TRAVAIL ENCADRÉ DE RECHERCHE

Modèles déterministes de prédiction de la température annuelle moyenne au Maroc en 2019

Alice JOSSIEN Camille GERMAIN

Encadrants : Caterina CALGARO Azzouz DERMOUNE

MASTER 1 MATHÉMATIQUES APPLIQUÉES ET STATISTIQUES

Table des matières

Table des figures

Liste des tableaux

Introduction

Le but de ce travail encadré de recherche est de prédire la température annuelle moyenne au Maroc en 2019. Pour ce faire, à partir de nos données, nous construirons des fonctions f polynomiales par morceaux basées sur des contraintes de régularité. Il s'agit de la méthode déterministe qui dans notre cas nous amènera à construire des splines qui nous permettront de déterminer des modèles de prédiction.
Dans un premier temps, nous définirons les coefficients des splines quadratique et cubique afin de les comparer à l'interpolation linéaire. Nous analyserons ensuite et comparerons leur efficacité afin d'approcher une fonction connue puis nous calculerons l'erreur entre cette fonction donnée et ses approximations. Finalement, nous utiliserons les températures au Maroc afin de déterminer les coefficients des trois approximations et les tracer.
Ensuite, nous présenterons plusieurs modèles de prédiction après avoir au préalable défini les matrices sur lesquelles reposent nos modèles pour l'interpolation linéaire et les deux splines. Pour finir, nous expérimenterons numériquement ces modèles afin de les valider ou non.

Chapitre 1

Splines

1 Définitions

Considérons les noeuds (t_i, p_i), $i = 1, \ldots, n$ où les t_i sont une partition de l'intervalle $[t_1; t_n]$ tels que $t_1 < \ldots < t_n$ et $p_i = f(t_i)$ avec $f : [t_1, t_n] \to \mathbb{R}$ une fonction que l'on cherche à approcher.

Définition 1. Une spline d'ordre k est une fonction qui est de classe C^{k+1} sur chaque sous-intervalle $[t_i, t_{i+1}]$, $i = 1, \ldots, n-1$ et qui est de classe C^k sur tout l'intervalle $[t_1, t_n]$.

1.1 Interpolation linéaire

Définition 2. L'interpolation linéaire de ces n points est la fonction f linéaire par morceaux de classe C^0 :

$$f(t) = \frac{(t - t_i)p_{i+1} + (t_{i+1} - t)p_i}{h_i}, \quad t \in [t_i, t_{i+1}], \quad i = 1, \ldots, n-1.$$

avec $h_i = t_{i+1} - t_i$, $i = 1, \ldots, n-1$.

Afin de simplifier l'implémentation des algorithmes dans la suite de notre travail, nous écrivons $f(t)$ sous la forme d'un produit matrice vecteur :

$$\begin{aligned} f(t) &= \frac{p_{i+1} - p_i}{h_i} t + \frac{p_i t_{i+1} - p_{i+1} t_i}{h_i} \\ &= (t, 1) \begin{pmatrix} \frac{p_{i+1} - p_i}{h_i} \\ \frac{p_i t_{i+1} - p_{i+1} t_i}{h_i} \end{pmatrix}, \quad t \in [t_i, t_{i+1}], \quad i = 1, \ldots, n-1 \end{aligned}$$

Le code présenté en A.1 permet donc de définir une matrice, appelée **M**, d'ordre $2 \times (n-1)$ contenant les $n-1$ coefficients de l'interpolation linéaire.

