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

Лабораторная работа №3. Вариант 1.
«Методы одномерного поиска. Методы стягивающихся отрезков. Методы интерполяции.»
по курсу
«Методы оптимизации»

Студент группы ИУ9-82 Белогуров А.А.

Преподаватель Каганов Ю.Т.

Москва, 2018

Содержание

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: 1 Вариант. Функция на множестве R^1

$$f(x) = 12x^2 - 2x + |x - 3|, x_0 = -3 \tag{1}$$

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

2.2 Задача 3.2

Дано: 1 Вариант. Функция на множестве R^1

$$f(x) = 12x^2 - 2x + |x - 3|, x_0 = -3 \tag{2}$$

1. Найти экстремум методами:
   (a) Квадратичной интерполяции
   (b) Кубической интерполяции
2. Найти все стационарные точни и значения функций, соотвествующие этим точкам.

3. Оценить скорость сходимости указанных алгоритмов.

4. Реализовать алгоритмы с помощью языка программирования высокого уровня.

3 Исследование

Найдем глобальные экстремумы функции

$$f(x) = 12x^2 - 2x + |x - 3| \tag{3}$$

с помощью сервиса WolframAlpha.com:

$$min(f(x)) = 45/16 \approx 2.8125, \quad x = 1/8 = 0.125 \tag{4}$$

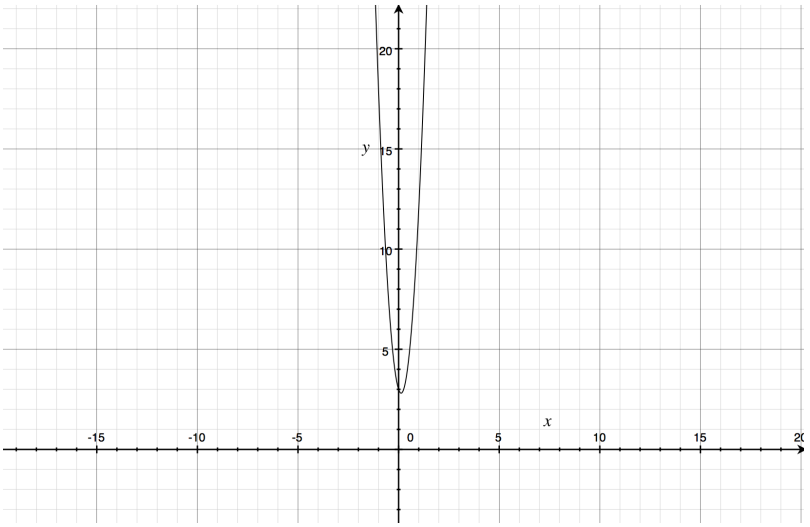

Рис. 1: График функции $f(x)$

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: $x^0-t, \quad x^0, \quad x^0+t$.

3. Проверить условия 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 п. **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 $f(x^{k+1}) < f(x^k)$ и $\triangle = t$, то $a_0 = x^k$;

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

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

   в первых двух случаях положить $k = k+1$ и перейти на п. **5**

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 зотолого 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.

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:

$$F_0 = F_1, \quad F_k = F_{k-1} + F_{k-2}, \quad k = 2, 3, 4, ... \tag{5}$$

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 (метод Пауэлла) 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 **Kotlin**. Использовался вспомогательный класс *Interval*.

Листинг 1. Класс *Interval*

```
data class Interval(var xStart: Float,
                    var xEnd: Float) {

    fun getLength() = (xEnd - xStart).absoluteValue

    fun getCenter() = (xEnd + xStart) / 2

    override fun toString(): String {
        return "[$xStart, $xEnd]"
    }
}
```

4.1 Задача 3.1

Листинг 2. Алгоритм Свенна.

```
fun svannMethod(xStart: Float,
                stepSize: Float,
                function: (x: Float) -> Float): Interval {
    PrintUtils.printInfoStart("Svann Method")

    var k = 0
    lateinit var interval: Interval

    val xValues = arrayListOf(xStart)

    val funResultWithoutStepSize = function(xStart - stepSize)
    val funResultOnStart = function(xStart)
    val funResultWithStepSize = function(xStart + stepSize)

    interval = Interval(funResultWithoutStepSize, funResultOnStart)

    if (funResultWithoutStepSize >= funResultOnStart && funResultOnStart <=
      ↪  funResultWithStepSize) {    // step 3.a
```

```
18          return interval
19
20      } else if (funResultWithoutStepSize <= funResultOnStart &&
          ↪  funResultOnStart >= funResultWithStepSize) {     // step 3.b
21          throw Exception("Interval can't be found, choose another xStart
              ↪  ($xStart) variable!")
22
23      } else {    // step 4
24          var delta = 0.0f
25          k += 1
26
27          if (funResultWithoutStepSize >= funResultOnStart && funResultOnStart
              ↪  >= funResultWithStepSize) {  // step 4.a
28              delta = stepSize
29              interval.xStart = xValues[0]
30
31              xValues.add(k, xStart + stepSize)
32          } else if (funResultWithoutStepSize <= funResultOnStart &&
              ↪  funResultOnStart <= funResultWithStepSize) { // step 4.b
33              delta = -stepSize
34              interval.xEnd = xValues[0]
35
36              xValues.add(k, xStart - stepSize)
37          }
38
39          while (true) {
40              xValues.add(k + 1, (xValues[k] + 2.0.pow(k) * delta).toFloat())
                  ↪  // step 5
41
42              if (function(xValues[k + 1]) >= function(xValues[k])) {
43                  if (delta > 0) {
44                      interval.xEnd = xValues[k + 1]
45                  } else if (delta < 0) {
46                      interval.xStart = xValues[k + 1]
47                  }
48              } else {
49                  if (delta > 0) {
50                      interval.xStart = xValues[k]
51                  } else if (delta < 0) {
52                      interval.xEnd = xValues[k]
53                  }
54              }
55
56              if (function(xValues[k + 1]) >= function(xValues[k])) {
57                  break
58              }
59              k += 1
60          }
61      }
```

```
62
63      PrintUtils.printInfoEnd(k, 0, interval)
64      return interval
65  }
```

