Условие домашнего задания №1

Домашнее задание №1
по курсу "Численные методы"(СМ7, 3-й семестр, Осень_2020)

Для заданной целевой функции на заданном отрезке (см. конец документа)найти:

1. Точку минимума (или в некоторых вариантах максимума).
2. Минимальное (максимальное) значение целевой функции. Решить задачу тремя методами:
   (a) методом дихотомии;
   (b) методом золотого сечения;
   (c) методом квадратичной аппроксимации.

При поиске точки минимума рассмотреть для каждого метода три варианта с различными значениями параметра точности поиска $\varepsilon = 0,01; \varepsilon = 0,00001$ и $\varepsilon = 10^{-17}$. Для каждого варианта вывести данные о количестве итераций и количестве вычисленных значений целевой функции (лучше в виде сводной таблицы) и построить графики изменения интервалов неопределенности.
Объяснить полученные результаты. Работа должна заканчиваться выводами.
Требования к выполнению, оформлению и сдачи домашнего задания.

1. Для выполнения домашнего задания использовать любые «математические пакеты» (MATLAB, SciLab, Octave, WolframMathematica, Mapleитд), а также любой язык программирования (Python, C/C++, JS... да хоть ассемблер). На худой конец ДЗ можно сделать даже в какой-нибудь электронной таблице (типа MicrosoftExcel).
2. Запрещено использовать только одну единственную программу: Mathcad.
3. Особо приветствуется в домашнем задании Julia ◡̈.
4. Запрещено использовать символьные вычисления и вычисления с произвольной точностью (например, VPA в MATLAB).
5. Работы без полностью заполненного титульного листа не принимаются.
6. Все страницы (кроме титульного листа) должны быть пронумерованы.

7. Все рисунки и таблицы в тексте должны быть пронумерованы, подписаны и оформлены согласно ГОСТ 7.32-2017.

8. ГОСТа 7.32-2017 и здравого смысла придерживаться при оформлении всего домашнего задания: заголовки относятся к тексту, идущему после них; висячие строки и разрывы полей таблиц запрещены итд. Помните, что ваш текст прочитают не менее двух человек!

9. **Сдача полностью одинаковых работ или работ с одинаковым кодом или выводами рассматривается как полное неуважение к Университету!**

10. Инициатива, расширение и углубление самого домашнего задания (например реализация связки МЗС+МКА, но не только) учитывается с повышенным коэффициентом.

При сдачи работы в электронном виде в гуглоклассе использовать **только формат pdf**.

Буду искать локальный максимум.
Отрезок - [-1, 0]

$$f(x) = \ln(2x^5 - 7x + \sqrt{11}) + \text{sh}\left(\frac{-4x^2 - 4x + 3 - 4\sqrt{2}}{3x^2 + 3x + 3\sqrt{2}}\right) - 1.0$$

Задаем саму функцию:

```
def F(x):
    return np.log(2 * x ** 5 - 7 * x + np.sqrt(11)) + np.sinh((-4 * x ** 2 - 4 * x
    + 3 - 4 * np.sqrt(2)) / (3 * x ** 2 + 3 * x + 3 * np.sqrt(2))) - 1.0
```

Также задам переменную max_iters = 10000, которая будет сигнализировать о зацикливании.

