Polytech

Rapport de calcul numérique : Interpolation de points par l'utilisation de splines lissantes

Gaoussou SISSOKO
Clément RICHEFORT

Tuteurs
M. BOULIER FRANCOIS
M. LEMAIRE FRANCOIS

23 décembre 2019

Table des matières

1 Introduction

Le but de ce projet est d'interpoler $n+1$ points à l'aide de splines lissantes. Une spline lissante se définit par l'équation suivante :

$$\forall i \in [\![0, n]\!]\ ,\ f_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3\ ,\ (x_i \leq x \leq x_{i+1}) \tag{1}$$

La première partie du projet consistait donc à trouver les vecteurs a, b, c, et d en fixant un certain paramètre p dont l'utilité est expliqué dans ce rapport.

La suite consiste à optimiser le coefficient p, et cela en essayant plusieurs méthodes, comme en fixant un paramètre S ou encore en fixant un certain degré de liberté.

Enfin, ces méthodes seront appliquées à un plus gros jeu de données : la production de poisson par pays de 1960 à nos jours.

2 Calcul des vecteurs de coefficients a, b, c, et d

Cette partie est consacrée au calcul des vecteurs a, b, c, et d.

Les formules de cette partie appellent les matrices Q $(n{+}1)\times(n{-}1)$ et T $(n{+}1)\times(n{-}1)$ ainsi que les vecteurs h et g de taille n. Ceux-ci sont définis dans le polycopié de cours [2] et l'objectif n'est pas de le paraphraser ici.

Ces vecteurs se définissent comme suit :

$$(Q^T\Sigma^2Q + pT)c = pQ^Ty \tag{2}$$

Or $(Q^T\Sigma^2Q + pT)$ est une matrice symétrique définie positive. Grâce à l'algorithme de Cholesky, il est possible de réécrire l'équation sous la forme :

$$LL^Tc = pQ^Ty \tag{3}$$

Descente sur :

$$L\omega = pQ^Ty \tag{4}$$

Puis remontée sur :

$$L^Tc = \omega \tag{5}$$

Il est ensuite possible de résoudre a par une équation en fonction de c :

$$a = y - \frac{1}{p}\Sigma^2Qc \tag{6}$$

Notons dés à présent que les dimensions implicites aux équations (6) et (2) des vecteurs a et c sont respectivement $(n+1)$ et $(n-1)$. Le vecteur c se vera attribuer deux 0 à chaques extrémités pour la suite et aura donc comme dimension $(n+1)$.

Le vecteur d de taille n s'obtient grâce à la formule :

$$\forall i \in [\![\, 0 \;,\; n-1 \,]\!] \;,\; d_i = \frac{c_{i+1} - c_i}{3 \times h_i} \tag{7}$$

Et enfin le vecteur b de taille n également :

$$\forall i \in [\![\, 0 \;,\; n-1 \,]\!] \;,\; b_i = \frac{a_{i+1} - a_i}{h_i} - c_i \times h_i - d_i \times h_i^2 \tag{8}$$

L'algorithme du calcul des splines en Fortran peut-être retrouvé en Annexe 7.1.

Voici donc quelques sorties écrans des splines calculées sur un exemple très basique, grâce aux fonctions *plot* de la librairie *Python matplotlib* :

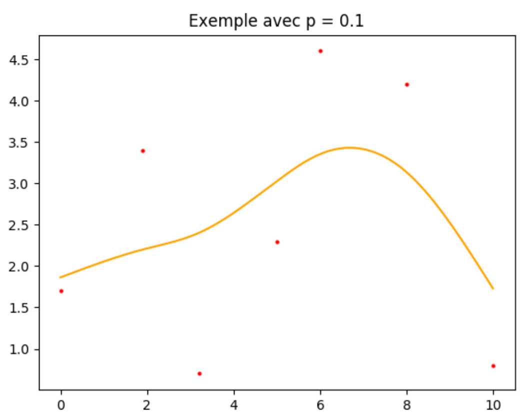

FIGURE 1 – *p = 0.1*

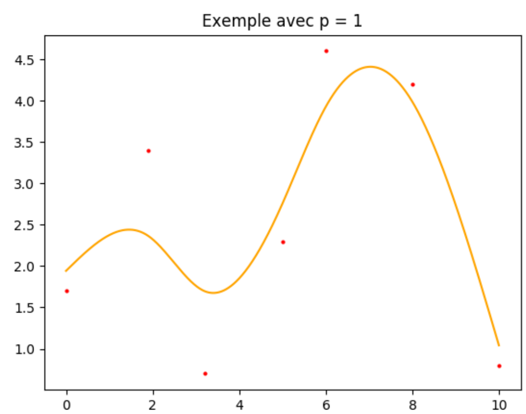

FIGURE 2 – *p = 1*

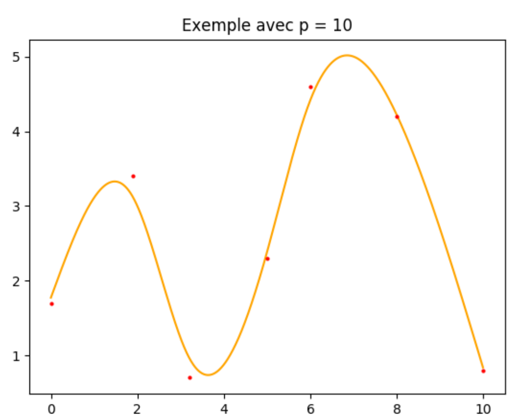

FIGURE 3 – *p = 10*

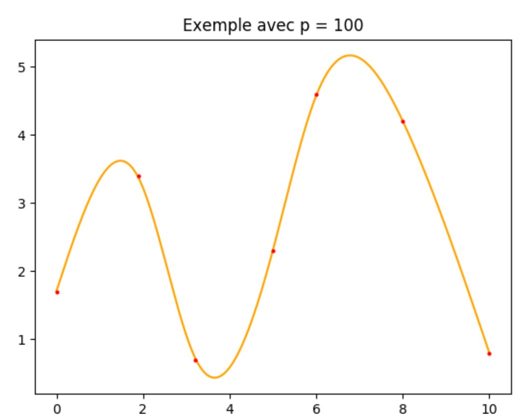

FIGURE 4 – *p = 100*

3 Qu'est ce qu'un multiplicateur de lagrange ?

3.1 Explication théorique

La méthode des multiplicateurs de Lagrange permet de trouver les extremums ou encore les points stationnaires d'une fonction dérivable d'une ou plusieurs variables sous certaines contraintes.

