**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики Изображение выглядит как текст, коллекция картинок, посуда

Автоматически созданное описание****УЧЕБНЫЙ ЦЕНТР ФИТиП**

Группа M32111 К работе допущен Студент Гаврилов А. Беляков Е. Работа выполнена Преподаватель Дорофеева Ю. А. Отчет принят

Отчет по лабораторной работе №1

**Алгоритмы одномерной минимизации функции без производной**

**1. Цель работы.**

1. Реализовать следующие алгоритмы:
   * + Метод дихотомии;
     + Метод золотого сечения;
     + Метод Фиббоначи;
     + Метод парабол;
     + Комбинированный метод Брента;
2. Протестировать алгоритмы на заданном полиноме;
3. Сравнить количество итераций для каждого метода;

**2. Метод дихотомии:**

**Описание:**

*"Если непрерывная функция на концах некоторого интервала имеет значения разных знаков, то внутри этого интервала у нее есть корень (как минимум, один, но м.б. и несколько)"*.

Обобщенный алгоритм:

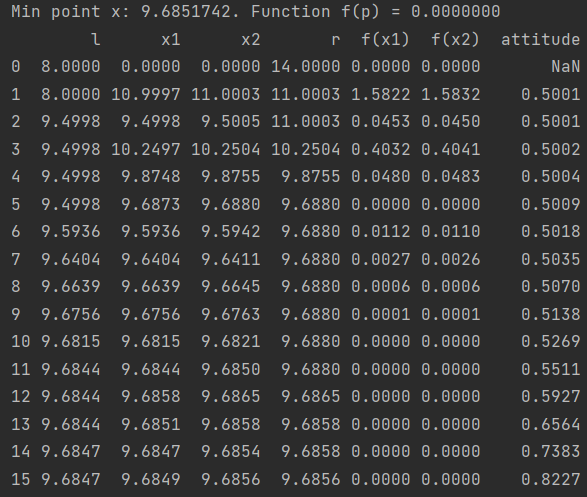
1. Задать начальный интервал [X_{left}..X_{right}];
2. Убедиться, что на концах функция имеет разный знак;
3. Повторять
   1. выбрать внутри интервала точку X;
   2. сравнить знак функции в точке X со знаком функции в одном из концов;
   3. если совпадает, то переместить этот конец интервала в точку X,
   4. иначе переместить в точку X другой конец интервала;
4. Повторять, пока не будет достигнута нужная точность.

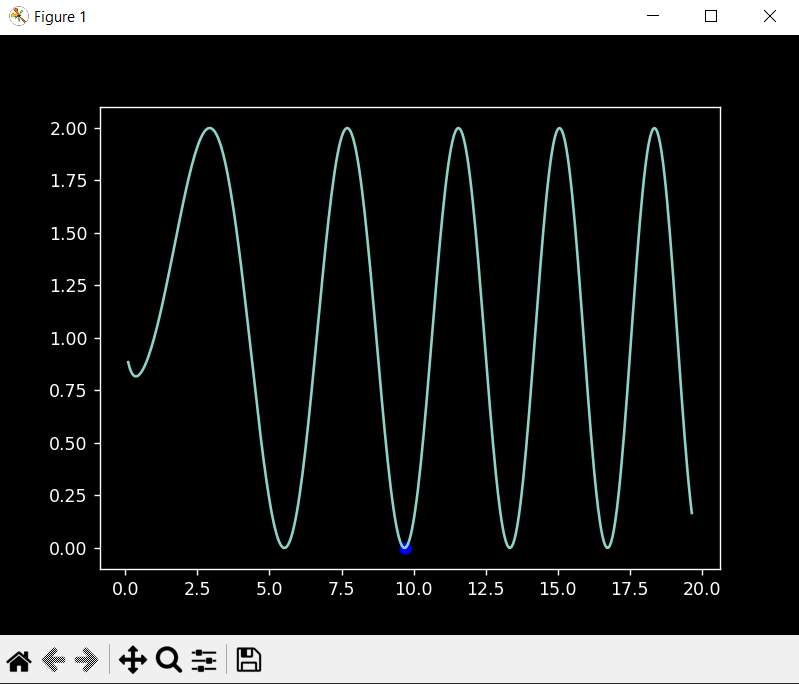
Варианты метода дихотомии различаются выбором точки деления.

