TP1 Optimisation Continue : Compte rendu

Constanza Corentin

Juin 2022

Table des matières

1 Objectif

Dans ce TP nous allons appliquer différentes méthodes de gradient. Elles sont definies par la relation suivantes :

$$v^{k+1} = v^{(k)} - \rho_k \nabla(v^k)$$

Nous allons nous placer dans 2 cas, tout d'abords le cas Quadratiques et dans un second temps le cas non Quadratique. Nous allons commencer par présenter les fonctions que nous allons utiliser tout au longs de ce TP.

2 Fonctions

2.1 Gradient à pas fixe

Pour un vecteur initial v^0 arbitraire, la suite $(v^k)_k$ est definie $\forall k \in \mathbf{N}$ par

$$v^{k+1} = v^{(k)} - \rho \nabla f(v^k)$$

où ρ est à determiner "au mieux".
Nous allons utiliser le code matlab suivant :

```
function x = pf(x0, rho, eps, Nmax, grad)
    fprintf("Algorithme : Pas fixe \n\n");


    x=x0;
    x_prec=x0;
    x_suiv=x_prec - rho*grad(x);
    GRAD=[];
    n=0;
    while (norm(x_suiv-x_prec) > eps) && n<Nmax
        x_prec = x_suiv;
        d=grad(x_prec);
        GRAD=[GRAD norm(d)];
        x_suiv = x_prec - rho*d;
        x=x_suiv;
        n=n+1;

    end
    figure()
    plot(GRAD) %tracer de l'evolution de la solution


    if (n==Nmax)
        fprintf("Nombre d'iteration max atteint : \n");
    else
        fprintf("L'algorithme a converg  en %d it rations :\n\n",n);
    end
end
```

2.2 Gradient à pas optimal

Pour un vecteur initial v^0 arbitraire, on va construire à chaques itérations la suite $(v^k)_k$ qui est definie $\forall k \in \mathbf{N}$ par :

$$v^{k+1} = v^{(k)} - \rho(v^{(k)}) \nabla f(v^k)$$

où $\rho(v^{(k)})$ est choisi tel que

$$\underset{\rho \in \mathbf{R}}{inf} \{f(v^{(k)}) - \rho \nabla f(v^k)\} = f(v^{(k)} - \rho \nabla f(v^k))$$

Nous allons utiliser le code matlab suivant :

```
function x = po_quad(x0, rho, eps, A, Nmax,grad)
    fprintf("Algorithme : Pas optimal \n\n");

    x=x0;
    x_prec=x0;
    x_suiv=x_prec - rho*grad(x);
    GRAD = [];
    n=0;
    while (norm(x_suiv-x_prec) > eps) && n<Nmax
        x_prec = x_suiv;
        d=grad(x_prec);
        rho = (norm(d)^2)/dot(A*d,d);
        GRAD=[GRAD norm(d)];
        x_suiv = x_prec - rho*d;
        x=x_suiv;
        n=n+1;

    end
    figure()
    plot(GRAD) %tracer de l'evolution de la solution

    if (n==Nmax)
        fprintf("Nombre d'iteration max atteint : \n");
    else
        fprintf("L'algorithme a converg  en %d it rations :\n\n",n);
    end
end
```

2.3 méthodes de recherche linéaire

2.3.1 Règle d'Armijo

La règles d'Armijo consiste à trouver le plus petit $j \in \mathbf{N}$ tel que :

$$f(v^{(k)} - \frac{1}{2^j}\nabla f(v^{(k)})) \leq f(v^{(k)}) - \frac{\omega}{2^j}||\nabla f(v^k)||$$

avec $w \in [0,1]$, ici $w = 0,5$.
Nous utiliserons le code matlab suivant :

```
function [X] = armijo(x0,eps,Nmax, f, Grad)
    fprintf("Algorithme : Armijo \n\n");


    x = x0;
    X = [];
    grad = Grad(x);
    GRAD = [];
    i=0;
    while norm(grad)>eps
        i=i+1;
        if(i>Nmax)
            fprintf("Le nombre d'it ration maximal a   t   atteint, on obtient :"
            break;
        end
        % Calcul de rho
        j=1; w=1/2;
        while f(x-1/(2^j)*Grad(x)) > f(x)-w/(2^j)*norm(Grad(x))^2
            j=j+1;
        end
        rho=1/(2^j);
        % Calcul de l'approximation
        x = x-rho*grad;
        X = [X x];
        grad = Grad(x);
        GRAD = [GRAD norm(grad)];
    end
```

```
    figure()
    plot(GRAD);
    if(i<=Nmax)
        fprintf("L'algorithme a converg  en %d it rations :\n\n",i);
    end
    disp(x);
end
```