— Principe :
On cherche à trouver l'extremum d'une fonction ϕ de n variables à valeurs dans les nombres réels, ou encore d'un espace euclidien de dimension n, parmi les points respectant une contrainte, de type $\psi(x) = 0$ où ψ est une fonction du même ensemble de départ que ϕ .La fonction ψ est à valeurs dans un espace euclidien de dimension m. Elle peut encore être vue comme m fonctions à valeurs réelles, décrivant m contraintes. Les fonctions ϕ et ψ doivent être dérivables et leurs dérivées doivent être continues ; elles doivent donc être de classe C^1.
On considère λ un vecteur pris dans l'ensemble d'arrivée de ψ. La fonction L , appelé le lagrangien, est définie par :

$$L(x, \lambda) = \phi(x) + \lambda\psi(x).$$

Si x_0 est une solution recherchée, on montre qu'il existe un vecteur λ_0 tel que la fonction L admet une dérivée qui s'annule au point (x_0, λ_0). Les coordonnées du vecteur λ_0 sont appelées multiplicateurs de Lagrange.
Cette technique permet de passer d'une question d'optimisation sous contrainte à une optimisation sans contrainte, celle de la fonction L, dans un espace de dimension n + m.

3.2 Exemple

En économie le multiplicateur de Lagrange est très souvent utilisé dans la recherche des points stationnaires d 'une fonction de profit ou d'utilité.
Nous prenons l'exemple suivant qui consiste à maximiser la fonction d'utilité f d'un consommateur dépendant de la quantité consommée du bien x et de la quantité consommée du bien y, sous la contrainte budgétaire h du consommateur qui est de 24 euros (le bien x coûtant 3 euros et le bien y 4 euros) :

$$f(x, y) = \sqrt{x} * \sqrt{y}$$

$$h(x, y) = 3x + 4y - 24 = 0.$$

Pour résoudre cela, il faut donc dans un premier temps poser la fonction du Lagrangien $L(x, y,\ \lambda) = f(x, y) - \lambda * h(x, y)$. Nous obtiendrons donc $L(x, y, \lambda) = \sqrt{xy} - \lambda[24 - 3x - 4y]$. En posant donc un Lagrangien à la place d'une fonction à optimiser et d'une contrainte, on se retrouve alors dans un problème de recherche de points stationnaires d'une fonction à 3 variables sans contrainte.

Et pour trouver les valeurs optimales $x^*,\ y^*\ et\ \lambda^*$ permettant de résoudre notre problème, il faut donc trouver les valeurs annulant chacune des dérivés partielles de premier ordre, c'est à dire un système tel que :

$$\begin{cases} \frac{\partial}{\partial x} L(x^*, y^*, \lambda^*) = f'_x(x^*, y^*) + \lambda^* h'_x(x^*, y^*) = 0 \\ \\ \frac{\partial}{\partial y} L(x^*, y^*, \lambda^*) = f'_y(x^*, y^*) + \lambda^* h'_y(x^*, y^*) = 0 \\ \\ \frac{\partial}{\partial \lambda^*} L(x^*, y^*, \lambda^*) = h(x^*, y^*) = 0 \end{cases}$$

Dans notre exemple, cela nous donne donc le système suivant :

$$\begin{cases} L'_x = \frac{1}{2\sqrt{x}}\sqrt{y} + 3\lambda = 0 \Longrightarrow \lambda = \frac{-1}{6\sqrt{x}}\sqrt{y} \\ \\ L'_y = \frac{1}{2\sqrt{y}}\sqrt{x} + 4\lambda = 0 \Longrightarrow \lambda = \frac{-1}{8\sqrt{y}}\sqrt{x} \\ \\ L'_\lambda = 3x + 4y - 24 = 0 \Longrightarrow 3x + 4y = 24 \end{cases}$$

$$\begin{cases} \lambda = \frac{-1}{6\sqrt{x}}\sqrt{y} = \frac{-1}{8\sqrt{y}}\sqrt{x} \Rightarrow y = \frac{6}{8}x \\ \\ 3x + 4y = 24 \Rightarrow x = 4,\ \ y = 3 \end{cases}$$

Donc pour maximiser son bien-être (sa fonction d'utilité), le consommateur doit donc consommer 4 unités du bien X, et 3 unités du bien Y

4 Optimisation du paramètre p

Toute cette partie est consacrée à l'optimisation du paramètre p. Ce paramètre est primordial pour le rendu des splines, puisqu'il permet de contrôler le degré de lissage voulu par l'utilisateur.

4.1 Calcul de p en fixant S

Cette partie explique le raisonnement pour arriver jusqu'à la méthode de newton. Même si la méthode du calcul de p en fixant le paramètre S est très bien expliqué dans [2], nous estimons qu'il est important d'en montrer le raisonnement ici.

On cherche une fonction f deux fois dérivables sur $[a,b]$ qui minimise

$$\int_a^b (g'')dx \tag{9}$$

parmi les fonctions g qui vérifient la contrainte suivante, où S est un paramètre positif donné.