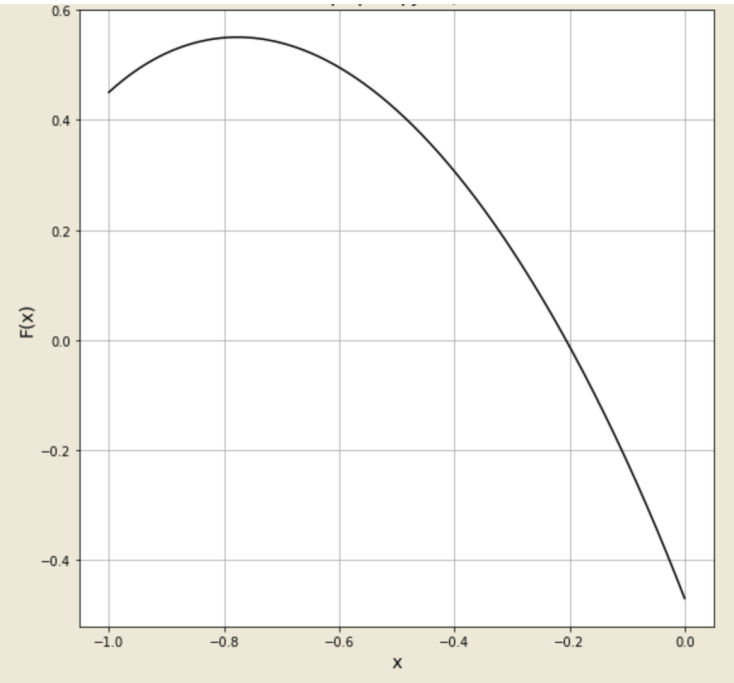

Рис. 1: График функции

1. Метод дихотомии.

Реализация на ЯП Python3:

```
def dichotomy(F, left, right, Eps, max_iters):
    counter = funcalls = 0  #Заведем перемнные для подсчета кол-ва
    итераций и кол-ва вызовов ф-ции
    dotes, iters = [], [] # Инициализируем массив границ промежутка и итераций
    dotes.append([left, right])# Занесем в массив начальные конца отрезка
    iters.append(counter) # Занесем в массив начальную итерацию
    sigma = Eps / 2 # sigma - константая различимость
    while abs(right - left) > Eps and counter <= max_iters:
     #|right - left| < Eps - условие выхода
        x1 = (left + right - sigma)/2
         # Возьмем левый конец чуть левее середины отрезка
        x2 = (left + right + sigma)/2
         # Возьмем правый конец чуть правее середины отрезка
        if F(x1) * F(x2) > 0:
            left = x1
        else:
            right = x2
        dotes.append([left, right]) # Добавим новые концы отрезка
        counter += 1
        iters.append(counter) # Добавим следующую итерацию
        funcalls += 2
         # Т.к. в конструкци if-else мы вызываем ф-цию дважды, прибавляем 2
    return [(left + right) / 2, counter, funcalls, dotes, iters]
```

Интервалы неопределенности для метода дихотомии.

Таблица 1: Результаты метода дихотомии при $\varepsilon = 0.01$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
0	-0.78	0.55	7	14	0.01

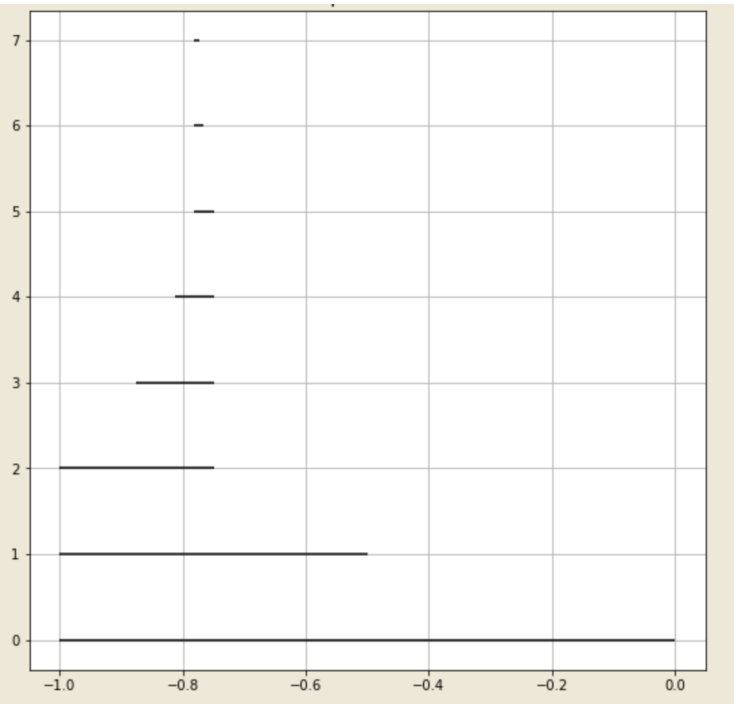

Рис. 2: График изменения интервала неопределенности, $\varepsilon = 0.01$

Таблица 2: Результаты метода дихотомии при $\varepsilon = 10^{-5}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
1	-0.77665	0.55052	17	34	1e-05

