Optimisation TP7 : Méthode de gradient

ZHONG Ming

5 avril 2019

Table des matières

1 Rappel du problème

le but de ce TP est d'implémenter en Scilab un code pour valider et expérimenter l'algorithme du gradient.

1.1 algorithme du gradient

L'algorithme du gradient est destiné à minimiser une fonction réelle différentiable définie sur un espace euclidien qui utilise la direction du gradient, fait partie de la famille des algorithmes à directions de descente.

1.2 3 versions d'algorithme du gradient

On considère l'algorithme du gradient à pas fixé/variable/optimal :

$$\begin{cases} x_0 \in \mathbb{R}^n, \text{ donné} \\ x_{k+1} = x_k - \rho \nabla f(x_k) \end{cases}$$

– GF(la méthode du gradient à pas fixé) :

$$\rho = constant$$

– GV(la méthode du gradient à pas variable) :

$$\rho = variable$$

– GO(la méthode du gradient à pas optimal) :
à chaque itération k le pas ρ_k minimise la fonction $f(\rho) = J(u_k - \rho \nabla J(u_k))$

2 Implémentation

2.1 fonctions de test

Pour les expérimentations et validations, on considérera les fonctions suivantes :

–
$$J_1(v) = \sum_{i=1}^{i=N} (v_i - 1)^2$$

–
$$J_2(v) = \sum_{i=1}^{i=N} (v_i - i)^2$$

–
$$J_R(v) = \sum_{i=1}^{i=N-1} \left\{ \left(v_{i+1} - v_i^2\right)^2 + (v_i - 1)^2 \right\}$$

–
$$J_H(x, y) = \left(x^2 + y - 2\right)^2 + \left(y^2 - 2x + 1\right)^2$$

2.2 méthode du gradient à pas fixé

Algorithm 1: gradient à pas fixé

Result: u_k
initialisation de N, epsg, Kmax, u_k, ρ
while *k = 1 :Kmax* **do**
  [J,G] = cost(uk)
  if *(norm(G)<epsg)* **then**
    break
  else
    u_k = u_k - ρ*G
  end
end

2.3 méthode du gradient à pas optimal

Construction l'approximation parabolique de la fonction $f(t) = J\left(u_k - t\nabla J\left(u_k\right)\right)$

$$f(t) = J\left(u_k - t\nabla J\left(u_k\right)\right) = a_0 + a_1 \cdot t + a_2 \cdot t^2$$

Donc on a :

$$a_0 = f\left(0\right) = J\left(u_k\right)$$
$$a_1 = f'\left(0\right) = -\left\|G\left(u_k\right)\right\|^2$$
$$a_2 = \left(J\left(u_k - t_{k-1} \cdot G\left(u_k\right)\right) - a_0 - a_1 \cdot t_{k-1}\right)/t_{k-1}^2$$
$$t_k = -a_1/2a_2$$

Algorithm 2: gradient à pas optimal

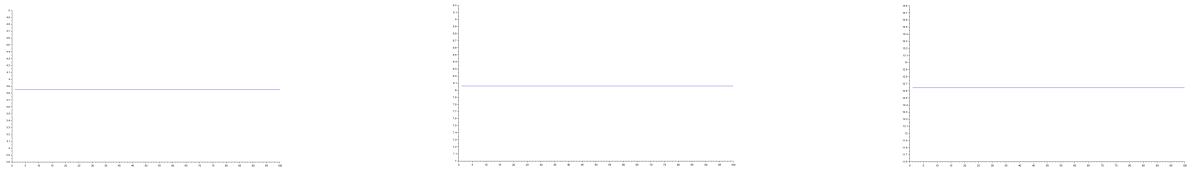
Result: u_k
initialisation de N, epsg, Kmax, u_k
while *k = 1 :Kmax* **do**
  [J,G] = cost(uk)
  if *(norm(G)<epsg)* **then**
    break
  else
    a0 = J
    a1 = -norm(G)^2
    J1,G1= cost(uk-t(k-1)*G)
    a2 = (J1-a0-a1*t(k-1)) / t(k-1)^2
    t(k) = -a1/(2*a2)
    u_k = u_k - t(k)*G
  end
end

