UR M2 STIM

Rapport du projet d'optimisation

HAO Chang
13/03/2016

Sommaire

I. Préambule

On considère dans ce projet le problème de l'approximation d'une séquence de points par une courbe paramétrique. Alors pour une suite de points $p_i = (xi,yi)^T$ avec $1 \leq i \leq n$, on a une trajectoire dépendant du temps dans R^2. La figure 1 illustre graphiquement cette trajectoire.

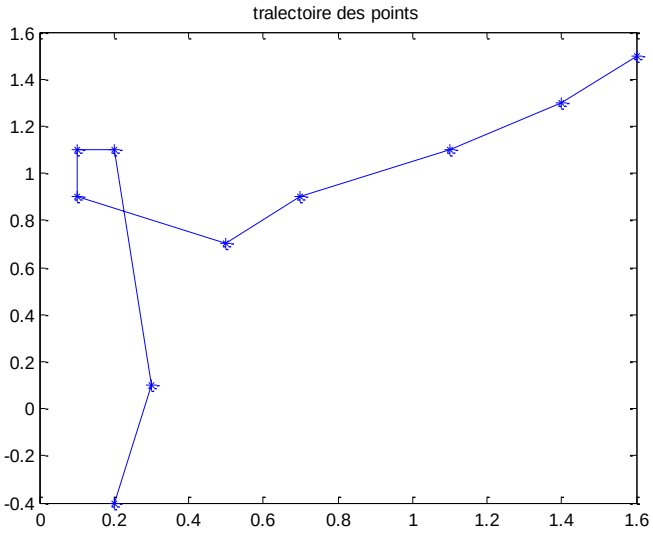

Figure 1 Trajectoire à modéliser par une courbe paramétrique m(t)

II. Partie 1: Descente de gradient et Gauss-Newton

I.1 Méthode du gradient

Le critère J est :

$$Ji(t) = \left|\left|m(t) - Pi\right|\right|^2 = (a_1*t^3 + b_1*t^2 + c_1*t + d_1 - xi)^2 + (a_2*t^3 + b_2*t^2 + c_2*t + d_2 - yi)^2$$

Dans le cas de la méthode du gradient, l'expression générale de $t_{i,n+1}$ est :

$$t_{i,n+1} = t_{i,n} - \alpha_n * \nabla J(tn)$$

$$t_{i,n+1} = t_{i,n} - \alpha_n * (2 * m'(t) * (m(t) - Pi))$$

Avec m(t) s'écrit sons forme polynomiale, on a :

$$\nabla J(tn) = 2 * m_x'(t) * (m\_x(t) - Pxi) + 2 * m\_y'(t) * (m\_y(t) - Pyi)$$

$M' = 3a*t^2 + 2*b*t + c$

Après 4 itération, nous avons obtenu le critère J_DG = 0.030421.

I.2 Méthode du gausse-Newton

Dans cette méthode, nous avons l'expression de t (i,n+1) suivant :

$$t_{i,n+1} = t_{i,n} - H(t_{n,i})^{-1} * \nabla J(tn), \text{ avec } H(t_{n,i}) = \nabla^2 J(tn)$$

$$H(t\,n, i) = 2(m'' * (m - Pi) + m'^{\,T} * m')$$

$M'' = 6*a*t + 2*b$

Après 4 itération, nous avons obtenu le critère J_NT = 0.017323.

```
DG: The 1th iteration, J = 2.740000, NT: The 1th iteration, J = 2.740000,
DG: The 2th iteration, J = 0.041504, NT: The 2th iteration, J = 0.853069,
DG: The 3th iteration, J = 0.039609, NT: The 3th iteration, J = 0.277115,
DG: The 4th iteration, J = 0.036285, NT: The 4th iteration, J = 0.081150,
DG: The 5th iteration, J = 0.030421, NT: The 5th iteration, J = 0.017323,
```

En comparant l'évolution de la valeur J, nous pouvons observer que J de la méthode du gradient varie suive la direction où la vitesse de descente est plus rapide. Avec la méthode de Gausse-Newton, J diminue plus que la méthode du gradient.