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i}\right)^2 \leq S \tag{10}$$

Pour appliquer la méthode du multiplicateur de Lagrange, il est nécessaire de transformer la contrainte d'inégalité (10) en une contrainte d'égalité :

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i}\right)^2 = S - z^2 \tag{11}$$

D'après la méthode du multiplicateur de *Lagrange* dont la théorie a été expliqué ci-dessus, minimiser (9) sous la contrainte (11) équivaut à minimiser la formule suivante en introduisant le multiplicateur de *Lagrange* p :

$$\mathcal{L} = \int_a^b (g'')dx + p\left(\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i}\right)^2 - S + z^2\right) \tag{12}$$

Toute solution optimale de $\mathcal{L}$ annule :

$$\frac{\partial \mathcal{L}}{\partial z} = 2 \times p \times z \tag{13}$$

Soit $p = 0$, mais ce cas de figure n'est pas intéressant puisque p est justement le paramètre à optimiser. La deuxième possiblité est que $z = 0$. Dans ce cas de figure, d'après la contrainte d'égalité (11) :

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i}\right)^2 = S \tag{14}$$

Soit donc finalement :

$$\sum_{i=0}^{n} \left(\frac{g(x_i) - y_i}{\sigma_i} \right)^2 = (\| \left(\Sigma^{-1}(a-y)\right\|_2)^2 = \left(\left\| \Sigma Q \left(Q^T \Sigma^2 Q + pT \right)^{-1} Q^T y \right\|_2 \right)^2 = F(p)^2 = S \tag{15}$$

Alors la fonction f suivante s'annule au paramètre p optimal :

$$f(p) = \frac{1}{F(p)} - \frac{1}{\sqrt{S}} \tag{16}$$

Il est donc possible de trouver le p optimal par la méthode de newton :

$$p^{(i+1)} = p^i - \frac{f(p)}{f'(p)} \tag{17}$$

D'après l'équation (16), la dérivée de f se définit par :

$$f'(p) = -\frac{(F(p)^2)' \times \frac{1}{\sqrt{F(p)^2}}}{F(p)^2} \tag{18}$$

Le calcul de $(F(p)^2)'$ et de $F(p)^2$ est suiffsament bien expliqué dans [2] pour ne pas le paraphraser ici.

Le calcul de p grâce à l'algorithme de Newton se trouve en Annexe 7.4, et la méthode d'inversion de matrice utilisée pour calculer $\left(Q^T \Sigma^2 Q + pT\right)^{-1}$ en Annexe 7.3.

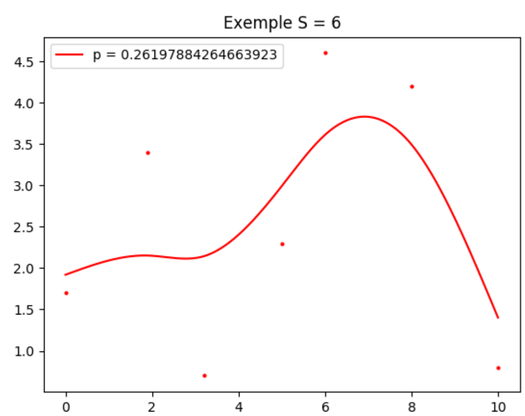

Figure 5 – *Calcul avec S fixé à 6*

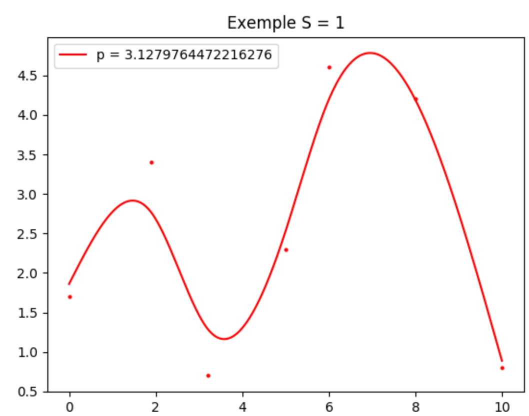

Figure 6 – *Calcul avec S fixé à 1*

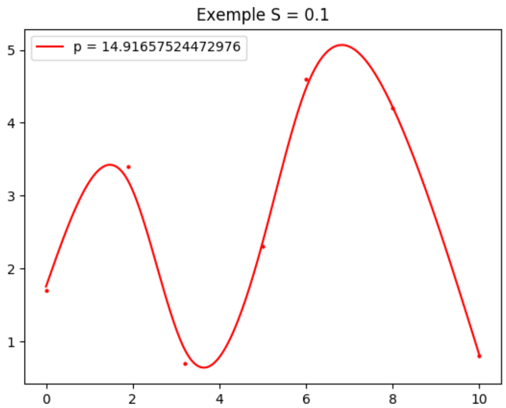

Figure 7 – *Calcul avec S fixé à 0.1*

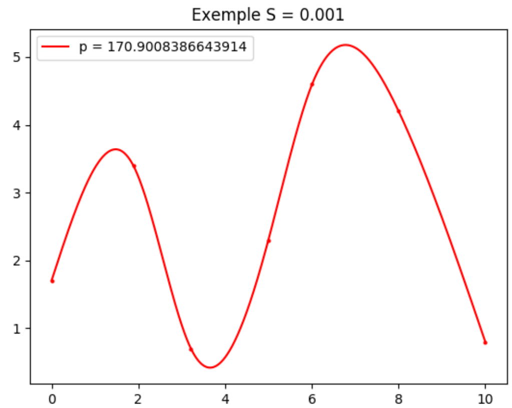

Figure 8 – *Calcul avec S fixé à 0.001*

4.1.1 Comparaison avec scipy sur le jeu de données de la production de poisson en Allemagne

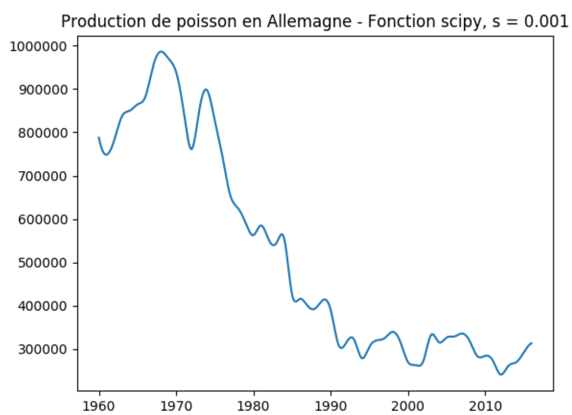

Figure 9 – *Calculé avec Scipy, Annexe 7.2*

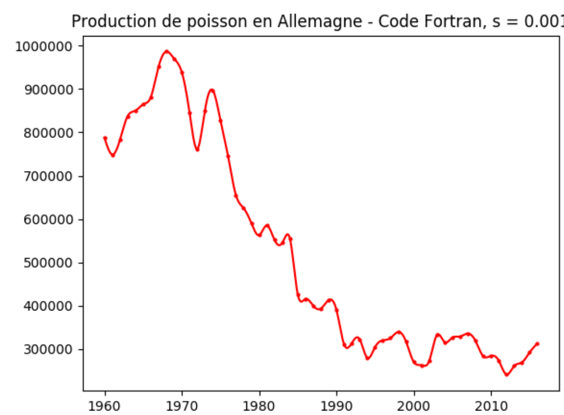

Figure 10 – *Calculé en Fortran, Annexe 7.4, 7.1*

4.2 Calcul de p en fixant le degré de liberté df

Cette partie est consacrée au calcul du paramètre p en fonction du degré de liberté df_p fixé par l'utilisateur. Comme expliqué dans [2], df_p est défini comme :