3 Résultats

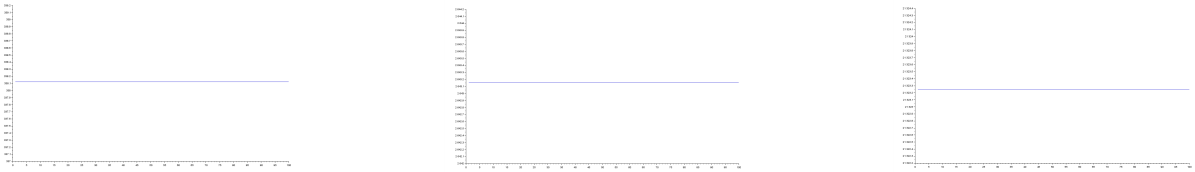
3.1 validation de GF pour J1 et J2

on prend le seuil de convergence $epsg = 10^{-6}$, un pas fixe t = 1, le nombre d'itération maximal Kmax = 100

cost function : J1 , N = 10, 20, 40 de gauche à droit

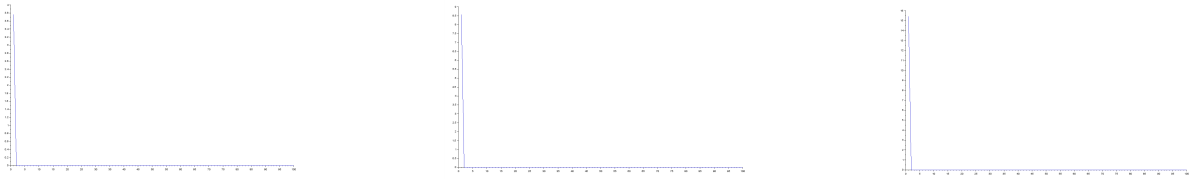


cost function : J2 , N = 10, 20, 40 de gauche à droit

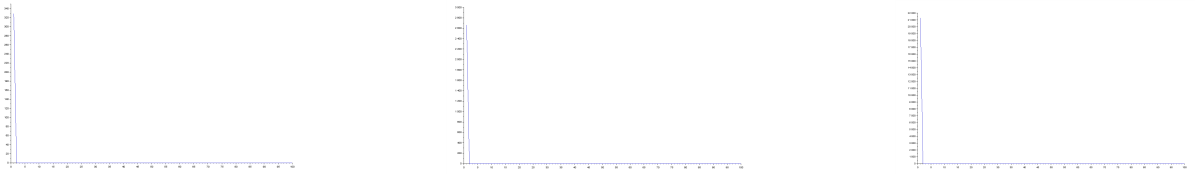


Ensuite, on prend un pas fixe t = 0.5

cost function : J1 , N = 10, 20, 40 de gauche à droit

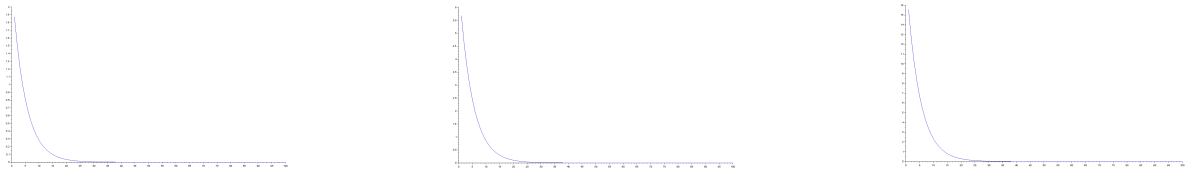


cost function : J2 , N = 10, 20, 40 de gauche à droit

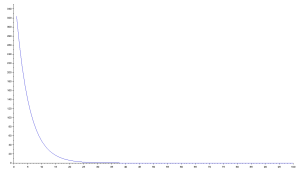
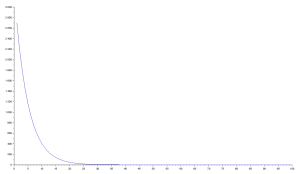
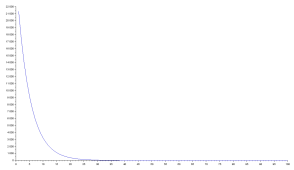


Ensuite, on prend un pas fixe t = 0.05

cost function : J1 , N = 10, 20, 40 de gauche à droit



cost function : J2 , N = 10, 20, 40 de gauche à droit

On constant que pour fonctions J1 et J2, le pas = 1 ou 0.5 est trop grand, il converge trop vite. En outre, pour différents N, les figures sont simulaires.

