Projet : phase 3

3 - FITTER UN NUAGE DE POINTS PAR UNE HQ : PROBLEME COMPLET

La phase 2 du projet a permis de comprendre la démarche générale d'identification des paramètres d'une hyper-quadrique passant au mieux par un nuage de points en travaillant sur un cas simplifié : il n'y avait que 2 coefficients à déterminer. La phase 3 va consister à traiter le problème général dans lequel il faut déterminer les coefficients $\{A_k, B_k, C_k, \ \forall k = 1, N_h\}$, pour une valeur de $N_h \geq 3$ choisie par l'utilisateur. Dans le cadre de ce projet, afin d'alléger la mise en œuvre du fit, on suppose que $\forall k = 1, N_h, \gamma_k = 4$.

Pour un traitement efficace du cas général, plusieurs adaptations sont nécessaires, d'une part sur la manière de poser le problème lui-même, d'autre part sur la manière de le résoudre. Ces adaptations sont présentées ci-dessous.

3.1 Amélioration du critère d'erreur de fitting :

Rappelons les données du problème. Les données à fitter sont un ensemble de points $\{(x_i, y_i)\}_{i=1,N}$. L'hyperquadrique est définie par l'équation $\varphi(x, y, \lambda) = 1$, où φ est définie par :

$$\varphi(x, y, \lambda) = \sum_{k=1}^{Nh} |A_k.x + B_k.y + C_k|^4$$

L'objectif est de déterminer le jeu de paramètres $\lambda = \{A_k, B_k, C_k\}_{1 \leq k \leq N_h}$, tels que l'hyperquadrique passe au mieux par le nuage de points. Pour ce faire, il faut définir un critère d'erreur de fit entre le nuage de points et une hyperquadrique de paramètres donnés, puis déterminer les paramètres qui minimisent ce critère.

Dans la phase 2 du projet, nous avons mis en œuvre – avec succès – un critère simple et intuitif :

$$J(\lambda) = \sum_{i=1}^{N} [\varphi(x_i, y_i, \lambda) - 1]^2 .$$

Nous allons voir que ce critère a besoin d'être affiné, d'une part pour améliorer son comportement numérique, d'autre part pour une évaluation plus juste de la distance entre un point (x, y) et une hyperquadrique donnée.

Amélioration pour un meilleur comportement numérique :

On note que le critère $J(\lambda)$ défini précedemment conduit à minimiser un polynôme de degré 8, dont les variations très rapides en λ sont sources d'instabilité numérique. En effet, les méthodes numériques itératives telles que la méthode du gradient ou la méthode de Newton exploitent des informations ponctuelles (gradient $\nabla J(\lambda_n)$ et matrice hessienne $H_J(\lambda_n)$ à l'itération n) pour déterminer un nouveau point λ_{n+1} tel que $J(\lambda_{n+1}) \leq J(\lambda_n)$. Pour que ces méthodes convergent de manière satisfaisante, il faut rechercher le nouveau point λ_{n+1} dans une région où les informations $\nabla J(\lambda_n)$ et $H_J(\lambda_n)$ n'ont « pas trop » changé. Pour la méthode du gradient, cela conduit à utiliser un pas très petit, et donc à un grand nombre d'itérations. Pour la méthode de Newton, cela nécessite de trouver un point initial très près de la solution. Ces inconvénients n'ont pas été trop génants pour le problème à 2 variables traité dans la phase 2. En revanche, les instabilités numériques sont rédhibitoires dans le cas général.

Une façon simple de pallier le problème de degré élevé du critère « naïf » consiste à modifier celui-ci en lui appliquant une fonction monotone qui ne modifiera pas la relation d'ordre entre différentes solutions mais atténuera les variations.

