# Nelder-Mead 알고리즘을 이용한 데이터의 Curve-Fitting

Yongbok Kee

2017.03.04

## 제 1 절 기본이론

### 1.1 Curve-Fitting

자료가 $\mathbf{x} = [x_1, x_2, \cdots, x_N]$, $\mathbf{y} = [y_1, y_2, \cdots, y_N]$과 같이 주어졌을 때, 자료들을 함수 $y = f(x)$의 꼴로 표현되는 어떤 적합한 곡선으로 맞춰야 할 필요가 있다. 이를 위해서, 식 (1)로 주어지는 데이터 $\mathbf{y}$와 함수 $f(x_i, \mathbf{c})$의 차이의 제곱의 합을 최소화하는 $\mathbf{c}$를 구해야 한다.

$$F(\mathbf{c}) = \sum_{i=1}^{N} [y_i - f(x_i, \mathbf{c})]^2 \tag{1}$$

다시 말해서, 곡선적합 문제는 최적화 문제로 볼 수 있으며, 최적화 문제를 풀기 위해 고안된 여러가지 알고리즘을 사용하여 해결할 수 있다. 본 보고서에서는, 도함수를 계산할 필요가 없이 최적해를 찾는 기법인 Nelder-Mead 알고리즘을 사용하겠다.

### 1.2 Nelder-Mead 알고리즘

Nelder-Mead 알고리즘(혹은 아메바 알고리즘)은 $n$변수 함수 $y = f(\mathbf{x}) = f(x_1, x_2, \cdots, x_n)$의 최적화 알고리즘의 하나로, $n+1$개의 꼭지점에서 $n$차원의 심플렉스(simplex)를 움직이면서 함수의 최소값을 탐색한다. 이 과정은 반사, 팽창, 수축의 3종류를 달리하면서 수행하게 된다. 이 알고리즘을 이용한 최적화의 예시를 그림 (1)로 나타내었다.

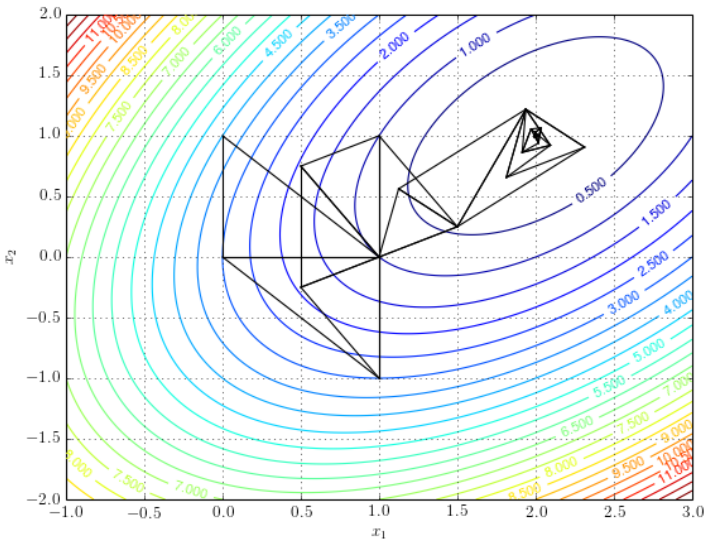


그림 1: Nelder-Mead 알고리즘을 이용한 최적화의 예시

위 알고리즘을 아래와 같은 의사코드로 나타낼 수 있다.

**Algorithm 1** Nelder-Mead 알고리즘

**procedure** NELDERMEAD ▷ 함수 $y = f(\mathbf{x}) = f(x_1, x_2, \cdots, x_n)$의 최적해 탐색
  $\mathbf{x}$ 입력 ($\mathbf{x} \in \mathbb{R}^n$)
  $\lambda$ 입력 ▷ simplex의 크기와 관련있음
  **for** $i = 1$ to $n$ **do**
    $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda \mathbf{e}_i$ ▷ $\mathbf{e}_i$: 단위 벡터, $\mathbf{e}_i \in \mathbb{R}^n$
  **end for**