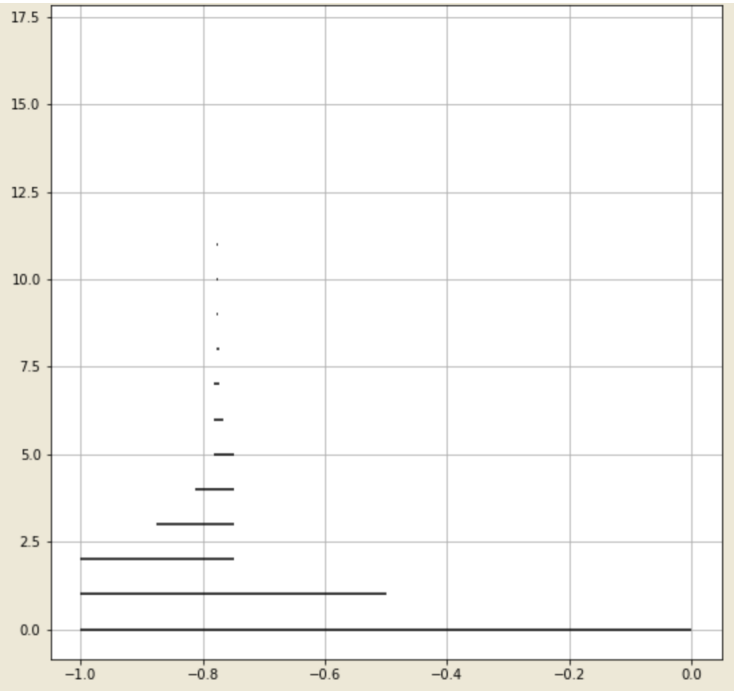

Рис. 3: График изменения интервала неопределенности, $\varepsilon = 10^{-5}$

Таблица 3: Результаты метода дихотомии при $\varepsilon = 10^{-17}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
2	-1.00000000000000000	0.45028971854510069	54	108	1e-17

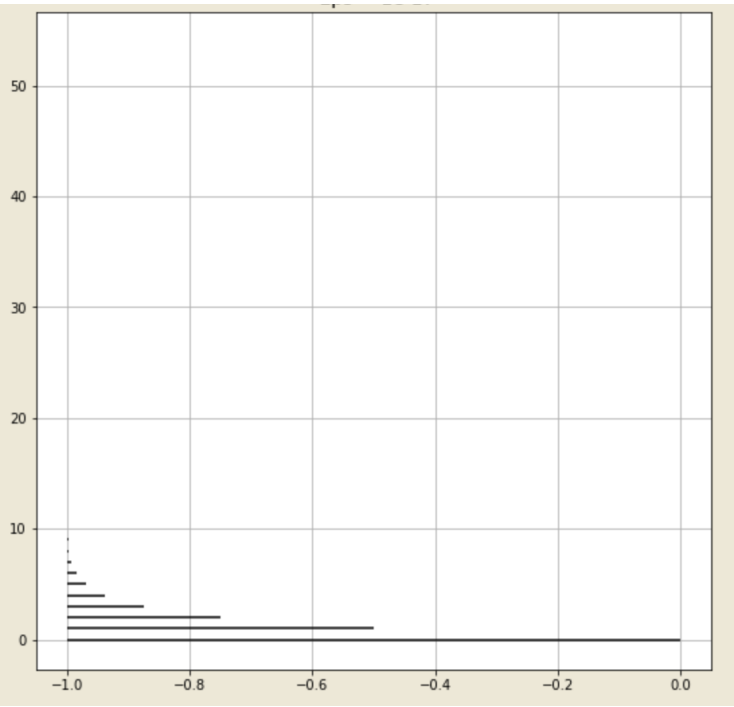

Рис. 4: График изменения интервала неопределенности, $\varepsilon = 10^{-17}$

2. Метод золотого сечения.

$\frac{b-a}{b-x_1} = \frac{b-a}{x_2-a} = \Phi = \frac{1+\sqrt{5}}{2}$, где Φ – пропорция золотого сечения.

