04.11.2024

Document Explicatif TP d'optimisation : Levenberg-Marquardt

INTRODUCTION

Idée générale de l'optimisation s

Lorsque l'on veut ajuster les paramètres d'un modèle pour qu'il corresponde à un jeu de données, que ce soit pour faire de la régression ou de la classification, on cherche en fait à minimiser une fonction à plusieurs paramètres d'entrées. Résoudre un tel problème revient à résoudre un problème d'optimisation.

La formulation du problème repose sur le choix d'un jeu de donnée (dataset), d'un type de modèle (polynôme, fraction rationnelle, somme d'exponentielles…). Les paramètres du modèle sont regroupés dans un vecteur X. Ainsi en passant le jeu de donnée dans le modèle et en choisissant tous les x_i du vecteur X tel que l'écart entre $y_{\text{SORTIE DATASET}}$ et $y_{\text{SORTIE MODELE}}$ soit minimum, on ajuste donc logiquement les paramètres du modèle pour qu'ils « s'alignent » avec les données initiales. En clair, on trouve les paramètres « optimaux » d'où le nom d'optimisation.

La fonction à minimiser qui dépend du vecteur X et qui quantifie l'écart $y_{\text{SORTIE DATASET}}$ et $y_{\text{SORTIE MODELE}}$ s'appelle la fonction de coût : on se ramène donc à un problème de mathématiques formel (illustration avec les formules dans « Le cas entièrement Numérique » ci-dessous).

Seulement, on ne peut pas toujours isoler le vecteur X des paramètres du modèle. Les fonctions manipulées sont par ailleurs rarement des fonctions de R ou R^2 vers R. C'est pourquoi la matière optimisation propose différents algorithmes par lesquels on peut trouver des minimums. Dans ce TP on s'intéresse à la méthode de Levenberg-Marquardt.

À noter qu'il n'y pas de méthodes meilleures qu'une autre, simplement des algorithmes plus ou moins adaptés à une situation selon le compromis précision, coût algorithmique, simplicité d'implémentation.

Mais il n'est parfois même pas nécessaire d'utiliser directement des algorithmes numériques. Récapitulons plusieurs cas pour commencer.

Le cas Convexe

On sait que « Toute fonction strictement convexe admet un minimum global, qui est le seul point critique. ». Par ailleurs démontrer qu'une fonction de coût est convexe peut se faire simplement en montrant que sa matrice Hessienne est symétrique définie positive.

Donc typiquement, dès que l'on a une forme quadratique, on sait directement quel est le minimum global (point d'annulation du gradient).

Le cas du modèle Linéaire

Dans le cas où on choisit un système linéaire, c'est-à-dire avec Y = AX, la matrice A contient alors les paramètres du modèle. Puisque c'est un modèle linéaire, le jeu de donnée prends une donnée d'entrée x et une donnée de sortie y. La taille de ces deux vecteurs est liée à la taille (nombre d'échantillon) du dataset.

Dans ce cas, il s'agit de résoudre le système en isolant A puisque les vecteurs X et Y sont connus via le jeu de données. L'idée est réécrire le système sous la forme Y = Z A. Où A = (a11 a12 a21 a22) (vecteur !) et Z = (x1 x2 0 0 ; 0 0 x1 x2 ; x1 x2 0 0 ; 0 0 x1 x2) (cas où M=2).

Dans le cas où on généralisait à un système de dimension M, cela induit simplement d'augmenter le nombre de données du jeu de données.

J'ai montré que la matrice Z pouvait se réécrire alors comme une matrice de taille $M^2 \times M^2$ tel que

$$\text{Pour tout } i,j : (Z_{i,j}) = x_{j-(i-1)\times M} \left(\text{SIMGA[from k=1 to i]} \left(\delta_{j,(i-1)\times M + k}\right)\right.$$

$$\text{Puis en inversant le système, on a : } (a_{i//M,\, i\%M}) = \text{SIMGA[from k=1 to } M^2\text{]} \left(z^{-1}_{i,k}\, y_k\right)$$

On voit donc que même si l'on possède une formule analytique, on devra tout de même inverser la matrice Z numériquement.