  $R = 1$, $K = 0.5$, $E = 2$, $S = 0.5$ ▷ Nelder-Mead 알고리즘의 Parameter

  **for** $i = 1$ to $(n + 1)$ **do**
    $f_i = f(\mathbf{x}_i)$
  **end for**

  $f_1 \leq f_2 \leq \cdots \leq f_{n+1}$이 되도록 정렬
  $N_{\text{itr}}$과 $\epsilon_{\text{tol}}$ 입력
  **for** $k = 1$ to $k = N_{\text{itr}}$ **do**
    $\mathbf{x}_m = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_n$ ▷ $\mathbf{x}_{n+1}$을 제외한 모든 점의 centroid 계산
    $\mathbf{x}_r = \mathbf{x}_m + R(\mathbf{x}_m - \mathbf{x}_{n+1})$
    **if** $f_1 \leq f_r < f_n$ **then** ▷ 반사
      $\mathbf{x}_{n+1} = \mathbf{x}_r$
    **else if** $f_r < f_1$ **then** ▷ 팽창
      $\mathbf{x}_e = \mathbf{x}_m + E(\mathbf{x}_r - \mathbf{x}_m)$
      $f_e = f(\mathbf{x}_e)$
      **if** $f_e < f_r$ **then**
        $\mathbf{x}_{n+1} = \mathbf{x}_e$
      **else**
        $\mathbf{x}_{n+1} = \mathbf{x}_r$
      **end if**
    **else** ▷ 수축
      **if** $f_r < f_{n+1}$ **then**
        $\mathbf{x}_{oc} = \mathbf{x}_m + K(\mathbf{x}_r - \mathbf{x}_m)$
        $f_{oc} = f(\mathbf{x}_{oc})$
        **if** $f_{oc} < f_r$ **then**
          $\mathbf{x}_{n+1} = \mathbf{x}_{oc}$
        **else**
          **for** $i = 2$ to $(n + 1)$ **do**
            $\mathbf{x}_i = \mathbf{x}_i + S(\mathbf{x}_i - \mathbf{x}_1)$
          **end for**
        **end if**

**Algorithm 1** Nelder-Mead 알고리즘 (계속)

```
            else
                x_ic = x_m + K(x_m − x_{n+1})
                f_ic = f(x_ic)
                if f_ic < f_{n+1} then
                    x_{n+1} = x_ic
                else
                    for i = 2 to (n + 1) do
                        x_i = x_i + S(x_i − x_1)
                    end for
                end if
            end if
        end if
        f_1 ≤ f_2 ≤ ··· ≤ f_{n+1}이 되도록 정렬
        r = |x_1 − x_{n+1}|                                  ▷ 잔차 계산
        if r ≤ ε_tol then
            즉시 for 루프를 빠져나감
        end if
    end for
end procedure
```

## 제 2 절 예제

이제, Nelder-Mead 알고리즘을 사용하여 주어진 자료를 Curve-Fitting해보자. 예를 들어, 온도에 따른 물의 증기압 데이터가 표 (1)과 같이 주어졌을 때를 살펴보자.

| $T$ [℃] | $P_{sat}$ [kPa] |
|---|---|
| 0 | 0.6113 |
| 5 | 0.8726 |
| 10 | 1.2281 |
| 15 | 1.7056 |
| 20 | 2.3388 |
| 25 | 3.169 |
| 30 | 4.2455 |
| 35 | 5.6267 |
| 40 | 7.3814 |
| 50 | 12.344 |
| 60 | 19.932 |
| 90 | 70.117 |
| 95 | 84.529 |
| 100 | 101.32 |

표 1: 온도에 따른 물의 증기압

온도에 따른 증기압의 관계를 설명하는 가장 대표적인 수학적인 모델은 식 (2)로 주어지는 Antoine 방정식이다.

$$P_{sat}(T) = 10^{A-\frac{B}{T+C}} \tag{2}$$

Nelder-Mead 알고리즘 수행시, $\lambda = 5$로 놓고, 최대반복횟수는 1000000회, 허용오차는 $10^{-8}$로 설정하였다. 이때, 초기 시작지점은 $A_0 = 1$, $B_0 = 1$, $C_0 = 1$로 설정하였다.

