科学计算和C程序集

蒋长锦　编著

中国科学技术大学出版社

```
    int i,iter,maxiter=200;
    static double delta,x[N]={ -2.0, 0.0, 1.0 };      /*   x[] -> x0[]   */
    delta=0.01;
    system("cls");
    printf("B-F-S method for non-linear equations:\n");
    iter=bfsr2( maxiter, delta, x );
    printf(" Iterative %d times!\n",iter);
    for(i=0;i<N;i++) printf("   %12.8f\n",x[i]);
    if( iter >= maxiter ) printf("It isn't successful!\n");
    getch();
}
/* * * * function f( x[N], f[N] ) * * * * */
void ff( double x[N], double f[N] )
{
    f[0]=x[0]*x[0]+x[1]*x[1]-x[2]-2.0;
    f[1]=x[0]+5.0*x[1]+1.0;
    f[2]=x[0]*x[2]-2.0*x[0]+1.0;
}
```

计算结果如下：

```
B-F-S method for non-linear equations:
Iterative 7 times!
  -2.10393732
   0.22078746
   2.47529933
```

9.6 最速下降法和共轭斜量法

对于非线性方程组

$$\boldsymbol{f}(\boldsymbol{x}) = 0 \tag{9.6.1}$$

这里 $\boldsymbol{f}(\boldsymbol{x})=(f_1(\boldsymbol{x}),f_2(\boldsymbol{x}),\cdots,f_n(\boldsymbol{x}))^T$，构造目标函数

$$F(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{f}^T(\boldsymbol{x})\boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{n} f_i^2(\boldsymbol{x}) \tag{9.6.2}$$

若非线性方程组(9.6.1)有解 $\boldsymbol{x}^*$，则显然成立

$$F(\boldsymbol{x}^*) = \frac{1}{2}\boldsymbol{f}^T(\boldsymbol{x}^*) * \boldsymbol{f}(\boldsymbol{x}^*) = \frac{1}{2}\sum_{i=1}^{n} f_i^2(\boldsymbol{x}^*) = 0$$

即 $\boldsymbol{x}^*$ 是目标函数的极小值点，且极小值为0。这样我们就将非线性方程组(9.6.1)的求解问题化为求目标函数(9.6.2)的极小点问题。

本节仅对由非线性方程(9.6.1)构造的目标函数(9.6.2)建立最速下降法和共轭斜量法的算法和程序，而对其收敛性不作详细介绍。目的仅在于求解非线性方程组。

一、最速下降法

最速下降法又称为梯度法，是最古老而又十分基本的数值方法。它具有算法简单，使用方便的优点，在最优化问题以及非线性方程组的求解等方面一直发挥着重要的作用。

1. 算法描述

对于目标函数(9.6.2)有

$$\frac{\partial F(x)}{\partial x_i}=\sum_{j=1}^{m} f_j(x)\frac{\partial f_j(x)}{\partial x_i},i=1,2,\cdots,n$$

所以

$$g(x)=grad(F(x))=\nabla F(x)=\left(\frac{\partial F(x)}{\partial x_1},\frac{\partial F(x)}{\partial x_2},\cdots,\frac{\partial F(x)}{\partial x_n}\right)^T$$

$$=\begin{pmatrix}\frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_1}\\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_2}\\ \cdots & \cdots & \cdots & \cdots\\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_n}{\partial x_n}\end{pmatrix}_{n}\begin{pmatrix}f_1\\ f_2\\ \vdots\\ f_n\end{pmatrix}=Df(x)^Tf(x) \tag{9.6.3}$$

梯度方向 $g(x)$是目标函数 $F(x)$在 x 点上升最快的方向，而反方向$-g(x)$就是 $F(x)$在该点下降最快的方向。最速下降法就是沿着梯度的负方向来逐步下降 $F(x)$的值。具体思想分析如下：

设 $x^{(0)}$是解的一个初始近似，计算 $F(x)$在 $x^{(0)}$的梯度

$$g_0=g(x^{(0)})=Df(x^{(0)})^Tf(x^{(0)}) \tag{9.6.4}$$

从 $x^{(0)}$出发沿负梯度方向$-g_0$跨出一个适当的步长 α 得新点

$$x^{(1)}=x^{(0)}-\alpha g_0$$

希望 $F(x^{(0)}-\alpha g_0)$在$-g_0$方向上取相对极小值即

$$F(x^{(1)})=\min_{\alpha}F(x^{(0)}-\alpha g_0) \tag{9.5.5}$$

实际计算时我们只要求得到相对极小值点的一个近似值 $x^{(1)}$，故可在 $\alpha=0$处对 $F(x^{(0)}-\alpha g_0)$作 Taylor 展开，并略去 α 的高阶项。如此得

$$F(x^{(0)}-\alpha g_0)=\frac{1}{2}f(x^{(0)}-\alpha g_0)^Tf(x^{(0)}-\alpha g_0)$$

$$\approx\frac{1}{2}(f(x^{(0)})-\alpha Df(x^{(0)})g_0)^T(f(x^{(0)})-\alpha Df(x^{(0)})g_0)$$

$$=\frac{1}{2}f(x^{(0)})^Tf(x^{(0)})-\alpha f^T(x^{(0)})Df(x^{(0)})g_0+\frac{1}{2}\alpha^2(Df(x^{(0)})g_0)^T(Df(x^{(0)})g_0)$$