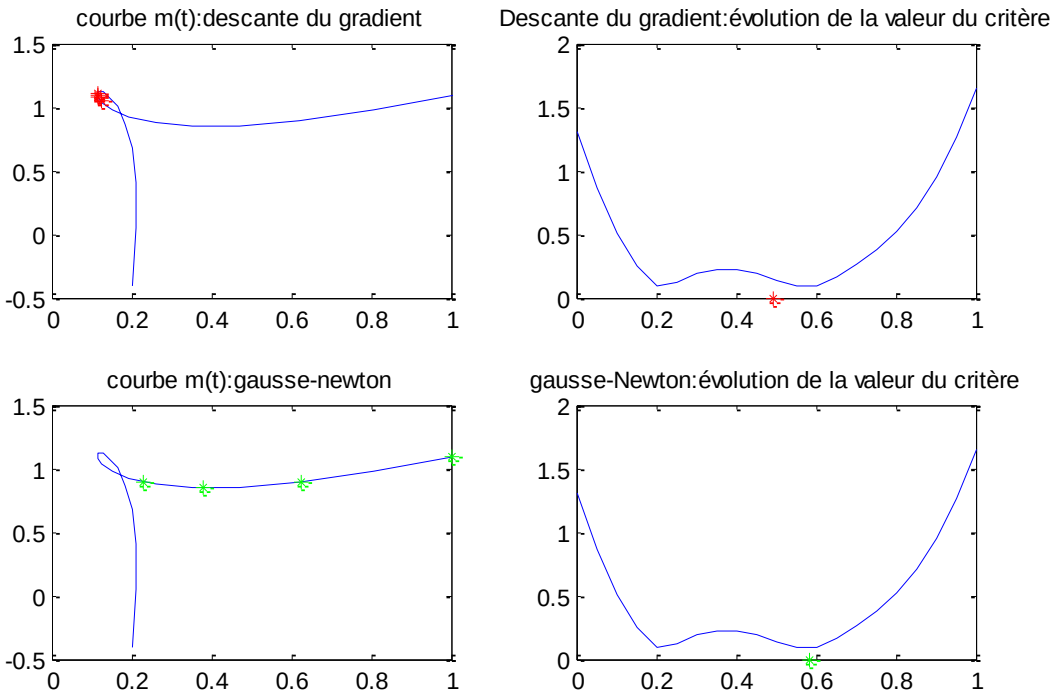

Figure 2: comparaison des deux méthodes

III. Partie 2 : Moindres carrées

Dans cette partie, nous considérons que les valeurs de ti sont fixées et nous cherchons à déterminer les valeurs optimales des paramètres de la fonction m(t) pour que la courbe la représentant passe au plus des points d'approximation fixés, au sens de la distance Euclidienne.

On reprend l'expression précédente de m(t) dont on cherche les valeurs optimales de a, b, c, et d :

M(t) = $at^3 + bt^2 + ct + d$

On cherche à minimiser J (θ)= ||Pi - Xθ||² avec $\theta = [a, b, c, d]^T$,

$$J(\theta) = (Pix - m_x)^2 + (Piy - m_y)^2 = Jx(a1, b1, c1, d1) + Jy(a2, b2, c2, d2)$$

On éstime les paramètres : $\theta x = [a1, b1, c1\ d1] et\ \theta y = [a2, b2\ c2, d2]$

$\theta = inv(X'*X)*X'*Ys$; d'où X =Ti, Ys = [Px, Py].