A cet effet, on introduit la fonction « inside-ouside » $F_{io}(x, y, \lambda)$:

$$F_{io}(x, y, \lambda) = \varphi(x, y, \lambda)^{\frac{1}{4}} = \left[\sum_{k=1}^{Nh} |A_k.x + B_k.y + C_k|^4\right]^{\frac{1}{4}}$$

puis on définit l'erreur de fit EoF_1 par :

$$EoF_1(\lambda) = \frac{1}{2}\sum_{i=1}^{N}\left[1 - F_{io\_i}(\lambda)\right]^2$$

où $F_{io\_i}(\lambda) = \varphi_i(\lambda)^{\frac{1}{4}}$ et $\varphi_i(\lambda) = \sum_{k=1}^{Nh}|A_k.x_i + B_k.y_i + C_k|^4$.

Le comportement asymptotique à l'infini de $F_{io\_i}(\lambda)$ est linéaire, et donc celui de $EoF_1(\lambda)$ est quadratique. De ce fait, les variations spatiales de $EoF_1(\lambda)$ sont beaucoup plus douces que celles du critère initial et les méthodes de minimisation convergeront plus facilement.

Amélioration pour une meilleure évaluation de la distance :

La quantité $|1 - F_{io}(x, y, \lambda)|$ est un critère de distance biaisé. En effet, à λ donné, deux points (x_1, y_1) et (x_2, y_2) tels que $|1 - F_{io}(x_1, y_1, \lambda)| = |1 - F_{io}(x_2, y_2, \lambda)|$ ne sont en général pas à la même distance géométrique de l'hyperquadrique. La Figure 1 illustre le propos : elle représente quelques isovaleurs de la fonction $|1 - F_{io}(x, y, \lambda)|$ utilisée en première approche pour évaluer la distance entre un point de coordonnées (x, y) et une hyperquadrique de paramètres λ. L'hyperquadrique correspond à l'isovaleur $|1 - F_{io}(x, y, \lambda)| = 0$, représentée en noir. Les points P_1 et P_2 sont tous deux situés sur la même isovaleur, correspondant à $|1 - F_{io}(x, y, \lambda)| = 0.32$, mais il est évident qu'ils ne sont pas à la même distance de l'hyperquadrique.

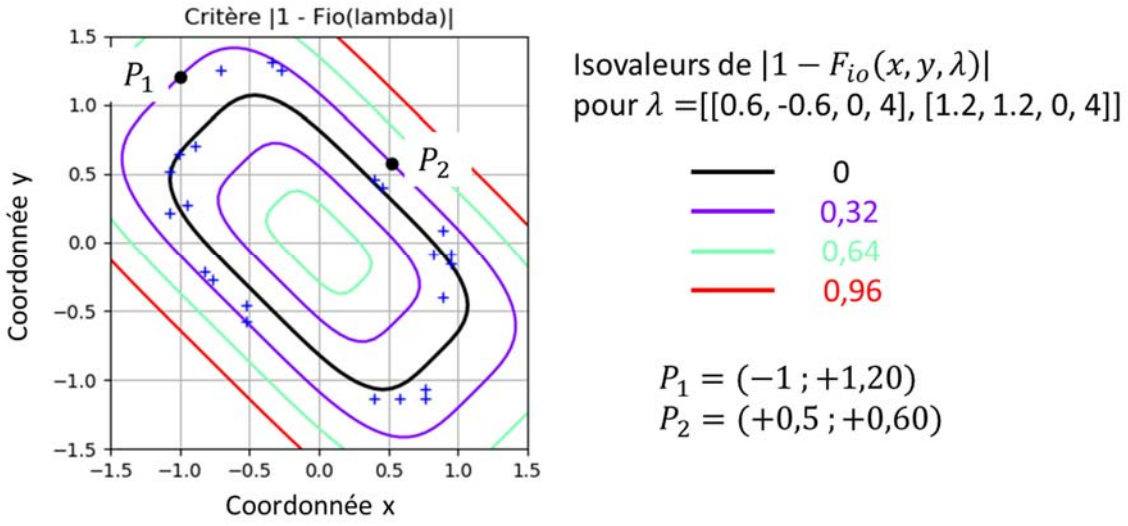

Figure 1 : Isovaleurs du critère $|1 - F_{io}(x, y, \lambda)|$

Il est possible d'utiliser la fonction $1 - F_{io}(x, y, \lambda)$ pour évaluer plus rigoureusement la distance entre un point de coordonnées (x_i, y_i) et l'hyperquadrique de paramètres λ. Pour des points P_i proches de HQ, la distance d_i peut être mesurée le long de la normale à l'isovaleur passant par P_i, comme cela est schématisé sur la Figure 2. Faisons un développement limité de $1 - F_{io}(x, y, \lambda)$ au voisinage de P_i :