**Код на языке Python:**

@painter\_decorator  
def dichotomous\_search(l, r, eps, s=1):  
 debug\_start(l, r, \*[0, 0, 0, 0])  
 delta = eps / 3  
  
 while r - l > eps:  
 m = (l + r) \* 0.5  
 x1 = m - delta  
 x2 = m + delta  
 if s \* f(x1) > s \* f(x2):  
 l = x1  
 else:  
 r = x2  
 debug\_tick(l, r, r - l, \*[x1, x2, f(x1), f(x2)])  
 return (l + r) \* 0.5

**Тесты: (8, 14, 0.001)**

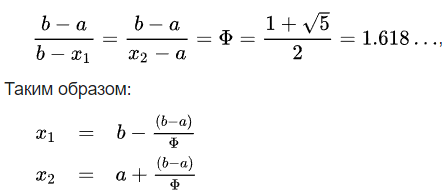


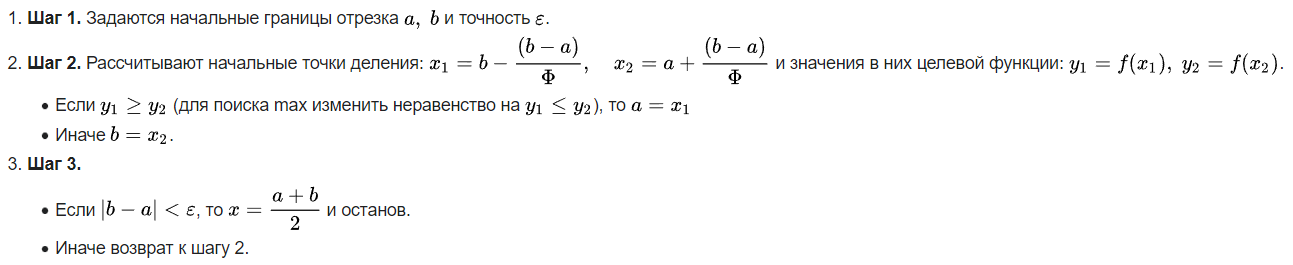


**3. Метод золотого сечения**

**Описание:**

Для того, чтобы найти неопределённое значение этой функции на заданном отрезке, отвечающее критерию поиска (пусть это будет минимум), рассматриваемый отрезок делится в пропорции золотого сечения в обоих направлениях, то есть выбираются две точки �1и �2такие, что:



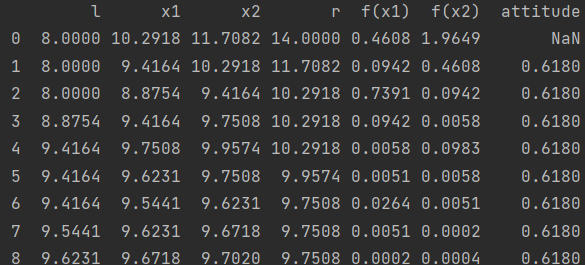
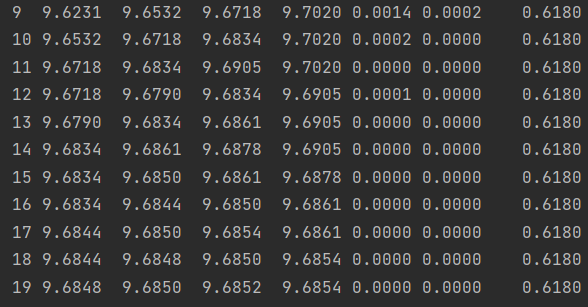


