Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский университет ИТМО

Мегафакультет трансляционных информационных технологий
Факультет информационных технологий и программирования

Прикладная математика

Лабораторная работа 1

Выполнили:
Кутузов Михаил (M32001)
Капанин Дмитрий (M32021)
Соколов Денис (M32021)

Преподаватель:
Москаленко Мария Александровна

Ссылка на код:
GitHub

Санкт-Петербург, 2022 г.

Содержание

Цель работы

1. Решить задачу в соответствии с номером варианта. Для решения реализовать алгоритмы одномерной минимизации функции без прихводной: метод дихотомии, метод золотого сечения, метод Фибоначчи, метод парабол и комбинированный метод Брента.

2. Сравнить методы по количеству итераций и количеству вычислений функции в зависимости от разной точности. Для каждого метода обязательно указывайте, как изменяется отрезок при переходе к следующей итерации.

3. Протестировать реализованные алогоритмы для задач минимизации многомодальных функций, например, на различных полиномах. Могут ли метод золотого сечения/Брента не найти локальный минимум многомодальной функции?

Описание задачи

Вариант 5

Разделение моря

«И простер Моисей руку свою на море, и гнал Господь море сильным восточным ветром всю ночь и сделал море сушею, и расступились воды. И пошли сыны Израилевы среди моря по суше: воды же были им стеною по правую и по левую сторону». Моисеевой рукой раздвинулись воды, и для его спутников со стороны берега море описывалось следующей функцией:

$$y(x) = e^{\sin(x)} \cdot x^2$$

Чтобы успешно пересечь море, необходимо найти наиболее низкую точку расступившихся волн, чтобы выйти максимально сухим из воды.

Исходные данные

$$\varepsilon = 10^{-5}$$
$$a = -\pi + \varepsilon$$
$$b = \pi$$
$$f(x) = e^{sin(x)} * x^2$$

Метод Дихотомии

Описание

1. Суть метода: строить вложенные отрезки $[a_i, b_i]$ до тех пор, пока соблюдаются некоторые условия. При $i = 0$ мы полагаем, что $a_0 = a$, $b_0 = b$.
2. Для каждого i-ого отрезка мы находим его середину $x_i = \frac{a_i+b_i}{2}$ и строим две дополнительные точки
$$c_i = x_i - \frac{\varepsilon}{2},$$
$$d_i = x_i + \frac{\varepsilon}{2}.$$
Если $f(c_i) < f(d_i)$, то
$$a_{i+1} = a_i,$$
$$b_{i+1} = d_i,$$
иначе
$$a_{i+1} = c_i,$$
$$b_{i+1} = b_i.$$
3. Для каждого построенного отрезка вычисляется его длина и проверяется условие. Если $l_i < 2\varepsilon$, то процесс построения завершается и за искомую точку выбирается $x_* = \frac{a_i+b_i}{2}$.
4. Также на каждом шаге должно проверяться условие $a_i < c_i < d_i < b_i$. Если оно не выполняется, то это значит, что решение задачи не может быть найдено.

Реализация

```
import math


class Dichotomy:
    def __init__(self, func):
        self._func = func

    @staticmethod
    def do_method(func, left, right, e):
        i = 0
        while abs(right - left) > e:
            i += 1
            middle = (right + left) / 2
```

```python
            fml = func(middle - e / 2)
            fmr = func(middle + e / 2)
            if (fml < fmr) > 0:
                right = middle
            else:
                left = middle
        return (left + right) / 2, i, right - left

    def do(self, left, right, e):
        return Dichotomy.do_method(self._func, left, right, e)


print(Dichotomy.do_method((lambda x: x**2 * math.e ** math.sin(x)),
-math.pi + 0.00001, math.pi, 0.00001))
```

Результат

№ ит	l_i	$\frac{a_i+b_i}{2}$
1	3.1415876535897933	-1.5707888267948964
2	1.5707938267948964	-0.7853919133974482
3	0.7853969133974482	-0.3926934566987241
4	0.3926984566987241	-0.19634422834936202
5	0.19634922834936205	-0.098169614174681
6	0.09817461417468103	-0.04908230708734048
7	0.04908730708734051	-0.024538653543670224
8	0.024543653543670257	-0.012266826771835096
9	0.012271826771835128	-0.006130913385917531
10	0.006135913385917564	-0.0030629566929587493
11	0.003067956692958782	-0.0015289783464793583
12	0.001533978346479391	-0.0007619891732396628
13	0.0007669891732396955	-0.000378494586619815
14	0.00038349458661984776	-0.00018674729330989112
15	0.00019174729330992388	-9.087364665492918e-05
16	9.587364665496194e-05	-4.2936823327448214e-05
17	4.793682332748097e-05	-1.896841166370773e-05
18	2.3968411663740485e-05	-6.9842058318374865e-06
19	1.1984205831870243e-05	-9.921029159023653e-07
20	5.992102915935121e-06	2.0039485420651954e-06

$$x_* = 2.0039485420651954e - 06$$

Метод Золотого Сечения

Описание

1. Суть метода: строить вложенные отрезки $[a_i, b_i]$ до тех пор, пока соблюдаются некоторые условия. При $i = 0$ мы полагаем, что $a_0 = a$, $b_0 = b$.