$$[1 - F_{io}(x, y, \lambda)] = [1 - F_{io}(x_i, y_i, \lambda)] - \nabla F_{io}(x_i, y_i, \lambda)^T.\begin{pmatrix} x - x_i \\ y - y_i \end{pmatrix}$$

La distance d_i est la norme du vecteur $\overrightarrow{P_iP}$ $\left(d_i = \left\|\overrightarrow{P_iP}\right\|\right)$, où P est à la fois sur la normale et sur l'hyperquadrique.

Le long de la normale $\nabla F_{io}(x_i, y_i, \lambda)$ et $\overrightarrow{P_iP}$ sont colinéaires, donc :

$$[1 - F_{io}(x_P, y_P, \lambda)] = [1 - F_{io}(x_i, y_i, \lambda)] \pm \|\nabla F_{io}(x_i, y_i, \lambda)\|.\left\|\overrightarrow{P_iP}\right\|$$

Le signe $\pm$ dépend de la position de P_i par rapport à HQ (intérieur ou extérieur).

Le point P est sur l'hyperquadrique, donc : $[1 - F_{io}(x_P, y_P, \lambda)] = 0$.

On en déduit que $d_i = \frac{|1-F_{io}(x_i,y_i,\lambda)|}{\|\nabla F_{io}(x_i,y_i,\lambda)\|}$.

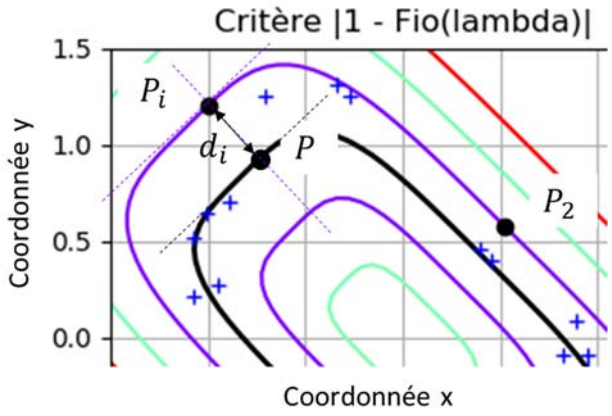

Figure 2 : Distance entre le point P_i et l'hyperquadrique

On définit ainsi un nouveau critère d'erreur de fit EoF_2 par :

$$EoF_2(\lambda) = \frac{1}{2}\sum_{i=1}^{N} {d_i}^2 = \frac{1}{2}\sum_{i=1}^{N} \left(\frac{1-F_{io}(x_i,y_i,\lambda)}{\|\nabla F_{io}(x_i,y_i,\lambda)\|}\right)^2$$

Remarque : La mise en œuvre du critère EoF_2 est un peu plus complexe que celle du critère EoF_1. Elle est justifiée dans le cas où la forme à fitter est très allongée, ou s'il y a une grande exigence de précision sur le fit. Dans le cadre de ce projet, on pourra se satisfaire du critère EoF_1.

3.2 Limitation du domaine de recherche des paramètres :

Les fonctions EoF_1 et EoF_2 possèdent en pratique un nombre infini de minima où l'erreur EoF nulle, sans pour autant que ces minima correspondent à la solution recherchée. En effet, prenons le cas particulier où $A_k = B_k = 0$.