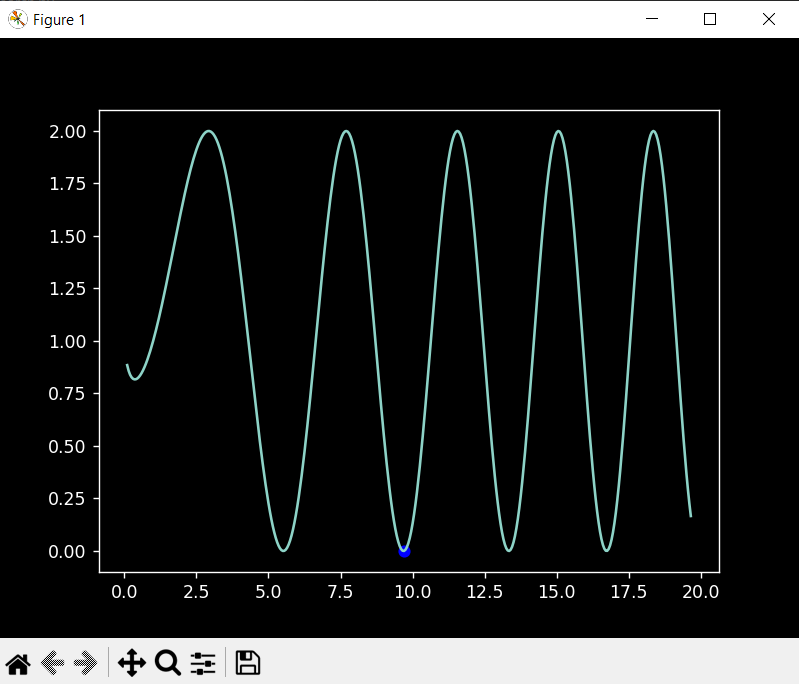
**Код на языке Python:**

@painter\_decorator  
def golden\_section\_search(l, r, eps, s=1):  
 d = lambda: gr \* (r - l)  
 gr = (math.sqrt(5) - 1) \* .5  
 x1, x2 = r - d(), l + d()  
 f1, f2 = s \* f(x1), s \* f(x2)  
 debug\_start(l, r, \*[x1, x2, f1, f2])  
 while r - l > eps:  
 if f1 > f2:  
 l, x1, f1 = x1, x2, f2  
 x2 = l + d()  
 f2 = s \* f(x2)  
 else:  
 r, x2, f2 = x2, x1, f1  
 x1 = r - d()  
 f1 = s \* f(x1)  
 debug\_tick(l, r, r - l, \*[x1, x2, f1, f2])  
  
