Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа №3
«Методы одномерного поиска. Методы стягивающихся отрезков. Методы интерполяции»
по курсу
«Методы оптимизации»

Студент группы ИУ9-82 Иванов Г.М

Преподаватель Каганов Ю.Т.

Москва, 2019

Contents

1 Цель работы

1. Изучение алгоритмов одномерного поиска.
2. Разработка программ реализации алгоритмов одномерного поиска.
3. Вычисление экстремумов функции.

2 Постановка задачи

2.1 Задача 3.1

Дано: Функция на множестве R^1

$$f(x) = 5x^6 - 36x^5 - \frac{165}{2}x^4 - 60x^3 + 36, x_0 = -13 \tag{1}$$

Найти экстремум методами:

1. Найти экстремум методами:
   (a) Половинного деления
   (b) Золотого сечения
   (c) Фибоначчи
2. Найти все стационарные точки и значения функций, соотвествествующие этим точкам.
3. Оценить скорость сходимости указанных алгоритмов
4. Реализовать алгоритм с помощью языка программирования высокого уровня

2.2 Задача 3.2

Дано: Функция на множестве R^1

$$f(x) = 5x^6 - 36x^5 - \frac{165}{2}x^4 - 60x^3 + 36, x_0 = -13 \tag{2}$$

1. Найти экстремум методами:
   (a) Квадратичной интерполяции

(b) Кубической интерполяции

2. Найти все стационарные точки и значения функций, соотвествествующие этим точкам.
3. Оценить скорость сходимости указанных алгоритмов
4. Реализовать алгоритм с помощью языка программирования высокого уровня

3 Исследование

Найдем глобальные экстремумы функции

$$f(x) = 5x^6 - 36x^5 - \frac{165}{2}x^4 - 60x^3 + 36, x_0 = -13 \tag{3}$$

с помощью сервиса WolframAlpha.com:

$$min(f(x)) = -9/2, \quad x = 3 \tag{4}$$

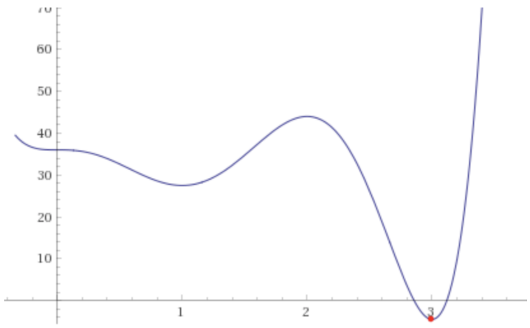

Figure 1: График функции $f(x)$

3.1 Задача 3.1

3.1.1 Алгоритм Свенна

Прежде, чем переходить к реализации самих алгоритмов поиска экстремума, необходимо найти интервал, где будет находиться стационарная точка.
Для этого воспользуемся алгоритмом Свенна.

1. Задать произвольно следующие параметры: x^0 - начальную точку, $t > 0$ - величину шага. Положить k = 0.
2. Вычислить значение функции в трех точках: $x^0 - t$, x^0, $x^0 + t$.
3. Проверить условие окончания:

(a) если $f(x^0 - t) \geq f(x^0) \leq f(x^0 + t)$, то начальный интервал неопределенности найден: $[a_0, b_0] = [x^0 - t, x^0 + t]$;

(b) если $f(x^0 - t) \leq f(x^0) \geq f(x^0 + t)$, то функция не является унимодальной, а требуемый интервал неопределенности не может быть найден. Вычисления при этом прекращаются (рекомендуется задать другую начальную точку x^0);

(c) если условие окончания не выполняется, то перейти на 4.

4. Определить величину $\triangle$:

   (a) если $f(x^0 - t) \geq f(x^0) \geq f(x^0 + t)$, то $\triangle = t; a_0 = x^0; x^1 = x^0 + t; k = 1$

   (b) если $f(x^0 - t) \leq f(x^0) \leq f(x^0 + t)$, то $\triangle = -t; b_0 = x^0; x^1 = x^0 - t; k = 1$

5. Найти следующую точку $x^{k+1} = x^k + 2^k \triangle$.

6. Проверить условие убывания функции:

   (a) если $f(x^{k+1}) < f(x)$ и $\triangle = t$, то $a_0 = x^k$;

   (b) если $f(x^{k+1}) < f(x)$ и $\triangle = -t$, то $b_0 = x^k$

   (c) если $f(x^{k+1}) \neq f(x)$, процедура завершается.