简记 $f(x^{(0)})$为 f_0，$Df(x^{(0)})$为 Df_0，$a^Tb=(a,b)$则有

$$F(x^{(0)}-\alpha g_0)\approx\frac{1}{2}(f_0,f_0)-\alpha(Df_0g_0,f_0)+\frac{1}{2}\alpha^2(Df_0g_0,Df_0g_0)$$

根据一元函数求极小值点的方法，应解方程

$$\frac{\partial F}{\partial \alpha} = -(Df_0 g_0, f_0) + \alpha(Df_0 g_0, Df_0 g_0) = 0$$

得

$$\alpha_0 = \alpha = \frac{(Df_0 g_0, f_0)}{(Df_0 g_0, Df_0 g_0)} = \frac{(g_0, g_0)}{(Df_0 g_0, Df_0 g_0)} \tag{9.6.6}$$

如果对求得的 α_0 成立

$$F(x^{(0)} - \alpha_0 g_0) < F(x^{(0)}) \tag{9.6.7}$$

则就以它作为 α_0，否则将 α_0 缩成其一半，再检查(9.6.7)，直到(9.6.7)满足为止。

根据以上分析现将最速下降法求解非线性方程组(9.6.1)的具体步骤归结如下：

给定精度要求 ε，取解的初始近似 $x^{(0)}$。假定由最速下降法已求得第 k 次近似 $x^{(k)}$，$k=0$，$1,\cdots$，以下求 $x^{(k+1)}$

(1)计算 $f_k = f(x^{(k)})$，$Df_k = Df(x^{(k)})$

(2)计算 $g_k = Df_k^T f_k$

若 $\|g_k\| < \varepsilon$，表示 $x^{(k)}$ 以满足精度要求，停止计算，否则

(3)计算 $Df_k g_k$，并求

$$\alpha_k = \frac{(g_k, g_k)}{(Df_k g_k, Df_k g_k)}$$

(4)计算 $x^{(k+1)} = x^{(k)} - \alpha_k g_k$

(5)计算 $f(x^{(k+1)})$

若 $\|f(x^{(k+1)}\|^2 < \|f(x^{(k)})\|^2$ (9.6.8)

表示 $x^{(k+1)}$ 已求得，返回(1)。

若(9.6.8)不满足，令 $\alpha_k := \frac{\alpha_k}{2}$，返回(4)重新计算 $x^{(k+1)}$。

一般来说最速下降法对初值 $x^{(0)} \in D$，这里 $f: D \subset R^n \to R^n$，精度要求不高，但其收敛速度是线性的。在完成几步计算后，其收敛速度就变得十分缓慢，特别是在解 x^* 附近。因此在实际使用中常与 Newton 迭代法相结合，使这两个方法相互取长补短以达到既能保证大范围的收敛性，又能加快收敛速度的目的。

2. 最速下降法程序

(1)子函数原型

```
void ff(double x[N], double f[N]);
void fd(double x[N], double Df[N][N]);
int steepdn(int maxiter, double x[N]);
```

关于子函数 ff()和 fd()参见 Newton 法迭代程序说明。

子函数 steepdn()用最速下降法求解非线性方程组。该子函数返回实际迭代次数。若迭代 maxiter 次仍不收敛，则停止迭代，返回主函数。

(2)形参和变量说明

符号常量 N，整型参数 maxiter，形参数 x[N]，fk[N]，Df[N][N]的说明参见 Newton 法迭代程序的相关说明。

变量 g[N]，dfg[N]分别对应 g_k，$Df_k g_k$。

变量 alpha 对应 α_k。gtg，dfgtdfg 分别对应$(\boldsymbol{g}_k,\boldsymbol{g}_k)$和$(\boldsymbol{Df}_k\boldsymbol{g}_k,\boldsymbol{Df}_k\boldsymbol{g}_k)$，即 $\boldsymbol{g}_k^T\boldsymbol{g}_k$ 和$(\boldsymbol{Df}_k\boldsymbol{g}_k)^T(\boldsymbol{Df}_k\boldsymbol{g}_k)$。normf0，normf1分别对应$\|\boldsymbol{f}(\boldsymbol{x}^{(k)})\|^2$和$\|\boldsymbol{f}(\boldsymbol{x}^{(k)}-\alpha_k\boldsymbol{g}_k)\|^2$。