La méthode est donc semi-numérique, semi-analytique dans le cas d'un tel modèle.

<u>Le cas entièrement Numérique</u>

Le cas entièrement numérique s'applique dès lors qu'il est impossible de trouver analytiquement la solution, car il n'existe parfois aucune formule explicite pour décrire certaines fonctions.

Dans cet exemple, on peut supposer de que x1,x2,x3,x4 sont des variables du prix d'un appartement (surface, emplacement,...) et y le prix. x1,x2,x3,x4, y constitue le jeu de données avec M échantillons (M exemple d'appartements). On postule que le modèle qui régit le prix d'un appartement serait de la forme suivante :

$$y = a*x1 + b*x2 + c*x3 + d * x4$$

$$\text{Donc } f(a,b,c,d) = \text{NORME\_2}\left(y_{DATA\_SET} - a*x1_{DATA\_SET} + b*x2_{DATA\_SET} + c*x3_{DATA\_SET} + d * x4_{DATA\_SET} \right)$$

La fonction de coût est une fonction de R^4 dans R. Dans le cas où il n'existe pas de simplification analytique, on utilise des méthodes d'optimisation numérique (gradient, Newton, etc.).

EXERCICE 1

On cherche dans cet exercice à trouver le minimum de la fonction coût associée au modèle x3*exp(x1*t) + x4 exp(x2*t) via la méthode de *Levenberg-Marquardt*. Le jeu de données comprends : le vecteur temps t et la donnée de sortie y, tous deux de vecteurs de dimensions M (nombre d'échantillons).

La fonction coût est donc telle que :

$$f(x1,x2,x3,x4) = f(x) = \text{norme}(h) = \text{norme}(y_{data} - x3*\exp(x1*t_{data}) + x4\, \exp(x2*t_{data}))$$

Le vecteur X qui permet d'obtenir le jeu de donnée est xv = [-4 -1 4 -5] (avec ajout d'un bruit blanc gaussien dans ce cas). On se doute donc que tout vecteur x0 (intial) « pris près » de xv atteindra le minimum global (« au bruit blanc près »).

La fonction h est de R^N vers R^M. La fonction f de R^N vers R.

<u>Principe de l'algorithme de Levenberg-Marquardt</u>

Levenberg-Marquardt est un algorithme itératif pour trouverun minimum local d'une fonction non linéaire. A chaque itération on recalcule un nouveau x_optimale qui va se rapprocher de plus en plus du minimum. La convergence est assurée par un critère d'arrêt (la différence entre la nouvelle et l'ancienne valeur de la fonction coût pour la nouvelle et l'ancienne valeur de x_optimale doit être inférieur à un epsilon) et par l'algorithme lui-même. Naturellement, en voulant être plus rigoureux : xk --> x_OPT à partir d'un certain rang.

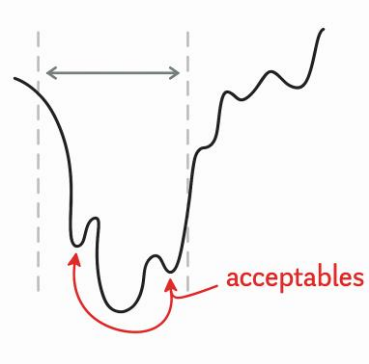
Dans chaque tour de boucle, on calcule la direction à l'itération k grâce à l'équation de Levenberg Marquardt, puis on met à jour des coefficients de cette équation, le x, selon la valeur d'un coefficient gamma (qui change à chaque itération, voir calcul du gamma dans dirLM.m).

Dans chaque itération on calcule également la matrice jacobienne de manière analytique pour éviter tout problème lié à l'utilisation de fonctions matlab inconnues.

Les problèmes de minima locaux & des conditions initiales

La plupart du temps, le problème d'optimisation font intervenir des fonctions de coût multimodale, c'est-à-dire qu'elle possède plusieurs minima locaux en raison de la nature non linéaire du modèle et des exponentielles impliquées.