$$df_p = \text{Trace}(S_p) \tag{19}$$

Où est S_p est le coefficient linéaire de l'équation :

$$a = S_p y \tag{20}$$

Et se défini par :

$$S_p = I_{n+1} - \Sigma^2 Q \left(Q^T \Sigma^2 Q + pT\right)^{-1} Q^T \tag{21}$$

df_p étant fixé, l'objectif est de trouver p par une recherche dichtomique. L'intervalle de recherche de p n'est pas rigoureusement fixé, il faudrait aussi étudier le sens de variation de la fonction df_p pour garantir l'existence et l'unicité du paramètre p dans l'intervalle de recherche [α,β]. Cependant le programme donné en Annexe 7.5 permet d'afficher la courbe pour $\alpha = 10^{-5}$ et $\beta = 10^5$, et d'en inférer négligemment son allure.

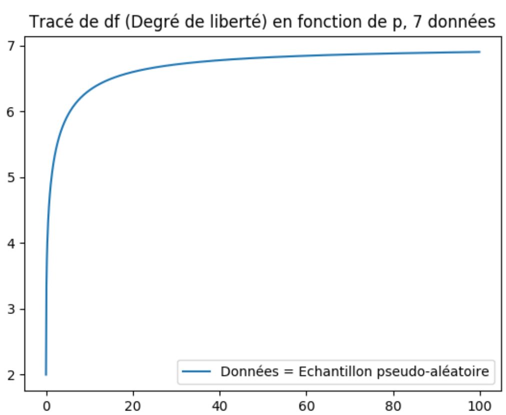

FIGURE 11 – *Exemple basique*

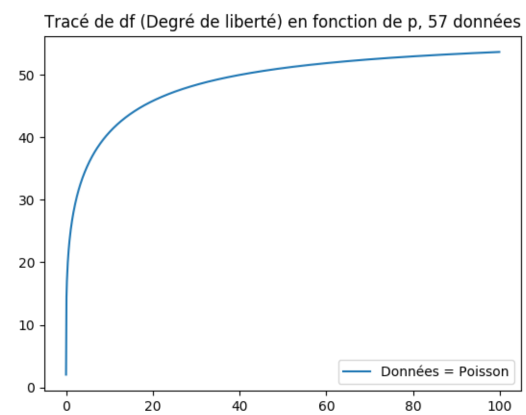

FIGURE 12 – Exemple de la production de poissons

Grâce à ces courbes, il est possible de faire l'hypothèse de monotonie croissante de df_p. D'ailleurs, df_p semble ici comprise entre 2 et $n + 1$, comme présenté dans [2]. Par cette hypothèse, il est possible de faire une recherche dichotomique du paramètre p en fixant df_p. Un tel algorithme de recherche dichotomique est simple, il suffit de calculer S_p, sa trace, et de restreindre peu à peu l'intervalle de recherche de p $[\alpha, \beta]$ en comparant la valeur attendue par la trace calculée. Si le df_p calculé est supérieur au df_p attendu, alors $\beta \leftarrow p$, sinon s'il est inférieur, alors $\alpha \leftarrow p$. D'ailleurs, p peut-être calculé de plusieurs manière, dont voici deux exemple :

$$p \leftarrow \sqrt{\alpha \times \beta}$$
$$p \leftarrow \frac{\alpha + \beta}{2}$$

Voici le rendu des courbes sur les deux jeux de données. Le code Fortran de la recherche dichotomique de p est donné en Annexe 7.6

