Методы оптимизации

Русначенко Николай

20 Декабря, 2015

1 Численные методы поиска безусловного экстремума

Требуется найти безусловный минимум функций f(x) многих переменных т.е.е найти такую точку $x^e \in R^n$, что $f(x^e) = min_{x \in R^n} f(x)$, $f(x) \in C^0(X)$.

$$f(x) = \sum_{i=1}^{2} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right)$$

2 Применение Метода Нелдера-Мида.

Необходимо определить значения следующих коэффициентов:

- коэффициент отражения $\alpha >$, обычно выбирается равным 1;
- коэффициент сжатия $\beta > 0$, обычно выбирается равным 0,5;
- коэффициент растяжения $\gamma > 0$, обычно выбирается равным 2.

1. «Подготовка». Вначале выбирается $n+1$ точка $x_i = \left(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(n)}\right), i = 1..n+1$, образующие симплекс n-мерного пространства. В этих точках вычисляются значения функции: $f_1 = f(x_1), f_2 = f(x_2), \ldots, f_{n+1} = f(x_{n+1})$.

2. «Сортировка». Из вершин симплекса выбираем три точки: x_h с наибольшим (из выбранных) значением функции f_h, x_g со следующим по величине значением f_g и x_l с наименьшим значением функции f_l. Целью дальнейших манипуляций будет уменьшение по крайней мере f_h;

3. Найдём центр тяжести всех точек, за исключением x_h: $x_c = \frac{1}{n} \sum\limits_{i \neq h} x_i$. Вычислять $f_c = f(x_c)$ не обязательно;

4. «Отражение». Отразим точку x_h относительно x_c с коэффициентом α (при $\alpha = 1$ это будет центральная симметрия, в общем случае — гомотетия), получим точку x_r и вычислим в ней функцию: $f_r = f(x_r)$. Координаты новой точки вычисляются по формуле: $x_r = (1 + \alpha)x_c - \alpha x_h$;

5. Далее смотрим, насколько нам удалось уменьшить функцию, ищем место f_r в ряду f_h, f_g, f_l.

    - Если $f_r < f_l$, то направление выбрано удачное и можно попробовать увеличить шаг. Производим «растяжение». Новая точка $x_e = (1 - \gamma)x_c + \gamma x_r$ и значение функции $f_e = f(x_e)$.

        - Если $f_e < f_r$, то можно расширить симплекс до этой точки: присваиваем точке x_h значение x_e и заканчиваем итерацию (на шаг 9).

  - Если $f_r < f_e$, то переместились слишком далеко: присваиваем точке x_h значение x_r и заканчиваем итерацию (на шаг 9).

- Если $f_l < f_r < f_g$, то выбор точки неплохой (новая лучше двух прежних). Присваиваем точке x_h значение x_r и переходим на шаг 9.
- Если $f_g < f_r < f_h$, то меняем местами значения x_r и x_h. Также нужно поменять местами значения f_r и f_h. После этого идём на шаг 6.
- Если $f_h < f_r$, то просто идём на следующий шаг 6. В результате (возможно, после переобозначения) $f_l < f_g < f_h < f_r$.

6. «Сжатие». Строим точку $x_s = \beta x_h + (1 - \beta) x_c$ и вычисляем в ней значение $f_s = f(x_s)$;

7. Если $f_s < f_h$, то присваиваем точке $\mathrm{x}_h \mathrm{x}_s$ и идём на шаг 9;

8. Если $f_s > f_h$, то первоначальные точки оказались самыми удачными. Делаем «глобальное сжатие» симплекса — гомотетию к точке с наименьшим значением x_l :$\mathrm{x}_i \leftarrow x_l + (x_i - x_l)/2, i \neq l$;

9. Последний шаг — проверка сходимости. Может выполняться по-разному, например, оценкой дисперсии набора точек. Суть проверки заключается в том, чтобы проверить взаимную близость полученных вершин симплекса, что предполагает и близость их к искомому минимуму. Если требуемая точность ещё не достигнута, можно продолжить итерации с шага 2.

3 Пример работы алгоритма.

