Analyse Numérique : Algorithmes codés

Nirina DEVEAUX

2013-2014

1. **Factorisation LU**

Commande OCTAVE : [L U] = lu(A) et A\b

```
function[L,U] = an_lu(A)
[m,n] = size(A);
for k = 1:n
        for j = k:n
                U(k,j) = A(k,j);
        end
        L(k,k) = 1;
        for i = k+1:n
                L(i,k) = A(i,k)/A(k,k);
        end
        for i = k+1:n
                for j = k+1:n
                        A(i,j) = A(i,j)-L(i,k)*U(k,j);
                end
        end
end
```

```
function y = an_lts(L,b)
n = size(L,2)
for i = 1:n
        y(i) = b(i);
        for j = 1:i-1;
                y(i) = y(i) - L(i,j)*y(j);
        end
        y(i) = y(i)/L(i,i);
end
```

```
function x = an_solveLU(A,b)
[L,U] = an_lu(A);
y = L\b;
x = U\y;
```

2. **Factorisation PA=LU**

Commande OCTAVE: [L U P] = lu(A)

```
P1 = eye(4);
P1([1,4], : ) = P1([4,1], :  )
A = P1*A
L1 = eye(4); L1(2:4,1) = -A(2:4,1)/A(1,1);
A = L1*A

P2 = eye(4);
P2([2,4], : ) = P2([4,2], :  )
A = P2*A
L2 = eye(4); L2(3:4,2) = -A(3:4,2)/A(2,2);
```

```
A = L2*A

P3 = eye(4);
P3([3,4], : ) = P3([4,3], :  )
A = P3*A
L3 = eye(4); L3(4,3) = -A(4,3)/A(3,3);
U = L3*A            % On a donc L3*P3*L2*P2*L1*P1*A=U

P = P3*P2*P1;

LP3 = L3
LP2 = P3*L2*P3
LP1 = P3*P2*L1*P2*P3

L = eye(4);
L(4,3) = -LP3(4,3);
L(3:4,2) = -LP2(3:4,2);
L(2:4,1) = -LP1(2:4,1);

Vérification
[l u p] = lu(Asv);
norm(L-l)
norm(U-u)
norm(P-p)
```

3. <u>Factorisation QR</u>

Commande OCTAVE: [Q R] = qr(A,O)

```
A = eye(4,4) + diag(ones(3,1),-1) -100* diag(ones(3,1) , 1);
A(4,1) = -100;

R = A;

x = R(1:4,1);
x(1) = x(1) + sign(x(1))*norm(x)            % vk(1) = e1
v1 = x ./ norm(x) ;
R(1:4,1:4) = (eye(4) - 2*v1*v1')*R(1:4,1:4)
H1 = eye(4) - 2*v1*v1'                      %pas utile


x = R(2:4,2);
x(1) = x(1) + sign(x(1))*norm(x)            % vk(1) = e1
v2 = x ./ norm(x) ;
R(2:4,2:4) = (eye(3) - 2*v2*v2')*R(2:4,2:4)
H2 = eye(3) - 2*v2*v2'                      %pas utile

x = R(3:4,3);
x(1) = x(1) + sign(x(1))*norm(x)            % vk(1) = e1
```

```
v3 = x ./ norm(x) ;
R(3:4,3:4) = (eye(2) - 2*v3*v3')*R(3:4,3:4)
H3 = eye(2) – 2*v3*v3'          %pas utile

Q = eye(4);                      % Calcul de Q
Q(3:4,3:4) = (eye(2) - 2*v3*v3')*Q(3:4,3:4);
Q(2:4,2:4) = (eye(3) - 2*v2*v2')*Q(2:4,2:4);
Q = (eye(4) - 2*v1*v1')*Q

Vérification
[q r] = qr(A)
Q
norm(A -Q*R)
```

4. Méthode itérative de JACOBI

```
function [x iter rr] = jacobi_3diag(A,b,tol,maxit,x0)
n = length(b);
iter = 0;
x = x0;
rr=1;
x(1) = (b(1) - A(1,2)*x0(2))/ A(1,1);      %ligne1
while iter < maxit && rr>tol

        for I = 2:n-1
                x(i) =( b(i)-A(i,i-1)*x0(i-1) - A(i,i+1)*x0(i+1))/ A(i,i);
        end
rr=norm(b-A*x)/norm(b)              % norme(residu) / norme(second membre)
iter=iter+1
x(n) = (b(n) - A(n,n-1)*x0(n-1))/ A(n,n);    %ligneN
x0 = x;
end
```