2. Для каждого i-ого отрезка $(i > 0)$ мы находим одну из двух дополнительных точек как одну из границ будущего отрезка (вторая граница соответственно будет либо a_i, либо b_i, а другая дополнительная точка на следующей итерации будет равна нынешней выбранной, поэтому мы всегда будем знать $f(c_i)$ и $f(d_i)$)

$$c_i = a_i + r(b_i - a_i),$$

$$d_i = b_i - r(b_i - a_i).$$

где $r = \frac{3-\sqrt{5}}{2}$

Если $f(c_i) < f(d_i)$, то

$$a_{i+1} = a_i,$$

$$b_{i+1} = d_i,$$

$$d_{i+1} = c_i,$$

иначе

$$a_{i+1} = c_i,$$

$$b_{i+1} = b_i,$$

$$c_{i+1} = d_i.$$

3. Для $i = 0$ вычисляются обе точки

4. Для каждого построенного отрезка вычисляется его длина и проверяется условие. Если $l_i < 2\varepsilon$, то процесс построения завершается и за искомую точку выбирается $x_* = \frac{a_i+b_i}{2}$.

5. Также на каждом шаге должно проверяться условие $a_i < c_i < d_i < b_i$. Если оно не выполняется, то это значит, что решение задачи не может быть найдено.

Реализация

```python
import math


class GoldenRatio:
    def __init__(self, func):
        self._func = func

    @staticmethod
    def do_method(func, left, right, e):
        i = 0
        phi = (1 + 5**0.5) / 2
        resphi = 2 - phi
        x1 = left + resphi * (right - left)
        x2 = right - resphi * (right - left)
        f1 = func(x1)
        f2 = func(x2)
```

```
        while abs(right - left) > e:
            i += 1
            if (f1 < f2) > 0:
                right = x2
                x2 = x1
                x1 = left + resphi * (right - left)
                f2 = f1
                f1 = func(x1)
            else:
                left = x1
                x1 = x2
                x2 = right - resphi * (right - left)
                f1 = f2
                f2 = func(x2)
        return (x1 + x2) / 2, i, right - left

    def do(self, left, right, e):
        return GoldenRatio.do_method(self._func, left, right, e)


print(GoldenRatio.do_method((lambda x: x**2 * math.e ** math.sin(x)),
-math.pi + 0.00001, math.pi, 0.00001))
```

Результат

№ ит	l_i	$\frac{a_i+b_i}{2}$
1	3.8832158971110458	-1.1999747050342706
2	2.399959410068541	-0.4583464615130179
3	1.4832564870425053	5.000000000310312e-06
4	0.9167029230260364	-0.2832717820082343
5	0.5665535640164692	-0.10819710250345059
6	0.3501493590095669	5.0000000001992895e-06
7	0.21640420500690158	-0.06686757700133247
8	0.13374515400266535	-0.025538051499214356
9	0.08265905100423648	5.000000000282556e-06
10	0.05108610299842928	-0.015781474002903324
11	0.03157294800580721	-0.006024896506592288
12	0.019513154992621906	5.000000000233117e-06
13	0.012059793013185043	-0.0037216809897181995
14	0.007453361979436865	-0.0014184654728441105
15	0.004606431033748278	5.00000000026445e-06
16	0.0028469309456887497	-0.0008747500440294999
17	0.0017595000880595286	-0.000331034615214889
18	0.0010874308576291592	5.0000000002452054e-06
19	0.0006720692304302684	-0.0002026808135992003
20	0.000415361627198891	-7.432701198351158e-05
21	0.000256707603231416	5.000000000257145e-06
22	0.00015865402396753744	-4.402678963168215e-05
23	9.80535792638786e-05	-1.3726567279852718e-05
24	6.060044470363501e-05	5.000000000249776e-06
25	3.745313456020499e-05	-6.573655071465238e-06
26	2.314731014343003e-05	5.792571369222428e-07
27	1.4305824416774962e-05	-3.841485726405292e-06
28	8.84148572665507e-06	-1.1093163813453444e-06

$$x_* = -1.1093163813453444e-06$$

Метод Фибоначчи

Описание

1. Суть метода: строить вложенные отрезки $[a_i, b_i]$ до тех пор, пока соблюдаются некоторые условия. Для i числа Фибоначчи имеем формулу:

$$F_i = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^i - (\frac{1-\sqrt{5}}{2})^i)$$

