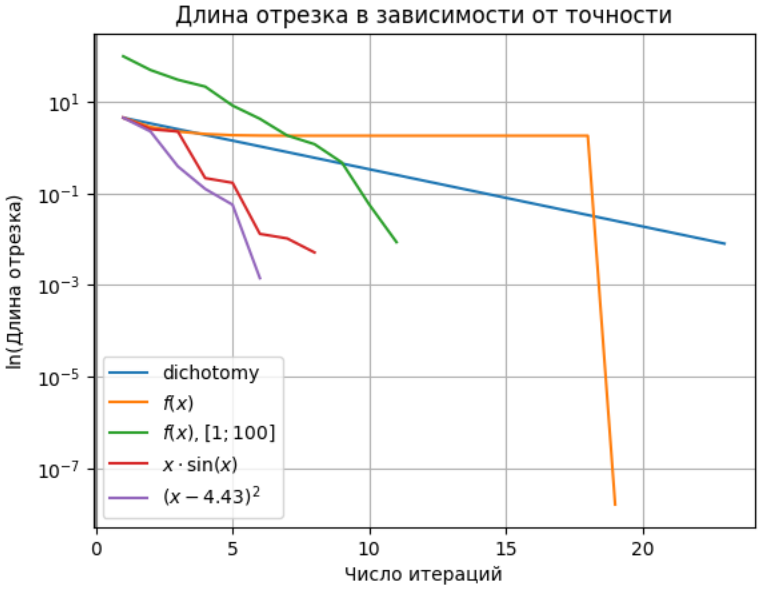
Найдём скорость сходимости методов, построив график зависимости длины текущего отрезка от номера итерации:



Выводы:

- Как было замечено в данной лекции, методы дихотомии, золотого сечения и Фибоначчи обладают линейной скоростью сходимости, а метод парабол — суперлинейной в малой окрестности точки минимума (то есть при большом числе итераций).
- Так как метод Брента является комбинацией методов параболы и золотого сечения, то он сходится нелинейно.
- Методы золотого сечения и Фибоначчи сходятся быстрее метода дихотомии при заданных параметрах.

Подробнее рассмотреть суперлинейную скорость сходимости метода парабол можно на следующем графике:



Выводы

Разные методы являются более или менее эффективными при разных параметрах точности, выбранной функции и выбранного параметра измерения эффективности. В целом, при больших ε (> 0.15) метод Фибоначчи показывает себя лучше других как по количеству итераций, так и по количеству вычислений функции. Однако при уменьшении ε на первое

место по обоим параметрам выходит комбинированный метод Брента, в частности за счёт высокой нелинейной скорости сходимости. Также стоит отметить, что многое зависит от поведения заданной функции, так как на некоторых функциях метод парабол (и соответсвенно метод Брента) работают значительно лучше.

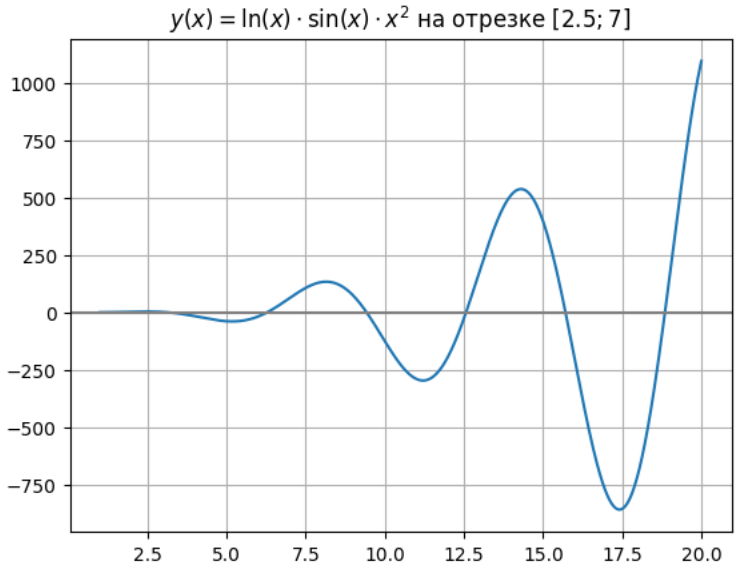
Минимизация многомодальных функций

Протестируем реализованные алгоритмы для задач минимизации многомодальных функций. Полиномы можно построить с помощью инструмента `Statistics: 4th order polynomial` в Desmos. Будем запускать алгоритмы на следующих функциях:

- Исходная функция $f(x)$, но на отрезке $[1; 20]$, где она многомодальна
- $f_1(x) = 0.0128x^5 - 0.2924x^4 + 2.2756x^3 - 6.6924x^2 + 4.3241x + 5.3353$ на отрезке $[0; 10]$
- $f_2(x) = 0.0633x^4 - 1.0101x^3 + 5.3388x^2 - 10.3956x + 5.4682$ на отрезке $[1; 7]$

Минимизация $f(x)$

График функции на отрезке $[1; 20]$:



С помощью производной найдём точки минимума функции на отрезке $[1; 20]$:

- $x_1 = 5.17891$
- $x_2 = 11.2077$
- $x_3 = 17.4129$ — абсолютный минимум на отрезке

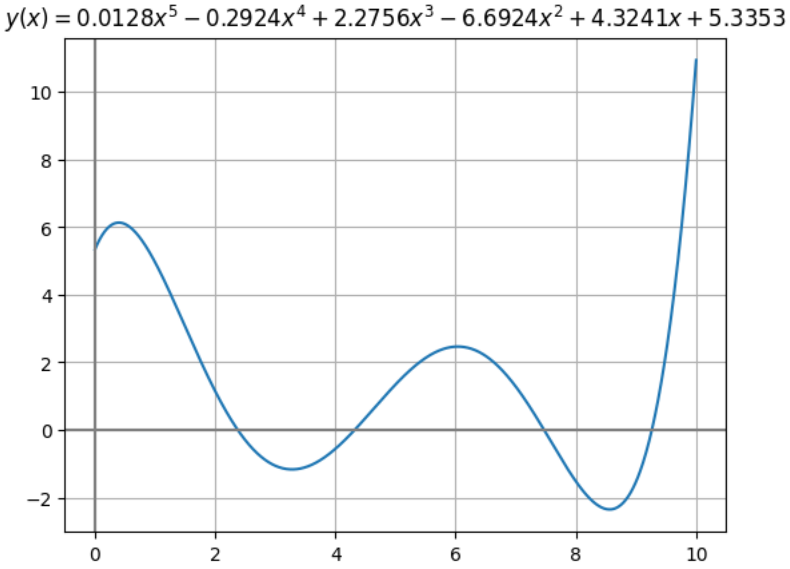
Результаты работы методов с точностью $\varepsilon = 0.001$:

1. Метод дихотомии: 11.207732973355204
2. Метод золотого сечения: 11.207601601817819
3. Метод Фибоначчи: 11.207767735631784
4. Метод парабол: 11.207747369994667
5. Метод Брента: 11.207725105475003

Все методы вернули одинаковую точку, которая действительно является точкой локального минимума, однако эта точка не является точкой абсолютного минимума.