Alors : $1 - F_{io}(x_i, y_i, \lambda) = 1 - \left[\sum_{k=1}^{Nh} {C_k}^4\right]^{\frac{1}{4}}$

La condition $\sum_{k=1}^{Nh} {C_k}^4 = 1$, réalisée par une infinité de combinaisons de coefficients C_k, annule $1 - F_{io}(x_i, y_i, \lambda)$ et donc l'erreur de fit. De manière évidente, ces jeux de paramètres ne correspondent pas à une solution du problème de fit, mais ce sont des solutions du problème de minimisation tel que nous venons de le poser.

Plus généralement, la fonction à minimiser possède de nombreux minima locaux dans des régions de l'espace des paramètres sans intérêt parce que ne correspondant pas à la forme recherchée. Pour éviter que l'algorithme de minimisation converge vers un point d'une zone sans intérêt, il est nécessaire de limiter l'espace de recherche à la région qui contient des valeurs « raisonnables » des paramètres. Pour cela, on peut exploiter les droites englobantes : en effet, le nuage de points est situés entre les droites englobantes, ce qui conduit à une condition de distance minimale entre droites englobantes parallèles.

soit S_k, la distance entre les deux droites englobantes associées au terme de rang k de l'hyperquadrique.

$$S_k = \frac{2}{\sqrt{{A_k}^2 + {B_k}^2}}$$

Soit S_{min} et S_{max}, les diamètres intérieur et extérieur d'une couronne qui contient le nuage de points (Figure 3).

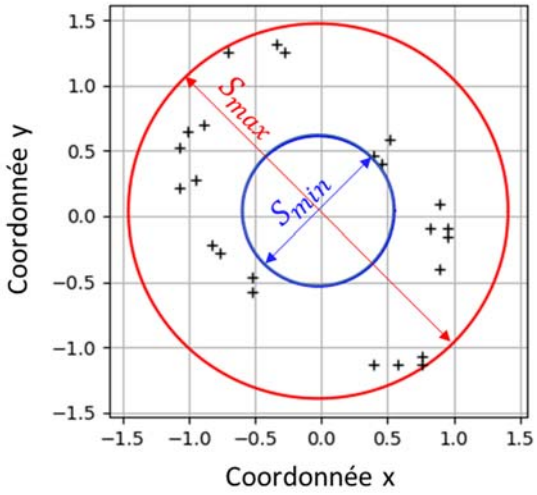

Figure 3 : définition des diamètres S_{min} et S_{max}

Le fait que chaque paire de droites englobantes encadre le nuage de points se traduit par la condition que toutes les valeurs de S_k sont supérieures à S_{min}. Par ailleurs, les droites englobantes ne doivent pas être trop éloignées du nuage de points. Donc toutes les valeurs de S_k doivent être inférieures à une certaine distance arbitrairement exprimée à partir de S_{max}.

Ces conditions se traduisent par :

$$\forall k = 1, N_h \quad k_2.S_{min} \leq S_k \leq k_1.S_{max}, \quad \text{où } k_1 \text{ et } k_2 \text{ sont des coefficients empiriques}$$

$$\Longrightarrow \quad k_2.S_{min} \leq \frac{2}{\sqrt{{A_k}^2+{B_k}^2}} \leq k_1.S_{max},$$

$$\Longrightarrow \quad \mu_1 \leq {A_k}^2 + {B_k}^2 \leq \mu_2, \qquad \text{avec } \mu_1 = \left(\frac{2}{k_1.S_{max}}\right)^2 \text{ et } \mu_2 = \left(\frac{2}{k_2.S_{min}}\right)^2$$

Cette inégalité double se traduit par deux inégalités simples :

$$\mu_1 - \left({A_k}^2 + {B_k}^2\right) \leq 0 \quad \text{et} \quad \left({A_k}^2 + {B_k}^2\right) - \mu_2 \leq 0$$

Ces contraintes sont prises en compte en ajoutant des termes de pénalité dans le critère à minimiser (voir TP3).

$$EoF(\lambda) = \frac{1}{2}\sum_{i=1}^{N}\left(\frac{1-F_{io}(x_i,y_i,\lambda)}{\|\nabla F_{io}(x_i,y_i,\lambda)\|}\right)^2 + \nu.\sum_{k=1}^{N_h} P_k(\lambda)$$

$$\text{Où } P_k(\lambda) = \left(\max\left(0, \mu_1 - \left({A_k}^2 + {B_k}^2\right)\right)\right)^2 + \left(\max\left(0, \left({A_k}^2 + {B_k}^2\right) - \mu_2\right)\right)^2$$

Dans le cadre de ce projet, vous utiliserez les valeurs suivantes : $k_1 = k_2 = 10$ et $\nu = 10^8$.