## 제 3 절 결과 및 평가

위 알고리즘을 C++로 구현하여 실행시켜서 1484회의 반복 끝에 아래와 같이 Antoine 계수를 얻었다.

$$A = 7.128198489,\ B = 1691.563241,\ C = 230.2243248 \tag{3}$$

표 (1)의 데이터와 데이터에 적합된 Antoine 방정식을 비교하기 위해 아래와 같이 그래프를 그려서 표시하였다. [그림 (2)]

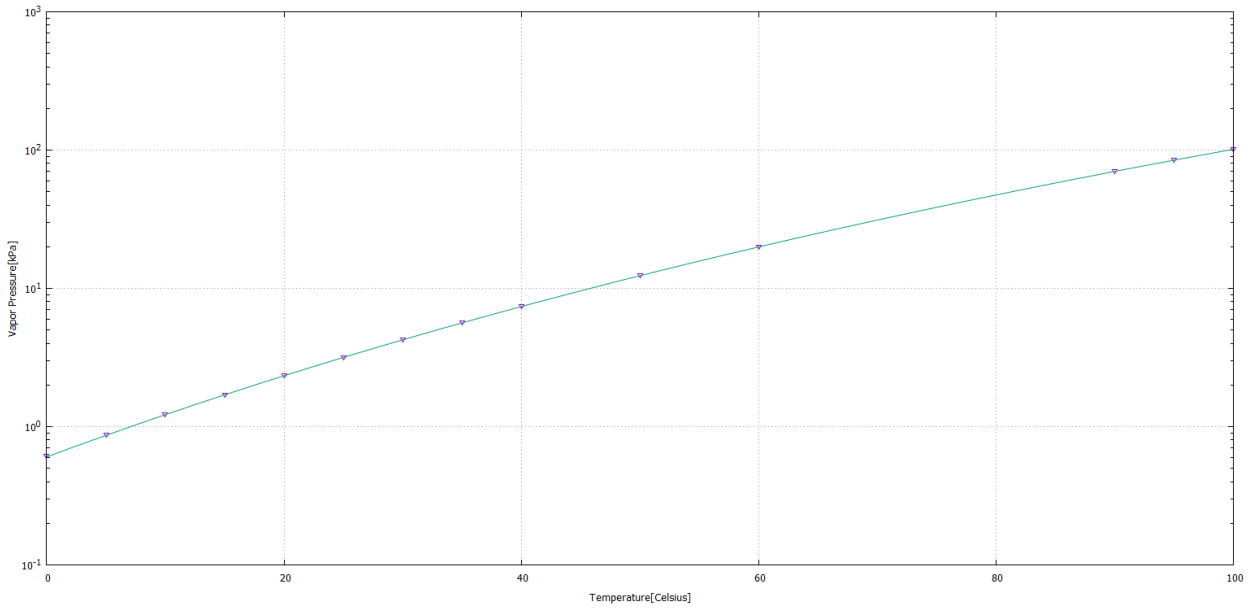


그림 2: curveFitting

반복계산을 수행하는 동안, 수렴하기까지의 Antoine 계수 $A$, $B$, $C$의 값과 잔차의 그래프를 그려서 그림 (3), (4)과 같이 나타내었다.