4.2.1 Exemple basique

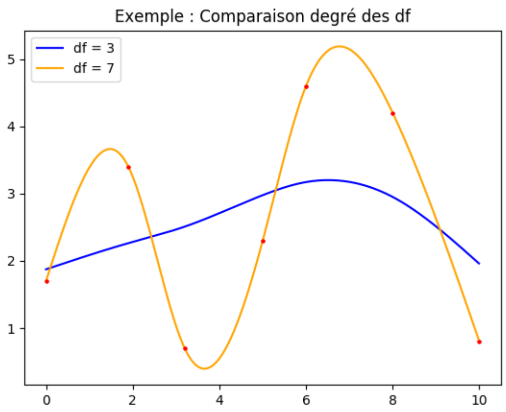

FIGURE 13 – *Exemple basique*

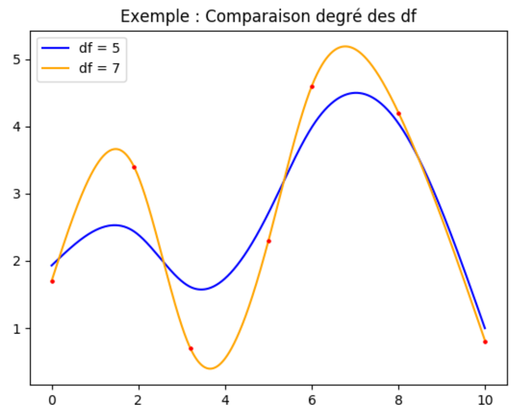

FIGURE 14 – Exemple de la production de poissons

4.2.2 Exemple sur la production de poisson dans le monde

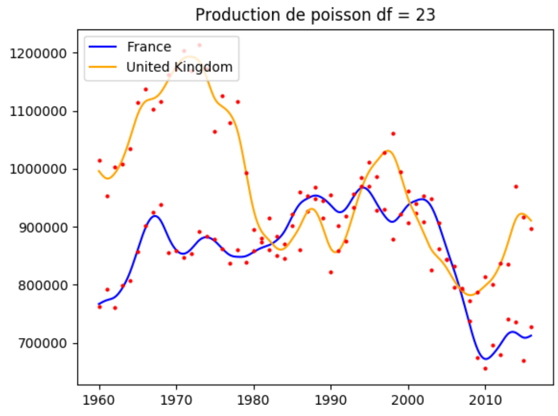

FIGURE 15 – *Exemple basique*

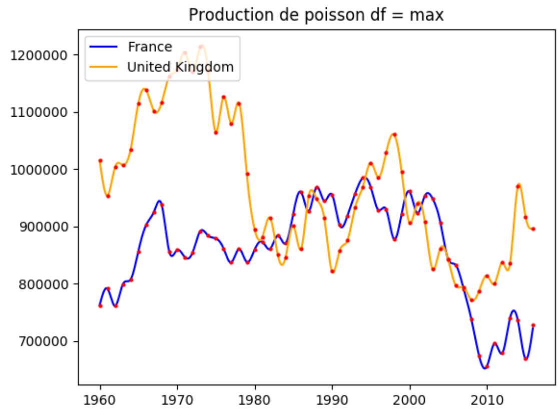

FIGURE 16 – Exemple de la production de poissons

5 Production de poisson de 1960 à nos jours [1]

Cette partie est illustrative, visualisons les résultats graphiques de tous ces calculs !

5.1 Comparaison de la production de poissons entre la France, l'Allemagne, et l'Espagne

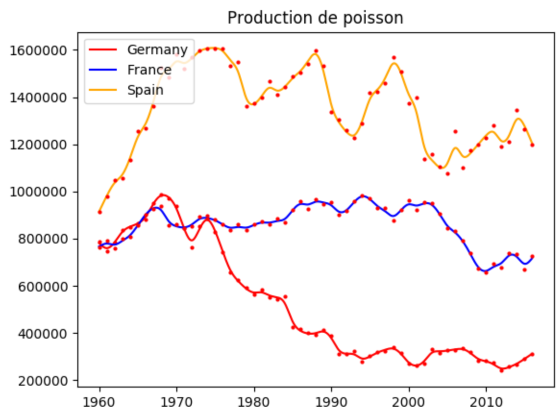

FIGURE 17 – *Degré de liberté fixé à 35*

5.2 Production de poissons dans le monde, en Chine et en Inde

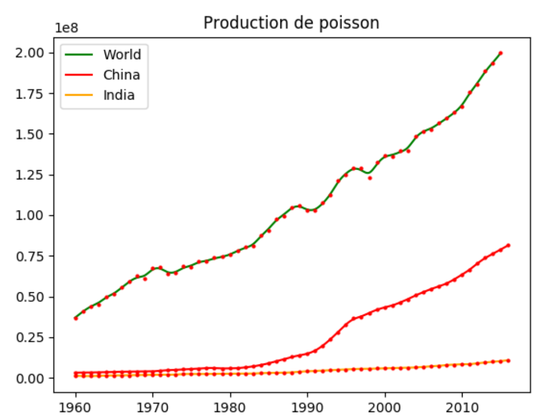

FIGURE 18 – *Degré de liberté fixé à 35*

5.3 Production de poissons en Amérique du Nord, aux Etats-Unis et au Canada

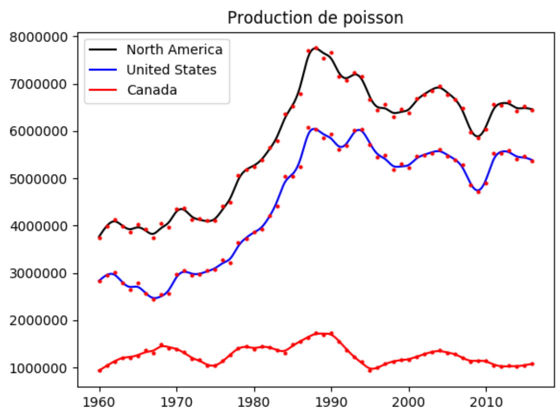

FIGURE 19 – *Degré de liberté fixé à 35*

5.4 Production de poissons en Afrique du Sud, au Burkina Faso, au Mali, et au Benin

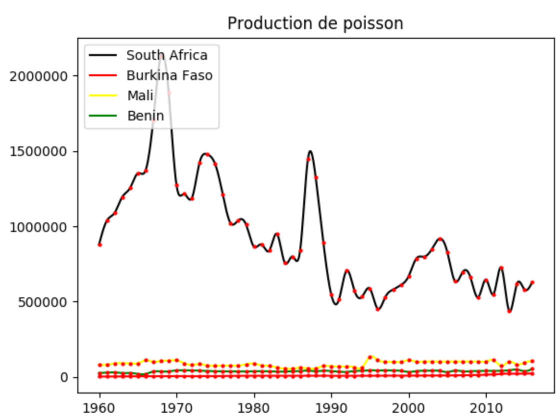

FIGURE 20 – *Degré de liberté fixé au maximum* $(n+1)$

6 Conclusion

Le code s'est architecturé autour d'un programme central écrit en Python permettant l'importation des données, l'affichage des splines et surtout l'appel des fonctions Fortran, reservé aux parties calculatoires. L'association de ces deux languages de programation est très simple d'utilisation grâce à la librairie *f2py3 - Fortran To Python 3*.

Il était tout à fait possible de faire l'ensemble du projet en Python, cependant, l'utilisation d'un language natif comme Fortran est parfaitement adéquat dans un contexte d'optimisation du temps de calcul et de performance en général. Ce choix s'argumente donc par une cohérence avec le contexte, mais aussi par la valeur esthétique que l'on attribue subjectivement à Fortran !

Nous sommes très satisfait du projet, notamment car nous avons réussi le calcul des splines, l'optimisation de p par deux méthodes différentes, mais aussi et surtout parce que tous les calculs ont été fais par nos propres programmes (hormis les multiplications de matrices qui sont réalisées grâce aux fonctions *DGEMM* [3] et *DGEMV*).