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

Реализация на ЯП Python3:

```
def GSC(F, left, right, Eps, max_iters):
    counter = funcalls = 0
    # Заведем перемнные для подсчета кол-ва итераций и кол-ва вызовов ф-ции
    dotes, iters = [], [] # Инициализируем массив границ промежутка и итераций
    dotes.append([left, right]) # Занесем в массив начальные конца отрезка
    iters.append(counter) # Занесем в массив начальную итерацию
    x1 = right - (right - left) / ((1 + math.sqrt(5))/2)
     #точка x2 делит отрезок [x1, b] в отношении золотого сечения
    x2 = left + (right - left) / ((1 + math.sqrt(5))/2)
    #точка x1 делит отрезок [a, x2] в отношении золотого сечения
    f1, f2 = F(x1), F(x2)
    funcalls += 2
    while True: # Реализуем цикл do-while
        if f1 >= f2:
            right = x2
            x2 = x1
            f2 = f1
            x1 = right - (right - left) / ((1 + math.sqrt(5))/2)
            f1 = F(x1)
            funcalls += 1
        else:
            left = x1
            x1 = x2
            f1 = f2
            x2 = left + (right - left) / ((1 + math.sqrt(5))/2)
            f2 = F(x2)
            funcalls += 1
        dotes.append([left, right]) # Добавим новые концы отрезка
        counter += 1 # Увеличиваем итерацию
        iters.append(counter) # Добавим следующую итерацию
        if abs(right - left) < Eps or counter >= max_iters:
            # |right - left| < Eps - условие выхода или же произойдет зацикл
            return [(right + left) / 2, counter, funcalls, dotes, iters]
```

Интервалы неопределенности для метода золотого сечения.

Таблица 4: Результаты метода золотого сечения при $\varepsilon = 0.01$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
0	-0.78	0.55	10	12	0.01

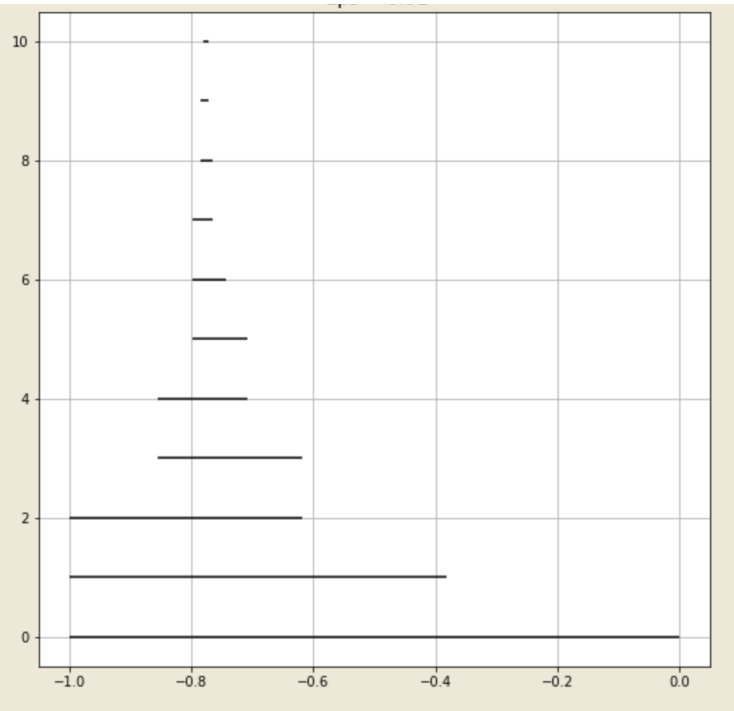

Рис. 5: График изменения интервала неопределенности, $\varepsilon = 0.01$

Таблица 5: Результаты метода золотого сечения при $\varepsilon = 10^{-5}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
1	-0.77665	0.55052	24	26	1e-05

