§6. 解线性方程组的迭代法

6.1 常用迭代法

将线性方程组 $Ax=b$ 等价变形为 $x=Bx+f$ ，建立迭代格式

$$x^{(k+1)}=Bx^{(k)}+f\,(k=0,1,\cdots)$$

其中 B 为迭代矩阵， f 是与 A 和 b 有关的向量。对方程组做不同的等价变形，将会得到不同的迭代格式，区别仅在于迭代矩阵 B 和向量 f 。设 $A=(a_{ij})_{n\times n}$ 可分裂为

$$A=\begin{bmatrix} a_{11} & & & \\ & a_{22} & & \\ & & \ddots & \\ & & & a_{nn}\end{bmatrix}-\begin{bmatrix} 0 & & & & \\ -a_{21} & 0 & & & \\ \vdots & \vdots & \ddots & & \\ -a_{n-1,1} & -a_{n-1,2} & \cdots & 0 & \\ -a_{n1} & -a_{n2} & \cdots & -a_{n,n-1} & 0\end{bmatrix}-\begin{bmatrix} 0 & -a_{12} & \cdots & -a_{1,n-1} & -a_{1n} \\ & 0 & \cdots & -a_{2,n-1} & -a_{2n} \\ & & \ddots & \vdots & \vdots \\ & & & 0 & -a_{n-1,n} \\ & & & & 0\end{bmatrix}=D-L-U$$

常见的迭代格式有以下几种。

Jacobi 迭代：设 $A=(a_{ij})_{n\times n}$ 的主对角元素 $a_{ii}\neq 0(i=1,2,\cdots,n)$ ，则对角阵 D 可逆，分裂 $A=D-(D-A)$ ，方程组等价于

$$x=D^{-1}(D-A)x+D^{-1}b$$

令 $B_J=D^{-1}(D-A)$ ，$f_J=D^{-1}b$ ，迭代格式为 $x^{(k+1)}=B_Jx^{(k)}+f_J\,(k=0,1,\cdots)$ ，称为 Jacobi 迭代。

Gauss-Seidel 迭代：设 $A=(a_{ij})_{n\times n}$ 的主对角元素 $a_{ii}\neq 0(i=1,2,\cdots,n)$ ，则下三角阵 $D-L$ 可逆，分裂 $A=(D-L)-U$ ，方程组等价于

$$x=(D-L)^{-1}Ux+(D-L)^{-1}b$$

令 $B_G=(D-L)^{-1}U$ ， $f_G=(D-L)^{-1}b$ ，迭代格式为 $x^{(k+1)}=B_Gx^{(k)}+f_G\,(k=0,1,\cdots)$ ，称为 Gauss-Seidel 迭代，也可以记为

$$Dx^{(k+1)}=Lx^{(k+1)}+Ux^{(k)}+b\,(k=0,1,\cdots)$$

逐次超松弛迭代（SOR 迭代）：设 $A=(a_{ij})_{n\times n}$ 的主对角元素 $a_{ii}\neq 0(i=1,2,\cdots,n)$ ，分裂 $\omega A=(D-\omega L)-((1-\omega)D+\omega U)$ ，其中 ω 为选定的非零参数，方程组 $Ax=b$ 等价于

$$(D-\omega L)x=((1-\omega)D+\omega U)x+\omega b$$

令 $B=(D-\omega L)^{-1}((1-\omega)D+\omega U)$， $f=\omega(D-\omega L)^{-1}b$，迭代格式为

$$x^{(k+1)}=Bx^{(k)}+f\,(k=0,1,\cdots),$$

称为逐次超松弛迭代，其中 ω 为松弛因子，也称 SOR 迭代。迭代格式还可以改写为

$$Dx^{(k+1)}=Dx^{(k)}+\omega(Lx^{(k+1)}+Ux^{(k)}-Dx^{(k)}+b)(k=0,1,\cdots)$$

Matlab 代码如下：

```
function [x,n]=SOR(A,b,x0,w,eps,M)
%--SOR迭代法解线性方程组
if nargin==4
    eps= 1.0e-6;
    M  = 10000;
elseif nargin<4
    error
    return
elseif nargin ==5
    M  = 10000;
end

if(w<=0 || w>=2)
    error;
    return;
end
D=diag(diag(A));
L=-tril(A,-1);
U=-triu(A,1);
B=inv(D-L*w)*((1-w)*D+w*U);
f=w*inv((D-L*w))*b;
x=B*x0+f;
n=1;
while norm(x-x0)>=eps
    x0=x;
    x =B*x0+f;
    n=n+1;
    if(n>=M)
        disp('Warning: 迭代次数太多，可能不收敛');
        return;
    end
end
```

