Методы оптимизации

Русначенко Николай

20 Декабря, 2015

1 Численные методы поиска условного экстремума (метод штрафов)

Даны дважды непрерывно дифференцируемые целевая функция $f(x) = f(x_1, x_2, \ldots, x_n)$ и функции ограничений $g_j(x) = 0, j = 1, \ldots, m$ и $g_j(x) \leq 0, j = m+1, \ldots, p$, определяющие множество допустимых решений X.

Требуется найти локальный минимум целевой функции на множестве X, т.е. такую точку, что:

$$f(x_e) = \min_{x \in X} f(x)$$

где:

$$f(x) = \sum_{i=1}^{2} (100(x_1^2 - x_2)^2 + (x_1 - 1)^2)$$

$$g_1(x) = -(x_1 + 1)^2 \leq 0$$

$$g_2(x) = x_1 + x_2 - 5 \leq 0$$

1.1 Решение поставленной задачи

1. Задать начальную точку x_0; начальное значение параметра штрафа $r_0 > 0$; число $C > 1$ для увеличения параметра; малое число $\epsilon > 0$ для условия останова. Положить $k > 0$.

2. Составляем вспомогательную ф-цию:

$$F(x, r_k) = f(x) + \frac{r_k}{2} \left[\sum_{j=1}^{m} [g_j(x)]^2 + \sum_{j=m+1}^{p} [g_j^+(x)]^2 \right]$$

где функция $g_j^+(x)$ представляет собой функцию срезки функции g.

$$g_j^+(x) = \begin{cases} g_j(x), & g_j(x) > 0 \\ 0, & g_j(x) \leq 0 \end{cases}$$

3. Реализуем поиск минимума функции F на основе безусловного алгоритма (Марка-

Левенберга). Это будет точка $x^*(r_k)$

$$F(x^*(r_k), r_k) = \min_{x \in R^n} F(x, r_k)$$

4. Вычислить $P(x^*(r_k), r_k)$:

$$P(x, r_k) = \frac{r_k}{2} \left[\sum_{j=1}^{m} [g_j(x)]^2 + \sum_{j=m+1}^{p} [g_j^+(x)]^2 \right]$$

5. Проверить условия останова:

   (a) Если $P(x^*(r_k), r_k) < \epsilon$ то конец, точка найдена.

   (b) Если $P(x^*(r_k), r_k) > \epsilon$, положить $r_{k+1} = Cr_k, x_{k+1} = x^*(r_k), k = k+1$, перейти на шаг 2.

1.2 Результат работы программы

Программа реализована на языке `Octave`, реализация алгоритма представлена в листинге 1. Минимум функции достигается в точке $(1, 1, 1)$, а значение функции в этой точке равно 0.

```
g1 = @(x) -(x (1) + 1) ^ 2
g2 = @(x) x (1) + x (2) - 5


ans =    1.9992
ans =    1.9977
...
ans =    4.9788e-06
xrk =
    0.99900    0.99804    0.99608


ans =     4.9788e-06
```