3.3 Algorithme de minimisation :

Schéma général :

Le fait d'avoir introduit une distance géométrique complique l'expression de la fonction à minimiser, avec l'apparition du terme $\|\nabla F_{io}(x_i, y_i, \lambda)\|$ au dénominateur. Le calcul exact de ∇EoF n'a rien d'évident… Par ailleurs, nous avons noté que le terme $\|\nabla F_{io}(x_i, y_i, \lambda)\|$ est un terme correctif qui ne joue pas un rôle fondamental dans la recherche de la solution. Cette constatation justifie l'algorithme suivant :

1. Choisir un jeu de paramètres λ_0 acceptable (dans le cadre de ce projet, ce jeu de paramètre sera fourni)
2. Calculer les poids $w_i = \frac{1}{\|\nabla F_{io}(x_i, y_i, \lambda)\|^2}$
3. Minimiser $J(\lambda) = \frac{1}{2}\sum_{i=1}^{N} w_i\left(1 - F_{io}(x_i, y_i, \lambda)\right)^2 + \nu.\sum_{k=1}^{N_h} P_k(\lambda)$ par la méthode de Levenberg-Marquardt
4. Tant que la qualité de la solution, évaluée à l'aide du critère $EoF_2(\lambda)$, n'est pas satisfaisante, aller en 2.

Méthode de Levenberg-Marquardt :

La méthode de Levenberg-Marquardt est une méthode de type « trust-region algorithm ». Il s'agit d'une classe de méthodes qui définissent à chaque itération une zone de recherche « de confiance », c'est-à-dire au sein de laquelle les dérivées ne varient « pas trop ». A chaque itération, l'algorithme ajuste la zone de confiance : il la diminue s'il détecte que la solution tend à se dégrader quand on considère une trop grande zone de travail, ou au contraire il l'étend s'il détecte que la taille de la zone de confiance est trop petite pour permettre une convergence rapide.

Par ailleurs, l'algorithme de Levenberg-Marquardt mixe le gradient et la matrice hessienne avec une pondération qui donne un comportement se rapprochant de la méthode de Newton au voisinage de la solution. Le principe est de passer automatiquement d'un comportement de type méthode de gradient loin de la solution à un comportement de type méthode de Newton au voisinage de la solution.

L'algorithme est détaillé ci-dessous. ∇J et H_J désignent le gradient et la matrice hessienne de J. I désigne la matrice identité.

Algorithme :

- Choix des paramètres de l'algorithme : $\lambda_0, \varepsilon, n_{max}$
- Initialisation : $\lambda_{n-1} \leftarrow \lambda_0$, $d\lambda \leftarrow 1, n \leftarrow 0, \beta \leftarrow 0{,}01$
- Tant que $d\lambda > \varepsilon$ et $n < n_{max}$:
  - Calculer $\nabla J_{n-1} = \nabla J(\lambda_{n-1})$
  - Calculer $HJ_{n-1} = H_J(\lambda_{n-1})$
  - (*) $\Delta\lambda \leftarrow$ solution du système $(HJ_{n-1} + \beta.I).\Delta\lambda = -\nabla J_{n-1}$
  - Calculer $J(\lambda_{n-1} + \Delta\lambda)$
  - Tant que $J(\lambda_{n-1} + \Delta\lambda) \geq J(\lambda_{n-1}), \beta \leftarrow 10 \times \beta$ et aller en (*)
  - $\beta \leftarrow 0{,}1 \times \beta$
  - $\lambda_n \leftarrow \lambda_{n-1} + \Delta\lambda$
  - $d\lambda \leftarrow \|\Delta\lambda\|$
  - $n \leftarrow n + 1$
- $converge \leftarrow (d\lambda \leq \varepsilon)$

Le paramètre β joue un double rôle : il gère la taille de la zone de confiance et la contribution relative de la matrice hessienne dans la recherche de la solution.