6.2 共轭梯度法

已知线性方程组 $Ax=b$ 中的系数矩阵 A 为实对称正定阵，则求解方程组的真实解 x^* 等价于求泛函 $\varphi(x)=\dfrac{1}{2}(Ax,x)-(b,x)$ 的极小值点 x^*，求解方法是构造一个向量序列 $\left\{x^{(k)}\right\}$ 使 $\varphi(x^{(k)})\to\varphi(x^*)$。通常的解法是从初始向量 $x^{(0)}$ 出发，找一个方向 $p^{(0)}$，令 $x^{(1)}=x^{(0)}+\alpha_0 p^{(0)}$，使 $\varphi(x^{(1)})=\min\limits_{\alpha\in R}(x^{(0)}+\alpha p^{(0)})$。一般的，令 $x^{(k+1)}=x^{(k)}+\alpha_k p^{(k)}$，使

$$\varphi(x^{(k+1)})=\min_{\alpha\in R}(x^{(k)}+\alpha p^{(k)})$$

由此得出 $\alpha_k=\dfrac{\left(r^{(k)},p^{(k)}\right)}{(p^{(k)},Ap^{(k)})}$。根据不同的方向 $p^{(k)}$，可以有不同的迭代格式。

一般的有，选取方向 $p^{(k)}$ 使 $\varphi(x)$ 在点 $x^{(k)}$ 沿 $p^{(k)}$ 下降最快，即为 $\varphi(x)$ 在 $x^{(k)}$ 的负梯度方向

$$p^{(k)}=-\nabla\varphi(x^{(k)})=-(Ax^{(k)}-b)=r^{(k)}$$

则 $\alpha_k=\dfrac{\left(r^{(k)},r^{(k)}\right)}{(r^{(k)},Ar^{(k)})}$，于是建立求 $\varphi(x)$ 的极小值点 x^* 的最速下降法：任取 $x^{(0)}\in R^n$，

$$\begin{cases} r^{(k)}=b-Ax^{(k)},\alpha_k=\dfrac{\left(r^{(k)},r^{(k)}\right)}{(r^{(k)},Ar^{(k)})} \\ x^{(k+1)}=x^{(k)}+\alpha_k r^{(k)}\,(k=0,1,2,\cdots) \end{cases}$$

在最速下降法中，当 $r^{(k)}$ 较小时，由于舍入误差的影响，实际计算的 $r^{(k)}$ 会偏离最速下降方向，则选取方向向量 $p^{(k)}$ 为

$$\begin{cases} p^{(0)}=r^{(0)} \\ p^{(k)}=r^{(k)}+\beta_k p^{(k-1)}\,(k=1,2,\cdots) \end{cases}$$

并要求 $[p^{(k)},Ap^{(k-1)}]=0$，即相邻两步的方向向量关于矩阵 A 共轭，可得到

$$\beta_k=-\frac{\left(r^{(k)},Ap^{(k-1)}\right)}{(p^{(k-1)},Ap^{(k-1)})}$$

于是建立求 $\varphi(x)$ 的极小值点 x^* 的共轭梯度法。任取 $x^{(0)}\in R^n$，

$$
\begin{cases}
r^{(k)} = b - Ax^{(k)}, \beta_k = -\dfrac{\left(r^{(k)}, Ap^{(k-1)}\right)}{(p^{(k-1)}, Ap^{(k-1)})} = \dfrac{\left(r^{(k)}, r^{(k)}\right)}{(r^{(k-1)}, r^{(k-1)})} \\
p^{(k)} = r^{(k)} + \beta_k p^{(k-1)}, \alpha_k = \dfrac{\left(r^{(k)}, p^{(k)}\right)}{(p^{(k)}, Ap^{(k)})} = \dfrac{\left(r^{(k)}, r^{(k)}\right)}{(p^{(k)}, Ap^{(k)})} \\
x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)} (k = 1, 2, \cdots)
\end{cases}
$$