Théorème 1. *Les splines linéaires sont globalement de classe C^0 et sont de classe C^∞ sur les intervalles $[t_i; t_{i+1}], i = 1, \ldots, n-1$.*

Démonstration. Ce théorème dérive de la construction des coefficients ci-dessus □

1.2 Splines quadratiques

Définition 3. Une spline quadratique sur l'intervalle $[t_1, t_n]$ est définie par la fonction quadratique par morceaux suivante :

$$f(t) = p_i + (t - t_i)q_i + \frac{(t - t_i)^2}{2}u_i, \quad t \in [t_i, t_{i+1}], , \quad i = 1, \ldots, n-1.$$

Nous souhaitons déterminer les coefficients q_i et u_i en fonction des données p_i pour $i = 1, \cdots, n-1$. Pour cela, nous nous plaçons dans le cas des splines naturelles autrement dit :

$$f'(t_1) = q_1 = 0$$

Nous utiliserons désormais les notations suivantes :

$$\begin{aligned} x^+ &= \lim_{s \to 0} \; x + s \\ x^- &= \lim_{s \to 0} \; x - s \end{aligned}$$

La fonction f est telle que pour chaque intervalle $[t_i, t_{i+1}]$, $i = 1, \ldots, n-1$ on a :

- $f(t_i^+) = p_i$,
- $f'(t_i^+) = q_i$
- $f(t_{i+1}^-) = p_i + h_i q_i + \frac{h_i^2}{2}u_i$,
- $f'(t_{i+1}^-) = q_i + h_i u_i$

Nous pouvons déterminer les coefficients q_i et u_i en imposant la continuité de f et de sa dérivée f' aux noeuds internes $t_2, \cdots, t_{n-1}$. En effet, cela implique :

- $p_i + h_i q_i + \frac{h_i^2}{2}u_i = p_{i+1}$, continuité de f au point t_{i+1}
- $q_i + h_i u_i = q_{i+1}$, continuité de f' au point t_{i+1}

pour $i = 1, \ldots, n-1$.

A partir des $n-1$ dernières équations on obtient :

$$u_i = \frac{q_{i+1} - q_i}{h_i}, \quad i = 1, \ldots, n-1. \tag{1.1}$$

On injecte ces relations dans les $n-1$ premières équations et on a donc :

$$\frac{h_i}{2}q_i + \frac{h_i}{2}q_{i+1} = p_{i+1} - p_i, \quad i = 1, \ldots, n-1.$$

C'est un système linéaire à $n-1$ équations et n inconnues $q_1, \ldots, q_n$. En fixant la valeur de q_1, il existe une unique spline quadratique f telle que $f(t_i) = p_i$ avec $i = 1, \ldots, n$.
La spline naturelle correspond au cas $f'(t_1) = q_1 = 0$.
Le vecteur des dérivées premières $\tilde{\mathbf{q}} = (q_2, \ldots, q_n)^\top$ est la solution du système

$$\begin{aligned} &\frac{h_1}{2}q_2 = p_2 - p_1, \\ &\frac{h_i}{2}q_i + \frac{h_i}{2}q_{i+1} = p_{i+1} - p_i, \quad i = 2, \ldots, n-1. \end{aligned} \tag{1.2}$$

En notant $\mathbf{p} = (p_1, \ldots, p_n)^\top$, nous obtenons :

$$\tilde{\mathbf{q}} = \mathbf{Q}\mathbf{p}, \tag{1.3}$$

où $\mathbf{Q}$ est une matrice d'ordre $n-1 \times n$ définie par $\mathbf{Q} = \mathbf{S^{-1}N}$; où $\mathbf{S}$ est une matrice de taille $n-1 \times n-1$ bidiagonale telle que :

$$\begin{aligned}
&\mathbf{S}_{i,i} = \frac{h_i}{2} \quad i = 1, \ldots, n-1, \\
&\mathbf{S}_{i,i-1} = \frac{h_i}{2} \quad i = 2, \ldots, n-1, \\
&\mathbf{S}_{i,j} = 0 \quad \text{si } j \neq i, i-1
\end{aligned}$$

et $\mathbf{N}$ est une matrice de taille $n-1 \times n$ telle que :

$$\begin{aligned}
&\mathbf{N}_{i,i} = -1 \quad i = 1, \ldots, n-1, \\
&\mathbf{N}_{i,i+1} = 1 \quad i = 1, \ldots, n-1, \\
&\text{et les autres sont nuls.}
\end{aligned}$$

Le code présenté en annexe A.2 permet de déterminer la matrice $\mathbf{M}$ de taille $3 \times (n-1)$ contenant les coefficients p_i, q_i et u_i de cette spline quadratique naturelle.