Нахождение минимума исходной функции: На вход программе необходимо ввести $n+1$ точек для определения симплекса, который в дальнешем будет преобразовываться для поиска ответа (минимума функции).

```
10 10 10
-10 10 10
10 -10 5
0 0 5

points: 4
1.00998 1.02016 1.04084
1.01012 1.02041 1.04122
1.01016 1.02033 1.04116
1.01007 1.02014 1.04055
minim: 0.000514108
```

4 Реализация алгоритма (язык C++).

```
// Nelder-Mead Minimization
//
// See: en.wikipedia.org/wiki/Nelder%E2%80%93Mead_method
//

#include <bits/stdc++.h>
#include <stdio.h>

using namespace std;

double eps = 1e-10;

vector<vector<double> > pts;

// function example
double f(vector<double>& args)
{
    double x1 = args[0], x2 = args[1], x3 = args[2];

    return 100*pow(pow(x1,2) - x2,2) + pow(x1-1,2) +
        100*pow(pow(x2,2) - x3,2) + pow(x2-1,2);
}

void xc_calc(vector<vector<double> >& pts, int length,
        int except_index, vector<double> *result)
{
    for (int j = 0; j < length; j++)
    {
        double m = 0;
        for(int i = 0; i < pts.size(); i++)
            if (except_index != i)
                m += pts[i][j]/(pts.size() - 1);

        result->push_back(m);
    }
}

void global_press(vector<vector<double> >& pts, int length,
        int min_index)
{
    for (int j = 0; j < length; j++)
        for(int i = 0; i < pts.size(); i++)
            if (min_index != i)
                pts[i][j] = pts[min_index][j] +(pts[i][j] - pts[min_index][j])/2;
}
void xr_calc(vector<double>& xc, vector<double>& xh, double alpha, vector<double>* xr)
{
    for (int i = 0; i < xc.size(); i++)
        xr->push_back( (1 + alpha)*xc[i] - alpha*xh[i]);
}

void xs_calc(vector<double>& xh, vector<double>& xc, double beta, vector<double>* xs)
{
    for (int i = 0; i < xc.size(); i++)
        xs->push_back( beta*xh[i] + (1 - beta)*xc[i]);
}
void press(vector<vector<double> > &pts, vector<double>& xc, double beta)
{
    vector<double>* xh = &pts[0];

    // 6
```

```
    vector<double> xs;
    xs_calc(*xh, xc, beta, &xs);

    // 7
    if (f(xs) < f(*xh))
        *xh = xs;
    // 8
    else if (f(xs) >= f(*xh))
        global_press(pts, xs.size(), pts.size()-1);
}

double mx(vector<vector<double> > pts)
{
    double m = 0;

    for (int i = 0; i < pts.size(); i++)
        m += f(pts[i])/(pts.size());

    return m;
}

bool cmp(vector<double> x, vector<double> y)
{
    return f(x) > f(y);
}

int main()
{
    double alpha = 1, beta = 0.5, gamma = 2;

    int n = 3;
    // выбираем n+1 точку
    for (int i = 0; i < n+1; i++)
    {
        vector<double> pt;
        double x1, x2, x3;

        cin >> x1 >> x2 >> x3;
        pt.push_back(x1); pt.push_back(x2); pt.push_back(x3);
        pts.push_back(pt);
    }

    cout << "Calculating..." << endl;

    while (true)
    {
        // 2
        sort(pts.begin(), pts.end(), cmp);
        vector<double>* xh = &pts[0];
        vector<double>* xg = &pts[1];
        vector<double>* xl = &pts[pts.size()-1];

        cout << f(*xh) << " " << f(*xg) << " " << f(*xl) << endl;

        // 3
        vector<double> xc;
        xc_calc(pts, n, 0, &xc);

        // 4
        vector<double> xr;
        xr_calc(xc, *xh, alpha, &xr);

        // 5
        if (f(xr) < f(*xl))
```

```
        {
            vector<double> xe;
            xr_calc(xc, xr, gamma, &xe);
            if (f(xe) < f(xr))
                pts[0] = xe;
            else
                pts[0] = xr;
        }
        else if (f(*xl) < f(xr) && f(xr) < f(*xg))
                pts[0] = xr;
        else if (f(*xg) < f(xr) && f(xr) < f(*xh))
        {
            vector<double> *c = &xr;
            xr = *xh; *xh = *c;
            press(pts, xc, beta);
        }
        else if (f(*xh) < f(xr))
            press(pts, xc, beta);

        // 9
        double diff = 0;
        double m = mx(pts);
        for (int i = 0; i < pts.size(); i++)
        {
             diff += pow(f(pts[i])-m, 2);
        }

        if (diff < eps)
            break;
    }

    // show
    cout << "points: " << pts.size() << endl;
    for (int i = 0; i < pts.size(); i++)
    {
        for (int j = 0; j < pts[i].size(); j++)
            cout << pts[i][j] << " ";
        cout << endl;
    }
    cout << "minim: " << mx(pts) << endl;

    return 0;
}
```