2.3.2 Règle de Wolfe

On pose $v^{(k)}$ le point courant, $g^{(k)} = \nabla f v^{(k)}$, $d^{(k)}$ la direction de descente avec $\langle g^{(k)}, d^{(k)} \rangle < 0$, ρ_k le pas de décente.
La règle de Wolfe à pour objectif de determiner ρ_k selon deux condition :
- La fonction doit décroite "suffisament vite" :

$$f(v^{(k)} - \rho_k d^{(k)}) \leq f(v^{(k)}) + \omega_1 \rho_k \langle g^{(k)}, d^{(k)} \rangle$$

- Le pas ρ_k doit être suffisament "grand" :

$$\langle \nabla f(v^{(k)} - \rho_k d^{(k)}), d^{(k)} \rangle \geq \omega_2 \langle g^{(k)}, d^{(k)} \rangle$$

On a donc le code matlab suivant :

```
function x = wolfe(x0,eps,Nmax, f, Grad)
    fprintf("Algorithme : Wolfe \n\n");

    x = x0;
    g = Grad(x);
    GRAD = [norm(g)];
    i=0;

    while norm(g)>eps
        i=i+1;
        if(i>Nmax)
            fprintf("Le nombre d'it ration maximal a   t   atteint, on obtien
            break;
        end


        % Calcul de rho
        w1 = 0.1; w2 = 0.9;
        rhoInf = 0; rhoSup = Inf; rho = 100;
        j=0;
        while 1
            j=j+1;
            if j==Nmax% On s'assure que la recherche du pas aboutis
                fprintf("La recherche du rho n'a pas aboutis, on a la solution
                disp(x);
                return;
            end
            if f(x-rho*g)>f(x)+w1*rho*-norm(g)^2 %premi  re condition de Wol
                rhoSup = rho;
                rho = 1/2*(rhoInf+rhoSup);
            elseif dot(Grad(x-rho*g), -g) < -w2*norm(g)^2 % deuxi  me condit
                rhoInf = rho;
                if rhoSup == Inf
                    rho = 2*rhoInf;
                else
                    rho = 1/2*(rhoInf+rhoSup);
                end
            else
                break;% On sort de la boucle si les deux conditions sont v   r
            end
        end

        % Calcul de l'approximation
```

```
        x = x-rho*g;
        g=Grad(x);
        GRAD = [GRAD norm(g)];


    end

    if(i<=Nmax)
        fprintf("L'algorithme a converg  en %d it rations \n\n:",i);
    end
    fprintf("On a la solution suivante :\n");
    disp (x)
    figure()
    plot(GRAD);
end
```

3 Tests

3.1 Cas quadratique

$$f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

Nous allons utiliser :

$$A = \begin{pmatrix} 2 & -1.5 & -0.5 \\ -1.5 & 2 & 0 \\ -0.5 & 0 & 2 \end{pmatrix}$$

$$b = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Avec le script suivant :

Listing 1 –

```
A = [2 -1.5 -0.5 ;-1.5 2 0; -0.5 0 2];
b = [1; 2; 1];
x0 = [1/2;1/2;1/2];
eps = 10e-6;
Nmax =15000;
rho=2/( max(eig(A))+min(eig(A)) );

fq = @(x)( (1/2) * dot(A*x,x)-dot(b,x) );
gradq = @(x) ( (1/2) * (A+A')*x - b ) ;

g = @(x) exp(x(1)) +  exp(x(2)) + exp(x(3));
gradg = @(x)[exp(x(1));exp(x(2)); exp(x(3))];

f_nq = @(x)( (1/2) * dot(A*x,x)-dot(b,x) +g(x) );
grad_nq = @(x) ( (1/2) * (A+A')*x - b +gradg(x));

fprintf("-----------Etudes du cas quadratique---------------\n\n");
fprintf("Condition initiale :\n");
disp(x0);

pf(x0, rho, eps, Nmax, gradq)

po_quad(x0, rho, eps, A, Nmax, gradq)

armijo(x0, eps, Nmax,fq,gradq);

wolfe(x0, eps, Nmax,fq,gradq);