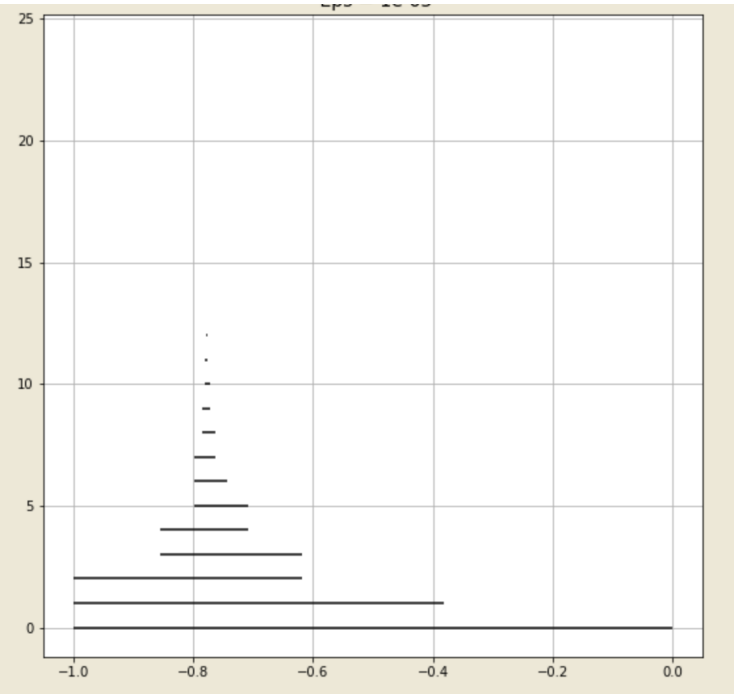

Рис. 6: График изменения интервала неопределенности, $\varepsilon = 10^{-5}$

Таблица 6: Результаты метода золотого сечения при $\varepsilon = 10^{-17}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
2	-0.77664964575113027	0.5505181509138641	100000	100002	1e-17

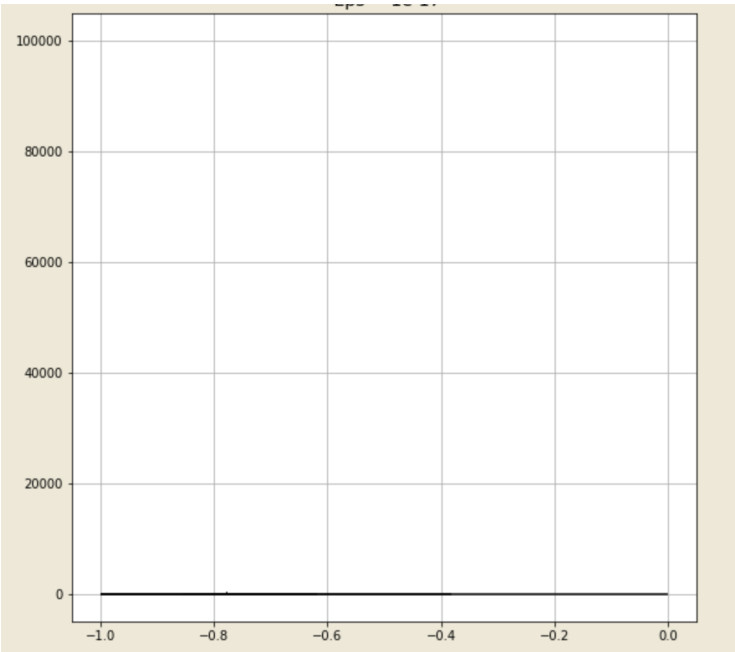

Рис. 7: График изменения интервала неопределенности, $\varepsilon = 10^{-17}$

3. Метод квадратичной аппроксимации.

Опишу формулы:

Начнем с того, что многочлен степени n (т.е. $w_0 + w_1 \cdot x = w_2 \cdot x_2 + ... + w_n \cdot x_n$) однозначно определяется любыми $n+1$ точками, через которые он проходит. Таким образом, в методе квадратичной аппроксимации требуется задать 3 точки.
Квадратичная функция, которая должна совпасть со значениями f(x) в каждой из трех указанных точек:

$$q(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2).$$

$$f_1 = f(x_1) = q(x_1) = a_0, \Rightarrow a_0 = f_1.$$

$$f_2 = f(x_2) = q(x_2) = f_1 + a_1(x_2 - x_1) \Rightarrow a_1 = \frac{f_2 - f_1}{x_2 - x_1}.$$

$$f_3 = f(x_3) = q(x_3) = f_1 = \frac{f_2 - f_1}{(x_2 - x_1)(x_3 - x_1) + a_2(x_3 - x_1)(x_3 - x_2)} \Rightarrow$$

$$a_2 = \frac{1}{x_3 - x_2} \cdot \left(\frac{f_3 - f_1}{x_3 - x_1} - \frac{f_2 - f_1}{x_2 - x_1} \right)$$