Cela signifie qu'une condition initiale différente peut donner des valeurs complètement différentes même si l'algorithme fonctionne. Si on se représentait la chose, je peux très bien atteindre le fond d'un lac situé à 2000m d'altitude, pour autant en étant sur un flanc de colline à 1000m, j'ai plus de chance d'atteindre le minimum global. Voici l'idée résumé sur un schéma en 1D :



On peut donc définir des minima locaux acceptables pour lesquels f_OPT va être « minimum » et d'autres pour lesquels le minimum local ne reflètera pas un minimum réel pour notre problème.

Dans l'éventualité où un algorithme local ne suffit pas, on pourrait utiliser les algorithmes génétiques, le Simulated Annealing (méthode probabiliste pour éviter les minimas locaux mais amoindri la précision du résultat) par exemple.

Paramètres de la méthode

Comme montrer ci-dessus, il arrive que l'on puisse, malgré nos précautions, obtenir un minimum local non optimal. Cela peut être dû à une convergence trop lente, non optimale. Il faut donc alors jouer sur les paramètres de la méthode de Levenberg-Marquardt pour l'adapter aux variables déjà définies et imposées (conditions initiales, bruit, jeu de données...). Deux choses sont possibles :

- **Modifier le critère d'arrêt :** on inclure des critères d'arrêt avec de plus grand nombre d'itérations, utiliser un critère d'arrêt basé sur le gradient pour détecter la convergence vers un minimum, utiliser un critère sur les paramètres. J'ai choisi le nombre d'itération car c'était le plus simple d'utilisation.
- **Améliorer la gestion du paramètre mu :** Pour Levenberg-Marquardt, il s'agit de modifier mu0 car la valeur de mu est régie par la routine de gamma. Lorsque mu est grand, on se rapproche d'un algorithme de gradient pour assurer la stabilité loin du minimum ; Lorsque mu est petit, l'algorithme adopte le comportement de Gauss-Newton avec des pas

accélérant la convergence près du minimum. Ainsi, mu0 est un contrôle initial sur la méthode utilisé.

Questions posées

Question 6 –

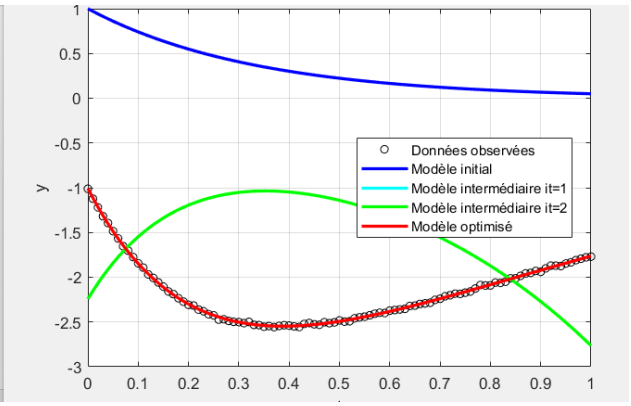
Le modèle présente une *symétrie* qui permet d'obtenir le même modèle en échangeant les paires de paramètres (x1, x3) et (x2, x4). Il en va donc de même pour l'inversion dans la condition initiale xv. Le *non-unicité de la décomposition* (on ne peut pas distinguer les composantes exponentielles) et la *sensibilité aux conditions initiales* sont deux autres caractéristiques de cet exemple.

Ceci implique que le x_opt obtenu peut permuter certains paramètres (x_opt = [x2,x1,x4,x3]) et que l'on peut tomber dans des minima locaux. D'où une indétermination inévitable.

Résultats (question 7)

Pour commencer on obtient :

```
--------- Nombre d'itérations avant convergence ------------
        1010

---------- xv initial -----------
    -4    -1     4    -5

---------- x0 initial -----------
    -3     2     1     0

------------ xOPT ---------------
   -3.9788   -1.0075    4.0297   -5.0372

------------ fOPT ---------------
    0.0098

>>
```



On a choisi une condition initiale proche de celle de xv. Comme expliqué ci-dessus, puisque xv est par définition du problème le minimum global, on voit que l'on arrive à une convergence (fopt quasi nul et xOPT proche de xv minimum global) dans un endroit « proche » au départ.