```

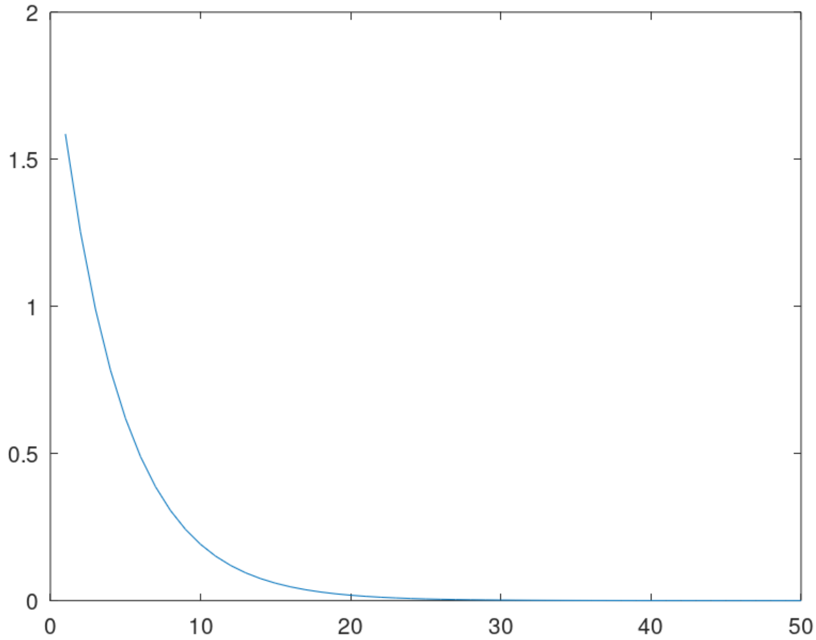
```
fprintf("-----------Etudes du cas non quadratique---------------\n\n");
fprintf("Condition initiale :\n");
disp(x0);

pf(x0, rho, eps, Nmax, grad_nq)

armijo(x0, eps, Nmax,f_nq,grad_nq);

wolfe(x0, eps, Nmax,f_nq,grad_nq);
```

On à la sortie suivante :

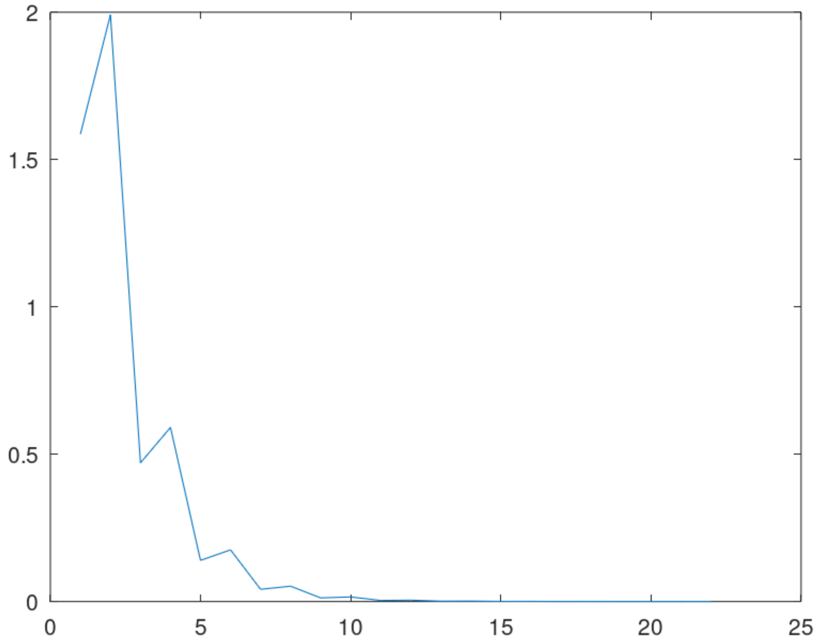
```
>> script

-----------Etudes du cas quadratique---------------

Condition initiale :
   0.5000
   0.5000
   0.5000
Algorithme : Pas fixe

L'algorithme a convergé en 50 itérations :

ans =

   3.6666
   3.7500
   1.4167

Algorithme : Pas optimal

L'algorithme a convergé en 22 itérations :

ans =

   3.6667
   3.7500
   1.4167

Algorithme : Armijo

L'algorithme a convergé en 52 itérations :

   3.6667
   3.7500
   1.4167
Algorithme : Wolfe

L'algorithme a convergé en 45 itérations

:On a la solution suivante :
   3.6667
   3.7500
   1.4167
```