По необходимому условию экстремума, найдем производную и с помощью нее выразим вершину параболы:

$$q'(x) = a_1 + a_2(x - x_2) + a_2(x - x_1) = 0 \Rightarrow x = \frac{x_2 - x_1}{2} - \frac{a_1}{2a_2}.$$

Реализация на ЯП Python3:

```
def powell(f, left, right, Eps, max_iters):
    h = 0.1 # Задаем шаг
    dotes, iters = [], [] # Инициализируем массив границ промежутка и итераций
    dotes.append([left, right]) # Занесем в массив начальные конца отрезка
    funcalls, count = 0, 0
    iters.append(count) # Занесем в массив начальную итерацию


    x1, x2 = left, left +  h # Определим начальные точки
    i, j = 0, 0
    # Индексация нам нужна, чтобы при выборе x_min мы
    перестраивали массив выбирая "лучшую" точку и две другие с индексами i и j.
    x = [x1, x2]
    f1, f2 = f(x[0]), f(x[1])
    if f1 > f2:
        x3 = left + 2 * h
```

```
else:
    x3 = left - h

fx = [f1, f2, f(x3)]
x.append(x3) # Занесем в массив x3, чтобы получить 3 начальные
точки x1, x2, x3.
funcalls += 3

dotes.append([x3, x1]) # Добавим новый интервал
count += 1 # Добавим счетчик итераций
iters.append(count) # Занесем в массив первую итерацию
while True:
 # Считаем коэффициенты многочлена:
    a1 = (fx[1] - fx[0]) / (x[1] - x[0])
    a2 = 1.0 / (x[2] - x[1]) * ((fx[2] - fx[0]) / (x[2]-x[0])-(fx[1]-fx[0]) /
     (x[1]-x[0]))
    x_max = (x[1] + x[0]) * 0.5 - a1 / (2 * a2)
    f_max = f(x_max)
    funcalls += 1
    # Здесь ищется x_max, который соответстует f_max= max(f1, max(f2, f3))
    if fx[0] >= fx[1]:
        if fx[0] >= fx[2]:
            i = 0
        else:
            i = 2
    else:
        if fx[1] >= fx[2]:
            i = 1
        else:
            i = 2
    count += 1
    iters.append(count)
    dotes.append([x[i], x_ max]])
    if ((abs((x_max - x[i]) / x_max) < Eps) and
    (abs((f_max - fx[i])/f_max) < Eps)) or count >= max_iters:
        return [x_max, count, funcalls, dotes, iters]

    if fx[0] < fx[1]:
        if fx[0] <= fx[2]:
            j = 0
        else:
            j = 2
    else:
        if fx[1] <= fx[2]:
            j = 1
        else:
```

```
            j = 2

        if (f_max > fx[i]):
            x[j] = x_max
            fx[j] = f_max
        else:
            x[j] = 2 * x[i] - x_max
            fx[j] = F(x[j])


    return [x_max, count, funcalls, dotes, iters]
```

Интервалы неопределенности для метода квадратичной аппроксимации.