5. Méthode itérative de GAUSS-SIEDEL

```
function [x iter rr] = gs_3diag(A,b,tol,maxit ,xo)
n = length(b);
x = xo;
for iter = 1:maxit
        x(1) = (b(1) - A(1,2)*x(2)) / A(1,1);
        for i = 2:n-1
                x(i) = (b(i) - A(i,i+1)*x(i+1) - A(i,i-1)*x(i-1))/A(i,i);
        end
        x(n) = (b(n) - A(n,n-1)*x(n-1)) / A(n,n);
        r = b – A*x;
        rr = norm(r)/norm(b);
        if (rr <= tol)
        break
        end
end
```

6. Recherche dichotomique

```
function[x it] = dico(f,a,b,tol)
if(f(a)*f(b)>0)
	error('f(a).f(b) doit etre <=0')
else
	it = 0;
	while abs(a-b) > tol
		x = (a+b)/2;
		it++;
		if f(x)*f(a) < 0
			b = x;
		elseif f(x)*f(b) < 0
			a = x;
		else
			return
		end
	end
end
```

7. Méthode de la fausse position

```
function[x it] = regfal(f,a,b,tol,maxit)
x = a - f(a)*(b-a)/(f(b)-f(a));
if(f(a)*f(b)>0)
	error('f(a).f(b) doit etre <= 0')
else
	it = 0;
	while norm(f(x)) < tol && it < maxit
		x = a - f(a)*(b-a)/(f(b)-f(a));
		it++;
		if f(x)*f(a)<0
			b = x;
		elseif f(x)*f(b)<0
			a = x;
		else
			return
		end
	end
end
```

8. Méthode de Newton (avec recherche linéaire)

```
function [x it] = newrl(f,df,x0,tol,maxit)
x = x0; fx = f(x);  r = norm(fx);
for it = 1:maxit
	if r<tol
		it--; return
	elseif (df(x)== 0)
		error('pente nulle')
	end
	d = df(x)\fx; a = 1;
	while(true)
		xn = x - d*a;
		if norm(f(xn)) < norm(fx)
			x = xn;
			break
		end
		a = a/2;
	end
	fx = f(x); r = norm(fx);
end
```

9. Méthode de Newton-Raphson

```
%remarque : ne pas ecrire x=x-f(x)/df(x)
% MAIS ecrire : x=x-df(x)\f(x)

function [x it] = newraph(f,df,x0,tol,maxit)
x = x0; fx = f(x);  r = norm(fx);
for it = 1:maxit
	if r < tol
		it--; return
	elseif (df(x) == 0)
		error('pente nulle')
	end
	x = x-df(x)\fx;
	fx = f(x); r = norm(fx);
end
```

10. Intégration numérique Trapèze

Commande Octave: quad(f,a,b)

```
function int = int_trapeze(f,a,b,n)
int = 0;
h = (b-a)/n;
for i = 1:n
	int = int+(f(a+(i-1)*h)+f(a+i*h));
end
int = int*h/2
```

11. Intégration numérique de Simpson

```
function int = int_simpson(f,a,b,n)
h = (b-a)/n;
int = 0;
for i = 1:n
        int = int+(f(a+(i-1)*h)+4*f(a+(i-0.5)*h)+f(a+i*h));
end
int = int*h/6
```

12. Méthode d'Euler progressive

```
function y = eulerp(f,y0,t)
y(:,1) = y0;
for i = 1:max(size(t)) - 1   % ou length(t)  plus rapide
        h = t(i+1) - t(i);
        y(:,i+1) = y(:,i) + h.*f(t(i),y(:,i));
end
```

13. Méthode d'Euler rétrograde

```
function y=eulerr(f,y0,t)
y(:,1) = y0;
for i = 1:max(size(t)) - 1   %ou length(t)  plus rapide
        h = t(i+1) - t(i);
        g = @(x) y(:,i) + f(t(i+1),x).*h-x;
        y(:,i+1) = fsolve(g,y(:,i));  % fsolve(...) raltentit eulerr
end
```

14. Méthode de Heun

```
function y = heun(f,y0,t)
y(:,1) = y0;
for i = 1:length(t) - 1
        h = t(i+1)-  t(i);
        fi = f(t(i),y(:,i));
        y(:,i+1) = y(:,i) + (fi + f(t(i+1),y(:,i)+h*fi)) * h/2;
end
```