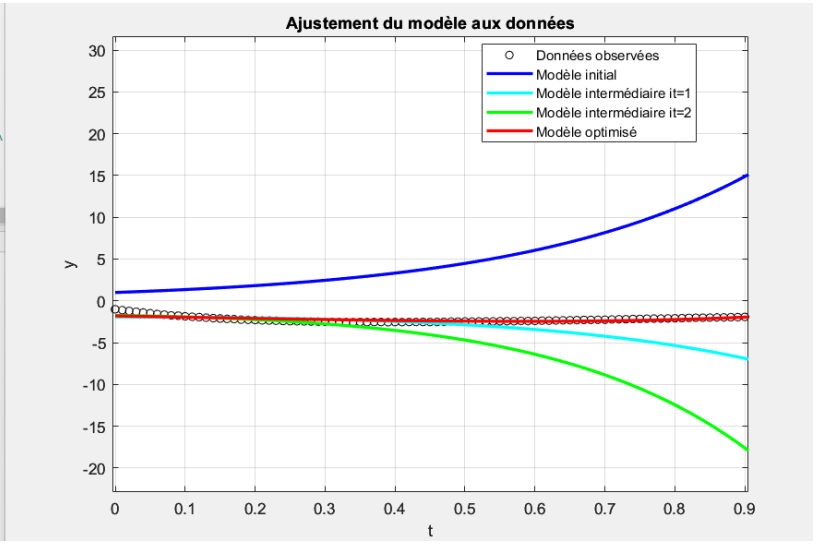
Puisqu'il s'agissait d'un endroit proche, mu prenait une valeur de départ (environ 1.7) relativement faible pour directement orienter la méthode de Levenberg-Marquardt vers un algorithme adapté orienté Gauss-Newton.

J'ai testé pour différents autres points initiaux : Pour x0 = [0 0 3 0] (fopt = 0.0092) ;

Pour x0 = [-1 3 0 1], f_opt n'est pas quasi nul, mais selon les cas on pourrait considérer que le modèle est acceptable (minimum local acceptable). Il s'agit d'un x_opt complètement différent de xv mais qui reproduit des caractéristiques similaires au modèle d'origine (dû aux propriétés discutées à la question 6).

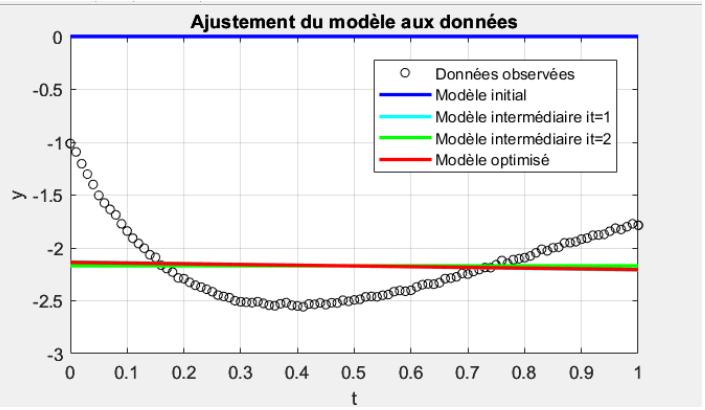
```
% Nombre d'itérations maximum
niter = 10*M;
% Critère d'arrêt
epsilon = 10^(-5);
% Valeur Intiale Coefficient d'amortissement
Jx0 = jacobienne(data,x0);
tau = 10^-3;
mu0 = tau*max(diag(transpose(Jx0)*Jx0)); % /!\
 disp(mu0)
%    mu0 = 100;
% Seuils pour GAMMA
```

```
and Window

--------- xv initial -----------
   -4    -1     4    -5

--------- x0 initial -----------
   -1     3     0     1

---------- xOPT ---------------
   1.6945    1.7115  -91.2429   89.4402

---------- fOPT ---------------
   4.7283
```



Pour x0 = [0 0 0 0] le modèle ne converge même plus de manière acceptable, on a fopt =12.1604 :