On définit Ti = (0.1: 0.1: 1). Le Résultat de l'estimation est ci-dessous et la courbe obtenue est illustrée dans la figure 3.

```
-2.2339    6.7541   -3.4386    0.5867
13.9083  -24.9184   14.2479   -1.6200
```

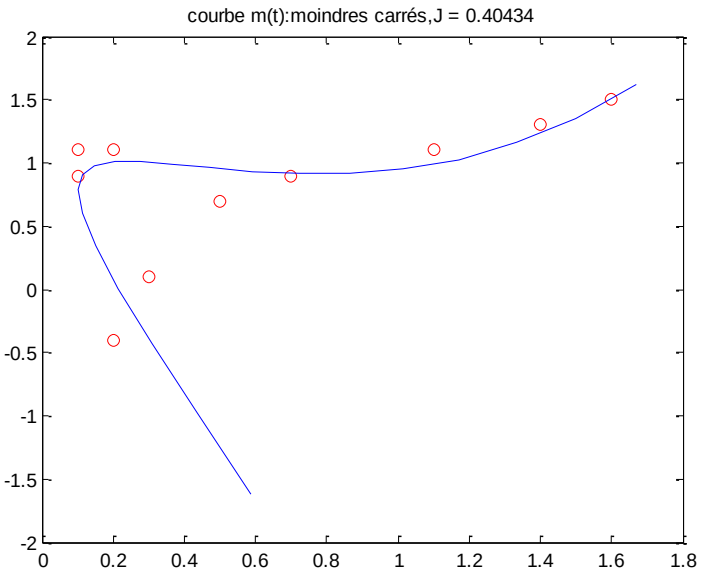

Figure 3: résultat de l'optimisation par moindres carrés

Le résultat obtenu par la méthode de Moindres carrés n'est pas optimal.

IV. Partie 3 : Moindres carrés et Gausse-Newton

Dans cette partie, nous allons combiner les outils développés dans la partie 1 et 2 pour approximer au mieux l'ensemble de points par une courbe paramétrique polynomiale de degré 3.

On définit le critère d'arrêt suivant : MSEnew- MSEold <epsilon, d'où MSE est la distance moyenne entre les points fixes Pi et la courbe m(t). Avec l'algorithme alternant des étapes d'optimisation, on a obtenu les paramètres estimés ci-dessous :

Algorithme :

Initialiser : ti = (0.1: 0.1: 1).

Tans que le critère d'arrêt n'est pas atteint, faire :

- Optimiser les paramètre à ti : [param,J(k)] = Estime_MC(t,P);
  Calculer MSE
- Optimiser les ti pour les paramètre obtenus :

```
for i = 1:length(t)
  told = t(i);
  [ tnew,~,m(:,k+1) ] = UpPasGaussienNewton( param',
told ,P(1,i),P(2,i) );
  t_optimiser = [t_optimiser tnew];
end
```

- Corriger les ti pour qu'ils sont dans [0 ,1]
- Calculer nouvelle MSE pour les ti

```
    9.8958   -11.1078     3.2558     0.0033
   21.3324   -36.0182    18.4480    -1.8517

t_optimiser =

    0.0953    0.1419    0.3486    0.4585    0.5781    0.7811    0.8326    0.8936    0.9305    0.9551
```