7. При $\triangle = t$ положить $b_0 = x^{k+1}$, а при $\triangle = -t$ положить $a_0 = x^{k+1}$. В результате имеем интервал $[a_0, b_0]$ — искомый начальный интервал неопределенности.

3.1.2 Метод деления интервала пополам

Метод относится к последовательным стратегиям и позволяет исключить из дальнейшего рассмотрения на каждой итерации в точности половину текущего интервала неопределенности. Задается начальный интервал неопределенности, а алгоритм уменьшения интервала, являясь, как и в общем случае, «гарантирующим» основан на анализе величин функции в трех точках, равномерно распределенных на текущем интервале. Условия окончания процесса поиска стандартные: поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

3.1.3 Метод золотого сечения

Для построения конкретного метода одномерной минимизации, работающего по принципу последовательного сокращения интервала неопределенности, следует задать правило выбора на каждом шаге двух внутренних точек. Конечно, желательно, чтобы одна из них всегда использовалась в качестве внутренней точки и для следующего интервала. Тогда число вычислений функции сократится вдвое и одна итерация потребует расчета только одного нового значения функции. В методе золотого сечения в качестве двух внутренних точек выбирают точки золотого сечения.

Точка производит **«золотое сечение»** отрезка, если отношение длины всего отрезка к большей части равно отношению большей части к меньшей.

Метод относится к последовательным стратегиям. Задается начальный интервал неопределенности и требуемая точность. Алгоритм уменьшения интервала опирается на анализ значений функции в двух точках. В качестве точек вычисления функции выбираются точки золотого сечения. Тогда с учетом свойств золотого сечения на каждой итерации, кроме первой, требуется только одно новое вычисление функции. Условия окончания процесса поиска стандартные: поиск заканчивается, когда длина текущего интервала неопределенности оказывается меньше установленной величины.

3.1.4 Метод Фибоначчи

Для построения эффективного метода одномерной минимизации, работающего по принципу последовательного сокращения интервала неопределенности, следует задать правило выбора на каждом шаге двух внутренних точек. Конечно, желательно, чтобы одна из них всегда использовалась в качестве внутренней точки и для следующего интервала. Тогда число вычислений функции сократится вдвое и одна итерация потребует расчета только одного нового значения функции. В методе Фибоначчи реализована стратегия, обеспечивающая максимальное гарантированное сокращение интервала неопределенности при заданном количестве вычислений функции и претендующая на оптимальность. Эта стратегия опирается на числа Фибоначчи.

Числа Фибоначчи определяются по формуле:

$$F_0 = F_1, F_k = F_{k-1} + F_{k-2}, k = 2, 3, 4, ... \qquad (5)$$

Последовательность чисел Фибоначчи имеет вид: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.

3.2 Задача 3.2

3.2.1 Метод квадратичной интерполяции

Метод квадратичной интерполяции относится к последовательным стратегиям. Задаётся начальная точка и с помощью пробного шага находятся три точки так, чтобы они были как можно ближе к искомой точке минимума. В полученных точках вычисляются значения функции. Затем строится интерполяционный полином второй степени, проходящий через эти три точки. В качестве приближения точки минимума берется точка минимума полинома. Процесс поиска заканчивается, когда полученная точка является наилучшей из трёх опорных точек не более чем на заданную величину.

3.2.2 Метод кубической интерполяции

Задаётся начальная точка и с помощью серии пробных шагов находятся две точки, первые производные в которых имеют противоположные знаки. По величине функции и её первых производных в полученных точках строится интерполяционный полином третьей степени. В качестве приближения точки минимума берётся точка минимума полинома. Процесс поиска заканчивается, если производная в точке минимума полинома достаточно мала или процедура становится неэффективной.

4 Практическая реализация

Все методы были реализованы на языке программирования **Python**. Использовался вспомогательный класс *Interval*.