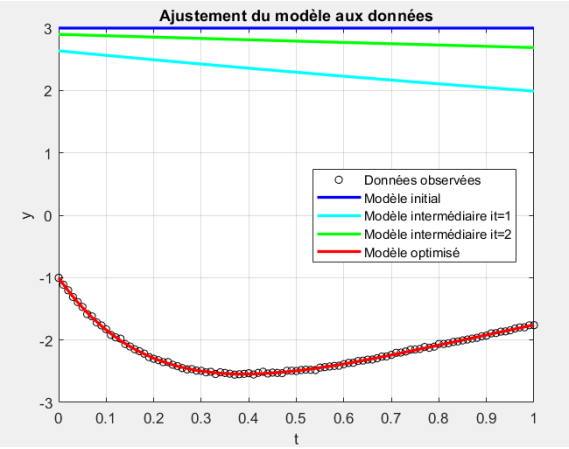
```
Command Window

---------- xv initial -----------
    -4    -1     4    -5

---------- x0 initial -----------
     0     0     0     0

------------ xOPT ---------------
    0.0315    0.0315   -1.0682   -1.0682

------------ fOPT ---------------
   12.1475

fx >>
```



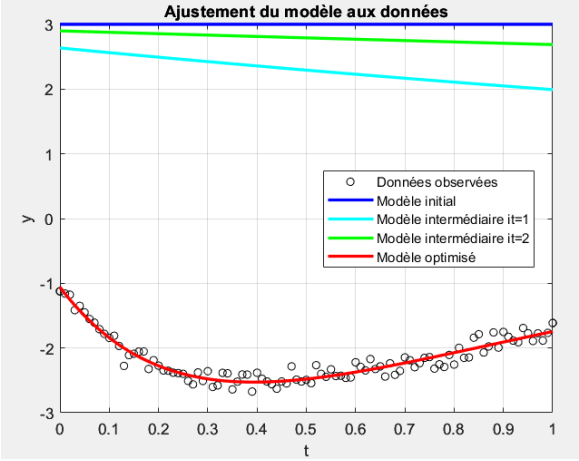
Ici l'algorithme converge, mais vers un minimum local non optimal.

<u>Cas du Bruit</u>

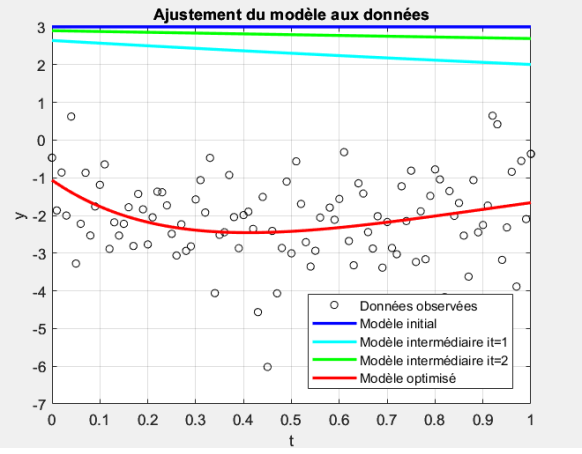
L'ajout du bruit, augmente l'incertitude sur les points (données observés) et impliquent nécessairement une erreur globale plus forte c'est-à-dire une valeur de f plus élevée. Pour autant, une valeur de f élevé peut être trompeuse dans la mesure où si la courbe « fait de son mieux » compte tenu du dataset, alors il s'agirait pourtant du modèle optimal ! Voici (cas x0 = [0 0 3 0]) :



Bruit 0.01, fOPT = 0.0099.



Bruit 0.1, fOPT = 0.9593



Bruit 1, fOPT = 115.6211

Un bruit trop élevé va impliquer des fausses prédictions, c'est-à-dire une prédiction au bruit près. Pour autant comme je l'ai dit, c'est ici le meilleur résultat que l'on pourrait avoir compte tenu du bruit.

Donc le bruit sur les données peut être toléré à jusqu'à un certain niveau.

