|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Отчёт

по лабораторной работе № 1

по дисциплине «Теория Систем и Системный Анализ»

**Тема: «Исследование методов прямого поиска экстремума унимодальной функции одного переменного»**

Вариант «данные из примера»

Выполнила: Чумичкина Евгения, студент группы ИУ8-32

Проверил: Коннова Н. С.,

доцент каф. ИУ8

г. Москва,

2020 г.

# 1. Цель работы

Исследовать функционирование и провести сравнительный анализ различных алгоритмов прямого поиска экстремума (пассивный поиск, метод дихотомии, золотого сечения, Фибоначчи) на примере унимодальной функции одного переменного.

# 2. Постановка задачи

Унимодальная функция: exp(-0.2x)sin(x) + 1

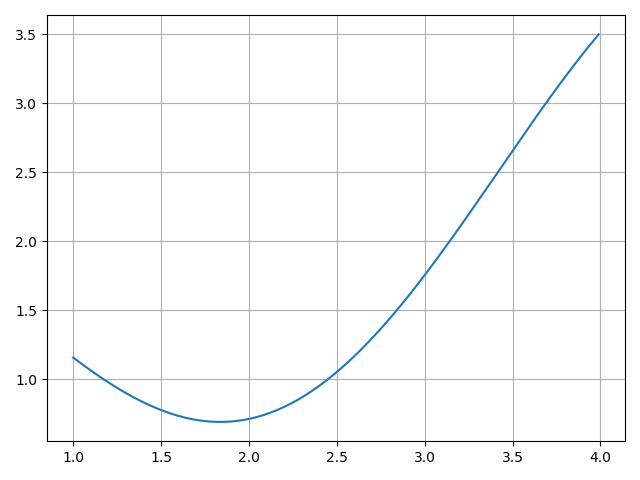
Отрезок поиска: 9,12

Методы поиска: оптимальный пассивный, Фибоначчи.

Наибольшая длина интервала неопределенности:

# 3. Ход работы

Рисунок 1 демонстрирует график унимодальной функции.



**Рисунок 1** – график функции

Реализация оптимального пассивного поиска:

Количество точек на 1 меньше, чем количество отрезков.

, где – наибольшая длина интервала неопределенности

Для интервала неопределенности 0.1 погрешность равна 0.05. Количество точек равно 59.

Точки расположены равномерно по отрезку, следовательно, координата точки с номером :

**Таблица 1** – применение оптимального пассивного поиска

|  |  |  |
| --- | --- | --- |
| N | x | f(x) |
| 1 | 9,05 | 1,06 |
| 2 | 9,1 | 1,05 |
| 3 | 9,15 | 1,04 |
| 4 | 9,2 | 1,04 |
| 5 | 9,25 | 1,03 |
| 6 | 9,3 | 1,02 |
| 7 | 9,35 | 1,01 |
| 8 | 9,4 | 1 |
| 9 | 9,45 | 1 |
| 10 | 9,5 | 0,99 |
| 11 | 9,55 | 0,98 |
| 12 | 9,6 | 0,97 |
| 13 | 9,65 | 0,97 |
| 14 | 9,7 | 0,96 |
| 15 | 9,75 | 0,95 |
| 16 | 9,8 | 0,95 |
| 17 | 9,85 | 0,94 |
| 18 | 9,9 | 0,94 |
| 19 | 9,95 | 0,93 |
| 20 | 10 | 0,93 |
| 21 | 10,05 | 0,92 |
| 22 | 10,1 | 0,92 |
| 23 | 10,15 | 0,91 |
| 24 | 10,2 | 0,91 |
| 25 | 10,25 | 0,91 |
| 26 | 10,3 | 0,9 |
| 27 | 10,35 | 0,9 |
| 28 | 10,4 | 0,9 |
| 29 | 10,45 | 0,89 |
| 30 | 10,5 | 0,89 |
| 31 | 10,55 | 0,89 |
| 32 | 10,6 | 0,89 |
| 33 | 10,65 | 0,89 |
| 34 | 10,7 | 0,89 |
| 35 | 10,75 | 0,89 |
| 36 | 10,8 | 0,89 |
| 37 | 10,85 | 0,89 |
| 38 | 10,9 | 0,89 |
| 39 | 10,95 | 0,89 |
| 40 | 11 | 0,89 |
| 41 | 11,05 | 0,89 |
| 42 | 11,1 | 0,89 |
| 43 | 11,15 | 0,89 |
| 44 | 11,2 | 0,9 |
| 45 | 11,25 | 0,9 |
| 46 | 11,3 | 0,9 |
| 47 | 11,35 | 0,9 |
| 48 | 11,4 | 0,91 |
| 49 | 11,45 | 0,91 |
| 50 | 11,5 | 0,91 |
| 51 | 11,55 | 0,92 |
| 52 | 11,6 | 0,92 |
| 53 | 11,65 | 0,92 |
| 54 | 11,7 | 0,93 |
| 55 | 11,75 | 0,93 |
| 56 | 11,8 | 0,93 |
| 57 | 11,85 | 0,94 |
| 58 | 11,9 | 0,94 |
| 59 | 11,95 | 0,95 |