Таблица 7: Результаты метода квадратичной аппроксимации при $\varepsilon = 0.01$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
0	-0.78	0.55	3	5	0.01

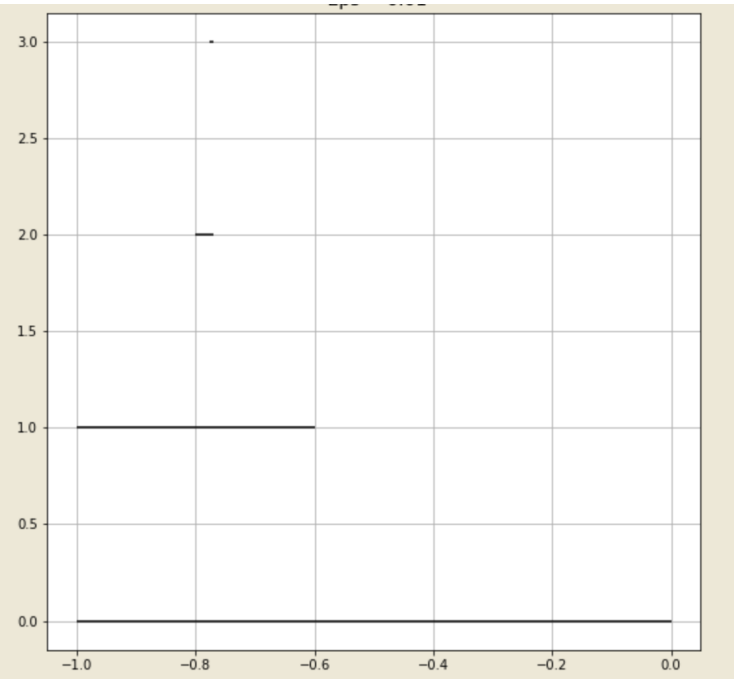

Рис. 8: График изменения интервала неопределенности, $\varepsilon = 0.01$

Таблица 8: Результаты метода квадратичной аппроксимации при $\varepsilon = 10^{-5}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
1	-0.77658	0.55052	6	8	1e-05

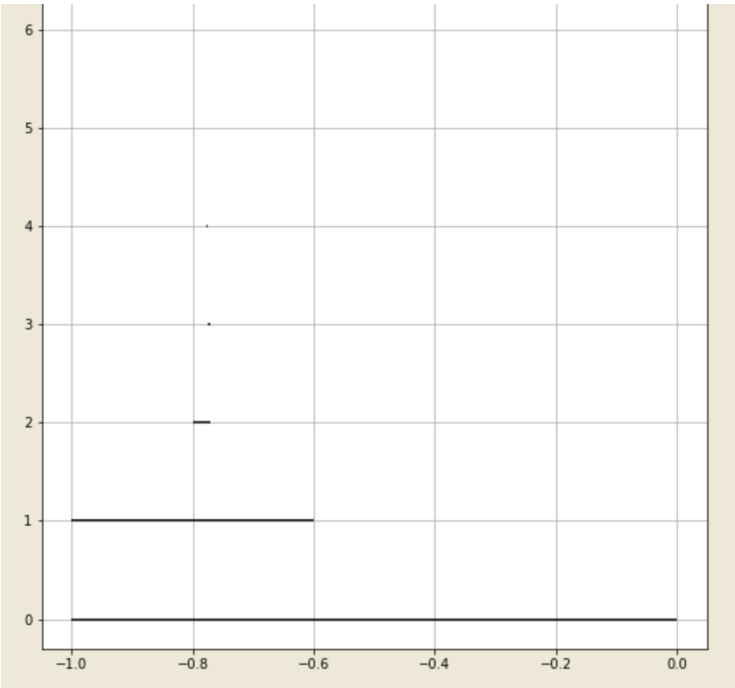

Рис. 9: График изменения интервала неопределенности, $\varepsilon = 10^{-5}$

Таблица 9: Результаты метода квадратичной аппроксимации при $\varepsilon = 10^{-17}$

№	X_max	F(X_max)	Num of iters	Num of func calls	Eps
2	-0.77664965664046193	0.5505181509138641	100000	100002	1e-17