3.2 validation de GO pour J1 et J2

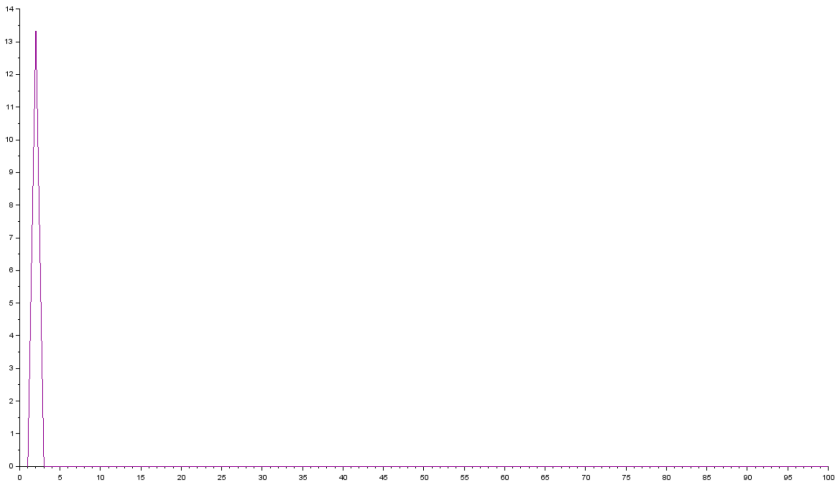


FIG. 1: cost function=J1,N=40

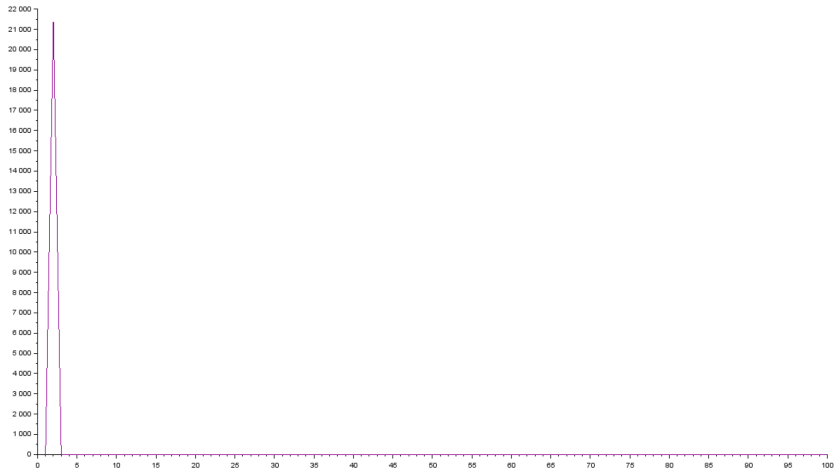


FIG. 2: cost function=J2,N=40

3.3 validation de GF et GO pour JR

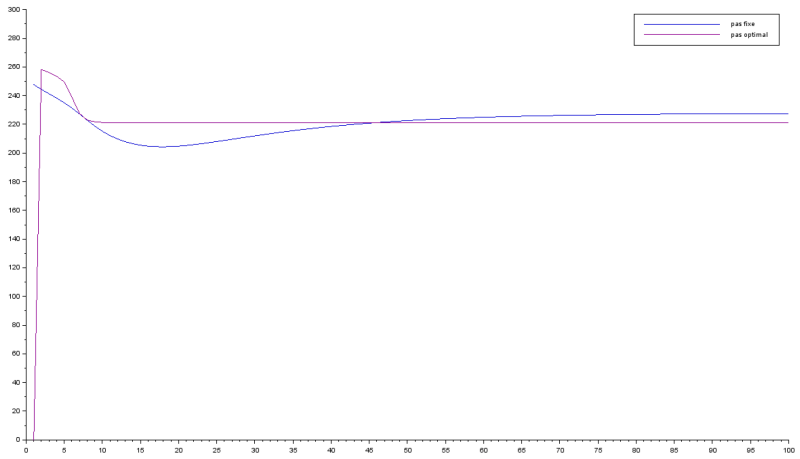


FIG. 3: cost function=JR,N=40,pasfixe=0.03

3.4 validation de GF et GO pour JH

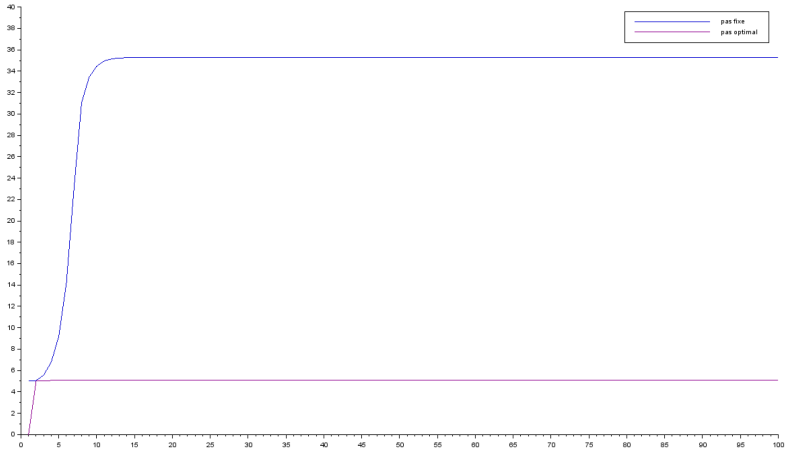


FIG. 4: cost function=JH,N=2,pasfixe=0.03,u0=(0,0)

