Методы оптимизации

Русначенко Николай

8 Января, 2016

1 Численное решение задач многокритериальной оптимизации

Даны дважды непрерывно дифференцируемые целевые функции, представленные в виде вектор-функции $F(x) = [f_1(x), f_2(x), \dots, f_l(x)]$ и функции ограничений $g_j(x) = 0, j = \overline{1, m}$ и $g_j(x) \leq 0, j = \overline{m+1, p}$, определяющие множество допустимых решений. Требуется найти недоминируемые (эффективные) решения на множестве X, т.е. такое множество решений $X_e \subset X$, что:

$$F(X_e) = \inf_{x \in X} [f_1(x), f_2(x), \dots, f_l(x)]$$

Где X удовлетворяет ораничениям $g_j(x) = 0, j = \overline{1, m}$ и $g_j(x) \leq 0, j = \overline{m+1, p}$. Рассматривается следующий вектор функции $F(x)$, а также ограничения:

$$\begin{cases} f_1(x) = 5(x_1^2 - x_2^2)^2 + 3(x_1 - x_2^2)^2 \\ f_2(x) = 4(x_1^2 - x_2)^2 + (3 - x_2^2) \\ g_1(x) = -(2x_1 + 1)^2 + 1 \leq 0 \\ g_2(x) = (x_1 + 2x_2)^2 - 16 \end{cases}$$

1.1 Решение задачи

Для решения поставленной задачи, необходимо рассмотреть свертку функций f_1, f_2, представляемую следующим образом:

$$F(x, r_k) = \sum_{j=1}^{l} w_j(f_j(x) - f_j^*) + P(x, r_k) \to \min_{x \in R^n}$$

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

Под f^* понимается вектор составленный из вычисленных минимальных значений функций. w – представляет собой вектор весовых коэффициентов. Этот вектор представляет собой собственный вектор матрицы A с максимальным собственным значением. Матрица A имеет следующий вид:

$$A = \begin{bmatrix} 1 & 1/a \\ a & 1 \end{bmatrix}$$

Для определения собственных значений матрицы А, необходимо вычислить $det(A - \lambda E)$. Собственные значения матрицы А:

$$\lambda_1 = 0$$

$$\lambda_2 = 2$$

Тогда для поиска вектора w, необходимо решить СЛАУ:

$$(A - \lambda E)\, x = 0$$

Решение будет иметь следующий вид (при условии, что рассматривается $\lambda = \lambda_2$:

$$w = \left[\frac{x_2}{a}, x_2 \in R \right]$$

Для поиска экстремума (минимума функции), будем искать минимум с помощью метода Нидлера-Мида с добавлением функций ограничений. Вектор f^* представляет собой минимум функции f_1^* и минимум функции f_2^* в точках, удовлетворяющих ограничениям g_1, g_2.

$$f^* = [f_1^*, f_2^*]$$

1.2 Результат работы программы

Прежде чем вычислить свертку функций f_1, f_2, необходимо определить минимальные значения (экстремумы) функций f_1, f_2:

- Для функции f_1, минимум достигается в точке `(1, -1)` или `(0, 0)`. $f_{1_{min}} = 3.4e^{-0.8} \approx 0$.
- Для функции f_2, минимум достигается в точке `(1.164, 1.417)`. $f_{1_{min}} = 1.0052$.

Исходя из полученных значений, вектор f^* будет иметь вид:

$$f^* = [0, 1.0052]$$

Значение a для матрицы A выбирается с равномерным распределением из интервала (0,1) при каждом запуске программы. Результаты работы алгоритма поиска минимума для свертки функций f_1, f_2 следующие:

Листинг 1: "Результат работы алгоритма для заданной функции f и граничными условиями."

```
a: 0.183853
% result simplex
1.02965 1.02731 -0.160674 0 0 0.256
1.02965 1.0269 -0.151224 0 0 0.256
1.03094 1.02822 -0.158597 0 0 0.256
1.03058 1.0278 -0.152598 0 0 0.256
1.02928 1.02673 -0.14623 0 0 0.256
1.03041 1.02788 -0.154032 0 0 0.256
1.02978 1.02729 -0.154396 0 0 0.256
minimum: 1.89642
-----------------------------------
a: 0.869594
1.07566 1.07806 -0.0756635 0 0 0.256
1.07887 1.0799 -0.0788747 0 0 0.256
1.07743 1.08003 -0.0774334 0 0 0.256
1.07738 1.07871 -0.0773835 0 0 0.256
1.07717 1.07865 -0.0771716 0 0 0.256
1.07817 1.08036 -0.078174 0 0 0.256
1.07679 1.07885 -0.0767944 0 0 0.256
minimum: 1.47005
-----------------------------------
a: 0.753412
0.00149757 0.00716231 -0.767758 0 0 1.024
0.0019838 0.0093893 -0.771089 0 0 1.024
0.00369031 0.00732168 -0.734181 0 0 1.024
0.0046733 0.00513931 -0.758748 0 0 1.024
0.00216721 0.00757979 -0.756801 0 0 1.024
0.00406691 0.00380676 -0.7353 0 0 1.024
0.00245188 0.00644172 -0.755824 0 0 1.024
minimum: 1.58226
-----------------------------------
a: 0.206507
0.0160532 -0.0810558 -3.29866 0 0 4.096
```

```
0.0150952  −0.0823098  −3.30678  0  0  4.096
0.0158779  −0.0814157  −3.30646  0  0  4.096
0.015441  −0.0818558  −3.30484  0  0  4.096
0.0147163  −0.0824471  −3.297  0  0  4.096
0.0144749  −0.0826463  −3.30265  0  0  4.096
0.0150046  −0.0820509  −3.29336  0  0  4.096
minimum:  1.33943
-----------------------------------
a:  0.388466
1.05049  1.04781  0.64242  0  0  0.256
1.04974  1.04699  0.646878  0  0  0.256
1.0509  1.0487  0.63395  0  0  0.256
1.05136  1.04867  0.647238  0  0  0.256
1.04932  1.04714  0.637952  0  0  0.256
1.0494  1.04707  0.645903  0  0  0.256
1.04991  1.0476  0.652672  0  0  0.256
minimum:  1.65171
```