Théorème 2. *Les splines quadratiques sont globalement de classe C^1 et sont de classe C^∞ sur les intervalles $[t_i; t_{i+1}]$, $i = 1, \ldots, n-1$.*

Démonstration. Ce théorème dérive de la construction des coefficients ci-dessus □

1.3 Splines cubiques

Définition 4. Une spline cubique sur l'intervalle $[t_1, t_n]$ est définie par la fonction cubique par morceaux suivante :

$$f(t) = p_i + (t-t_i)q_i + \frac{(t-t_i)^2}{2}u_i + \frac{(t-t_i)^3}{6}v_i, \quad t \in [t_i, t_{i+1}], , \quad i = 1, \ldots, n-1.$$

Les coefficients $\{(q_i, u_i, v_i) : \quad i = 1, \ldots, n\}$ sont liés aux noeuds $\{(t_i, p_i) : \quad i = 1, \ldots, n\}$ par les relations qui imposent la continuité de f , f' et f'' aux noeuds internes $t_2, \ldots, t_{n-1}$:

$$\begin{aligned}
p_i + h_i q_i + \frac{h_i^2}{2}u_i + \frac{h_i^3}{6}v_i &= p_{i+1}, \quad \text{continuité de } f \\
q_i + h_i u_i + \frac{h_i^2}{2}v_i &= q_{i+1}, \quad \text{continuité de } f', \\
u_i + h_i v_i &= u_{i+1}, \quad \text{continuité de } f'' \\
\text{avec} \quad i &= 1, \ldots, n-1.
\end{aligned} \tag{1.4}$$

A partir des $n-1$ dernières équations, on exprime les coefficients v_i en fonction des coefficients u_i. En effet,

$$v_i = \frac{u_{i+1} - u_i}{h_i}, \quad i = 1, \ldots, n-1$$

A partir des $n-1$ premières équations on exprime les q_i en fonction des p_i et u_i.

Dans le cas d'une spline cubique naturelle, en imposant $f''(t_1) = u_1 = f''(t_n) = u_n = 0$ on exprime les $(u_2, \ldots, u_{n-1})$ en fonction des données p_i de la façon suivante :

$$\begin{aligned}
p_{i+1} &= p_i + h_i q_i + \frac{h_i^2}{2} u_i + \frac{h_i^3}{6} v_i \\
&= p_i + h_i q_i + \frac{h_i^2}{2} u_i + \frac{h_i^3}{6} \frac{u_{i+1} - ui}{h_i} \\
&= p_i + h_i q_i + \frac{3h_i^2}{6} u_i + \frac{h_i^2}{6} u_{i+1} - ui \\
&= p_i + h_i q_i + \frac{h_i^2}{6} u_i + \frac{h_i^2}{6} (\frac{2}{h_i}(q_{i+1} - q_i)) \\
&= p_i + \frac{2}{3} h_i q_i + \frac{1}{3} h_i q_{i+1} + \frac{1}{6} h_i^2 u_i \\
\iff p_{i+1} - p_i &= \frac{2}{3} h_i q_i + \frac{1}{3} h_i q_{i+1} + \frac{1}{6} h_i^2 u_i \\
\iff \frac{p_{i+1} - p_i}{h_i} &= \frac{2}{3} q_i + \frac{1}{3} q_{i+1} + \frac{1}{6} h_i u_i \quad \text{pour } i = 1, \ldots, n-1 \\
\text{ou encore : } \frac{p_i - p_{i-1}}{h_{i-1}} &= \frac{2}{3} q_{i-1} + \frac{1}{3} q_i + \frac{1}{6} h_{i-1} u_{i-1} \quad \text{pour } i = 2, \ldots, n
\end{aligned}$$