5 Применение метода Левенберга-Марквардта.

1. Задать $x^0, \epsilon_1 > 0, M$ — предельное число итераций. Найти градиент $f(x^0)$ и матрицу Гессе $H(x^0)$;
2. Положить $k = 0, \gamma^k = \gamma^0$;
3. Вычислить $\bigtriangledown f(x^k)$;
4. Проверить выполнение условия:
   (a) Если неравенство выполнено, то $\{x^e = x^k\}$, КОНЕЦ;
   (b) Иначе переход на Шаг 5.
5. Проверить выполнение условия $k \geq M$:
   (a) Если неравенство выполнено, то $\{x^e = x^k\}$, КОНЕЦ;
   (b) Иначе переход на Шаг 6.
6. Вычислить $H(x^k)$;
7. Вычислить $\left[H(x^k) + \gamma^k E\right]$;
8. Вычислить $\left[H(x^k) + \gamma^k E\right]^{-1}$;
9. Вычислить $d^k = -\left[H(x^k) + \gamma^k E\right]^{-1} \bigtriangledown f(x^k)$;
10. Вычислить $x^{k+1} = x^k + \alpha^k d^k$;
11. Проверить выполнение условия $f(x^{k+1} < f(x^k)$:
    (a) Если неравенство выполнено, то перейти на Шаг 12;
    (b) Иначе перейти на Шаг 13.
12. Положить $k = k + 1, \gamma^{k+1} = \dfrac{\gamma^k}{2}$ и перейти на Шаг 2;
13. Положить $\gamma^{k+1} = 2\gamma^k$ и перейти на Шаг 7.

6 Пример работы алгоритма.

В качестве параметров были рассмотрены следующие значения: $\gamma = 10^4, \alpha = 1$

```
ans = 0
xk =
   1   1   1
```

Минимум функции достигается в точке с координатами $(1, 1, 1)$.

7 Реализация на языке `Octave`.

```
% function
f = @(x) 100*(x(1)^2 - x(2))^2 + (x(1) - 1)^2 + 100*(x(2)^2 - x(3))^2 + (x(2) - 1)^2;
% gradient
df = @(x) [400*x(1)*(x(1)^2 - x(2)) + 2*(x(1) - 1), -200*(x(1)^2 - x(2)) + 400*x(2)*(x(2)^2 - x(3)) + 2*(x(2) - 1), -200*(x(2)^
% hesse
H = @(x) [400*(x(1)^2 - x(2)) + 800*x(1)^2 + 2, -400*x(1),                    0;
          -400*x(1), 200 + 400*(x(2)^2 - x(3)) + 800*x(2)^2 + 2, -400*x(2);
          0,                   -400*x(2),                          200];

%1
xk = [0, 0, 0];
eps = 1e-20;

% 2
k = 0;
gamma = 10^4;
alpha = 1;
M = 128;

while true
    % 3-4
    if (abs(df(xk)) < eps)
        break;
    end
    % 5
    if (k >= M)
        break;
    end
    while true
        % 6-9
        dk = -( H(xk) + gamma*eye(length(xk)) )^(-1) * df(xk)';
␣␣␣␣␣␣␣␣%␣10
␣␣␣␣␣␣␣␣xn␣=␣xk␣+␣alpha*dk';
        % 11
        f(xn)
        if (f(xn) < f(xk))
            k++;
            xk = xn;
            gamma /= 2;
            break;
        else
            xk = xn;
            gamma *= 2;
        end
    end
end

% show result
```

```
xk
% minimum
f(xk)
```