1.3 Реализация алгоритма на языке Octave

Листинг 1: "Реализация программы"

```
function xk = minsearcher(f, df, H, xk, eps, gamma, alpha, M)
    k = 0;
    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
                % added to halt infinity recurse
                k++;
                xk = xn;
                gamma *= 2;
            end
        end
    end
endfunction

% start parameters
C = 4;
xk = [0, 0, 0];
gamma = 10^4;
alpha = 1;
M = 30;
r = 0.001;
eps = 10^-1;

% border cases
g1 = @(x) -(x(1)+1)^2
g2 = @(x) x(1) + x(2) - 5
g = @(x) [g1(x) * (1 && !(g1(x) <= 0)), g2(x) *(1 && !(g2(x) <= 0))];

% penality function
P = @(x) r/2*(g(x)(1)^2 + g(x)(2)^2);

% 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 + P(x);

% gradient
dg1 = @(x) [4*(x(1) + 1)^3, 0, 0];
dg2 = @(x) [2*(x(1) + x(2)), -10, 0];
df = @(x) [400*x(1)*(x(1)^2 - x(2)) + 2*(x(1) - 1) + (r/2)*(dg1(x)(1) + dg2(x)(1)), -200*(x(1)^2 - x(2)) + 400*x(2)*(x(2)^2 - x

% hesse
```

```
Hfg = @(x) [400*(x(1)^2 - x(2)) + 800*x(1)^2 + 2 + (r/2)*(12*(x(1) + 1) + 2), -400*x(1) + (r/2)*2,  0;
            -400*x(1) + (r/2)*2, 200 + 400*(x(2)^2 - x(3)) + 800*x(2)^2 + 2, -400*x(2);
            0,  -400*x(2),  200];

% algorithm
while true
    xrk = minsearcher(f, df, Hfg, xk, eps, gamma, alpha, M);

    p = P(xrk, r);

    if (p <= eps)
        p
        break
    else
        r = C*r
        xk = xrk;
    end
end

% show minimum
xrk
f(xrk)
```

2 Численные методы поиска условного экстремума (метод модифицированных функций Лагранжа)

Даны дважды непрерывно дифференцируемые целевая функция $f(x) = f(x_1, x_2, \ldots, x_n)$ и функции ограничений $g_j(x) = 0, j = 1, \ldots, m$ и $g_j(x) \leq 0, j = m+1, \ldots, p$, определяющие множество допустимых решений X.

Требуется найти локальный минимум целевой функции на множестве X, т.е. такую точку, что:

$$f(x_e) = \min_{x \in X} f(x)$$

где:

$$f(x) = \sum_{i=1}^{2} (100(x_1^2 - x_2)^2 + (x_1 - 1)^2)$$

$$g_1(x) = -(x_1 + 1)^2 \leq 0$$

$$g_2(x) = x_1 + x_2 - 5 \leq 0$$

2.1 Решение поставленной задачи

1. Задать начальную точку x_0; начальное значение параметра штрафа $r_0 > 0$; число $C > 1$ для увеличения параметра; малое число $\epsilon > 0$ для условия останова. Положить $k > 0$.

2. Составим модифицированную функцию Лагранжа (для рассматриваемой задачи, $m = 0, p = 2$):

$$L(x, \lambda_k, \mu_k, r_k) = f(x) + \sum_{j=1}^{m} \lambda_{k_j} g_j(x) + \frac{r_k}{2} \sum_{j=1}^{m} g_j(x) +$$
$$+ \frac{1}{2r_k} \sum_{j=m+1}^{p} \left(\left[max(0, \mu_{k_j} + r_k g_j(x)) \right]^2 - (\mu_{k_j})^2 \right)$$

3. Найти точку $x^*(\lambda_k, \mu_k, r_k)$ безусловного минимума функции по с помощью какого-либо метода (нулевого, первого или второго порядка).

4. Вычислить $P(x^*(\lambda_k, \mu_k, r_k), \mu_k, r_k)$ где:

$$P(x^*(\lambda_k, \mu_k, r_k), \mu_k, r_k) = \frac{r_k}{2} \sum_{j=1}^{m} g_j(x)^2 +$$
$$+\frac{1}{2r_k} \sum_{j=m+1}^{p} \left(\left[max(0, \mu_{k_j} + r_k g_j(x))\right]^2 - (\mu_{k_j})^2 \right)$$

5. Проверка условий останова:

   (a) Если $P(x^*(\lambda_k, \mu_k, r_k), \mu_k, r_k) < \epsilon$ то конец, точка найдена;

   (b) Если $P(x^*(\lambda_k, \mu_k, r_k), \mu_k, r_k) > \epsilon$, положить $r_{k+1} = Cr_k, ,\quad \lambda_{k+1} = \lambda_k + r_k g(x^*(\lambda_k, \mu_k, r_k))$, $max(0, \mu_{j_k} + r_k g_j(x^*(\lambda_k, \mu_k, r_k))),\quad x_{k+1} = x^*(\lambda_k, \mu_k, r_k),\quad k = k+1$, перейти на шаг 2.