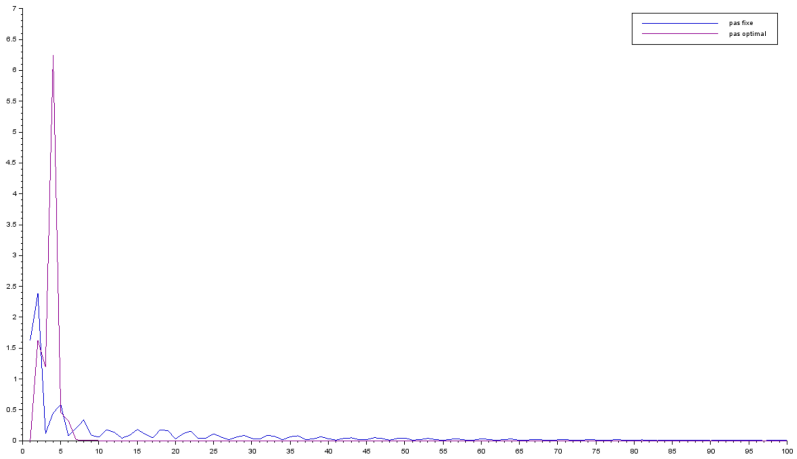


FIG. 5: cost function=JH,N=2,pasfixe=0.07,u0=(1.5,-1.5)

On peut constater que quand u0 = (0,0), il tombe dans un piège(optimum local), GF et GO tombe dans 2 différents pièges. Quand u0 = (1.5,-1.5), ces deux méthodes marchent très bien.

4 Code en scilab

```
// MAM_OPT_TP7
```

```
//auteur : ZHONG Ming
//
clear()

function [J,G]=cost1(v)
    J = 0
    n = length(v)
    for i=1:n
        J = J + (v(i)-1)^2
    end
    G = 2*v-2
endfunction



function [J,G]=cost2(v)
    J = 0
    n = length(v)
    G = zeros(n)
    for i=1:n
        J = J + (v(i)-i)^2
        G(i) = 2*(v(i)-i)
    end

endfunction



function [J,G]=costR(v)
    J = 0
    n = length(v)
    G = zeros(n-1)
    for i=1:n-1
        J = J + (v(i+1)-v(i)^2)^2 + (v(i)-i)^2
        G(i) = -4*v(i)*(v(i+1)-v(i)^2)+2*(v(i)-1)
    end
    J = J + (v(n)-1)^2
    G(n) = 2*(v(n)-i)
endfunction




function [J,G]=costH(v)
    x = v(1)
    y = v(2)
    J = (x^2+y-2)^2+(y^2-2*x+1)^2
    G(1) = 4*x*(x^2+y-2)+4*(y^2-2*x+1)
    G(2) = 2*(x^2+y-2)+4*y*(y^2-2*x+1)
```

```
endfunction

//MAM_OPT_TP7
//auteur : ZHONG Ming
//
//exec('optim_fonction.sci')

function [J,G]=cost(v)
    [J,G]=costH(v)
endfunction


N = 10
epsg = 10^-6
Kmax = 100
uk = [1.5;-1.5]
J_cost_gf = zeros(Kmax,1)
J_cost_go = zeros(Kmax,1)


pas_fixe = 0.07
for k = 1:Kmax
    [J,G] = cost(uk)
    J_cost_gf(k) = J
//    if(norm(G)<epsg) then break; end
        u1 = uk - pas_fixe*G
        uk = u1
end

//parabolique : f(t) = a0 + a1*t + a2*t^2 = J(uk - t*G(uk))
// f'(t) = a1 + 2*a2*t  :  f'(tk) = 0  ====>   tk = -a1/(2*a2)
//f(0) = a0 = J(uk)
//f'(0) = a1 = -G(uk)*G(uk)
//f(tk-1) = a0 + a1*tk-1 + a2*tk-1^2 = J(uk - tk-1*G(uk))   ===> a2 = (J(uk -
    tk-1*G(uk)) - a0 - a1*tk-1) / tk-1^2
//                                                              = (J(uk -
    tk-1*G(uk)) - J(uk) - G(uk)*tk-1) / tk-1^2


u = [1.5;-1.5]
kappa = 1
t = zeros(Kmax,1)
t(1) = 1
for k = 2:Kmax

    [J,G] = cost(u)
    if(norm(G)<epsg) then break
        else
        J_cost_go(k) = J

        a0 = J
```

```
        a1 = -norm(G)^2

        [J1,G1] = cost(u-t(k-1)*G)
        a2 = (J1-a0-a1*t(k-1)) / t(k-1)^2
        t(k) = -a1/(2*a2)
        t(k) = t(k) * kappa
        u2 = u - t(k)*G
        u = u2
    end

end



scf(1)
plot2d(J_cost_gf,style=11)
plot2d(J_cost_go,style=22)
legend("pas fixe","pas optimal")
```