Voici les graphiques qui suivent l'evolution de la norme du gradient :

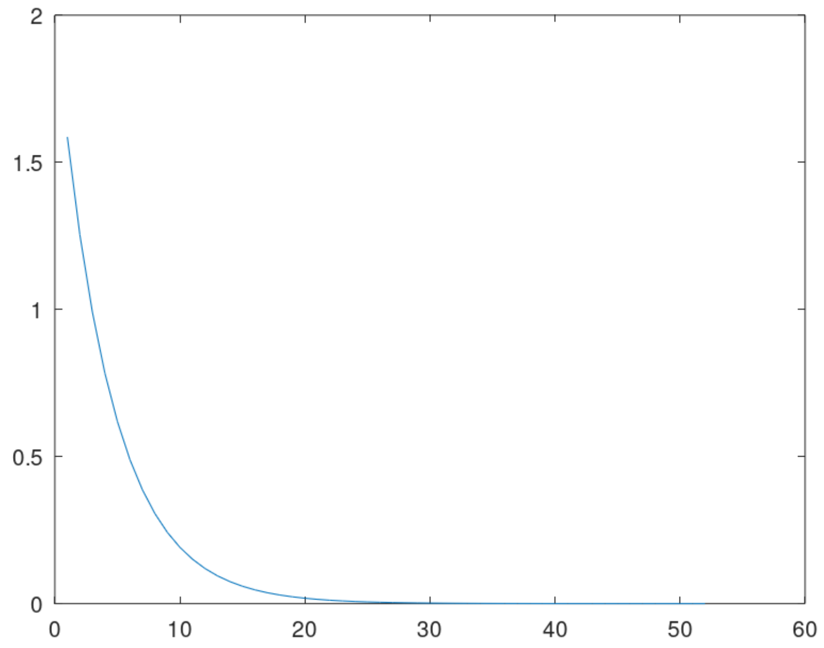
Pour le pas fixe :



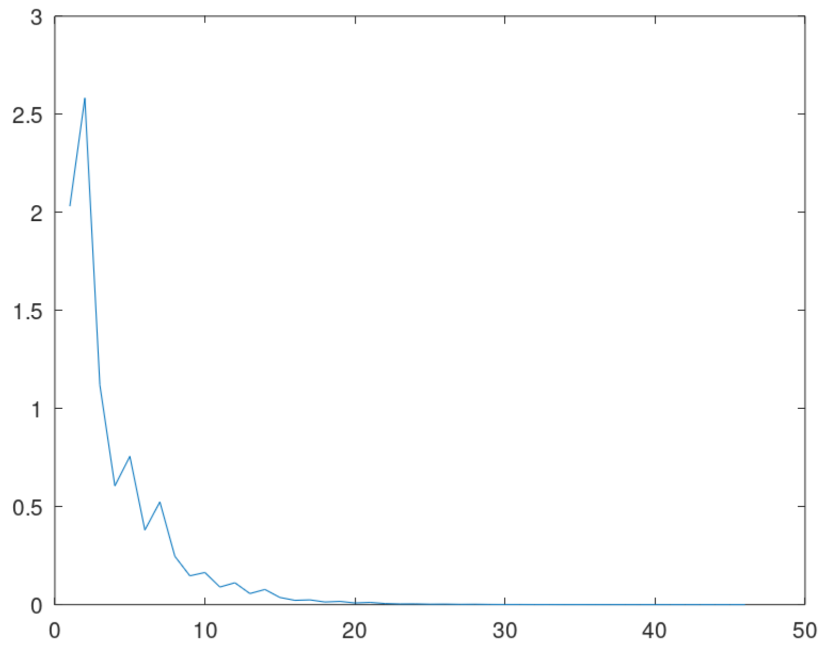
Pour le pas optimal :