Минимум функции в точке , значение функции в этой точке 0.8869

Реализация метода Фибоначчи:

На первой итерации необходимо определить положение левой и правой точки:

На следующих итерациях происходит исключение отрезка. Оставшаяся точка внутри нового отрезка поиска уже будет расположена в правильном отношении. Следовательно, необходимо найти только одну координату для новой точки.

После выполнения итераций (то есть, когда все точки будут расставлены) происходит вычисление интервала неопределенности. Если длина интервала неопределенности удовлетворяет условию, то вычисления прекращаются. Иначе, происходит переход к первой итерации с N, большим чем предыдущий на 1.

**Таблица 2** – последовательный поиск (метод Фибоначчи)

| Iters || Result || Partition |

| || x | f(x) | precision || left | right | f(left) | f(right) |

|--------||----------|----------|-----------||----------|----------|----------|----------|

| 1 || +2.0000 | +0.7141 | 1.0000 || +1.0000 | +3.0000 | +1.1585 | +1.7556 |

| 2 || +1.6000 | +0.7356 | 0.6000 || +1.0000 | +2.2000 | +1.1585 | +0.8008 |

| 3 || +1.7500 | +0.6983 | 0.3750 || +1.3750 | +2.1250 | +0.8498 | +0.7605 |

| 4 || +1.9231 | +0.6984 | 0.2308 || +1.6923 | +2.1538 | +0.7087 | +0.7749 |

| 5 || +1.8571 | +0.6927 | 0.1429 || +1.7143 | +2.0000 | +0.7041 | +0.7141 |

| 6 || +1.7941 | +0.6938 | 0.0882 || +1.7059 | +1.8824 | +0.7058 | +0.6941 |

| 7 || +1.8182 | +0.6927 | 0.0545 || +1.7636 | +1.8727 | +0.6966 | +0.6934 |

| 8 || +1.8427 | +0.6924 | 0.0337 || +1.8090 | +1.8764 | +0.6930 | +0.6937 |

В Таблице 2 показан результат (минимум в точке , а значение функции равно ) и итоговый отрезок неопределенности. Итерирование заканчивается, когда (т.е. отрезок неопределенности не больше 0.1).

Минимум функции в точке , значение функции в этой точке

В Таблице 3 показано выполнение метода Фибоначчи для фиксированного количества итераций (например, 8)

**Таблица 3** – последовательный поиск (метод Фибоначчи)

| N || Search part || Generated part |

| || start | end | length || left | right | f(left) | f(right) |

|-----||----------|----------|----------||----------|----------|----------|----------|