 return (l + r) \* 0.5

**Тесты: (8, 14, 0.001)**

**Min point x: 9.6851149. Function f(p) = 0.0000000**



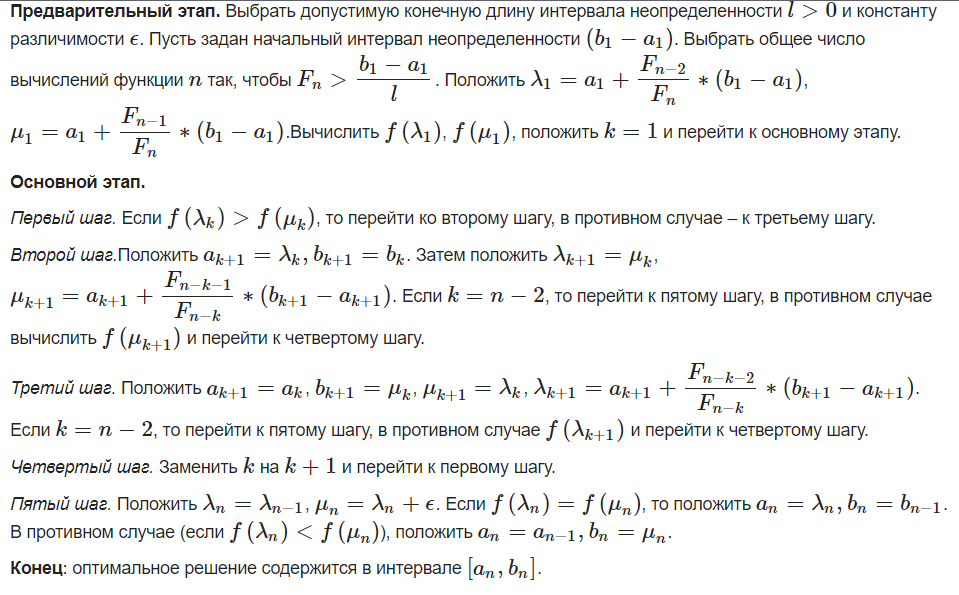


**4. Метод Фибоначчи:**

**Описание:**

Метод Фибоначчи— это улучшение реализации поиска с помощью золотого сечения, служащего для нахождения минимума/максимума функции.

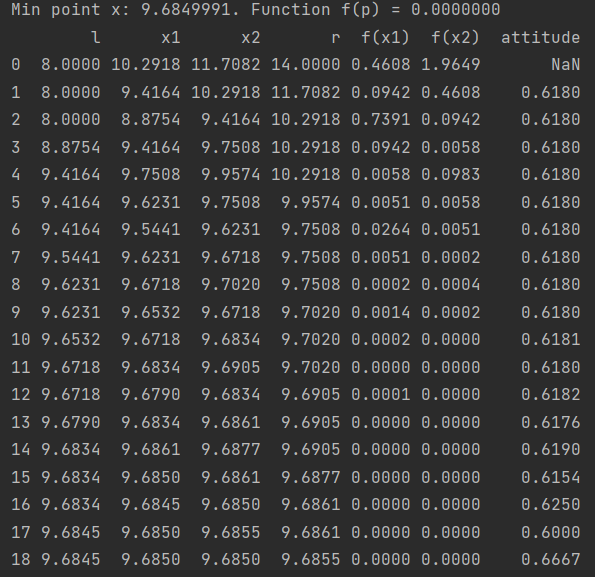
Подобно методу золотого сечения, он требует двух вычислений функции на первой итерации, а на каждой последующей только по одному. Однако этот метод отличается от метода золотого сечения тем, что коэффициент сокращения интервала неопределенности меняется от итерации к итерации.

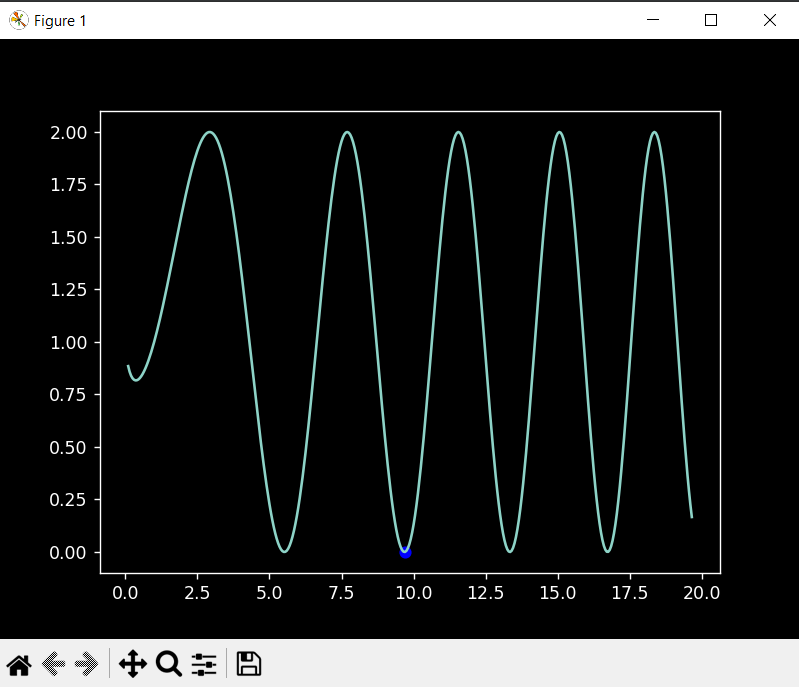


**Код на языке Python:**

@painter\_decorator  
def fibonacci\_search(l, r, eps, s=1):  
 fib = np.array([1, 1])  
 while fib[-1] <= (r - l) / eps:  
 fib = np.append(fib, fib[-1] + fib[-2])  
  
