Минимизация скалярных функций. Методы дихотомии и касательных парабол.

А. Климовский

13 Мая 2001

1 Постановка задачи

Рассмотрим следующую задачу на безусловный минимум (здесь и далее $y(x) : \mathbb{R} \mapsto \mathbb{R}$):

$$\min_{x \in [a;b]} y(x) = ? \tag{1}$$

Тестовая задача:

$$\min_{x \in [a;b]} 0.5x^4 + 8x^2 \sin x + 2 \sin^2 x + 1 = ? \tag{2}$$

2 Краткое изложение метода и вычислительная схема решения

2.1 Метод деления пополам

Рассмотрим следующий алгоритм:

```
while ((b-a) > eps) {
    c = (a+b)/2;
    if (f((a+c)/2) < f((c+b)/2))
        b = c;
    else
        a = c;
}
```

Легко видеть, что этот алгоритм отыскивает локальное решение задачи (1) в случае непрерывности функции $y(x)$.

2.2 Метод касательных парабол

Рассмотрим $h > 0$, и $x_i \in [a;b]$. Рассмотрим итерационную формулу Ньютона для нахождения решения уравнения $y'(x) = 0$:

$$x_{i+1} = x_i - \frac{y'(x_i)}{y''(x_i)} \tag{3}$$

При некоторых условиях на функцию y(x) последовательность $\{x_i\}_{i=1}^{\infty}$, получающаяся итерациями по формуле (3), сходится к локальному минимуму функции $y(x)$.

Заменяя производные в формуле (3) конечными разностями с точностью до $O(h^2)$ получаем соотношение:

$$x_{i+1} = x_i - \frac{h(y(x_i+h) - y(x_i-h))}{2(y(x_i+h) - 2y(x_i) + y(x_i-h))} \tag{4}$$

3 Результаты счета

Ниже приведен вывод генерируемый программой:

```
        Dihotomy x0: -2.02029 f0: -18.4571 f0': -0.000410515
Parabola approx. x0: -2.02028 f0: -18.4571 f0': -1.99493e-12
```

4 Приложение: исходные тексты программы

```
//---------------------------------------------------
// Scalar function 1d minimization.
// Dihotomy and parabola methods.
//
// Author: Anton Klimovsky, root@ludus.kharkiv.com
//---------------------------------------------------
#include <iostream.h>
#include <math.h>

const double eps = 1e-5;
const double a = -3;
const double b = 2;
const int MAX_ITERATIONS = 1000;

inline double sqr(double x)
{
    return x*x;
}

double f(double x)
{
    return 0.5*sqr(sqr(x))+8*sqr(x)*sin(x)+2*sqr(sin(x))+1;
}

void dihotomy(
    double a,
    double b,
    double (*f)(double),
    double eps,
    double* x0
)
{
    double c;

    while ((b-a) > eps) {
        c = (a+b)/2;
        if (f((a+c)/2) < f((c+b)/2))
            b = c;
```

```
        else
            a = c;
    }
    *x0 = c;
}

inline double derive(double x0, double (*f)(double), double h)
{
    return (f(x0+h)-f(x0-h))/(2*h);
}

inline double derive2(double x0, double (*f)(double), double h)
{
    return (f(x0-h)-2*f(x0)+f(x0+h))/sqr(h);
}

void parabola(
    double a,
    double b,
    double (*f)(double),
    double eps,
    double* x0
)
{
    double xNew;
    double xOld;

    xNew = (a+b)/2;

    do {
        xOld = xNew;
        xNew = xOld-derive(xOld, f, eps)/derive2(xOld, f, eps);
        if (xNew < a || xNew > b) {
            xNew = a;
            break;
        }
    } while (fabs(xOld-xNew) > eps);
    *x0 = xNew;
}

void findGlobalMinimum(
    double a,
    double b,
    int n0,
    void (*method)(double, double, double (*f)(double), double, double*),
    double (*f)(double),
    double eps,
    double* resultX0,
    double* resultF0
)
{
    double h;
    int i;
    int n;
```

```
    double f0;
    double x0;
    double tempX;
    double tempF;
    double x;
    double oldF0;

    oldF0 = f(a);

    for (n = n0, h = (b-a)/n0; n < MAX_ITERATIONS; n <<= 1, h /= 2) {

        method(a, a+h, f, eps, &x0);
        f0 = f(x0);

        for (x = a+h, i = 1; i < n; i++, x += h) {
            method(x, x+h, f, eps, &tempX);
            if (f0 > (tempF = f(tempX))) {
                x0 = tempX;
                f0 = tempF;
            }
        }

        if (fabs(oldF0-f0) < eps)
            break;

        oldF0 = f0;
    }

    *resultX0 = x0;
    *resultF0 = f0;
}

void main()
{
    double f0;
    double x0;

    findGlobalMinimum(a, b, 10, dihotomy, f, eps, &x0, &f0);

    cout << "x0: " <<  x0 << " f0: " <<
         f0 << " f0': " << derive(x0, f, eps) << endl;

    findGlobalMinimum(a, b, 10, parabola, f, eps, &x0, &f0);

    cout << "x0: " <<  x0 << " f0: " <<
         f0 << " f0': " << derive(x0, f, eps) << endl;
}
```