Листинг 3. Метод деления интервала пополам.

```
1   fun bisectionMethod(epsilon: Float,
2                       interval: Interval,
3                       function: (x: Float) -> Float): Float {
4       PrintUtils.printInfoStart("Bisection method")
5
6       var k = 0
7       var xMiddle = interval.getCenter()
8
9       do {
10          val xLeftMiddle = interval.xStart + interval.getLength() / 4
11          val xRightMiddle = interval.xEnd - interval.getLength() / 4
12
13          when {
14              function(xLeftMiddle) < function(xMiddle) -> {
15                  interval.xEnd = xMiddle
16                  xMiddle = xLeftMiddle
17              }
18              function(xRightMiddle) < function(xMiddle) -> {
19                  interval.xStart = xMiddle
20                  xMiddle = xRightMiddle
21              }
22              else -> {
23                  interval.xStart = xLeftMiddle
24                  interval.xEnd = xRightMiddle
25              }
26          }
27
28          k += 1
29      } while (interval.getLength() > epsilon)
30
31      PrintUtils.printInfoEndFunction(k, 0, xMiddle, function)
32      return xMiddle
33  }
```

Листинг 4. Метод золотого сечения.

```
fun goldenSectionMethod(epsilon: Float,
                        interval: Interval,
                        function: (x: Float) -> Float): Float {
    PrintUtils.printInfoStart("Golden Section method")

    var k = 0
    val phi = (1 + Math.sqrt(5.0)) / 2

    while (interval.getLength() > epsilon) {
        val z = (interval.xEnd - (interval.xEnd - interval.xStart) /
          ↪  phi).toFloat()
        val y = (interval.xStart + (interval.xEnd - interval.xStart) /
          ↪  phi).toFloat()
        if (function(y) <= function(z)) {
            interval.xStart = z
        } else {
            interval.xEnd = y
        }

        k += 1
    }

    PrintUtils.printInfoEndFunction(k, 0, interval.getCenter(), function)
    return interval.getCenter()
}
```

Листинг 5. Метод Фибоначчи.

```
fun fibonacciMethod(eps: Float,
                    interval: Interval,
                    function: (x: Float) -> Float): Float {
    PrintUtils.printInfoStart("Fibonacci method")

    var k = 0

    var y: Float
    var z: Float
    var n = 3
    val fibArr = arrayListOf(1.0, 1.0, 2.0, 3.0)
    var f1 = 2.0
    var f2 = 3.0
    while (fibArr[fibArr.size - 1] < interval.getLength() / eps) {
        fibArr.add(f1 + f2)
        f1 = f2
```

```
17          f2 = fibArr[fibArr.size - 1]
18          ++n
19      }
20      for (i in 1 until n - 3) {
21          y = (interval.xStart + fibArr[n - i - 1] / fibArr[n - i + 1] *
              ↪   (interval.xEnd - interval.xStart)).toFloat()
22          z = (interval.xStart + fibArr[n - i] / fibArr[n - i + 1] *
              ↪   (interval.xEnd - interval.xStart)).toFloat()
23          if (function(y) <= function(z))
24              interval.xEnd = z
25          else
26              interval.xStart = y
27
28          k += 1
29      }
30
31      PrintUtils.printInfoEndFunction(k, timeExecution, interval.getCenter(),
          ↪   function)
32      return interval.getCenter()
33  }
```