 d = lambda k: (r - l) \* (fib[n - k] / fib[n - k + 1])  
 n = len(fib) - 1  
 x1, x2 = r - d(1), l + d(1)  
 f1, f2 = s \* f(x1), s \* f(x2)  
 debug\_start(l, r, \*[x1, x2, f1, f2])  
 for k in range(1, n):  
 if f1 > f2:  
 l, x1, f1 = x1, x2, f2  
 x2 = l + d(k)  
 f2 = s \* f(x2)  
 else:  
 r, x2, f2 = x2, x1, f1  
 x1 = r - d(k)  
 f1 = s \* f(x1)  
 debug\_tick(l, r, r - l, \*[x1, x2, f1, f2])  
 return (l + r) \* 0.5

**Тесты: (8, 14, 0.001, s=1)**





**5. Метод Парабол:**

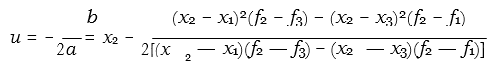
**Описание:**

В методе парабол предлагается аппроксимировать оптимизируемую функцию f (x) с помощью квадратичной функции: p(x) = ax2 + bx + c.

Для того, чтобы найти коэффициенты аппроксимируемой параболы a, b, c необходимо решить систему линейных уравнений: ax2 + bxi + ci = fi = f (xi), i = 1, 2, 3.

Для того, чтобы получить систему, используем три точки: x1<x2<x3,xmin ∈ [x1, x3].

Решив эту систему, получим, что минимум такой параболы равен:



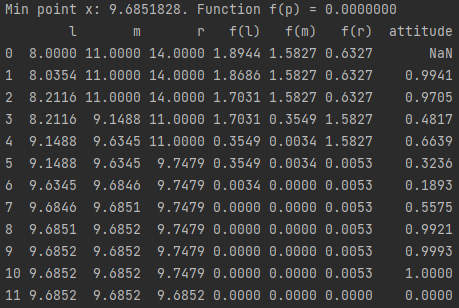
Если f2<f1 и f2<f3, то точка m гарантированно попадает в интервал [x1, x3].

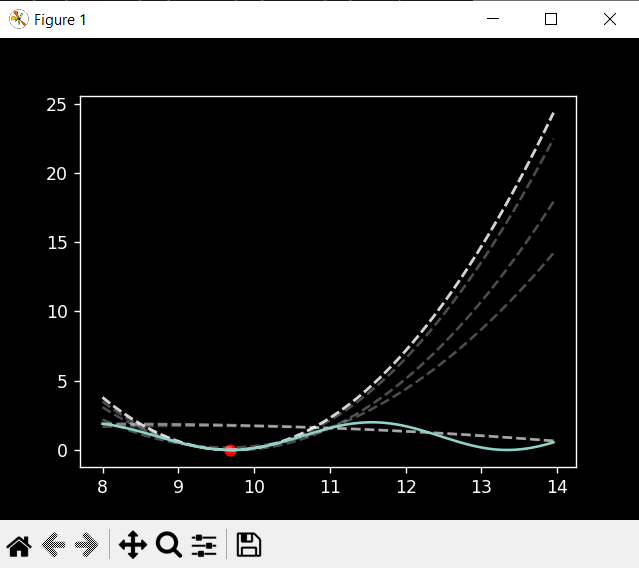
Таким образом, внутри интервала у нас определены две точки x2 и u, с помощью сравнения значений функции f в которых можно сокращать интервалы поиска.