Pour Armijo :



Pour Wolfe :



3.2 Cas non Quadratique

$f(x) = \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle + \exp(x_1) + \exp(x_2) + \exp(x_2)$
avec le script suivant :

Listing 2 –

```
A = [2 -1.5 -0.5 ;-1.5 2 0; -0.5 0 2];
b = [1; 2; 1];
x0 = [1/2;1/2;1/2];
eps = 10e-6;
Nmax =15000;
rho=2/( max(eig(A))+min(eig(A)) );
```

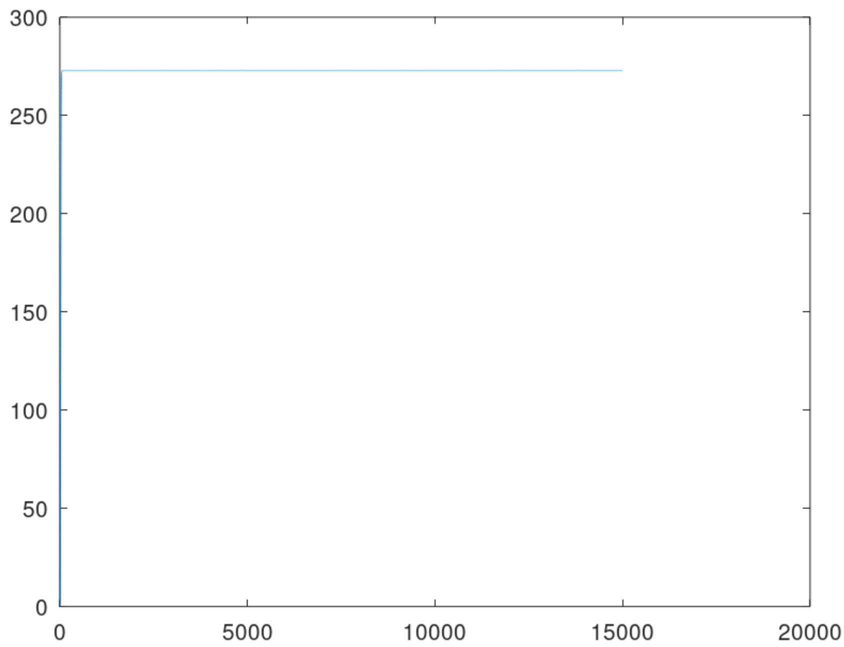
```
fq = @(x)( (1/2) * dot(A*x,x)-dot(b,x) );
gradq = @(x) ( (1/2) * (A+A')*x - b ) ;

g = @(x) exp(x(1)) +  exp(x(2)) + exp(x(3));
gradg = @(x)[exp(x(1));exp(x(2)); exp(x(3))];

f_nq = @(x)( (1/2) * dot(A*x,x)-dot(b,x) +g(x) );
grad_nq = @(x) ( (1/2) * (A+A')*x - b +gradg(x));

fprintf("-----------Etudes du cas quadratique---------------\n\n");
fprintf("Condition initiale :\n");
disp(x0);

pf(x0, rho, eps, Nmax, gradq)

po_quad(x0, rho, eps, A, Nmax, gradq)

armijo(x0, eps, Nmax,fq,gradq);

wolfe(x0, eps, Nmax,fq,gradq);


fprintf("-----------Etudes du cas non quadratique---------------\n\n");
fprintf("Condition initiale :\n");
disp(x0);

pf(x0, rho, eps, Nmax, grad_nq)

armijo(x0, eps, Nmax,f_nq,grad_nq);

wolfe(x0, eps, Nmax,f_nq,grad_nq);
```

On à la sortie suivante :