| 1 || +1.0000 | +4.0000 | +3.0000 || +2.1461 | +2.8539 | +0.7708 | +1.5207 |

| 2 || +1.0000 | +2.8539 | +1.8539 || +1.7079 | +2.1461 | +0.7054 | +0.7708 |

| 3 || +1.0000 | +2.1461 | +1.1461 || +1.4382 | +1.7079 | +0.8113 | +0.7054 |

| 4 || +1.4382 | +2.1461 | +0.7079 || +1.7079 | +1.8764 | +0.7054 | +0.6937 |

| 5 || +1.7079 | +2.1461 | +0.4382 || +1.8764 | +1.9775 | +0.6937 | +0.7085 |

| 6 || +1.7079 | +1.9775 | +0.2697 || +1.8090 | +1.8764 | +0.6930 | +0.6937 |

| 7 || +1.7079 | +1.8764 | +0.1685 || +1.7753 | +1.8090 | +0.6954 | +0.6930 |

| 8 || +1.7753 | +1.8764 | +0.1011 || +1.8090 | +1.8427 | +0.6930 | +0.6924 |

| 9 || +1.8090 | +1.8764 | +0.0674 || ---- | ---- | ---- | ---- |

Search part – отрезок поиска на начало итерации. Generated part – точки, установленные на отрезок поиска (и значения функции в них).

Строка является этапом обработки результата, а не итерацией, так как не происходит создания новых точек.

Реализация метода дихотомии:

Отрезок поиска делится пополам точкой x. Вычисляются значения функции на границах окрестности точки x: . Исключается левая половина, если значение функции в точке левой границы окрестности больше, чем в правой. Иначе, исключается правая половина. Эти действия повторяются до тех пор, пока отрезок поиска будет больше отрезка неопределенности.

**Таблица 4** – последовательный поиск (метод дихотомии)

| N || Search part || Generated part |

| || start | end | length || left | right | f(left) | f(right) |

|-----||----------|----------|----------||----------|----------|----------|----------|

| 1 || +1.0000 | +4.0000 | +3.0000 || +2.4900 | +2.5100 | +1.0430 | +1.0646 |

| 2 || +1.0000 | +2.5000 | +1.5000 || +1.7400 | +1.7600 | +0.6997 | +0.6970 |

| 3 || +1.7500 | +2.5000 | +0.7500 || +2.1150 | +2.1350 | +0.7558 | +0.7653 |

| 4 || +1.7500 | +2.1250 | +0.3750 || +1.9275 | +1.9475 | +0.6990 | +0.7023 |

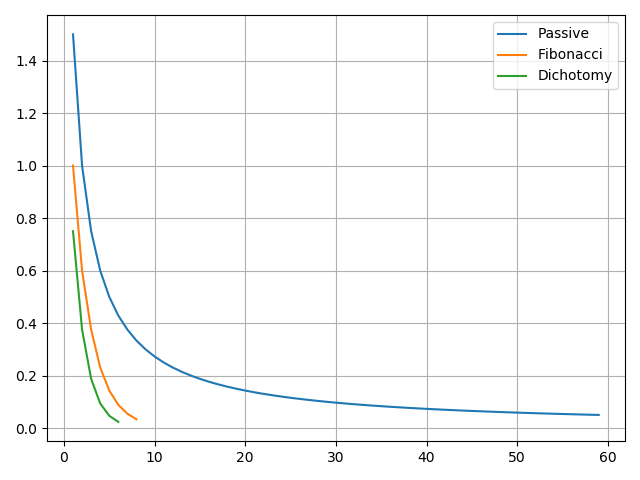
| 5 || +1.7500 | +1.9375 | +0.1875 || +1.8337 | +1.8538 | +0.6924 | +0.6926 |

| 6 || +1.7500 | +1.8438 | +0.0938 || ---- | ---- | ---- | ---- |

Итерирование завершается при

Минимум функции в точке: , а значение функции в этой точке

На Рисунке 2 приведен график зависимостей погрешности от количества итераций.



**Рисунок 2** – график зависимости погрешности от количества итераций

# 4. Выводы

В данной лабораторной работе был найден минимум унимодальной функции с помощью метода оптимального пассивного поиска и метода Фибоначчи. Из приведенного выше хода работы можно сделать вывод, что для достижения заданной погрешности методу Фибоначчи нужно меньше точек, чем методу оптимального пассивного поиска. Однако первый предполагает априорное задание количества шагов поиска.

Касательно метода дихотомии. Из результатов эксперимента можно сделать вывод о том, что скорость сходимости метода дихотомии выше, чем у метода Фибоначчи. А также для выполнения метода дихотомии не нужно предварительное задание количества точек.