Pour une très grande valeur de β, résoudre le système $(HJ_{n-1} + \beta.I).\Delta\lambda = -\nabla J_{n-1}$ donne une solution proche de $\Delta\lambda = -\frac{1}{\beta}.\nabla J_{n-1}$. La méthode fonctionne comme une descente de gradient à pas adaptatif.

Pour une valeur très petite de β, la solution du système $(HJ_{n-1} + \beta.I).\Delta\lambda = -\nabla J_{n-1}$ est proche de la solution de la méthode de Newton $HJ_{n-1}.\Delta\lambda = -\nabla J_{n-1}$.

Lors des premières itérations, on peut s'attendre à ce que la valeur de β augmente (prédominance de la méthode du gradient). Dans un deuxième temps, lorsque λ_n se rapproche de la solution la valeur de β diminue (prédominance de la méthode de Newton).

Approximation de la matrice hessienne :

Un dernier point porte sur le calcul de la matrice hessienne. La forme particulière du critère permet une approximation efficace dont nous allons présenter le principe.

Soit $f(x) = \frac{1}{2}\sum_{i=1}^{N} r_i(x)^2$, une fonction de la variable $x = \left(x_j\right)_{j=1,m} = \begin{pmatrix} x_1 \\ \dots \\ x_m \end{pmatrix}$.

Gradient : $$\nabla f(x) = \left(\sum_{i=1}^{N} \frac{\partial r_i}{\partial x_j}(x).r_i(x)\right)_{j=1,m} = \begin{pmatrix} \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_1}(x).r_i(x) \\ \dots \\ \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_m}(x).r_i(x) \end{pmatrix}$$

Matrice hessienne : $$\mathrm{H}_f(x) = \left(\sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial x_j}(x).\frac{\partial r_i}{\partial x_k}(x) + \frac{\partial^2 r_i}{\partial x_j \partial x_k}(x).r_i(x)^2\right)\right)_{\substack{j=1,m \\ k=1,m}}$$

$$\mathrm{H}_f(x) = \begin{pmatrix} \sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial x_1}(x)^2 + \frac{\partial^2 r_i}{\partial x_1^2}(x).r_i(x)^2\right) & \dots & \sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial x_1}(x).\frac{\partial r_i}{\partial x_m}(x) + \frac{\partial^2 r_i}{\partial x_1 \partial x_m}(x).r_i(x)^2\right) \\ \dots & \dots & \dots \\ \sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial x_m}(x).\frac{\partial r_i}{\partial x_1}(x) + \frac{\partial^2 r_i}{\partial x_m \partial x_1}(x).r_i(x)^2\right) & \dots & \sum_{i=1}^{N}\left(\frac{\partial r_i}{\partial x_m}(x)^2 + \frac{\partial^2 r_i}{\partial x_m^2}(x).r_i(x)^2\right) \end{pmatrix}$$

Approximation : $$\mathrm{H}_f(x) \approx \left(\sum_{i=1}^{N} \frac{\partial r_i}{\partial x_j}(x).\frac{\partial r_i}{\partial x_k}(x)\right)_{\substack{j=1,m \\ k=1,m}}$$

$$\mathrm{H}_f(x) \approx \begin{pmatrix} \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_1}(x)^2 & \dots & \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_1}(x).\frac{\partial r_i}{\partial x_m}(x) \\ \dots & \dots & \dots \\ \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_m}(x).\frac{\partial r_i}{\partial x_1}(x) & \dots & \sum_{i=1}^{N} \frac{\partial r_i}{\partial x_m}(x)^2 \end{pmatrix}$$

Cette approximation est valide dans les conditions suivantes : si $\frac{\partial^2 r_i}{\partial x_j \partial x_k}(x)$ ou si $r_i(x)^2$ sont « petits ». La première condition correspond à des fonctions $r_i(x)$ linéaires, ce qui correspond au comportement asymptotique de la distance définie en 3.1. La deuxième condition tend à devenir vraie au voisinage du point x^* qui minimise $f(x)$ et donc chacun des termes $r_i(x)$ de la somme.