Листинг 1. Класс *Interval*

```
class Interval:

    def __init__(self, x_start, x_end):
        self.x_start = x_start
        self.x_end = x_end

    @property
    def length(self):
        return abs(self.x_end - self.x_start)

    @property
    def center(self):
        return (self.x_end + self.x_start) / 2

    def __str__(self):
        return "[" + str(self.x_start) + " , " + str(self.x_end) + "]"
```

4.1 Задача 3.1

Листинг 2. Алгоритм Свенна.

```
def method_svann(x_start, step_size, function):
    print_function.print_start("Svann Method")
```

```python

    k = 0

    x_values = [x_start]

    fun_result_without_step_size = function(x_start - step_size)
    fun_result_on_start = function(x_start)
    fun_result_with_step_size = function(x_start + step_size)

    interval = Interval(fun_result_without_step_size, fun_result_on_start)

    if fun_result_without_step_size >= fun_result_on_start and fun_result_on_start <= fun_result_with_step_size:
        return interval
    elif fun_result_without_step_size <= fun_result_on_start and fun_result_on_start >= fun_result_with_step_size:
        raise Exception("Interval can't be found, choose another x_start (%f) variable!" % (str(x_start)))
    else:
        delta = float(0.0)
        k += 1

        if fun_result_without_step_size >= fun_result_on_start >= fun_result_with_step_size:
            delta = step_size
            interval.x_start = x_values[0]

            x_values.insert(k, x_start + step_size)

        elif fun_result_without_step_size <= fun_result_on_start <= fun_result_with_step_size:
            delta = -step_size
            interval.x_end = x_values[0]

            x_values.insert(k, x_start - step_size)

        while True:

            x_values.insert(k + 1, (x_values[k] + pow(2.0, k) * delta))

            if function(x_values[k + 1]) >= function(x_values[k]):
                if delta > 0:
                    interval.x_end = x_values[k + 1]
                elif delta < 0:
                    interval.x_start = x_values[k + 1]
            else:
                if delta > 0:
                    interval.x_start = x_values[k]
                elif delta < 0:
```

```
                    interval.x_end = x_values[k]

            if function(x_values[k + 1]) >= function(x_values[k]):
                break

            k += 1

    print_function.print_end(k, interval)
    return interval
```

Листинг 3. Метод деления интервала пополам.

```
def bisection_method(epsilon, interval, function):
    print_function.print_start("Bisection method")

    k = 0
    x_middle = interval.center

    while True:
        x_left_middle = interval.x_start + interval.length / 4
        x_right_middle = interval.x_end - interval.length / 4

        if function(x_left_middle) < function(x_middle):
            interval.x_end = x_middle
            x_middle = x_left_middle
        elif function(x_right_middle) < function(x_middle):
            interval.x_start = x_middle
            x_middle = x_right_middle
        else:
            interval.x_start = x_left_middle
            interval.x_end = x_right_middle

        k += 1

        if not interval.length > epsilon:
            break

    print_function.print_end_function(k, x_middle, function)
    return x_middle
```

Листинг 4. Метод золотого сечения.

```
def golden_section_method(epsilon, interval, function):
    print_function.print_start("Golden Section method")
```

```

    k = 0

    phi = (1 + math.sqrt(5.0)) / 2

    while interval.length > epsilon:
        z = (interval.x_end - (interval.x_end - interval.x_start) / phi)
        y = (interval.x_start + (interval.x_end - interval.x_start) / phi)
        if function(y) <= function(z):
            interval.x_start = z
        else:
            interval.x_end = y

        k += 1

    print_function.print_end_function(k, interval.center, function)
    return interval.center
```

Листинг 5. Метод Фибоначчи.

```
def fibonacci_method(eps, interval, function):
    print_function.print_start("Fibonacci method")

    k = 0

    n = 3
    fib_arr = [1.0, 1.0, 2.0, 3.0]
    f1 = 2.0
    f2 = 3.0
    while fib_arr[len(fib_arr) - 1] < interval.length / eps:
        fib_arr.append(f1 + f2)
        f1 = f2
        f2 = fib_arr[len(fib_arr) - 1]
        n = n + 1

    for i in range(1, n - 3):
        y = (interval.x_start + fib_arr[n - i - 1] / fib_arr[n - i + 1] *
        ↪  (interval.x_end - interval.x_start))
        z = (interval.x_start + fib_arr[n - i] / fib_arr[n - i + 1] *
        ↪  (interval.x_end - interval.x_start))
        if function(y) <= function(z):
            interval.x_end = z
        else:
            interval.x_start = y

        k += 1
```

```
    print_function.print_end_function(k, interval.center, function)
    return interval.center
```