2. Для $i = 0$ вычисляются две точки, симметричные относительно середины отрезка

$$c_0 = a_0 + \frac{F_i}{F_{i+2}}(b_0 - a_0)$$

$$d_0 = a_0 + \frac{F_{i+1}}{F_{i+2}}(b_0 - a_0)$$

3. Новый отрезок получается путём исключения отрезков аналогично методу Золотого Сечения

   Если $f(c_i) < f(d_i)$, то

$$a_{i+1} = a_i,$$
$$b_{i+1} = d_i,$$
$$d_{i+1} = c_i,$$

   иначе

$$a_{i+1} = c_i,$$
$$b_{i+1} = b_i,$$
$$c_{i+1} = d_i.$$

4. Особенность метода Фибоначчи в том, что у нас есть фиксированное N, и, когда мы доходим до этапа $i = N$, вычисления прекращаются и за искомую точку берётся $x_* = \frac{a_i+b_i}{2}$.

5. Также на каждом шаге должно проверяться условие $a_i < c_i < d_i < b_i$. Если оно не выполняется, то это значит, что решение задачи не может быть найдено.

Реализация

```
import math


class Fibonacci:
    def __init__(self, func):
        self._func = func

    @staticmethod
    def _get_fibonacci_list(left, right, e):
        a = [1, 1]
        b = (right - left) / e
        while a[-1] < b:
            a.append(a[-1] + a[-2])
        return a
```

```python
    @staticmethod
    def _last_part(func, left, right, e, x1, f1, n):
        x2 = x1 + e
        f2 = func(x2)
        if f1 - e < f2 < f1 + e:
            return (x1 + right) / 2, n, right - x1
        else:
            return (left + x2) / 2, n, x2 - left

    @staticmethod
    def do_method(func, left, right, e):
        fibonacci_list = Fibonacci._get_fibonacci_list(left, right, e)
        n = len(fibonacci_list)
        x1 = left + (right - left) * fibonacci_list[-3] / fibonacci_list[-1]
        x2 = left + (right - left) * fibonacci_list[-2] / fibonacci_list[-1]
        f1 = func(x1)
        f2 = func(x2)
        for i in range(0, n):
            if f1 <= f2:
                right = x2
                x2 = x1
                x1 = left + (right - left) * fibonacci_list[-3 - i] / fibonacci_list[-1 - i]
                f2 = f1
                f1 = func(x1)
                if i == n - 3:
                    return Fibonacci._last_part(func, left, right, e, x1, f1, n)
            else:
                left = x1
                x1 = x2
                x2 = left + (right - left) * fibonacci_list[-2 - i] / fibonacci_list[-1 - i]
                f1 = f2
                f2 = func(x2)
                if i == n - 3:
                    return Fibonacci._last_part(func, left, right, e, x1, f1, n)

    def do(self, left, right, e):
        return Fibonacci.do_method(self._func, left, right, e)


print(Fibonacci.do_method((lambda x: x**2 * math.e ** math.sin(x)),
-math.pi + 0.00001, math.pi, 0.00001))
```