Cependant, le calcul de p par la dernière méthode proposé n'a pas pu être réalisé par manque de temps. Comme dit précédemment, le fait que tous les programmes ont été écrits par nos soins et en Fortran demandait plus de travail personnel.

Ce projet peut faire l'objet d'améliorations sur notre temps personnel, notamment en implémentant les splines dans l'enveloppe de *Graham* vu en cours de structures de données en *GIS3*, ou encore pour modéliser des trajectoires sur une séquence d'images, etc.

7 Annexes

7.1 Calcul des splines - Fortran

```fortran
      SUBROUTINE SPLINES(Y,LDY,H,LDH,Q,LDQ,T,LDT,
     $                   SIG,LDSIG,N,P,A,LDA,B,LDB,C,LDC,
     $                   D,LDD)
!variables données
      DOUBLE PRECISION Y(LDY),H(LDH)
      DOUBLE PRECISION Q(LDQ,*), T(LDT,*), SIG(LDSIG,*)
      INTEGER N
      DOUBLE PRECISION P
!variables résultats
      DOUBLE PRECISION A(LDA), B(LDB), C(LDC), D(LDD)
!variables locales
      DOUBLE PRECISION QTQ(N-1,N-1),QSIG(N-1,N+1),M(N-1,N-1)
      DOUBLE PRECISION L(N-1,N-1), TEMP(N-1), SIGQ(N+1,N-1)
!début
      CALL DGEMM('T','N',N-1,N+1,LDQ,1D0,Q,LDQ,SIG,LDSIG,
     $           0D0,QSIG,N-1)
      CALL DGEMM('N','N',N-1,N-1,LDQ,1D0,QSIG,N-1,Q,LDQ,
     $           0D0,QTQ,N-1)
      CALL ADDITION_MATRICE_CARRE(QTQ,N-1,T,LDT,M,N-1,P)
      CALL CHOLESKY(M,N-1,L,N-1)
      CALL DGEMV('T',LDQ,N-1,P,Q,LDQ,Y,1,
     $           0D0,TEMP,1)
      CALL DESCENTE(L,N-1,TEMP,N-1)
      CALL REMONTEE(L,N-1,TEMP,N-1)
      CALL DGEMM('N','N',LDSIG,N-1,LDQ,1D0,SIG,LDSIG,Q,LDQ,
     $           0D0,SIGQ,N+1)
      CALL DGEMV('N',LDQ,N-1,(1/P)*1D0,SIGQ,N+1,TEMP,1,0D0,A,1)
*f2py intent(inplace) a,b,c,d
      DO I=1,LDA
        A(I) = Y(I) - A(I)
      END DO
      C(1) = 0
      C(LDC) = 0
      DO I=1,LDC-2
        C(I+1) = TEMP(I)
      END DO
      DO I=1,LDD
        D(I) = (C(I+1) - C(I))/(3.0*H(I))
      END DO
      DO I=1,LDB
        B(I) = (A(I+1) - A(I))/H(I) - C(I)*H(I) - D(I)*H(I)*H(I)
      END DO
      DO I=1,LDB
      END DO

      END SUBROUTINE
```

7.2 Affichage des splines avec Scipy

```
f = interpolate.interp1d(vecteur_final1[1][0],
    vecteur_final1[1][1], fill_value = "interpolate",kind="cubic")
f = interpolate.UnivariateSpline(vecteur_final1[1][0]
    ,vecteur_final1[1][1],s=0.01)
xnew = np.arange(min(vecteur_final1[1][0]),
                 max(vecteur_final1[1][0]),
                 0.001)
plt.plot(xnew,f(xnew),'-')
```

7.3 Méthode d'inversion de matrice

Soit la définition de l'inverse d'une matrice A $(n)\times(n)$ symétrique définie positive :

$$AA^{-1} = I_n \tag{22}$$

La matrice inverse A^{-1} étant l'inconnue de l'équation, on a :

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,j} & \dots & a_{1,n} \\ \vdots & \ddots & \dots & \iddots & \vdots \\ a_{i,1} & \dots & a_{i,j} & \dots & a_{i,n} \\ \vdots & \iddots & \dots & \ddots & \vdots \\ a_{n,1} & \dots & a_{n,j} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} \sigma_{1,1} & \dots & \sigma_{1,j} & \dots & \sigma_{1,n} \\ \vdots & \ddots & \dots & \iddots & \vdots \\ \sigma_{i,1} & \dots & \sigma_{i,j} & \dots & \sigma_{i,n} \\ \vdots & \iddots & \dots & \ddots & \vdots \\ \sigma_{n,1} & \dots & \sigma_{n,j} & \dots & \sigma_{n,n} \end{pmatrix} = I_n \tag{23}$$

Comme la matrice A est SDP, on applique la méthode de Cholesky pour la décomposer en deux matrices LL^T. Ainsi, l'inversion de la matrice A revient à résoudre n systèmes d'équations linéaires sur chaque colonne de A^{-1}. Par exemple, pour la première colonne de A^{-1}, on obtient le système :

$$LL^T \begin{pmatrix} \sigma_{1,1} \\ \vdots \\ \sigma_{i,1} \\ \vdots \\ \sigma_{n,1} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{24}$$