Cette approximation simplifie grandement l'évaluation de la matrice hessienne de $\frac{1}{2}\sum_{i=1}^{N} w_i\left(1 - F_{io}(x_i, y_i, \lambda)\right)^2$.

En revanche, on n'échappe pas au calcul de la matrice hessienne du terme de pénalité $\sum_{k=1}^{N_h} P_k(\lambda)$.

Calcul des dérivées de $P_k(\lambda)$:

Dérivées d'ordre 1 :

$$\frac{\partial P_k}{\partial A_k}(\lambda) = 4A_k.\left[-\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right) + \max(0, (A_k^2 + B_k^2) - \mu_2)\right]$$

$$\frac{\partial P_k}{\partial B_k}(\lambda) = 4B_k.\left[-\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right) + \max(0, (A_k^2 + B_k^2) - \mu_2)\right]$$

Toutes les autres dérivées partielles sont nulles.

Dérivées d'ordre 2 :

$$\frac{\partial^2 P_k}{\partial A_k^2}(\lambda) = 4\left[-\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right) + \max(0, (A_k^2 + B_k^2) - \mu_2)\right] \ldots$$

$$+8A_k^2\left[sign\left(\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right)\right) + sign(\max(0, (A_k^2 + B_k^2) - \mu_2))\right]$$

$$\frac{\partial^2 P_k}{\partial B_k^2}(\lambda) = 4\left[-\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right) + \max(0, (A_k^2 + B_k^2) - \mu_2)\right] \ldots$$

$$+8B_k^2\left[sign\left(\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right)\right) + sign(\max(0, (A_k^2 + B_k^2) - \mu_2))\right]$$

$$\frac{\partial^2 P_k}{\partial A_k \partial B_k}(\lambda) = \frac{\partial^2 P_k}{\partial B_k \partial A_k}(\lambda) = +8A_k B_k\left[sign\left(\max\left(0, \mu_1 - (A_k^2 + B_k^2)\right)\right) + sign(\max(0, (A_k^2 + B_k^2) - \mu_2))\right]$$

Toutes les autres dérivées partielles sont nulles.

3.4 Travail demandé

Implémenter l'algorithme de fit décrit dans ce document. Outre quelques fichiers de données, une fonction python qui détermine le point de départ λ_0 de l'algorithme est également fournie.

L'hyperquadrique ainsi obtenue sera alors superposée au nuage de points pour vérifier le résultat.

Le test des algorithmes devra être fait très progressivement, avec quelques itérations de calcul pour commencer. Une fois l'algorithme mis au point sur quelques itérations, les méthodes seront appliquées avec un critère d'arrêt $\varepsilon = 10^{-6}$.

Documents à rendre :

- Présentation de la mise en œuvre des algorithmes de minimisation.

- Code python (fichiers .py)

- Résultats commentés, sous forme d'un notebook ou d'un rapport classique, à votre convenance.

Bibliographie :

[1]S. Kumar et D. Goldgof, « A robust technique for the estimation of the deformable hyperquadrics from images », in Proceedings of 12th International Conference on Pattern Recognition, Jerusalem, Israel, 1994, vol. 1, p. 74-78, doi: 10.1109/ICPR.1994.576229.

[2]S. Kumar, Song Han, D. Goldgof, et K. Bowyer, « On recovering hyperquadrics from range data », IEEE Trans. Pattern Anal. Machine Intell., vol. 17, n° 11, p. 1079-1083, nov. 1995, doi: 10.1109/34.473234.