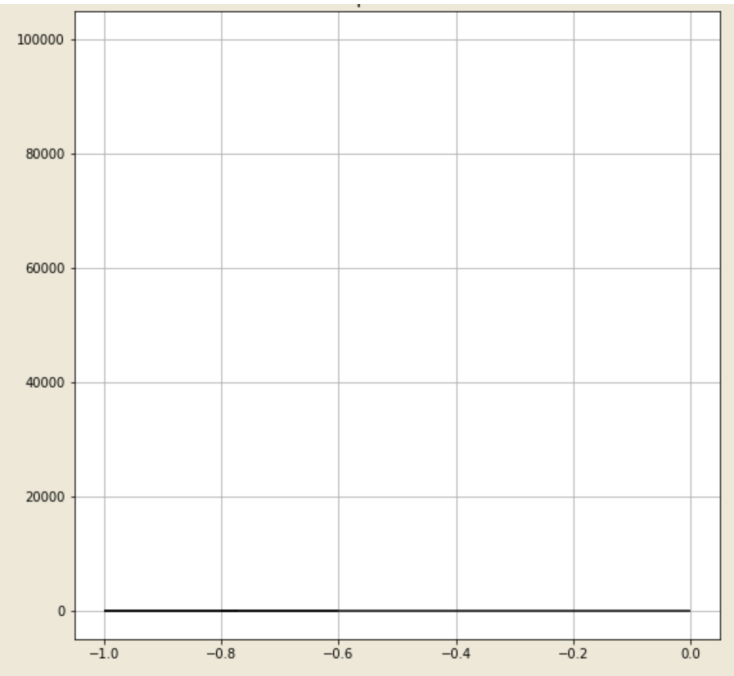

Рис. 10: График изменения интервала неопределенности, $\varepsilon = 10^{-17}$

Выводы:

1. Метод дихотомии:
   (a) Сам метод прост, как в реализации, так и в действии;
   (b) Так как функция будет рассматривать точки x_1 и x_2, вне зависимости от данных, алгоритм будет работать с любыми данными также, как и в в худшем. Т.е. мы будем рассматривать множество лишних данных;
   (c) При увелечении количества итераций элементы дихотомического деления, будут более схожи между собой, т.е. x_1 и x_2 будут очень близки по значению, что затрудняет поиск локального экстремума;
   (d) При уменьшении Eps, наш отрезок [left, right] может становиться большей длины, что повлияет на точность вычисления x_{max}.
2. Метод золотого сечения :
   (a) Аналогично, метод довольно таки прост;
   (b) Eps начинает влиять на результат, лишь увеличивает при очень малых значениях(например, 1e-17);
   (c) При малых Eps происходит зацикливание, например, при Eps = $1e-17$, это происходит потому что при интервал неопределенности уже достигает точности в $1e-16$ и не может выйти из цикла, ибо не меньше $1e-17$, но и уменшиться не может, ибо он дошел до своего конечного ответа.
   (d) Алгоритм универсален, в плане поступающих данных. Т.е. поступят плохие данные - отработает плохо, иначе - хорошо. Под словами 'хорошо' и 'плохо', я имею в виду, что может достигаться как нижняя, так и верхняя ассимптотическая граница алгоритма.
3. Метод квадратичной аппроксимации:
   (a) Алгоритм средней сложности в реализцаии;
   (b) Eps начинает влиять на результат, лишь увеличивает при очень малых значениях(например, 1e-17);
   (c) Алгоритм быстродейственный;
   (d) Зацикливание происходит при очень малой точности, например при Eps = $1e-17$, это происходит потому что при малых eps теряется точность чисел с плавающей точкой. Данное Eps будет превышать конечный интервал поиска, поэтому цикл не остановится. Наш реузльтат точно еще меньше, чем шаг между ними, поэтому когда он находит новое значение он не может найти, потому что зацикливается около старого значения.

(e) Алгоритм не справиться с линейными функциями, так как мы аппроксимируем функцию параболой, наша функция должна быть многочленом как минимум второй степени.

Из-за того что на больших отрезках метод квадратичной аппроксимации не очень точный, в голову приходит мысль оптимизации.
Если мы модифицируем алгоритм, т.е. при больших интервалах будем использовать метод золотого сечения, а при малых метод Пауэлла, мы получим самый оптимальный алгоритм из вышепредложенных.

Источники

- Метод дихотомии
  http://www.machinelearning.ru/wiki/index.php?title=Методы_дихотомии
- Метод золотого сечения
  https://ru.wikipedia.org/wiki/Метод_золотого_сечения
- Метод квадратичной аппроксимации
  http://knigechka.blogspot.com/2010/04/blog-post_6723.html