# Приложение 1. Исходный код программы. Файл variant\_data\_example.py

**from** math **import** sqrt, sin  
**from** typing **import** Tuple  
  
  
**def** function(x):  
 *"""  
 Функция из варианта* **:type** *x: float* **:rtype***: float  
 """* **return** - sqrt(x) \* sin(x) + 2  
  
  
**def** get\_partition():  
 *"""* **:rtype***: Tuple[float, float]  
 """* **return** 1, 4

# Приложение 2. Исходный код программы. Файл Dichotomy.py

**from** ICalculation **import** ICalculation  
  
  
**class** Dichotomy(ICalculation):  
 EPSILON = 0.01  
  
 **def** \_\_init\_\_(self, function):  
 super().\_\_init\_\_(function)  
  
 self.log\_result\_partitions = **None** self.log\_partitions = **None** self.clear\_logs()  
  
 **def** clear\_logs(self):  
 super().clear\_logs()  
  
 self.log\_result\_partitions = []  
 self.log\_partitions = []  
  
 **def** eval\_by\_precision(self, begin, end, precision, min\_points=1):  
 length = end - begin  
  
 **while** length > precision \* 2:  
 point = begin + length / 2  
  
 self.log\_result\_partitions.append((begin, end))  
  
 left = point - Dichotomy.EPSILON  
 right = point + Dichotomy.EPSILON  
 left\_value = self.function(left)  
 right\_value = self.function(right)  
  
 **if** left\_value < right\_value:  
 end = point  
 **else**:  
 begin = point  
  
 length = end - begin  
  
 self.log\_partitions.append((left, right, left\_value, right\_value))  
  
 self.log\_result\_partitions.append((begin, end))  
 self.log\_partitions.append((**None**, **None**, **None**, **None**))  
  
 **return** (begin + end) / 2, len(self.log\_partitions) - 1, length/2

# Приложение 3. Исходный код программы. Файл Fibonacci.py

**from** ICalculation **import** ICalculation  
  
  
**class** Fibonacci(ICalculation):  
 **class** NumberCache:  
 @staticmethod  
 **def** generator(length):  
 *"""  
 Генератор чисел Фибоначчи  
 """* x, y = 1, 1  
 **for** i **in** range(0, length):  
 **yield** x  
 x, y = y, x + y  
  
 **def** \_\_init\_\_(self, length):  
 *"""  
 Кэшированный контейнер чисел Фибоначчи  
 """* self.numbers = list(self.generator(length))  
  
 **def** get\_left\_offset\_multiplier(self, index):  
 *"""  
 Получить коэффициент отступа от левой границы для точки слева  
 """* **return** self.numbers[index - 2] / self.numbers[index]  
  
 **def** get\_right\_offset\_multiplier(self, index):  
 *"""  
 Получить коэффициент отступа от левой границы для точки справа  
 """* **return** self.numbers[index - 1] / self.numbers[index]  
  
 **class** Partition:  
 **def** \_\_init\_\_(self, begin=**None**, end=**None**, begin\_value=**None**, end\_value=**None**):  
 *"""  
 Структура, содержащая границы отрезка и значения функции на границах отрезка  
 """* self.begin = begin  
 self.end = end  
 self.begin\_value = begin\_value  
 self.end\_value = end\_value  
  
 **def** recalculate(self, function):  
 *"""  
 Пересчитать значения функции на границах отрезка* **:param** *function: Функция  
 """* self.begin\_value = function(self.begin)  
 self.end\_value = function(self.end)  
  
 **def** swap(self):  
 *"""  
 Поменять начало и конец отрезка и значения функции на них местами  
 """* self.begin, self.end = self.end, self.begin  
 self.begin\_value, self.end\_value = self.end\_value, self.begin\_value  
  
 @property  
 **def** is\_empty(self):  
 **return** self.begin **is None or** self.end **is None  
  
 def** \_\_init\_\_(self, function):  
 *"""  
 Поиск минимума методом Фибоначчи  
 """* super().\_\_init\_\_(function)  
  
 self.log\_result\_partitions = **None** self.log\_partitions = **None** self.clear\_logs()  
  
 **def** clear\_logs(self):  
 super().clear\_logs()  
  
 self.log\_result\_partitions = []  
 self.log\_partitions = []  
  