4.2 Задача 3.2

Листинг 6. Метод квадратичной интерполяции.

```
def quadratic_interpolation(eps, delta, x_start, step_size, function):
    print_function.print_start("Quadratic Interpolation method")

    a1 = x_start
    k = 0

    while True:
        # Step 2
        a2 = a1 + step_size

        # Step 3
        f1 = function(a1)
        f2 = function(a2)

        # Step 4
        a3 = (a1 + 2 * step_size) if f1 > f2 else (a1 - 2 * step_size)

        while True:
            k += 1

            # Step 5
            f3 = function(a3)

            # Step 6
            f_min = min(f1, min(f2, f3))

            a_min = None
            if f_min == f1:
                a_min = a1
            elif f_min == f2:
                a_min = a2
            elif f_min == f3:
                a_min = a3
            else:
                raise Exception("Cannot find f_min")
```

```
            # Step 7
            det = 2 * ((a2 - a3) * f1 + (a3 - a1) * f2 + (a1 - a2) * f3)
            if det == 0.0:
                a1 = a_min
            else:
                a = ((pow(a2, 2) - pow(a3, 2)) * f1 + (pow(a3, 2) - pow(a1, 2)) * f2 + (pow(a1, 2) - pow(a2, 2)) * f3) / det

                # Step 8
                if abs((f_min - function(a)) / function(a)) < eps and abs((a_min - a) / a) < delta:
                    print_function.print_end_function(k, a1, function)
                    return a1
                else:
                    if a1 <= a <= a3:
                        if a < a2:
                            a3 = a2
                            a2 = a
                        else:
                            a1 = a2
                            a2 = a
                    else:
                        a1 = a
                        break
```

Листинг 7. Метод кубической интерполяции.

```
def cubic_interpolation(eps, delta, x_start, step_size, function, derivative_function):
    print_function.print_start("Cubic Interpolation")

    a1 = x_start
    k = 0

    # Step 2
    df = derivative_function(x_start)

    m = 0.0
    a2 = a1
    # Step 3
    if df < 0:
        while True:
            a1 = a2
            a2 += pow(m, 2) * step_size
            m += 1.0
```

```
            if not (derivative_function(a1) * derivative_function(a2) > 0):
                break
    else:
        while True:
            a1 = a2
            a2 -= pow(m, 2) * step_size
            m += 1.0
            if not (derivative_function(a1) * derivative_function(a2) > 0):
                break

    while True:
        k += 1

        # Step 4
        f1 = function(a1)
        f2 = function(a2)
        df1 = derivative_function(a1)
        df2 = derivative_function(a2)

        # Step 5
        z = 3 * (f1 - f2) / (a2 - a1) + df1 + df2
        w = math.sqrt(pow(z, 2) - df1 * df2) if a1 < a2 else (-math.sqrt(pow(z, 2) - df1 * df2))

        mu = (df2 + w - z) / (df2 - df1 + 2*w)

        if mu < 0:
            a = a2
        elif 0.0 < mu <= 1.0:
            a = a2 - mu * (a2 - a1)
        elif mu > 1:
            a = a1
        else:
            raise Exception("Error in range")

        # Step 6
        while function(a) > function(a1) and ((a - a1) / a).absoluteValue > delta:
            a -= (a - a1) / 2

        # Step 7
        if abs(derivative_function(a)) <= eps and (abs(a - a1) / a) <= delta:
            print_function.print_end_function(k, a, function)
            return a
        else:
            if derivative_function(a) * derivative_function(a1) <= 0:
                a2 = a1
                a1 = a
            else:
```

```
            a1 = a
```

5 Результаты.

При последовательном запуске всех алгоритмов со следующими параметрами:

$$stepSize = 0.01, \quad epsilon = 0.01, \quad delta = 0.01 \tag{6}$$

были получены следующие результаты:

Листинг 8. Результаты выполнения программ.

```
Start Svann Method
        Iteration(s): 4
        Result: [-6.0 , 18.0]
Start Bisection method
        Iteration(s): 12
        f(3.000000) = {-4.500000}

Start Golden Section method
        Iteration(s): 17
        f(2.999726) = {-4.499980}

Start Fibonacci method
        Iteration(s): 13
        f(2.995356) = {-4.494216}

Start Quadratic Interpolation method
        Iteration(s): 12
        f(3.007932) = {-4.482817}

Start Cubic Interpolation
        Iteration(s): 4
        f(3.000000) = {-4.500000}
```

Все результаты с небольшой погрешностью совпадают с результатами полученными с помощью сервиса WolframAlpha.com в пункте 3. Наилучшая точность получена с помощью метода кубической инерполяции.