7.4 Calcul du paramètre p en fixant S - Fortran

```
      DOUBLE PRECISION FUNCTION NEWTON_P(Q,LDQ,T,LDT,SIG,LDSIG,Y,
     $                                   LDY,N,S,ITERATION)
      IMPLICIT NONE
!variables données :
      INTEGER LDQ,LDT,LDSIG,LDY,N,ITERATION,I,J
      DOUBLE PRECISION Q(LDQ,*),T(LDT,*),Y(LDY),SIG(LDSIG,*),S
!variables locales :
      DOUBLE PRECISION QTQ(N-1,N-1),M(N-1,N-1),INVERSE(N-1,N-1)
      DOUBLE PRECISION QQT(N-1,N+1),SIGR(N+1,N+1),U(N-1),V(N+1)
      DOUBLE PRECISION SIGQ(N+1,N-1)
      DOUBLE PRECISION UMAT(1,N-1), QQ(1,N-1),QSIG(N-1,N+1)
      DOUBLE PRECISION FP2, FP2P, NORME2, FP, FPP
      NEWTON_P = 0.0001D0
      DO I=1,N+1
        DO J=1,N+1
            IF (I==J) THEN
                SIGR(I,I) = SQRT(SIG(I,I))
            ELSE
                SIGR(I,J) = 0
            END IF
        END DO
      END DO
      DO I=1,ITERATION
        CALL DGEMM('T','N',N-1,LDSIG,LDQ,1D0,Q,LDQ,SIG,LDSIG,
     $           0D0,QSIG,N-1)
        CALL DGEMM('N','N',N-1,N-1,N+1,1D0,QSIG,N-1,Q,LDQ,
     $           0D0,QTQ,N-1)
        CALL ADDITION_MATRICE_CARRE(QTQ,N-1,T,LDT,M,N-1,NEWTON_P)
        CALL INVERSION(M,N-1,INVERSE,N-1,N-1)
        DO J=1,N-1
            WRITE(*,*) INVERSE(:,J)
        END DO
        CALL DGEMM('N','T',N-1,N+1,N-1,1D0,INVERSE,N-1,Q,LDQ,
     $           0D0,QQT,N-1)
        CALL DGEMV('N',N-1,N+1,1D0,QQT,N-1,Y,1,0D0,U,1)
        CALL DGEMM('N','N',N+1,N-1,N+1,1D0,SIGR,N+1,Q,LDQ,0D0,
     $             SIGQ,N+1)
        CALL DGEMV('N',N+1,N-1,1D0,SIGQ,N+1,U,1,0D0,V,1)
        FP2 = NORME2(V,N+1,V,N+1)
        UMAT(1,:) = U
        CALL DGEMM('N','N',1,N-1,N-1,1D0,UMAT,1,T,LDT,0D0,
     $             QQ,1)
        UMAT(1,:) = QQ(1,:)
        FP2P = -1D0 * NORME2(UMAT,N-1,U,N-1)
        CALL DGEMM('N','N',1,N-1,N-1,1D0,UMAT,1,INVERSE,N-1,0D0,
     $             QQ,1)
        UMAT(1,:) = QQ(1,:)
        CALL DGEMM('N','N',1,N-1,N-1,1D0,UMAT,1,T,LDT,0D0,
     $             QQ,1)
        FP2P = 2D0 * (NEWTON_P * NORME2(QQ,N-1,U,N-1) + FP2P)
        FP = (1/SQRT(FP2)) - (1/SQRT(S))
        FPP = (-0.5D0 * FP2P * (1/SQRT(FP2)))/FP2
        NEWTON_P = NEWTON_P - (FP/FPP)
      END DO
      RETURN
      END FUNCTION
```

7.5 Affichage de la courbe df_p - Fortran

```
      SUBROUTINE DFP(Q,LDQ,T,LDT,SIG,LDSIG,N,RES,LDRES,
     $                             IND,LDIND,A,B,H)
! Cette fonction permet de calculer un vecteur de df en fonction de p
      INTEGER LDQ,LDT,LDSIG,N,I,INDICE,P_LOOP
      DOUBLE PRECISION Q(LDQ,*), T(LDT,*), SIG(LDSIG,*)
      DOUBLE PRECISION RES(LDRES),IND(LDIND)
!variables locales
      DOUBLE PRECISION IDENT(N+1,N+1),M(N-1,N-1),
     $                 QSIG(N-1,N+1),SIGQ(N+1,N-1),
     $                 QQ(N-1,N-1),
     $                 INVERSE(N-1,N-1), TEMP(N+1,N-1)
      DOUBLE PRECISION A,B,H,P,DF
      INDICE = 1
      DF = 0
*f2py intent(inplace) res
*f2py intent(inplace) ind
      CALL IDENTITE(IDENT,N+1)
      CALL DGEMM('T','N',LDQ-2,N+1,LDQ,1D0,Q,LDQ,SIG,LDSIG,
     $              0D0,QSIG,N-1)
      CALL DGEMM('N','N',N-1,N-1,N+1,1D0,QSIG,N-1,Q,LDQ,
     $              0D0,QQ,N-1)
      DO P_LOOP = 1,INT(B*H)
        P = A + (P_LOOP - 1)/(H * 1D0)
        CALL ADDITION_MATRICE_CARRE(QQ,N-1,T,LDT,M,N-1,P)
        CALL INVERSION(M,N-1,INVERSE,N-1,N-1)
        CALL DGEMM('N','N',LDSIG,N-1,N+1,1D0,SIG,LDSIG,Q,LDQ,
     $              0D0,SIGQ,N+1)
        CALL DGEMM('N','N',N+1,N-1,N-1,1D0,SIGQ,N+1,INVERSE,
     $              N-1,0D0,TEMP,N+1)
        CALL DGEMM('N','T',N+1,N+1,N-1,1D0,TEMP,N+1,Q,
     $              LDQ,0D0,IDENT,N+1)
        DO I = 1,N+1
            DF = DF + (1-IDENT(I,I))
        END DO
        RES(INDICE) = DF
        IND(INDICE) = P
        INDICE = INDICE + 1
        DF = 0D0
      END DO
      END SUBROUTINE
```