Результат

№ ит	l_i	$\frac{a_i+b_i}{2}$	$\frac{l_i}{l_{i-1}}$
0	-3.8832158971151047	-1.1999747050322407	0.6180339887505408
1	-2.399959410073558	-0.4583464615114673	0.6180339887505409
2	-1.4832564870506229	5.000000000254801e-06	0.6180339887519853
3	-0.9167029230376207	-0.2832717820065008	0.6180339887543226
4	-0.5665535640130023	-0.10819710249419162	0.618033988738303
5	-0.3501493590190088	5.00000280512225e-06	0.6180339887703414
6	-0.21640420498491703	-0.06686757701424076	0.6180339886704432
7	-0.13374515403409176	-0.025538051538828134	0.618033988957902
8	-0.08265905095429216	5.0000010716755305e-06	0.6180339882312468
9	-0.05108610308540918	-0.015781473933369813	0.6180339901755971
10	-0.03157294786888298	-0.0060248963251067125	0.618033985017358
11	-0.01951315521438355	5.000002143002585e-06	0.6180339984539399
12	-0.01205979265103255	-0.003721681279532497	0.6180339631667066
13	-0.007453362563350999	-0.0014184662356917215	0.6180340557275541
14	-0.00460643008900578	5.000001480888125e-06	0.618033813577794
15	-0.002846932476487868	-0.0008747488047780678	0.6180344478216819
16	-0.0017594976125179119	-0.00033103137279308966	0.6180327868852459
17	-0.0010874348631515372	5.0000018900976835e-06	0.6180371348133711
18	-0.0006720627480421518	-0.000202686055664595	0.6180257510729614
19	-0.0004153721151093855	-7.434073919821183e-05	0.6180555555555556
20	-0.00025669063343857043	5.000001637195709e-06	0.6179775293076009
21	-0.0001586814824892981	-4.4004573837440463e-05	0.6181818181818182
22	-9.800915094927234e-05	-1.3668408067427592e-05	0.6176470588235294
23	-6.067233122734685e-05	5.000001793535155e-06	0.619047615857316
24	-3.733681921682883e-05	-6.667754211723856e-06	0.6153846153846153
25	-2.3335512010518018e-05	3.328993914315504e-07	0.625
26	-1.4001307206310812e-05	-4.3342030106720525e-06	0.6000000000000001
27	4.667102594521381e-06	3.3289929522266326e-07	0.5000000206142655

$$x_* = 1.2242348284034293e-06$$

Метод Парабол

Описание

1. Суть метода всё та же: строить вложенные отрезки $[a_i,\ b_i]$ до тех пор, пока соблюдаются некоторые условия. Пусть парабола проходит через 3 точки: $(x_1, f_1), (x_2, f_2), (x_3, f_3)$. Тогда вершина этой параболы вычисляется по формуле:

$$u = x_2 - \frac{(x_2 - x_1)^2 (f_2 - f_3) - (x_2 - x_3)^2 (f_2 - f_1)}{2[(x_2 - x_1)(f_2 - f_3) - (x_2 - x_3)(f_2 - f_1)]}.$$

   Если $x_1 < x_2 < x_3$, $f_2 < f_1$ и $f_2 < f_3$, то вершина параболы точно лежит в (x_1, x_3).

2. Для $i = 0$

$$a_0 = a,$$
$$b_0 = b,$$
$$e_0 = \frac{a+b}{2},$$
$$x_0 = \frac{a+b}{2}.$$

3. Строим параболу, проходящую через точки $(a_i, f(a_i)), (e_i, f(e_i)), (b_i, f(b_i))$, а вершину обозначим как x_{i+1}.

4. Вычислим две точки

$$c_{i+1} = min\{e_i, x_{i+1}\},$$
$$d_{i+1} = max\{e_i, x_{i+1}\}.$$

   Если $f(c_i) < f(d_i)$, то

$$a_{i+1} = a_i,$$
$$b_{i+1} = d_{i+1},$$
$$e_{i+1} = c_{i+1},$$

   иначе

$$a_{i+1} = c_{i+1},$$
$$b_{i+1} = b_i,$$
$$e_{i+1} = d_{i+1}.$$

5. Если $|x_{i+1} - x_i| < \varepsilon$, то x_{i+1} - искомая точка. Если $x_{i+1} \notin [a_i, b_i]$, то вычисления прекращаются.

Реализация

```python
import math


class Parabolic:
    def __init__(self, func):
        self._func = func
```

```
    @staticmethod
    def _top_of_parabola(m, l, r, fm, fl, fr):
        denominator = 2 * ((m - l) * (fm - fr) - (m - r) * (fm - fl))
        if denominator == 0:
            return None
        return m - ((fm - fr) * (m - l) ** 2 - (fm - fl) * (m - r) ** 2) / denominator


    @staticmethod
    def do_method(func, left, right, e):
        i = 0
        middle = (left + right) / 2
        fl = func(left)
        fm = func(middle)
        fr = func(right)
        while fm < fl and fm < fr and right - left > e:
            i += 1
            u = Parabolic._top_of_parabola(middle, left, right, fm, fl, fr)
            fu = func(u)
            if fu <= fm:
                if u < middle:
                    right = middle
                    fr = fm
                else:
                    left = middle
                    fl = fm
                middle = u
                fm = fu
            else:
                if u < middle:
                    left = u
                    fl = fu
                else:
                    right = u
                    fr = fu
            if i > 1 and abs(xi - u) < e:
                break
            xi = u
        return u, i, right - left

    def do(self, left, right, e):
        return Parabolic.do_method(self._func, left, right, e)


print(Parabolic.do_method((lambda x: x ** 2 * math.e ** math.sin(x)),
-math.pi + 0.00001, math.pi, 0.00001))
```