Par somme des deux dernières égalités, pour $i = 2, \ldots, n-1$, on obtient :

$$\begin{aligned}
(\frac{p_{i+1} - p_i}{hi} - \frac{p_i - p_{i-1}}{h_{i-1}}) &= \frac{2}{3}(q_i - q_{i-1}) + \frac{1}{3}(q_{i+1} - q_i) + (\frac{h_i}{6} u_i - \frac{h_{i-1}}{6} u_{i-1}) \\
&= \frac{h_{i-1}}{3}(u_i + u_{i-1}) + \frac{1}{3}\frac{h_i}{2}(u_{i+1} + u_i) + \frac{h_i}{6} u_i - \frac{h_{i-1}}{6} u_{i-1} \\
&= u_i(\frac{h_{i-1}}{3} + \frac{2h_i}{6}) + u_{i+1}(\frac{h_i}{6}) + u_{i-1}(\frac{h_{i-1}}{3} - \frac{h_{i-1}}{6}) \\
\iff 6(\frac{p_{i+1} - p_i}{hi} - \frac{p_i - p_{i-1}}{h_{i-1}}) &= u_i(2h_{i-1} + 2h_i) + u_{i+1} h_i + u_{i-1} h_{i-1}
\end{aligned}$$

Nous pouvons donc écrire :

$$\mathbf{u} = \mathbf{T}\mathbf{p},$$

avec $\mathbf{u} = (u_1, \ldots, u_n)^\top$ et la matrice $\mathbf{T}$ de dimension $n \times n$ est définie par le produit $\mathbf{V}^{-1}\mathbf{Z}$ où la matrice $\mathbf{V}$ est de taille $n \times n$ ayant pour coefficients non nuls :

$$\begin{aligned}
&\mathbf{V}_{1,1} = \mathbf{V}_{n,n} = 1 \\
&\mathbf{V}_{i,i} = 2(h_i + h_{i-1}) \quad i = 2, \ldots, n-1, \\
&\mathbf{V}_{i,i-1} = h_{i-1} \quad i = 2, \ldots, n-1, \\
&\mathbf{V}_{i,i+1} = h_i \quad i = 2, \ldots, n-1,
\end{aligned}$$

et la matrice $\mathbf{Z}$ est de dimension $n \times n$, qui a sa première et dernière ligne nulle (car nous

sommes dans le cas des splines naturelles), et le reste de ses coefficients définis par :

$$\begin{aligned}
\mathbf{Z}_{i,i} &= \frac{-1}{h_i} - \frac{1}{h_{i-1}} \quad i = 2, \ldots, n-1, \\
\mathbf{Z}_{i,i-1} &= \frac{1}{h_{i-1}} \quad i = 2, \ldots, n-1, \\
\mathbf{Z}_{i,i+1} &= \frac{1}{h_i} \quad i = 2, \ldots, n-1, \\
\mathbf{Z}_{i,j} &= 0 \quad \text{sinon, pour } j = 1, \ldots, n
\end{aligned}$$

Le code présenté en annexe A.3 permet de déterminer une matrice $\mathbf{M}$ de taille $4 \times (n-1)$ contenant les coefficients p_i, q_i, u_i et $v_i$de cette spline quadratique naturelle.

Théorème 3. *Les splines cubiques sont globalement de classe C^2 et sont de classe C^∞ sur les intervalles $[t_i; t_{i+1}]$, $i = 1, \ldots, n-1$.*

Démonstration. Ce théorème dérive de la construction des coefficients ci-dessus □

2 Efficacité des splines

Afin de tester l'efficacité de l'interpolation linéaire et des splines quadratiques et cubiques, nous allons utiliser des fonctions données. En effet, dans un premier temps nous définirons les noeuds (t_i, p_i). Nous choisirons tout d'abord les t_i puis calculerons les p_i, c'est-à-dire les valeurs de la fonction aux points t_i. Ensuite, nous déterminerons les trois approximations et nous les tracerons afin de comparer visuellement les trois splines. Enfin, nous déterminerons la norme l_2 de la différence des valeurs de la série et de la spline étudiée sur chaque intervalle $[t_i, t_{i+1}]$.