2.2 Результат работы программы

Реализация описанного алгоритма на языке C++ представлена в листинге 3. В качестве значений дополнительных параметров функции L, были взяты следующие значения (минимум достигается в точке (1,1,1)):

- $\epsilon = 10^{-6}$;
- $C = 4$;
- $r = 10^{-4}, \quad m_1 = m_2 = 1$.

Листинг 2: "Результат работы алгоритма для заданной функции f и граничными условиями."

```
points: 7
1.01848 1.03501 1.07146 0 0 1.024
1.01545 1.02828 1.05576 0 0 1.024
1.01442 1.02677 1.05167 0 0 1.024
1.0147 1.02664 1.05312 0 0 1.024
1.01432 1.02777 1.05423 0 0 1.024
1.01213 1.02232 1.04331 0 0 1.024
1.01423 1.02786 1.0549 0 0 1.024
minim: 0.00177785
```

2.3 Реализация алгоритма на языке C++

Листинг 3: "Реализация программы"

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

using namespace std;

double eps = 1e-6;

vector<vector<double> > pts;

// function example
double f1(vector<double>& xargs)
{
    double x1 = xargs[0], x2 = xargs[1], x3 = xargs[2];

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

// Boundaries
double g1(vector<double>& xargs)
{
    double x1 = xargs[0], x2 = xargs[1], x3 = xargs[2];

    return -(x1+1)*(x1+1);
}
double g2(vector<double>& xargs)
{
    double x1 = xargs[0], x2 = xargs[1], x3 = xargs[2];
    return x1 + x2 - 5;
}

// Lagrange function
double f(vector<double>& args)
{
    vector<double> xargs;

    // x arguments
    xargs.push_back(args[0]);
    xargs.push_back(args[1]);
    xargs.push_back(args[2]);

    double m1 = args[3], m2 = args[4];
    double r = args[5];

    return f1(xargs) + (1.0/(2*r))*(
            ( pow(max(0.0,(const double) m1 + r*g1(xargs)), 2) - pow(m1, 2) ) +
            ( pow(max(0.0,(const double) m2 + r*g2(xargs)), 2) - pow(m2, 2) ) );
}

double P(vector<double>& args)
{
     vector<double> xargs;

    // x arguments
    xargs.push_back(args[0]);
    xargs.push_back(args[1]);
    xargs.push_back(args[2]);

    double m1 = args[3], m2 = args[4];
```

```
    double r = args[5];

    return abs((1/(2*r))*(
            ( pow(max(0.0,(const double) m1 + r*g1(xargs)), 2) - pow(m1, 2) ) +
            ( pow(max(0.0,(const double) m2 + r*g2(xargs)), 2) - pow(m2, 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 = 6;
    double C = 4, r = 0.001, m1 = 0.3, m2 = 0.3;
    // выбираем n+1 точку
    for (int i = 0; i < n+1; i++)
    {
        vector<double> pt;
        for (int j = 0; j < n; j++)
            if (j == 5)
            {
                // r
                pt.push_back(r);
            }
            else if (j == 3)
            {
                // m
                pt.push_back(m1);
            }
            else if (j == 4)
            {
                // m2
                pt.push_back(m2);
            }
            else
            {
                pt.push_back(rand()%2);
            }

        pts.push_back(pt);
    }

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

    while (true)
    {
        // min algo
        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;
        cout << "P(pt): " << P(pts[0]) << endl;
        if (P(pts[0]) > eps)
        {
            cout << "optimize" << endl;
            r = C*r;
            m1 = max(0.0, m1 + r*g1(pts[0]));
            m2 = max(0.0, m2 + r*g2(pts[0]));
            for (int i = 0; i < pts.size(); i++)
            {
                pts[i][3] = m1;
                pts[i][4] = m2;
                pts[i][5] = r;
            }
        }
        else break;
    }
    return 0;
}
```