EXERCICE 2

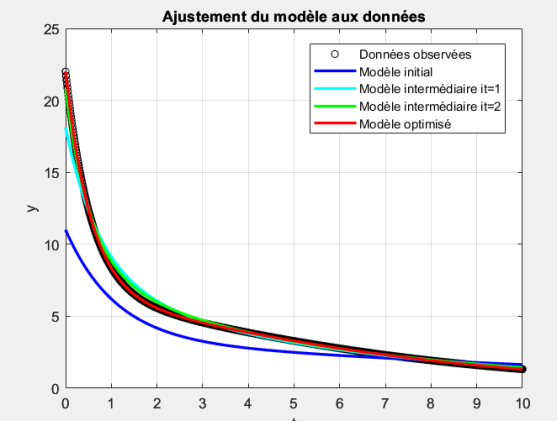
Dans le cadre de cet exercice, on choisit d'augmenter le nombre de variable à 10. Ainsi la fonction f devient une fonction de coût de R^10 vers R. L'augmentation à 10 paramètres a pour effet d'augmenter l'espace de recherche ce qui rend l'optimisation plus complexe. Cela acroît les risques de se retrouver bloquer dans les minima locaux et de converger lentement. C'est pourquoi j'ai choisi d'augmenter le nombre d'itérations.

Le comportement est donc sensiblement le même mais les paramètres/conditions doivent subir plus de contraintes pour que l'algorithme fonctionne :

- **Conditions initiales :** L'augmentation du nombre de paramètres augmente la probabilité de les choisir éloignés du minimum global et favoriser des résultats non optimaux.
- **Bruit :** L'ajout de bruit sur 10 variables augmente davantage l'incertitude qu'avec 4. Ainsi il faut prendre un bruit beaucoup plus faible si l'on veut que ça marche. (Je parle ici du bruit sur les paramètres et non sur les données).
- **Risque de généralisation :** De manière intuitive, on pourrait penser que l'augmentation du nombre de paramètres pourrait provoquer un surajustement du jeu de données d'entrainement et pourrait nuire à la généralisation/prédiction.

Remarque : on s'aperçoit ici de l'utilité de calculer la jacobienne analytiquement puisqu'un calcul d'une jacobienne dans R^10 aurait impliqué des erreurs supplémentaires.

```
>> tp1exo2
---------- xv initial -----------
   -2.0000   -1.5000   -1.0000   -0.5000   -0.2000   10.0000    5.0000    2.0000   -5.0000   10.0000

---------- x0 initial -----------
   -1.0000   -0.7500   -0.5000   -0.2500   -0.1000    5.0000    2.5000    1.0000   -2.5000    5.0000

------------ xOPT ---------------
   -2.0000   -1.0000   -1.5000   -0.5000   -0.2000   10.0000    2.0000    5.0000   -5.0000   10.0000

------------ fOPT ---------------
   5.8617e-28
```



On obtient donc une convergence mais j'ai dû choisir des relations entre les paramètres « cohérents entre eux » c'est-à-dire 0.5*xv.

BONUS : Méthodes du gradient

Voici les résultats obtenus pour x0 = [0 0 3 0] pour les méthodes du gradient.

```
--- Résultats ---
Paramètres réels xv :
    -4    -1     4    -5

Paramètres estimés par descente de gradient à pas fixe :
   -1.5161   -0.1753    0.8112   -2.8189

Nombre d'itérations : 1000, f(x) = 10.1999
Paramètres estimés par descente de gradient avec recherche linéaire :
    0.0297    0.0297   -0.9147   -1.2247

Nombre d'itérations : 252, f(x) = 12.1992
Paramètres estimés par descente de gradient avec pas optimal :
     0     0     3     0

Nombre d'itérations : 1000, f(x) = 2713.403
```

Les méthodes de descente de gradient n'ont pas convergé vers les paramètres réels xv=[−4,−1,4,−5] malgré un nombre important d'itérations.

Les méthodes du gradient usuelles n'ont pas réussi à échapper aux minima locaux près du point x0 = [0 0 3 0] contrairement à l'algorithme de Levenberg-Marquardt (qui y arrive grâce à la combinaison des méthodes de descente et de Gauss-Newton).

On pourrait montrer de la même manière que Gauss-Newton se bloque dans certains minima locaux pour lesquels Levenberg-Marquardt ne se bloque pas.

Conclusion

En conclusion, ce TP nous a permis de montrer l'efficacité et les limites de la méthode de Levenberg-Marquardt.