2.1 Graphiques

Afin de tracer les trois splines, nous avons codé un algorithme en C++. En effet, nous obtenons les coefficients des splines comme expliqué dans la partie précédente puis nous traçons la fonction par morceaux sur l'intervalle étudié.

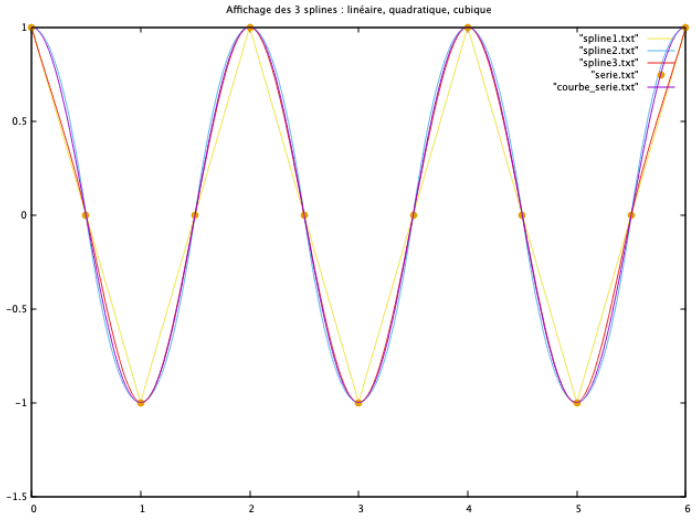


FIGURE 1.2.1 – $cos(\pi x)$ pour $x \in [0, 6]$ avec h=0.5

Sur la figure 1.2.1, nous avons tracé la fonction $cos(\pi x)$ pour $x \in [0, 6]$ en violet. Nous avons également déterminé puis tracé les splines linéaires, quadratiques et cubiques respectivement en jaune, bleu et rouge en prenant comme données initiales :

$$i = 0, \cdots, 12; \quad h = 0.5; \quad x_i = ih; \quad y_i = cos(\pi x_i)$$

Nous constatons que les trois splines réprésentent plutôt bien la courbe théorique. Cependant, la courbe définissant la spline cubique semble le mieux approcher la fonction donnée. En effet, nous observons des erreurs commises par l'interpolation linéaire sur chaque subdivison de l'intervalle dues à sa linéarité. Enfin, nous constatons également deux erreurs notables pour la spline quadratique sur les première et dernière subdivisions de notre intervalle.

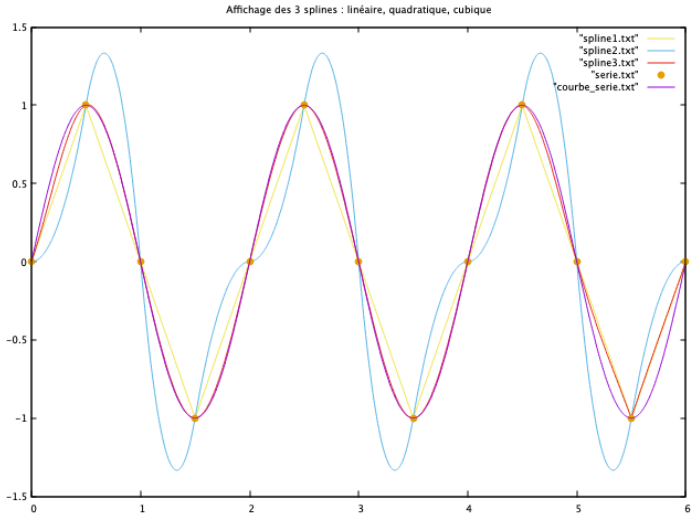


FIGURE 1.2.2 – $sin(\pi x)$ pour $x \in [0, 6]$ avec h=0.5

Sur la figure 1.2.2, nous avons tracé la fonction $sin(\pi x)$ pour $x \in [0, 6]$ en violet. Nous avons également déterminé puis tracé les splines linéaires, quadratiques et cubiques respectivement en jaune, bleu et rouge en prenant comme données initiales :