§7. 非线性方程的数值解法

7.1 不动点迭代法

将方程 $f(x) = 0$ 改写为等价形式 $x = \varphi(x)$，若 x^* 满足 $f(x^*) = 0$，则 $x^* = \varphi(x^*)$，则称 x^* 为 $\varphi(x)$ 的一个不动点，求 $f(x)$ 的零点等价于求 $\varphi(x)$ 的不动点。对于初始近似值 $x_0 \in [a, b]$，构造迭代格式

$$x_{k+1} = \varphi(x_k)(k = 0, 1, 2, \cdots)$$

若得到的点列 $\{x_k\}$ 有 $\lim\limits_{k \to \infty} x_k = x^*$，则迭代收敛，且 x^* 为 $\varphi(x)$ 的不动点，上述迭代格式称为不动点迭代。

如果把 Aitken 加速和不动点迭代结合，可得到

$$
\begin{cases}
y_k = \varphi(x_k), z_k = \varphi(y_k) \\
x_{k+1} = x_k - \dfrac{(y_k - x_k)^2}{z_k - 2y_k + x_k} (k = 0, 1, 2, \cdots)
\end{cases}
$$

称为 Steffensen 迭代法。

Matlab 代码如下：

```
function [root,n]=SteffenStablePoint(g,x0,eps)
%--Steffensen迭代，g为不动点迭代的迭代函数
if(nargin==2)
    eps=1.0e-4;
end
```

```
tol=1;
root=x0;
n=0;
while(tol>eps)
    n=n+1;
    r1=root;
    y=subs(sym(g),findsym(sym(g)),r1)+r1;
    z=subs(sym(g),findsym(sym(g)),y)+y;
    root=r1-(y-r1)^2/(z-2*y+r1);
    tol=abs(root-r1);
end
```

7.2 Newton 法

设已知方程 $f(x)=0$ 有近似根 x_k（假定 $f^{'}(x_k)\neq 0$），将函数 $f(x)$ 在点 x_k 展开，

$$f(x)\approx f(x_k)+f^{'}(x_k)(x-x_k)$$

也是方程 $f(x)=0$ 可近似的表示为

$$f(x_k)+f^{'}(x_k)(x-x_k)=0$$

这是个线性方程，其根记为 x_{k+1}，则 x_{k+1} 的计算公式为

$$x_{k+1}=x_k-\frac{f(x_k)}{f^{'}(x_k)}(k=0,1,2,\cdots)$$

这就是 Newton 法。

为了防止迭代发散，将 Newton 法与下山法结合起来，即在下山法保证函数稳定下降的前提下，用 Newton 法加快收敛速度，将 Newton 法的结果 $\overline{x}_{k+1}=x_k-f(x_k)/f^{'}(x_k)$ 与前一步的近似值 x_k 适当加权平均作为新的改进值 $x_{k+1}=\lambda\overline{x}_{k+1}+(1-\lambda)x_k$，其中 $\lambda(0<\lambda\leq 1)$ 为下山因子，

$$x_{k+1}=x_k-\lambda\frac{f(x_k)}{f^{'}(x_k)}(k=0,1,2,\cdots)$$

称为 Newton 下山法。选择 λ 时，从 $\lambda=1$ 开始，逐次将 λ 减半进行，直到满足下降条件 $\left|f(x_{k+1})\right|<\left|f(x_k)\right|$ 为止。

Matlab 代码如下：

```
function root=NewtonDown(f,a,b,x0,eps)
%--Newton下山法
if(nargin==3)
    eps=1.0e-4;
end
fa=subs(sym(f),findsym(sym(f)),a);
fb=subs(sym(f),findsym(sym(f)),b);
if(fa==0)
    root=a;
end
if(fb==0)
    root=b;
end
if(fa*fb>0)
    disp('两端点函数值乘积大于0!');
    return;
else
    tol=1;
    fun=diff(sym(f));
    root=x0;
    while(tol>eps)
        r1=root;
        fx=subs(sym(f),findsym(sym(f)),r1);
        dfx=subs(sym(fun),findsym(sym(fun)),r1);
        toldf=1;
        lamda=2;
        while toldf>0
            lamda=lamda/2;
            root=r1-lamda*fx/dfx;
            fv=subs(sym(f),findsym(sym(f)),root);
            toldf=abs(fv)-abs(fx);
        end
        tol=abs(root-r1);
    end
end
```