7.6 Recherche dichotomique de p - Fortran

```
      DOUBLE PRECISION FUNCTION PDF(Q,LDQ,T,LDT,SIG,
     $                              LDSIG,N,DFGOAL,A,B)
! Cette fonction permet de calculer p selon un df donné par
! recherche dichotomique
      INTEGER LDQ,LDT,LDSIG,N,I,INDICE
      DOUBLE PRECISION Q(LDQ,*), T(LDT,*), SIG(LDSIG,*)
      DOUBLE PRECISION SEUIL,DF,A,B,DFGOAL
!variables locales
      DOUBLE PRECISION IDENT(N+1,N+1),M(N-1,N-1),
     $                 QSIG(N-1,N+1),SIGQ(N+1,N-1),
     $                 QQ(N-1,N-1),
     $                 INVERSE(N-1,N-1), TEMP(N+1,N-1)
      SEUIL = 0.001D0
      DF = DFGOAL + SEUIL + 1D0
      CALL IDENTITE(IDENT,N+1)
      CALL DGEMM('T','N',LDQ-2,N+1,LDQ,1D0,Q,LDQ,SIG,LDSIG,
     $              0D0,QSIG,N-1)
      CALL DGEMM('N','N',N-1,N-1,N+1,1D0,QSIG,N-1,Q,LDQ,
     $              0D0,QQ,N-1)
      INDICE = 0
      DO WHILE((ABS(DFGOAL - DF) .gt. SEUIL .and. INDICE .lt. 300))
        PDF = SQRT(A*B)
        DF = 0D0
        CALL ADDITION_MATRICE_CARRE(QQ,N-1,T,LDT,M,N-1,PDF)
        CALL INVERSION(M,N-1,INVERSE,N-1,N-1)
        CALL DGEMM('N','N',LDSIG,N-1,N+1,1D0,SIG,LDSIG,Q,LDQ,
     $              0D0,SIGQ,N+1)
        CALL DGEMM('N','N',N+1,N-1,N-1,1D0,SIGQ,N+1,INVERSE,
     $              N-1,0D0,TEMP,N+1)
        CALL DGEMM('N','T',N+1,N+1,N-1,1D0,TEMP,N+1,Q,
     $              LDQ,0D0,IDENT,N+1)
        DO I = 1,N+1
            DF = DF + (1D0-IDENT(I,I))
        END DO
        IF(DF > DFGOAL) THEN
            B = PDF
        ELSE IF(DF < DFGOAL) THEN
            A = PDF
        END IF
        INDICE = INDICE + 1
      END DO
      WRITE(*,*)PDF,DF,(ABS(DFGOAL - DF))
      RETURN
      END FUNCTION
```

7.7 Utilisation du logiciel

Le logiciel s'architecture autour d'un squelette Python appellant des programmes Fortran. Le logiciel utilise *f2py3*. Les fichiers fortran sont inscrits ligne 10 du *Makefile*. Instructions d'utilisation :

1. Compiler avec *make*.
2. Insérer dans les variables $pays_i$ les noms correpondant aux pays à visualiser
3. Insérer la valeur du df souhaité, ou *"max"* pour $df = n + 1$, ou *None* pour calculer p en fonction de S
4. Vérifier que les bibliothèques *numpy* et *matplotlib* sont bien installées sur le système.
5. Lancer avec la commande *python3 spline.py*

7.8 Variation de Σ^2

Voici ici quelques images illustrant des variations sur la matrice Σ^2.

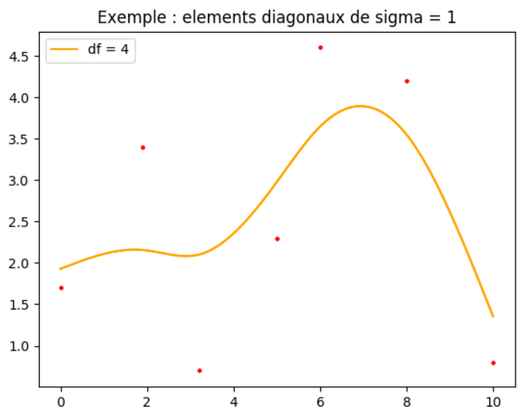

FIGURE 21 – $\Sigma^2_{i,i} = 1$

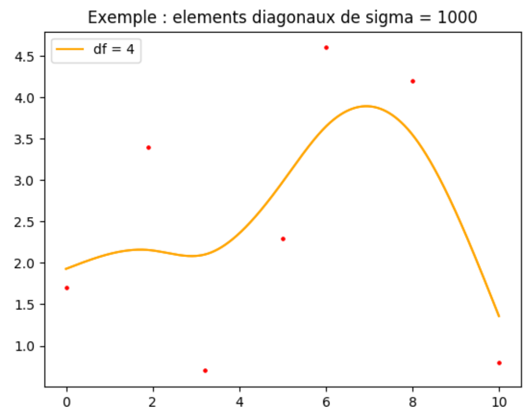

FIGURE 22 – $\Sigma^2_{i,i} = 1000$

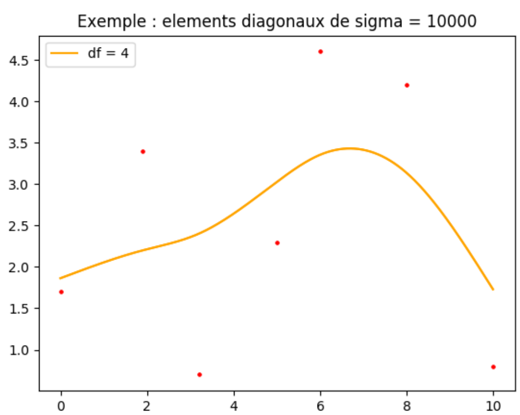

FIGURE 23 – $\Sigma^2_{i,i} = 10000$

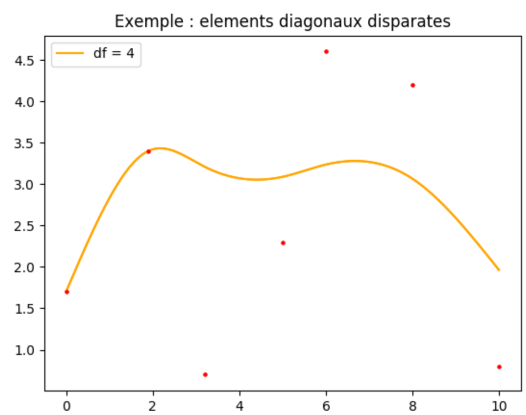

FIGURE 24 – $\Sigma^2_{i,i}$ pseudo-aléatoire

8 Références

Références

[1] Data World Bank. "Total fisheries Production". In : (2019). url : https://data.worldbank.org/indicator/ER.FSH.PROD.MT?view=chart.

[2] François Boulier. "Notes de cours - Calcul numérique". In : (2019). url : https://pro.univ-lille.fr/francois-boulier/enseignements/calcul-numerique/.

[3] Intel. "Multiplying Matrices Using dgemm". In : (2019). url : https://software.intel.com/en-us/mkl-tutorial-c-multiplying-matrices-using-dgemm.