Результат

№ ит	l_i	u_i	$\frac{l_i}{l_{i-1}}$
1	3.1416005075589273	-7.853969133947032e-06	0.500002045775982
2	1.2853969133979788e-05	-2.3707409359717492e-11	4.091535223225286e-06

$$x_* = -2.3707409359717492e - 11$$

Комбинированный метод Брента

Описание

1. Суть метода заключается в комбинировании методов Парабол и Золотого Сечения.
2. Введём параметры:
   - a, b - левая и правая граница поиска точки минимума функции соответственно;
   - x - точка, в которой функция принимает наименьшее из всех вычисленных на текущий момент значений;
   - w - точка, в которой функция либо принимает второе снизу из всех вычисленных на текущий момент значений, либо принимает значение $f(x)$;
   - v - предыдущее значение w;
   - u - текущее приближение к точке минимума функции;
   - d_{cur} - принимает значение $|u - x|$;
   - d_{prv} - предыдущее значение d_{cur}.
3. Начальные значения:
$$x, w, v = a + r(b - a),$$
$$d_{cur}, d_{prv} = b - a.$$
   где $r = \frac{3-\sqrt{5}}{2}$.
4. Параболы строятся по 3м точкам: x, w, v - три наилучших приближения к точке минимума. Если парабола не может быть построена или вершина отвергается, то выполняется шаг Золотого Сечения. Вершина параболы может быть отвергнута в двух случаях:
   - $u \notin [a, b]$ - вершина вышла за границы отрезка поиска.
   - $|u - x| > \frac{d_{prv}}{2}$ - вершина параболы сильно отдалилась от точки текущего наименьшего значения функции.

Реализация

```python
import math


class Brent:
    def __init__(self, func):
        self._func = func

    @staticmethod
    def _top_of_parabola(x, w, v, fx, fw, fv):
        denominator = 2 * ((w - x) * (fw - fv) - (w - v) * (fw - fx))
        if denominator == 0:
            return None
        return w - ((fw - fv) * (w - x) ** 2 - (fw - fx) * (w - v) ** 2) / denominator

    @staticmethod
    def do_method(func, left, right, e):
        r = (3 - 5 ** 0.5) / 2
        x = w = v = left + r * (right - left)
```

```
        fx = func(x)
        fv = fx
        fw = fx
        d_cur = d_prv = right - left
        i = 0
        while right - left > e:
            i += 1
            if max(x - left, right - x) < e:
                break
            g = d_prv / 2
            d_prv = d_cur
            u = Brent._top_of_parabola(x, w, v, fx, fw, fv)
            if u is None or abs(u - x) > g/2 or u < left or u > right:
                if x < (left + right) / 2:
                    u = x + r * (right - x)
                    d_prv = right - x
                else:
                    u = x - r * (x - left)
                    d_prv = x - left
            fu = func(u)
            d_cur = abs(u - x)
            if fu > fx:
                if u < x:
                    left = u
                else:
                    right = u
                if fu <= fw or w == x:
                    v = w
                    fv = fw
                    w = u
                    fw = fu
                else:
                    if fu <= fw or v == x or v == w:
                        v = u
                        fv = fu
            else:
                if u < x:
                    right = x
                else:
                    left = x
                v = w
                fv = fw
                w = x
                fw = fx
                x = u
                fx = fu
        return x, i, right - left

    def do(self, left, right, e):
        return Brent.do_method(self._func, left, right, e)


print(Brent.do_method((lambda x: x**2 * math.e ** math.sin(x)),
-math.pi + 0.00001, math.pi, 0.00001))
```

Результат

№ ит	l_i	u_i	$\frac{l_i}{l_{i-1}}$
1	3.883215897111045	-0.7416232435212526	0.6180339887498947
2	2.3999594100685404	-0.7416232435212526	0.6180339887498949
3	1.4832564870425047	-0.4826392410157896	0.6180339887498949
4	1.2242724845370416	-0.015008763413919823	0.8253949975827468
5	0.7566420069351717	-0.015008763413919823	0.6180339887498948
6	0.34150378212653526	-0.015008763413919823	0.451341293499721
7	0.28901152933330193	-0.009733807821277739	0.8462908596022995
8	0.2837365737406595	0.0009535166967194995	0.9817482866347554
9	0.010687324517996689	6.194659541360406e-05	0.037666362066404095
10	0.009795754416690994	-4.9338739612007606e-06	0.9165768663799309
11	6.688046937480482e-05	2.701488955186317e-08	0.006827495517940622
12	4.960888850752624e-06	-1.520439123792205e-10	0.07417544908291998

$$x_* = -1.520439123792205e - 10$$

Общие результаты