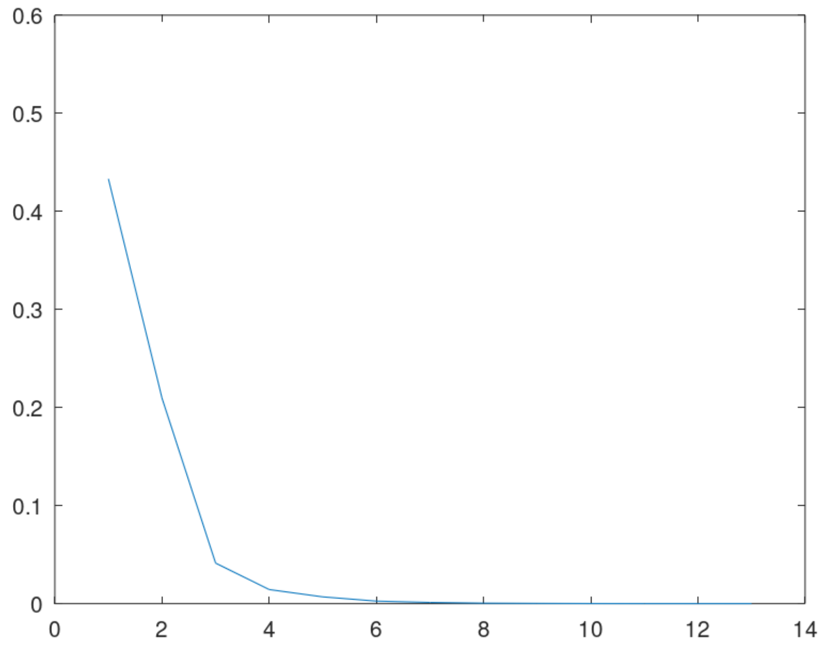
```
-----------Etudes du cas non quadratique---------------

Condition initiale :
   0.5000
   0.5000
   0.5000
Algorithme : Pas fixe

Nombre d'iteration max atteint :
ans =

  -92.3906
    3.7500
    1.4167

Algorithme : Armijo

L'algorithme a convergé en 13 itérations :

   0.199474
   0.402116
   0.033062
Algorithme : Wolfe

L'algorithme a convergé en 14 itérations

:On a la solution suivante :
   0.199472
   0.402111
   0.033061
```

Voici les graphiques qui suivent l'evolution de la norme du gradient :

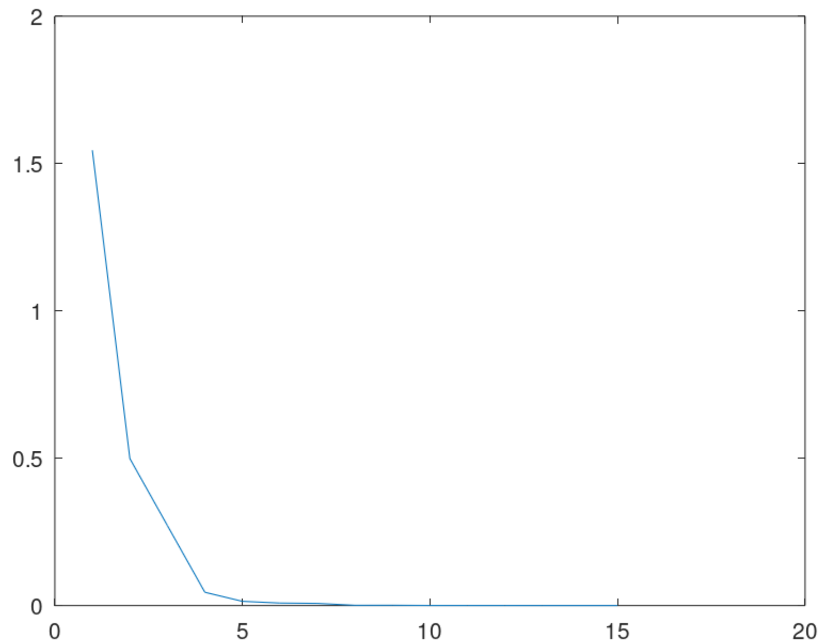
Pour le pas fixe :



On voit bien la divergences
Pour Armijo :



Pour Wolfe :



4 Comparaison des méthodes

On peut voir que c'est dans le cas quadratiques la méthode du pas optimal la plus rapide. Cependant elle est inapplicable dans le cas non quadratique. De plus comme la convergence de la méthodes à pas fixes n'est pas assuré dans le cas non quadratique (c'est divergent dans notre test d'ailleurs), il faut passer par des méthodes de recherches linéaire, plus couteuse en itération mais qui converge. Des deux méthodes de recherche linéaire, Wolfe est la plus optimal (Dans le cas quadratique comme non quadratique).

On priviligiera donc le pas optimal dans le cas quadratique et Wolfe dans le non quadratique.