(3)子函数程序

```
/*   steepdn.c: Steepest descent method for non-linear equations.   */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define DIM N
void ff( double x[N], double f[N] );
void fd( double x[N], double Df[N][N] );
int steepdn( int maxiter, double x[N] )
{
  int i,j,k;
  double alpha,gtg,dfgtdfg,normf0,normf1;
  static double fk[N],Df[N][N],xx[N],g[N],dfg[N];
  k=0;
  ff( x, fk );
  normf0=0.0;
  for(i=0;i<N;i++) normf0=normf0+fk[i]*fk[i];
  again: k=k+1;
  ff( x, fk );
  fd( x, Df );
  gtg=0.0;
  for(i=0;i<N;i++)
  {
    g[i]=0.0;
    for(j=0;j<N;j++) g[i]=g[i]+Df[j][i]*fk[j];
    g[i]=2.0*g[i];
    gtg=gtg+g[i]*g[i];
  }
  dfgtdfg=0.0;
  for(i=0;i<N;i++)
  {
    dfg[i]=0.0;
    for(j=0;j<N;j++) dfg[i]=dfg[i]+Df[i][j]*g[j];
    dfgtdfg=dfgtdfg+dfg[i]*dfg[i];
  }
  alpha=0.5*gtg/dfgtdfg;
```

```
    do
    {
      for(i=0;i<N;i++) xx[i]=x[i]-alpha * g[i];
      ff( xx, fk );
      normf1=0.0;
      for(i=0;i<N;i++) normf1=normf1+fk[i] * fk[i];
      alpha=0.5 * alpha;
    }
    while( normf1 > normf0 );
    normf0=normf1;
    for(i=0;i<N;i++) x[i]=xx[i];
    normf1=sqrt(normf1);
    if( normf1 < EPSILON ) goto endd;
    if( k > maxiter ) goto endd;
    goto again;
    endd:return k;
}
```

(4)计算实例和主函数

例9.9 用最速下降法求解非线性方程组

$$\begin{cases} x^2+y^2+z^2=1 \\ 2x^2+y^2-4z=0 \\ 3x^2-4y+z^2=0 \end{cases}$$

取初始近似 $(x^{(0)},y^{(0)},z^{(0)})^T=(1,1,1)^T$。

解 该方程组的 Jacobi 矩阵为

$$Df=\begin{pmatrix} 2x & 2y & 2z \\ 4x & 2y & -4 \\ 6x & -4 & 2z \end{pmatrix}$$

这时 N=3,取 maxiter=1000,$\varepsilon=10^{-8}$。主函数程序如下:

```
/*   steepdnm.c:Steepest descent method for non-linear equations.   */
#define N 3
#define EPSILON 1.0e-8
#include "steepdn.c"
void ff( double x[N], double f[N] );
void fd( double x[N], double Df[N][N] );
extern int steepdn( int maxiter, double x[N] );
void main( )
{
  int i,iter,maxiter=1000;
```

```
    static double x[N]={ 1.0, 1.0, 1.0 };
    system("cls");
    printf("Steepest descent method for non-linear equations:\n");
    iter=steepdn( maxiter, x );
    printf(" Iterative %d times!\n",iter);
    for(i=0;i<N;i++) printf("   %12.8f\n",x[i]);
    if( iter > maxiter ) printf("It isn't successful!\n");
    getch();
}
/* * * * function f( x[N], f[N] ) * * * * /
void ff( double x[N], double f[N] )
{
    f[0]=x[0]*x[0]+x[1]*x[1]+x[2]*x[2]-1.0;
    f[1]=2.0*x[0]*x[0]+x[1]*x[1]-4.0*x[2];
    f[2]=3.0*x[0]*x[0]-4.0*x[1]+x[2]*x[2];
}
/* * * * function fd( x[N], Df[N][N] ) * * * * /
void fd( double x[N], double Df[N][N] )
{
    Df[0][0]= 2.0*x[0];
    Df[0][1]= 2.0*x[1];
    Df[0][2]= 2.0*x[2];
    Df[1][0]= 4.0*x[0];
    Df[1][1]= 2.0*x[1];
    Df[1][2]= -4.0;
    Df[2][0]= 6.0*x[0];
    Df[2][1]=-4.0;
    Df[2][2]= 2.0*x[2];
}
```

计算结果如下：

```
Steepest descent method for non-linear equations:
Iterative 70 times!
     0.78519694
     0.49661140
     0.36992283
```