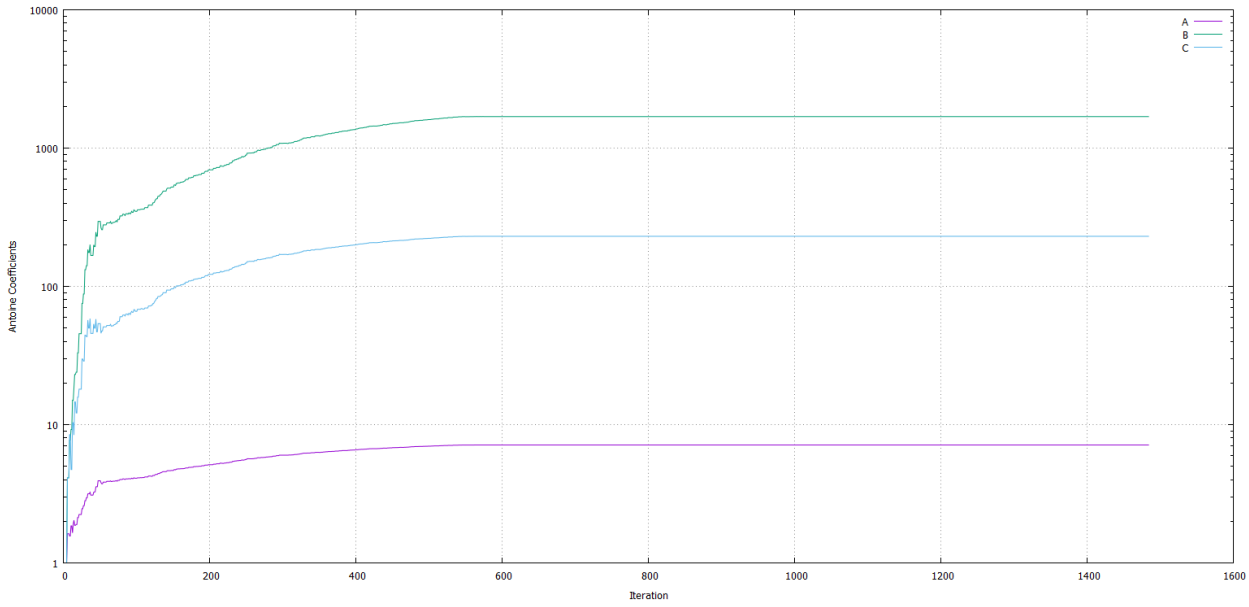


그림 3: Antoine 계수

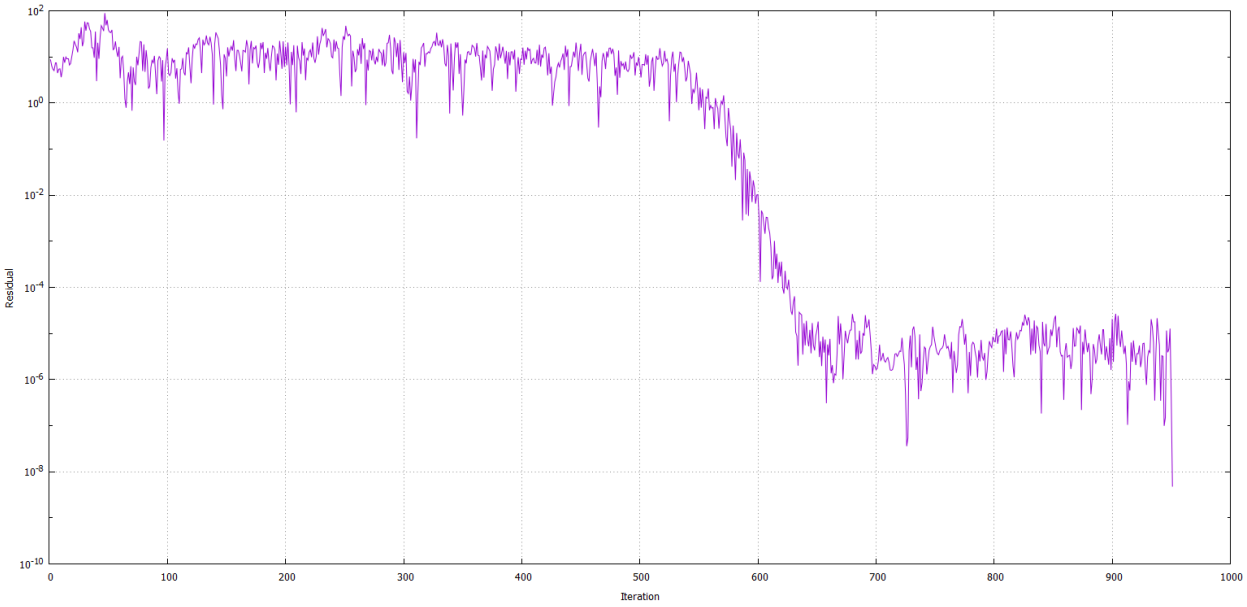


그림 4: 잔차

주어진 데이터가 Antoine 방정식에 잘 적합하는지 평가하기 위해서, 식 (4)으로 주어지는 결정계수 $R^2$를 계산하면 된다.

$$R^2 = 1 - \frac{\mathrm{SSRes}}{\mathrm{SSTot}} = 1 - \frac{\sum_{i=1}^{N_{\mathrm{data}}} \left[y_i - f(x_i, \mathbf{c})\right]^2}{\sum_{i=1}^{N_{\mathrm{data}}} \left(y_i - \bar{y}\right)^2} \tag{4}$$

일반적으로, 결정계수는 1에 가까울수록, 데이터가 곡선에 잘 적합하다고 판단할 수 있다.
본 보고서에서는 $R^2 = 0.9999999628$을 얻었는데, 1에 매우 가까운 수치이다. 따라서, 표 (1)의 증기압 데이터는 Antoine Equation에 잘 적합하다고 판단할 수 있다.