$$i = 0, \cdots, 12 \quad h = 0.5 \quad x_i = ih \quad y_i = sin(\pi x_i)$$

Nous constatons que la courbe de la spline cubique est quasiment superposée avec la courbe de la fonction donnée. Pour l'interpolation linéaire, l'erreur est importante mais pour l'interpolation quadratique elle l'est encore plus. La spline cubique est sûrement la plus efficace dans cet exemple.

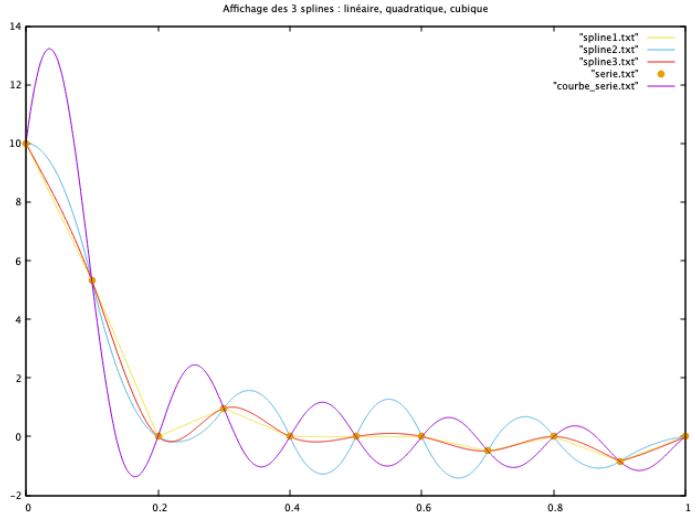


FIGURE 1.2.3 – $\sum_{i=1}^{10} cos(i\pi x) + sin(i\pi x)$ pour $x \in [0, 1]$ avec h=0.1

Sur la figure 1.2.3, nous avons tracé la fonction $\sum_{i=1}^{10} cos(i\pi x) + sin(i\pi x)$ pour $x \in [0, 1]$ en violet. Nous avons également déterminé puis tracé les splines linéaires, quadratiques et cubiques respectivement en jaune, bleu et rouge en prenant comme données initiales :

$$i = 0, \cdots, 1 \quad h = 0.1 \quad x_i = ih \quad y_i = cos(\pi x_i) + sin(\pi x_i)$$

Nous remarquons trois zones sur ce graphique. Premièrement, sur l'intervalle $[0; 0.2]$ la courbe représentant la spline quadratique est celle se rapprochant le plus de la courbe représentant la série. Puis, sur l'intervalle $[0.2; 0.8]$ il s'agit de la courbe représentant l'interpolation linéaire. Et enfin, sur l'intervalle $[0.8; 1]$, c'est la courbe représentant la spline cubique. Nous pouvons penser qu'en augmentant le nombre de noeuds, la spline cubique s'améliore, ce que nous allons vérifier dans la partie suivante.