**Код на языке Python:**

@painter\_decorator  
def parabolic\_interpolation(l, r, eps, s=1):  
 m = (l + r) / 2  
 xx = np.arange(l, r, 0.05)  
 yy = np.array([f(x) for x in xx])  
 parables = []  
  
 f1, f2, f3 = s \* f(l), s \* f(m), s \* f(r)  
 debug\_start(l, r, \*[m, f1, f2, f3])  
 while r - l > eps:  
 p = ((m - l) \*\* 2) \* (f2 - f3) - ((m - r) \*\* 2) \* (f2 - f1)  
 q = 2 \* ((m - l) \* (f2 - f3) - (m - r) \* (f2 - f1))  
 u = m - p / q  
 fu = s \* f(u)  
 lagr = lagrange([l, m, r], [f1, f2, f3])  
 parables.append(s \* lagr(xx))  
 if m > u:  
 if f2 < fu:  
 l, f1 = u, fu  
 else:  
 r, f3 = m, f2  
 m, f2 = u, fu  
 else:  
 if f2 > fu:  
 l, f1 = m, f2  
 m, f2 = u, fu  
 else:  
 r, f3 = u, fu  
 debug\_tick(l, r, r - l, \*[m, f1, f2, f3])  
  
 p = (l + r) / 2  
 plt.scatter(p, f(p), c='r')  
 for i in parables:  
 plt.plot(xx, i, c='w', linestyle='--', alpha=.3)  
 plt.plot(xx, yy)  
 plt.show()  
 return p

**Тесты: (8, 14, 0.001)**





**6. Комбинированный метод Брента**

**Описание:**

В данном методе на каждой итерации отслеживаются значения в шести точках (не обязательно различных): *a, c, x, w, v, u*. Точки *a, c* задают текущий интер- вал поиска решения, *x* точка, соответствующая наименьшему значению функции, *w* точка, соответветствующая второму снизу значению функции, *v* предыдущее значение *w*. В отличие от метода парабол, в методе Брента аппроксимирующая па- рабола строится с помощью трех наилучших точек *x, w, v* (в случае, если эти три точки различны и значения в них также различны). При этом минимум аппрокси- мирующей параболы *u* принимается в качестве следующей точки оптимизационного процесса, если:

1. *u* попадает внутрь интервала [*a, c*] и отстоит от границ интервала нt менее, чем на *ε*;
2. *u* отстоит от точки *x* не более, чем на половину от длины, предыдущего шага.

Если точка *u* отвергается, то следующая точка находится с помощью золотого сечения большего из интервалов [*a, x*] и [*x, c*].

Код на языке Python:

@painter\_decorator  
def brents\_method(l, r, eps, s=1):  
 gr = (math.sqrt(5) - 1) / 2  
 m = w = v = l + gr \* (r - l)  
 fm = fw = fv = s \* f(m)  
 d = e = 0  
 u = float('+inf')  
 algo\_type = None  
 debug\_start(l, r, \*[m, w, v, fm, fw, fv, algo\_type])  
 while r - l > eps:  
 g, e = e, d  
 if len({m, w, v}) == len({fm, fw, fv}) == 3:  
 p = ((m - w) \*\* 2) \* (fm - fv) - ((m - v) \*\* 2) \* (fm - fw)  
 q = 2 \* ((m - w) \* (fm - fv) - (m - v) \* (fm - fw))  
 u = m - p / q  
  
 if l + eps <= u <= r - eps and 2 \* abs(u - m) < g:  
 algo\_type = 'spi'  
 d = abs(u - m)  
 else:  
 algo\_type = 'gss'  
 if m < (r + l) \* .5:  
 d = r - m  
 u = m + gr \* d  
 else:  
 d = m - l  
 u = m - gr \* d  
  
 if abs(u - m) < eps:  
 return u  
  
 fu = s \* f(u)  
  
 if fu <= fm:  
 if u >= m:  
 l = m  
 else:  
 r = m  
 v, w, m = w, m, u  
 fv, fw, fm = fw, fm, fu  
  
 else:  
 if u >= m:  
 r = u  
 else:  
 l = u  
  
 if fu <= fw or w == m:  
 v, w = w, u  
 fv, fw = fw, fu  
 elif fu <= fv or v == m or v == w:  
 v = u  
 fv = fu  
  
 debug\_tick(l, r, d, \*[m, w, v, fm, fw, fv, algo\_type])  
  
 return (l + r) / 2

**Тесты: (8, 14, 0.001)**