# 제 4 절 소스코드

## 4.1 sort.cpp

```
1         void sort(double* a, double* b, double* c, double* F, int n){
2                 for(int i=0; i<n-1; i++){
3                         for(int j=i+1;j<n;j++){
4                                 if(F[i]>F[j]){
5                                         double tmpF = F[i];
6                                         F[i]=F[j];
7                                         F[j]=tmpF;
8                                         double tmpA = a[i];
9                                         a[i]=a[j];
10                                        a[j]=tmpA;
11                                        double tmpB = b[i];
12                                        b[i]=b[j];
13                                        b[j]=tmpB;
14                                        double tmpC = c[i];
15                                        c[i]=c[j];
16                                        c[j]=tmpC;
17                                }
18                        }
19                }
20        }
```

## 4.2 FitRoutine.cpp

```
1         #include <cmath>
2
3         double FitFunc(double a, double b, double c, double x){
4                 return pow(10.0,(a-b/(x+c)));
5         }
6
7         double FitDev(double a, double b, double c, double* x, double* y, int n){
8                 double sum = 0;
9                 for(int i=0; i<n; i++){
10                        sum += (y[i]-FitFunc(a,b,c,x[i]))*(y[i]-FitFunc(a,b,c,x[i]));
11                }
12                return sum;
13        }
```