2.2 Calcul erreur par la norme l_2

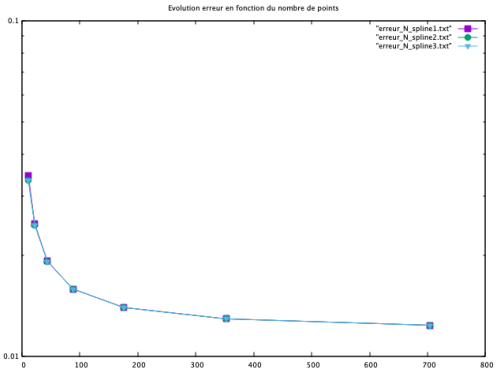
Afin de confirmer ce que nous avons constaté graphiquement sur les trois exemples précédents, nous allons calculer l'erreur entre la courbe théorique et les trois approximations. En effet, afin de tracer ces différentes courbes, nous avons subdivisé à nouveau chaque intervalle $[t_i, t_{i+1}]$ en 100. Ainsi, sur chaque intervalle $[t_1, t_n]$, nous disposons de cinq vecteurs $v^1, \cdots, v^5$ de taille $100n$ tels que pour $i = 1, \cdots, n$ et $j = 0, \cdots, 99$:

- $v_j^1 = \frac{jh}{100n}$ où $h = t_{i+1} - t_i$
- $v_j^2 = f(v_j^1)$ où f représente la fonction théorique étudiée
- $v_j^3 = s_0(v_j^1)$ où s_0 représente la fonction obtenue avec l'interpolation linéaire
- $v_j^4 = s_1(v_j^1)$ où s_1 représente la fonction obtenue avec la spline quadratique
- $v_5(j) = s_2(v_j^1)$ où s_2 représente la fonction obtenue avec la spline cubique

Nous avons ensuite utilisé la norme l_2 vectorielle définie par :

$$\|u\|_2 = \sqrt{\frac{1}{n}\sum_{i=1}^{n} u_i^2} \quad , u \in \mathbb{R}^n$$

En effet, nous avons calculé les normes $\|f - s_0\|_2$, $\|f - s_1\|_2$ et $\|f - s_2\|_2$ sur chaque subdivsion $[t_1, t_n]$ pour différentes valeurs de n et tracé leurs évolutions en fonction de ces n en utilisant une échelle logarithmique pour l'axe des ordonnées.
Les résultats suivants proviennent des algorithmes C++ que nous avons implémentés, visibles en annexe A.4.



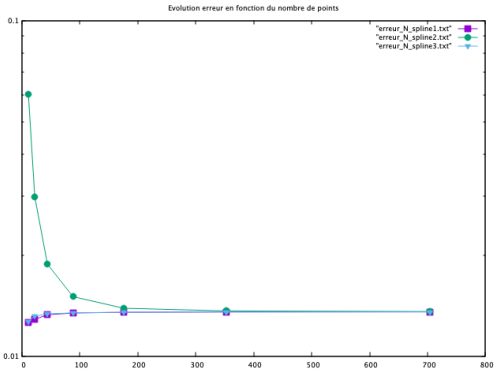
(a) Graphique des erreurs en fonction de N

N	Interpolation linéaire	Spline quadratique	Spline cubique
11	0.0344446	0.0333716	0.0333592
22	0.0247453	0.0245547	0.0245399
44	0.0191672	0.0191161	0.019113
88	0.0158526	0.0158365	0.0158363
176	0.0139498	0.0139453	0.0139453
352	0.0129067	0.0129055	0.0129055
704	0.0123555	0.0123552	0.0123552

(b) Tableau des erreurs en fonction de N

FIGURE 1.2.4 – $cos(\pi x)$ pour $x \in [0, 6]$

Nous remarquons que pour ces trois approximations, les erreurs sont très proches. Plus N augmente, plus les approximations sont meilleures. En effet, nous observons que les erreurs des trois approximations convergent vers la même valeur (environ 0.012).