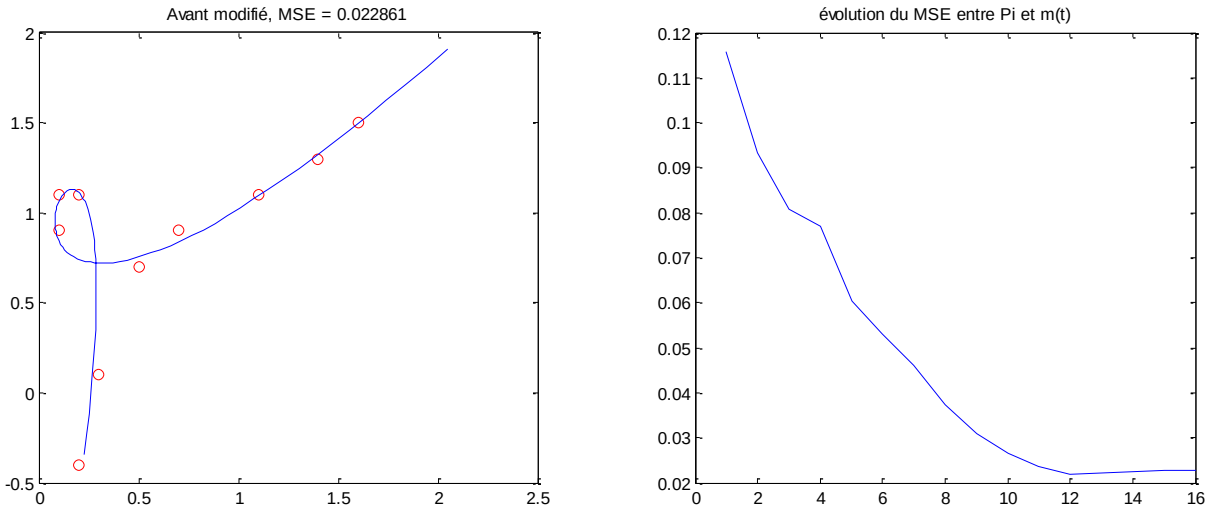

Figure 4 : résultat obtenu et les MSE de chaque itération

Pour remédier au problème que cet algorithme ne prendre pas en compte d'éventuels optimal locaux, on propose d'injecter une part d'aléatoires dans les ti. Cette modification permet de prendre en compte les voisinages de ti pour éviter de tomber dans un optimal local.

```
if abs(J(k+1)-J(k)) < 10e-3
     t= (0.8+0.4*rand(1))*t;
end
```

Les ti modifiés sont dans [0,1].

Après avoir appliqué la modification sur ti, nous avons obtenu :

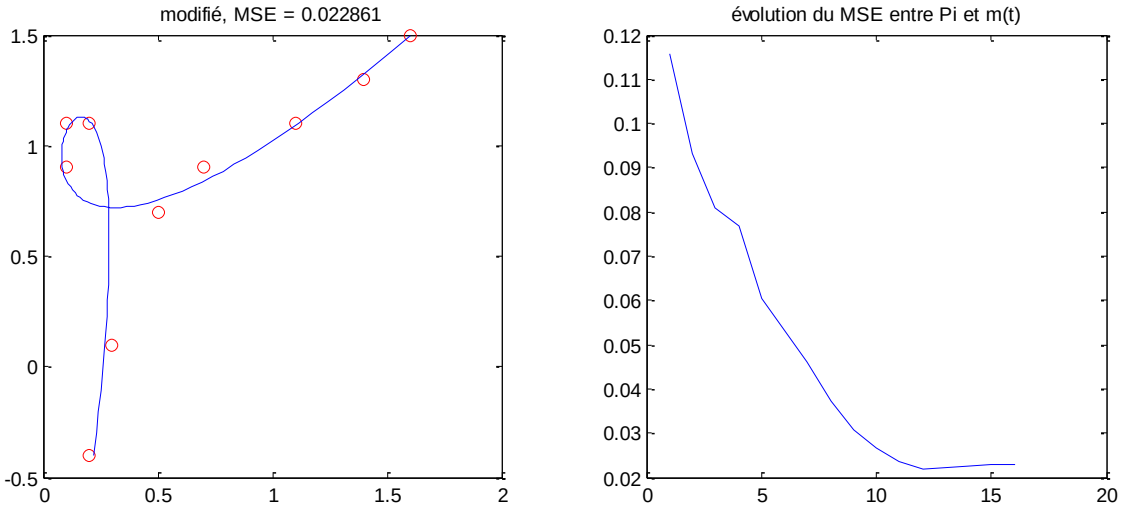

Figure 5:résultat obtenu par l'algorithme modifié et les MSE de chaque itération

```
    6.2933   -6.1225    1.2113    0.2212
   13.5664  -22.1266   10.4614   -0.4025
```

```
t_optimiser =

    0.0000    0.0543    0.2946    0.4224    0.5614    0.7975    0.8574    0.9284    0.9712    0.9998
```

Conclusion :

Pour la quatrième partie, j'ai essayé de programmer la méthode de recuit simulé. Mais à cause de temps, je n'ai pas pu réaliser une méthode méta-heuristique. Donc je ne pourrais pas montrer le résultat obtenu par la méthode méta-heuristique.