 **def** eval\_by\_precision(self, begin, end, precision, min\_points=1):  
 **return** super().eval\_by\_precision(begin, end, precision, min\_points)  
  
 **def** eval\_by\_points(self, begin, end, points):  
 fibonacci\_cache = self.NumberCache(points + 3) *# Точка ожидает существование k-1, k-2 элемента* part = self.Partition()  
  
 *# Первая итерация* length = end - begin  
  
 *# Сдвиг до левой и правой точек, соответственно* left\_offset = fibonacci\_cache.get\_left\_offset\_multiplier(points + 2) \* length  
 right\_offset = fibonacci\_cache.get\_right\_offset\_multiplier(points + 2) \* length  
  
 *# Запомнить обе точки, вычислить значения функции в них* part.begin = begin + left\_offset  
 part.end = begin + right\_offset  
 part.recalculate(self.function)  
  
 *# Запомнить итерацию* self.log\_result\_partitions.append((begin, end))  
 self.log\_partitions.append((part.begin, part.end, part.begin\_value, part.end\_value))  
  
 **for** i **in** range(points+1, 1, -1):  
 **if** part.begin\_value < part.end\_value:  
 *# Сузить отрезок поиска* end = part.end  
 length = end - begin  
  
 part.swap()  
  
 *# Изменить левую границу нового отрезка* left\_offset = fibonacci\_cache.get\_left\_offset\_multiplier(i) \* length  
 part.begin = begin + left\_offset  
  
 **if** i != 2:  
 part.begin\_value = self.function(part.begin)  
  
 **else**:  
 *# Сузить отрезок поиска* begin = part.begin  
 length = end - begin  
  
 part.swap()  
  
 *# Изменить правую границу нового отрезка* right\_offset = fibonacci\_cache.get\_right\_offset\_multiplier(i) \* length  
 part.end = begin + right\_offset  
  
 **if** i != 2:  
 part.end\_value = self.function(part.end)  
  
 self.log\_result\_partitions.append((begin, end))  
 self.log\_partitions.append((part.begin, part.end, part.begin\_value, part.end\_value))  
  
 length = end - begin  
 **return** dict(  
 minimum=begin + length / 2,  
 minimum\_value=self.function(begin + length / 2),  
 precision=length / 2,  
 )

# Приложение 4. Исходный код программы. Файл ICalculation.py

**from** typing **import** Dict  
  
  
**class** ICalculation:  
 **def** \_\_init\_\_(self, function):  
 *"""  
 Метод поиска минимума* **:param** *function: Функция  
 """* self.log\_precision = **None** self.function = function  
  
 self.clear\_logs()  
  
 **def** clear\_logs(self):  
 self.log\_precision = []  
  
 **def** eval\_by\_precision(self, begin, end, precision, min\_points=1):  
 *"""  
 Поиск минимума. Через заданную погрешность. Перебор количества точек* **:type** *begin: float* **:param** *begin: Начало отрезка* **:type** *end: float* **:param** *end: Конец отрезка поиска* **:type** *precision: float* **:param** *precision: Длина отрезка неопределенности* **:type** *min\_points: int* **:param** *min\_points: Минимальное количество точек* **:rtype***: Tuple[float, int]* **:return***: Значение аргумента с предполагаемым минимумом, количество точек для достижения указанного отрезка  
 неопределенности  
 """* points = min\_points  
 current\_precision = **None** *# Текущая точность* minimum = **None  
 while** current\_precision **is None or** current\_precision > precision:  
 result\_data = self.eval\_by\_points(begin, end, points)  
  
 current\_precision = result\_data[**'precision'**]  
 minimum = result\_data[**'minimum'**]  
 minimum\_value = self.function(minimum)  
 left = result\_data[**"minimum"**] - current\_precision  
 right = result\_data[**"minimum"**] + current\_precision  
 left\_value = self.function(left)  
 right\_value = self.function(right)  
  
 self.log\_precision.append(dict(points=points,  
 precision=current\_precision,  
 minimum=minimum,  
 minimum\_value=minimum\_value,  
 left=left,  
 right=right,  
 left\_value=left\_value,  
 right\_value=right\_value))  
  
 points += 1  
  
 **return** minimum, points - 1, current\_precision  
  
 **def** eval\_by\_points(self, begin, end, points):  
 *"""  
 Поиск минимума. Через заданное количество точек* **:type** *begin: float* **:param** *begin: Начало отрезка поиска* **:type** *end: float* **:param** *end: Конец отрезка поиска* **:type** *points: int* **:param** *points: Количество точек* **:rtype***: Dict[str, any]* **:return***: Значение аргумента с предполагаемым минимумом, точность  
 """* **pass**