Исходя из полученных результатов, можно сделать вывод что засчет изменения весовых коэффициентов для каждой функции, возможны несколько точке, в которых достигается минимум одновременно двух функций (при условии ограничений g_1, g_2):

$$x_1 = (0, 0)$$

$$x_2 = (1, 1)$$

Заметим, что x_1, x_2 являются минимумами функции f_1, в то время как только x_2 является минимумом для функции f_2.

1.3 Реализация программы на языке $C++$

Листинг 2: "Реализация программы."

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

#include <bits/stdc++.h>
#include "nedler_mead.h"
#include <stdio.h>
#include <time.h>

using namespace std;

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

    return 5*pow(pow(x1, 2) - pow(x2, 2), 2) + 3*pow(x1 - pow(x2, 2) , 2);
}

double f2(vector<double>& xargs)
{
    double x1 = xargs[0], x2 = xargs[1], x3 = xargs[2];

    return 4*pow(pow(x1, 2) - x2, 2) + (3 - pow(x2, 2));
}

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

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

    return pow(x1 + 2*x2, 2) - 16;
}

double lambda = 2;
double a;
double w2;

// 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];
    double w1 = lambda * (w2 / a);
    double fm1 = 0, fm2 = 1.0052;
```

```
    return w1*(f1(xargs) - fm1) + w2*(f2(xargs) - fm2) + (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) ) );
}

// Penality function
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) )));
}

vector<vector<double> > simplex;

int main()
{
    srand(time(NULL));
    int n = 6;
    double eps = 1e-08;
    double alpha = 1, beta = 0.5, gamma = 2;
    double C = 4, r = 0.001, m1 = 1, m2 = 1;

    a = (double) rand() / (RAND_MAX);
    w2 = lambda * ((double) rand() / (RAND_MAX));

    cout << "a: " << a << endl;

    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);

        simplex.push_back(pt);
    }

    double result = nedler_mead_f_g2(f, P, g1, g2, simplex,
            n, C, r, m1, m2, alpha, beta, gamma, eps);

    cout << "minimum: " << result << endl;

    return 0;
```

```
}
```

Листинг 3: "Заголовочный файл."

```
#ifndef _NEDLER_MEAD_H_
#define _NEDLER_MEAD_H_

#include <bits/stdc++.h>

using namespace std;

double nedler_mead_f_g2(double (*f)(vector<double>&),
        double (*P)(vector<double>&), double (*g1)(vector<double>&),
        double (*g2)(vector<double>&), vector<vector<double> > simplex,
        double n, double C, double r, double m1, double m2, double alpha,
        double beta, double gamma, double eps);

#endif
```

Листинг 4: "Реализация алгоритма минимизации."

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

using namespace std;

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

        result->push_back(m);
    }
}

void global_press(vector<vector<double> >& simplex, int length, int min_index)
{
    for (int j = 0; j < length; j++)
        for(int i = 0; i < simplex.size(); i++)
            if (min_index != i)
                simplex[i][j] = simplex[min_index][j] +(simplex[i][j] -
                    simplex[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(double (*f)(vector<double>&), vector<vector<double> > &simplex,
```

```
        vector<double>& xc, double beta)
{
    vector<double>* xh = &simplex[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(simplex, xs.size(), simplex.size()-1);
}

double mx(double (*f)(vector<double>&), vector<vector<double> > simplex)
{
    double m = 0;

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

    return m;
}

double (*_f)(vector<double>&);

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

double nedler_mead_f_g2(double (*f)(vector<double>&),
        double (*P)(vector<double>&), double (*g1)(vector<double>&),
        double (*g2)(vector<double>&), vector<vector<double> > simplex,
        double n, double C, double r, double m1, double m2, double alpha,
        double beta, double gamma, double eps)
{
    _f = f;
    while (true)
    {
        // min algo
        while (true)
        {
            // 2
            sort(simplex.begin(), simplex.end(), cmp);
            vector<double>* xh = &simplex[0];
            vector<double>* xg = &simplex[1];
            vector<double>* xl = &simplex[simplex.size()-1];

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

            // 3
            vector<double> xc;
            xc_calc(simplex, 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))
                    simplex[0] = xe;
                else
                    simplex[0] = xr;
            }
            else if (f(*xl) < f(xr) && f(xr) < f(*xg))
                    simplex[0] = xr;
            else if (f(*xg) < f(xr) && f(xr) < f(*xh))
            {
                vector<double> *c = &xr;
                xr = *xh; *xh = *c;
                press(f, simplex, xc, beta);
            }
            else if (f(*xh) < f(xr))
                press(f, simplex, xc, beta);

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

            if (diff < eps)
                break;
        }

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

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