Минимизация $f_1(x)$

График функции на отрезке $[0; 10]$:



С помощью производной найдём точки минимума функции на отрезке $[0; 10]$:

- $x_1 = 3.27902$
- $x_2 = 8.56103$ — абсолютный минимум на отрезке

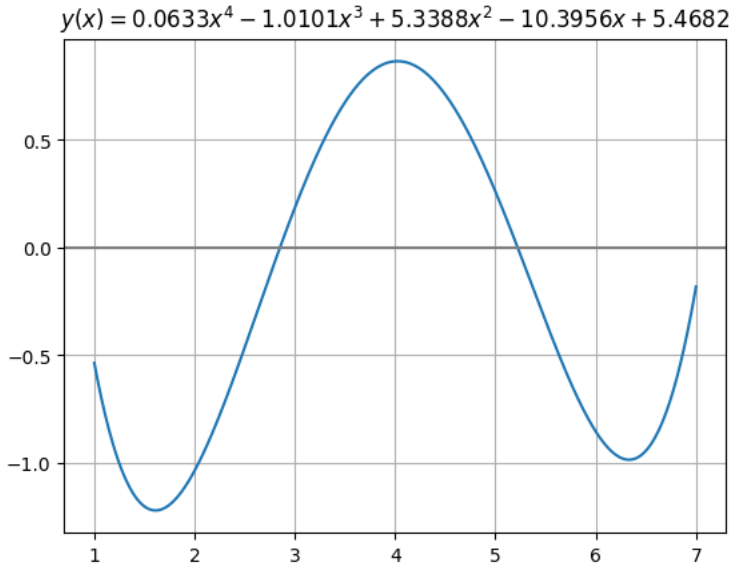
Результаты работы методов с точностью $\varepsilon = 0.001$:

1. Метод дихотомии: 3.2789924620315114
2. Метод золотого сечения: 3.2791031987336696
3. Метод Фибоначчи: 3.2788233144527688
4. Метод парабол: 3.2786611898181732
5. Метод Брента: 3.278871843148635

Все методы вернули одинаковую точку, которая действительно является точкой локального минимума, однако эта точка не является точкой абсолютного минимума.

Минимизация $f_2(x)$

График функции на отрезке $[1; 7]$:



С помощью производной найдём точки минимума функции на отрезке $[1; 7]$:

- $x_1 = 1.61098$ — абсолютный минимум на отрезке
- $x_2 = 6.33236$

Результаты работы методов с точностью $\varepsilon = 0.001$:

1. Метод дихотомии: 1.6110191812771233
2. Метод золотого сечения: 1.6110799664080753
3. Метод Фибоначчи: 1.6110864745011089
4. Метод парабол: 3.6213652185179455
5. Метод Брента: 1.6110111016023154

Все методы, кроме метода парабол вернули одинаковую точку, которая действительно является точкой абсолютного минимума. Метод парабол же вернул точку, которая вовсе не является точкой экстремума. Это может быть обусловлено тем, что на одном из этапов были выбраны точки $x_1 < x_2 < x_3$ удовлетворяющие условию $f(x_1) > f(x_2) < f(x_3)$, однако на этом отрезке функция не была унимодальной. В таком случае минимум данной параболы не будет иметь отношение к реальному минимуму заданной функции, и метод придёт к неправильному результату.

Корректность работы метода золотого сечения и метода Брента для многомодальной функции

Данные алгоритмы могут сработать некорректно, если сложится ситуация, в которой отрезок с минимумом функции был откинут на одной из итераций. Однако это может произойти только тогда, когда функция будет убывать на каком-либо другом отрезке. В этом случае метод вернёт минимум на этом отрезке, или, в худшем случае, его наименьшую границу, что в целом тоже можно считать приемлемым результатом.

Такое поведение можно наблюдать, если искать минимум функции f_2 на отрезке $[2; 7]$. В таком случае метод Брента вернёт точку, отличающуюся от 2 не более чем на ε, однако на отрезке есть минимум — точка 6.33.

Выводы

Нами были реализованы и протестированы методы одномерной оптимизации. Как показали эксперименты, самыми эффективными являются методы Фибоначчи и комбинированный метод Брента. Все эти методы работают для нахождения минимума унимодальных функций, однако могут найти и локальный минимум многомодальной функции, или, в худшем случае, одну из границ исходного отрезка. Однако в некоторых случаях многомодальной функции метод парабол может не сработать совсем, и вернуть точку, не являющуюся экстремумом или границей.