# Приложение 5. Исходный код программы. Файл Sequential.py

**from** ICalculation **import** ICalculation  
  
  
**class** Sequential(ICalculation):  
 **def** \_\_init\_\_(self, function):  
 *"""  
 Метод оптимального поиска  
 """* super().\_\_init\_\_(function)  
  
 self.log\_points = **None** self.clear\_logs()  
  
 **def** clear\_logs(self):  
 super().clear\_logs()  
  
 self.log\_points = []  
  
 **def** eval\_by\_precision(self, begin, end, precision, min\_points=1):  
 points = int((end - begin) / precision - 1)  
 **return** super().eval\_by\_precision(begin, end, precision, points)  
  
 **def** eval\_by\_points(self, begin, end, points):  
 *# Длина отрезка* step = (end - begin) / (points + 1) *# +1, т.к. отрезков на 1 больше, чем точек  
  
 # Минимум (x, f(x)) инициализируется первой точкой* minimum = begin + step  
 minimum\_value = self.function(minimum)  
  
 self.log\_points.append((minimum, minimum\_value))  
  
 **for** i **in** range(2, points + 1): *# начиная со второй точки, заканчивая points точкой* current = begin + i \* step *# x* current\_value = self.function(current) *# f(x)* self.log\_points.append((current, current\_value))  
  
 **if** minimum\_value > current\_value:  
 minimum = current  
 minimum\_value = current\_value  
  
 **return** dict(  
 minimum=minimum,  
 minimum\_value=minimum\_value,  
 precision=step  
 )

# Приложение 6. Исходный код программы. Файл main.py

**from** Dichotomy **import** Dichotomy  
**from** Fibonacci **import** Fibonacci  
**from** Sequential **import** Sequential  
**from** variant\_data\_example **import** \*  
  
  
**def** input\_precision():  
 *"""  
 Получение точности из консоли. С обработкой неправильного ввода* **:rtype***: float  
 """* precision = **None  
  
 while** precision **is None**:  
 print(**'Enter precision: '**, end=**''**)  
 **try**:  
 precision = float(input())  
  
 **if** precision <= 0:  
 print(**'Excepted positive number. Try again.'**)  
 precision = **None  
 except** ValueError:  
 print(**'Excepted positive number. Try again.'**)  
  
 **return** precision  
  
  
**def** print\_logs\_sequential(log):  
 print(**'| N '**,  
 **' x '**,  
 **' f(x) '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
 print(**'|-----'**,  
 **'----------'**,  
 **'----------'**,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
 **for** i, v **in** enumerate(log):  
 print(**f'| {**i + 1**: ^03} '**,  
 **f' {**v[0]**: >+08.4f} '**,  
 **f' {**v[1]**: >+08.4f} '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
  
**def** print\_logs\_points(log):  
 print(**'| Iters '**,  
 **' Result '**,  
 **' Partition '**,  
 sep=**'||'**,  
 end=**'|\n'**)  
 print(**'| '**,  
 **'| x '**,  
 **' f(x) '**,  
 **' precision '**,  
 **'| left '**,  
 **' right '**,  
 **' f(left) '**,  
 **' f(right) '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
 print(**'|--------'**,  
 **'|----------'**,  
 **'----------'**,  
 **'-----------'**,  
 **'|----------'**,  
 **'----------'**,  
 **'----------'**,  
 **'----------'**,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
 **for** v **in** log:  
 print(**f'| {**v[**"points"**]**: ^06} '**,  
 **f'| {**v[**"minimum"**]**: >+08.4f} '**,  
 **f' {**v[**"minimum\_value"**]**: >+08.4f} '**,  
 **f' {**v[**"precision"**]**: >09.4f} '**,  
 **f'| {**v[**"left"**]**: >+08.4f} '**,  
 **f' {**v[**"right"**]**: >+08.4f} '**,  
 **f' {**v[**"left\_value"**]**: >+08.4f} '**,  
 **f' {**v[**"right\_value"**]**: >+08.4f} '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
  