§9. 常微分方程的初值问题

9.1 Euler 法

一阶常微分方程的初值问题

$$\begin{cases} y' = f(x,y), x \in [x_0,b] \\ y(x_0) = y_0 \end{cases}$$

的解 $y = y(x)$ 称作它的积分曲线，积分曲线上一点 (x,y) 的切线斜率等于函数 $f(x,y)$ 的值。如果按函数 $f(x,y)$ 在 xy 平面上建立一个方向场，积分曲线上每一点的切线方向均与方向场在该点的方向一致。一般的，从初始点 $P_0(x_0,y_0)$ 依方向场在该点的方向推进到 $x = x_1$ 上的一点 $P_1(x_1,y_1)$，再重复上述步骤，作出一条折线 $\overline{P_0P_1P_2\cdots}$。而折线上的两个顶点 $P_n(x_n,y_n)$ 和 $P_{n+1}(x_{n+1},y_{n+1})$ 的坐标满足 $(y_{n+1}-y_n)/(x_{n+1}-x_n) = f(x_n,y_n)$，即

$$y_{n+1} = y_n + hf(x_n,y_n)$$

上述方法称为 Euler 法。

Matlab 代码如下：

```
function y = DEEuler(f, h,x0,b,y0,varvec)
format long;
N = (b-x0)/h;
y = zeros(N+1,1);
x = x0:h:b;
y(1) = y0;
for i=2:N+1
    y(i) = y(i-1)+h*Funval(f,varvec,[x(i-1), y(i-1)]);
end
format short;
function fv = Funval(f,varvec,varval)
var = findsym(f);
if length(var) > 4
    if var(1) == varvec(1)
        fv = subs(f,varvec(1),varval(1));
    else
        fv = subs(f,varvec(2),varval(2));
    end
else
    fv = subs(f,varvec,varval);
end
```

9.2　Runge-Kutta 方法

由 Euler 法和改进的 Euler 法，可总结类似的公式

$$y_{n+1} = y_n + h\varphi(x_n, y_n, h)$$

其中 $\varphi(x_n, y_n, h)$ 为增量函数，表达式为

$$\begin{cases} \varphi(x_n, y_n, h) = \sum_{i=1}^{r} c_i K_i \\ K_1 = f(x_n, y_n) \\ K_i = f(x_n + \lambda_i h, y_n + h\sum_{j=1}^{i-1} \mu_{ij} K_j)(i = 2, \cdots, r) \end{cases}$$

这里 c_i, λ_i, μ_{ij} 均为常数，上两式合称为 r 级显式 Runge-Kutta 法，精度记为 p 阶。

当 $r = 1$ 时，$\varphi(x_n, y_n, h) = f(x_n, y_n)$，就是 Euler 法，此时方法的阶为 $p = 1$。当 $r = 2$ 时，改进的 Euler 法公式就是其中一种，常用的计算公式还有中点方法，方法的阶均为 $p = 2$。

中点方法 Matlab 代码如下：

```
function y = RungeKutta2_mid(f,h,a,b,y0,varvec)
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
    K1 = Funval(f,varvec,[x(i-1) y(i-1)]);
    t = y(i-1) + h*K1/2;
    K2 = Funval(f,varvec,[x(i)+h/2 t]);
    y(i) = y(i-1)+h*K2;
end
format short;
```

常用的三阶 Runge-Kutta 公式有：

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}(K_1 + 4K_2 + K_3) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1) \\ K_3 = f(x_n + h, y_n - hK_1 + 2hK_2) \end{cases}$$

Matlab 代码如下：

```
function y = RungeKutta3(f, h,a,b,y0,varvec)
format long;
N = (b-a)/h;
y = zeros(N+1,1);
y(1) = y0;
x = a:h:b;
var = findsym(f);
for i=2:N+1
   K1 = Funval(f,varvec,[x(i-1) y(i-1)]);
   K2 = Funval(f,varvec,[x(i-1)+h/2 y(i-1)+K1*h/2]);
   K3 = Funval(f,varvec,[x(i-1)+h y(i-1)-h*K1+K2*2*h]);
   y(i) = y(i-1)+h*(K1+4*K2+K3)/6;
end
format short;
```

常用的还有经典四阶 Runge-Kutta 公式：

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_1) \\ K_3 = f(x_n + \dfrac{h}{2}, y_n + \dfrac{h}{2}K_2) \\ K_4 = f(x_n + h, y_n + hK_2) \end{cases}$$