## 4.3 fit.cpp

```
1         #include <iostream>
2         #include <fstream>
3         #include <cmath>
4         #include "FitRoutine.h"
5         #include "sort.h"
```

```
6
7           using namespace std;
8
9           int main(){
10                  //data input ("*.csv")
11                  ifstream is;
12                  is.open("samp.csv");
13                  if(is.bad()){
14                          return -1;
15                  }
16                  double X,Y;
17                  int N = 0;
18                  is.ignore(256,'\n');
19                  while(!(is.fail())){
20                          is>>X>>Y;
21                          N++;
22                  }
23                  is.close(); is.clear();
24                  int n = N - 1;
25                  double* x = new double[n];
26                  double* y = new double[n];
27                  is.open("samp.csv");
28                  is.ignore(256,'\n');
29                  for(int j=0; j<n; j++){
30                          is>>x[j]>>y[j];
31                  }
32                  is.close(); is.clear();
33
34                  double a0,b0,c0;
35                  cout<<"Fit is designed to fit data into the function\n";
36                  cout<<"y=10**(a-b/(x+c))\n";
37                  cout<<"Enter the initial value of a,b,c\n";
38                  cin>>a0>>b0>>c0;
39
40                  //coefficients for Nelder-Mead algorithm
41                  double R=1; double K=0.5; double E=2; double S=0.5;
42
43                  double* a = new double[4];
44                  double* b = new double[4];
45                  double* c = new double[4];
46                  double* F = new double[4];
47
48                  double lambda;
49                  cout<<"Enter the lambda\n";
50                  cin>>lambda;
51
```

```
52                  double ao,bo,co;
53                  double ar,br,cr,Fr;
54                  double ae,be,ce,Fe;
55                  double aoc,boc,coc,Foc;
56                  double aic,bic,cic,Fic;
57
58                  //generating ititial simplex
59                  a[0]=a0; b[0]=b0; c[0]=c0;
60                  a[1]=a0+lambda; b[1] = b0; c[1]=c0;
61                  a[2]=a0; b[2]=b0+lambda;c[2]=c0;
62                  a[3]=a0;b[3]=b[0];c[3]=c0+lambda;
63
64                  for(int i=0; i<4;i++){
65                          F[i]=FitDev(a[i],b[i],c[i],x,y,n);
66                  }
67
68                  sort(a,b,c,F,4);
69
70                  int itrMax;
71                  double tol;
72                  cout<<"Enter the maximum iteration number\n"; cin>>itrMax;
73                  cout<<"Enter the tolerance error\n"; cin>>tol;
74                  double resid;
75                  cout.precision(10);
76                  ofstream out;
77                  out.open("log.txt");
78                  out<<"#itr"<<'\t'<<'a'<<'\t'<<'b'<<'\t'<<'c'<<'\t'<<"resid"<<endl;
79                  for(int itr=1; itr<=itrMax; itr++){
80                          ao = (1./3)*(a[0]+a[1]+a[2]);
81                          bo = (1./3)*(b[0]+b[1]+b[2]);
82                          co = (1./3)*(c[0]+c[1]+c[2]);
83
84                          ar = ao + R*(ao-a[3]);
85                          br = bo + R*(bo-b[3]);
86                          cr = co + R*(co-c[3]);
87                          Fr = FitDev(ar,br,cr,x,y,n);
88
89                          //reflection
90                          if(F[0]<=Fr && Fr<F[2]){
91                                  a[3] = ar;
92                                  b[3] = br;
93                                  c[3] = cr;
94                          }
95
96                          //expansion
97                          else if(Fr<F[0]){
```

```
98                      ae = ao + E*(ar-ao);
99                      be = bo + E*(br-bo);
100                     ce = co + E*(cr-co);
101                     Fe = FitDev(ae,be,ce,x,y,n);
102                     if(Fe<Fr){
103                             a[3]=ae;
104                             b[3]=be;
105                             c[3]=ce;
106                     }
107                     else
108                     {
109                             a[3]=ar;
110                             b[3]=br;
111                             c[3]=cr;
112                     }
113             }
114
115             //shrinkage
116             else{
117                     if(Fr<F[3]){
118                             aoc = ao + K*(ar-ao);
119                             boc = bo + K*(br-bo);
120                             coc = co + K*(cr-co);
121                             Foc = FitDev(aoc,boc,coc,x,y,n);
122                             if(Foc<Fr){
123                                     a[3]=aoc; b[3]=boc; c[3]=coc;
124                             }
125                             else{
126                                     for(int i=1; i<4; i++){
127                                             a[i] = a[i] + S*(a[i]-a[0]);
128                                             b[i] = b[i] + S*(b[i]-b[0]);
129                                             c[i] = c[i] + S*(c[i]-c[0]);
130                                     }
131                             }
132                     }
133                     else{
134                             aic = ao - K*(ao-a[3]);
135                             bic = bo - K*(bo-b[3]);
136                             cic = co - K*(co-c[3]);
137                             Fic = FitDev(aic,bic,cic,x,y,n);
138                             if(Fic<Fr){
139                                     a[3]=aic; b[3]=bic; c[3]=cic;
140                             }
141                             else{
142                                     for(int i=1; i<4; i++){
143                                             a[i] = a[i] + S*(a[i]-a[0]);
```

```
                                                b[i] = b[i] + S*(b[i]-b[0]);
                                                c[i] = c[i] + S*(c[i]-c[0]);
                                        }
                                }
                        }
                }

                for(int i=0; i<4; i++){
                        F[i]=FitDev(a[i],b[i],c[i],x,y,n);
                }

                sort(a,b,c,F,4);

                //calculating residual
                resid =
                        (a[0]-a[3])*(a[0]-a[3])+(b[0]-b[3])*(b[0]-b[3])
                        +(c[0]-c[3])*(c[0]-c[3]);
                resid = sqrt(resid);

                cout<<"itr: "<<itr<<", a="<<a[0]<<", b="<<b[0]<<", c="<<c[0]<<
                        ", resid="<<resid<<endl;
                out<<itr<<'\t'<<a[0]<<'\t'<<b[0]<<'\t'<<c[0]<<'\t'<<resid<<endl;

                if(resid<=tol){
                        cout<<"Convergence succeeded\n";
                        break;
                }
        }
        out.close();

        //calculating average of data y
        double mean = 0;
        for(int i=0; i<n; i++){
                mean += y[i];
        }
        mean /= n;

        //calculating  total sum of squares, SSTot
        double SSTot = 0;
        for(int i=0; i<n; i++){
                SSTot += (y[i]-mean)*(y[i]-mean);
        }

        //calculating sum of squares of residuals, SSRes
        double SSRes = FitDev(a[0],b[0],c[0],x,y,n);

```

```
190                 //calculating coefficient of determination, R2
191                 double R2 = 1- (SSRes/SSTot);
192                 cout<<"R2="<<R2<<endl;
193 
194                 delete[] a;
195                 delete[] b;
196                 delete[] c;
197                 delete[] F;
198 
199                 return 0;
200         }
```