**def** print\_logs(log\_result\_partitions, log\_partitions):  
 print(**'| N '**,  
 **'| Search part '**,  
 **'| Generated part '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
 print(**'| '**,  
 **'| start '**,  
 **' end '**,  
 **' length '**,  
 **'| left '**,  
 **' right '**,  
 **' f(left) '**,  
 **' f(right) '**,  
 sep=**'|'**,  
 end=**'|\n'**)  
 print(**'|-----'**,  
 **'|----------'**,  
 **'----------'**,  
 **'----------'**,  
 **'|----------'**,  
 **'----------'**,  
 **'----------'**,  
 **'----------'**,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
 **for** i, v **in** enumerate(log\_result\_partitions):  
 local = log\_partitions[i]  
  
 **if** i == len(log\_result\_partitions) - 1:  
 *# На последней итерации не генерируются новые точки и не считается значение функций в них* generated = (  
 **'| ---- '**,  
 **' ---- '**,  
 **' ---- '**,  
 **' ---- '**,  
 )  
 **else**:  
 generated = (**f'| {**local[0]**: >+08.4f} '**,  
 **f' {**local[1]**: >+08.4f} '**,  
 **f' {**local[2]**: >+08.4f} '**,  
 **f' {**local[3]**: >+08.4f} '**,)  
  
 print(**f'| {**i + 1**: ^03} '**,  
 **f'| {**v[0]**: >+08.4f} '**,  
 **f' {**v[1]**: >+08.4f} '**,  
 **f' {**v[1] - v[0]**: >+08.4f} '**,  
 \*generated,  
 sep=**'|'**,  
 end=**'|\n'**)  
  
  
**def** eval\_advanced\_search(template, undefined\_length, clear\_logs=**True**):  
 name = template.\_\_name\_\_  
  
 search = template(function)  
 value, points, precision = search.eval\_by\_precision(  
 \*get\_partition(),  
 undefined\_length,  
 )  
  
 print(**f'\n================================[INFO ({**name**} search)]================================'**)  
 print\_logs\_points(search.log\_precision)  
  
 print(**f'\n============[RESULT ({**name**} search)]============'  
 f'\nX: {**value**:0.4f}'  
 f'\nF(X): {**function(value)**:0.4f}\nPoints: {**points**}'  
 f'\nPrecision: {**precision**:0.4f}\n\n'**)  
  
 **if** clear\_logs:  
 search.clear\_logs()  
  
 search.eval\_by\_points(  
 \*get\_partition(),  
 points  
 )  
  
 print(**f'========================[ INFO ({**name**} search. Points: {**points**}) ]========================'**)  
 print\_logs(search.log\_result\_partitions, search.log\_partitions)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 *# undefined\_length = input\_precision()* undefined\_length = 0.1  
  
 eval\_advanced\_search(Fibonacci, undefined\_length / 2)  
 eval\_advanced\_search(Dichotomy, undefined\_length / 2, **False**)  
  
 sequential = Sequential(function)  
 value, points, \_ = sequential.eval\_by\_precision(  
 \*get\_partition(),  
 undefined\_length / 2,  
 )  
  
 print(**f'\n==[ INFO (Sequential search. Points: {**points**}) ]=='**)  
 print\_logs\_sequential(sequential.log\_points)  
  
 print(**f'\n==[RESULT (Sequential search)]== \nX: {**value**:0.4f}'  
 f'\nF(X): {**function(value)**:0.4f}\nPoints: {**points**}'  
 f'\nPrecision: {**undefined\_length / 2**:0.4f}\n\n'**)  
  
 *# %%* **import** matplotlib.pyplot **as** plot  
 **import** numpy **as** np  
  
 x = np.arange(\*get\_partition(), 0.01)  
 y = np.array([function(i) **for** i **in** x])  
  
 plot.plot(x, y)  
 plot.grid()  
 plot.show()