4.2 Задача 3.2

Листинг 6. Метод квадратичной интерполяции.

```
1   fun quadraticInterpolation(eps: Float,
2                              delta: Float,
3                              xStart: Float,
4                              stepSize: Float,
5                              function: (x: Float) -> Float): Float {
6       PrintUtils.printInfoStart("Quadratic Interpolation method")
7
8       var a1 = xStart
9       var k = 0
10
11      while (true) {
12          // Step 2
13          var a2 = a1 + stepSize
14
15          // Step 3
16          val f1 = function(a1)
17          val f2 = function(a2)
18
```

```
19          // Step 4
20          var a3 = if (f1 > f2) {
21              a1 + 2 * stepSize
22          } else {
23              a1 - 2 * stepSize
24          }
25
26          while (true) {
27              k += 1
28
29              // Step 5
30              val f3 = function(a3)
31
32              // Step 6
33              val fMin = min(f1, min(f2, f3))
34              val aMin = when (fMin) {
35                  f1 -> a1
36                  f2 -> a2
37                  f3 -> a3
38                  else -> throw Exception("Cannot find fMin")
39              }
40
41              // Step 7
42              val det = 2 * ((a2 - a3) * f1 + (a3 - a1) * f2 + (a1 - a2) * f3)
43              if (det == 0f) {
44                  a1 = aMin
45              } else {
46                  val a = ((a2.pow(2) - a3.pow(2)) * f1 + (a3.pow(2) -
                     ↪  a1.pow(2)) * f2 + (a1.pow(2) - a2.pow(2)) * f3) / det
47
48                  // Step 8
49                  if (((fMin - function(a)) / function(a)).absoluteValue < eps
                     ↪  && ((aMin - a) / a).absoluteValue < delta) { // a)
50                      PrintUtils.printInfoEndFunction(k, 0, a1, function)
51                      return a
52                  } else {
53                      if (a in a1..a3) { // b)
54                          if (a < a2) {
55                              a3 = a2
56                              a2 = a
57                          } else {
58                              a1 = a2
59                              a2 = a
60                          }
61                      } else { // c)
62                          a1 = a
63                          break
64                      }
65                  }
```

```
            }
        }
    }
}
```

Листинг 7. Метод кубической интерполяции.

```
fun cubicInterpolation(eps: Float,
                       delta: Float,
                       xStart: Float,
                       stepSize: Float,
                       function: (x: Float) -> Float,
                       derivativeFunction: (x: Float) -> Float): Float {
    PrintUtils.printInfoStart("Cubic Interpolation")

    var a1 = xStart
    var k = 0

    // Step 2
    val df = derivativeFunction(xStart)

    var m = 0f
    var a2 = a1
    // Step 3
    if (df < 0) {
        do {
            a1 = a2
            a2 += m.pow(2) * stepSize
            m += 1f
        } while (derivativeFunction(a1) * derivativeFunction(a2) > 0)
    } else {
        do {
            a1 = a2
            a2 -= m.pow(2) * stepSize
            m += 1f
        } while (derivativeFunction(a1) * derivativeFunction(a2) > 0)
    }

    while (true) {
        k += 1

        // Step 4
        val f1 = function(a1)
        val f2 = function(a2)
        val df1 = derivativeFunction(a1)
        val df2 = derivativeFunction(a2)
```

```
40
41          // Step 5
42          val z = 3 * (f1 - f2) / (a2 - a1) + df1 + df2
43          val w = if (a1 < a2) {
44              sqrt(z.pow(2) - df1 * df2)
45          } else {
46              - sqrt(z.pow(2) - df1 * df2)
47          }
48          val mu = (df2 + w - z) / (df2 - df1 + 2*w)
49
50          var a = when {
51              mu < 0 -> a2
52              mu in 0.0..1.0 -> a2 - mu * (a2 - a1)
53              mu > 1 -> a1
54              else -> throw Exception("Error in range")
55          }
56
57          // Step 6
58          while (function(a) > function(a1) && ((a - a1) / a).absoluteValue >
             ↪  delta) {
59              a -= (a - a1) / 2
60          }
61
62          // Step 7
63          if (derivativeFunction(a).absoluteValue <= eps && ((a - a1) /
             ↪  a).absoluteValue <= delta) {
64              PrintUtils.printInfoEndFunction(k, 0, a, function)
65              return a
66          } else {
67              if (derivativeFunction(a) * derivativeFunction(a1) <= 0) {
68                  a2 = a1;
69                  a1 = a;
70              } else {
71  //                  a2 = a2;
72                  a1 = a;
73              }
74          }
75
76      }
77  }
```

5 Результаты.

При последовательном запуске всех алгоритмов со следующими параметрами -

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

были получены следующие результаты:

Листинг 8. Результаты выполнения программ.

```
Start Svann Method:
         Iteration(s): 2
         Result: [-2.0, 4.0]

Start Bisection method:
         Iteration(s): 10
         f(0.12402344) = 2.8125114

Start Golden Section method:
         Iteration(s): 14
         f(0.12597115) = 2.8125114

Start Fibonacci method:
         Iteration(s): 10
         f(0.13934426) = 2.814969

Start Quadratic Interpolation method:
         Iteration(s): 3
         f(0.12498383) = 2.8125

Start Cubic Interpolation:
         Iteration(s): 2
         f(0.125) = 2.8125
```

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.