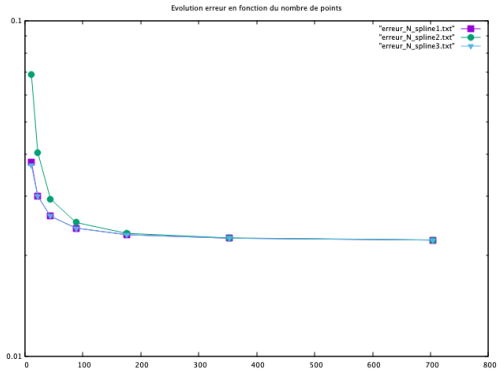
(a) Graphique des erreurs en fonction de N

N	Interpolation linéaire	Spline quadratique	Spline cubique
11	0.0126047	0.0603762	0.0126904
22	0.0128592	0.0298157	0.0131464
44	0.0132738	0.0187767	0.0133548
88	0.0134335	0.015029	0.013454
176	0.013499	0.01388	0.0135041
352	0.0135276	0.0136267	0.0135288
704	0.0135409	0.0135667	0.0135412

(b) Tableau des erreurs en fonction de N

FIGURE 1.2.5 – $sin(\pi x)$ pour $x \in [0, 6]$

Pour cette fonction, nous constatons que la spline cubique réalise la meilleure approximation puisque son erreur est très petite peu importe la valeur de N. L'erreur commise par l'interpolation linéaire devient rapidement très proche de celle commise par la spline cubique. En revanche, nous constatons que l'approximation par spline quadratique est un mauvais modèle ; en effet, son erreur pour un N petit est jusqu'à 5 fois plus grande que celle de la spline cubique. Ce n'est que lorsque N devient grand que les erreurs commises par les splines quadratique et cubique ont une différence de l'ordre de 10^{-4}.



(a) Graphique des erreurs en fonction de N

N	Interpolation linéaire	Spline quadratique	Spline cubique
11	0.0378218	0.0692124	0.037011
22	0.0299631	0.0404419	0.0299412
44	0.0261094	0.0294061	0.0261153
88	0.0240618	0.0249993	0.0240636
176	0.0229906	0.023205	0.0229911
352	0.0224403	0.0224967	0.0224404
704	0.0221611	0.0221769	0.0221611

(b) Tableau des erreurs en fonction de N

FIGURE 1.2.6 – $\sum_{i=1}^{10} cos(i\pi x) + sin(i\pi x)$ pour $x \in [0, 1]$

Pour ce dernier graphique, nous constatons que les erreurs pour l'interpolation linéaire et pour la spline cubique sont très proches pour n'importe quelle valeur de N. La spline quadratique réalise une erreur jusqu'à deux fois plus importante que les autres approximations. Nous observons que l'erreur pour ces trois approximations tend vers 0.0222.

Prouvons pour les splines cubiques que l'erreur converge :
Théorème 4. *On suppose* $f \in C^4([a,b])$ *et* $\|f^{(4)}\| = \max|f^{(4)}| \leq L$. *Soit* Δ *la partition telle que* $h = \max_{0\leq i\leq n-2} h_{i+1} = \max 0 \leq i \leq n-1 x_{i+1} - x_i$ *et* $\frac{h}{h_i} \leq K (i = 0, \ldots, n-1)$. *Si* s *est la spline cubique telle que* $s(x_i) = f(x_i)$ i=0, …, n *et* $s'(a) = f'(a)$, $s'(b) = f'(b)$ *alors il existe une constante* $C_l \leq 2$ *qui ne dépend pas de la partition* Δ , *telle que* $x \in [a, b]$. *On a :*

$$|f^{(l)} - s^{(l)}| \leq C_l L K h^{4-l}, \quad l = 0, 1, 2, 3.$$

Démonstration. Admis. La preuve est disponible dans le cours présenté en annexe. □

Application :

— Pour la fonction $f(x) = \cos(\pi x)$
Prenons $l = 0$ pour mesurer l'erreur entre la fonction et la spline cubique. Comme nous prenons un pas constant, nous avons $K = 1$ et $h = \frac{6}{N}$. $f^{(4)} = \pi^4 \cos(\pi x)$, donc $L = \pi^4$. Ce qui nous donne :

$$|f - s| \leq C_l(\pi \frac{6}{N})^4$$

En ajustant la constante C_l, nous concluons que l'erreur est bornée.

— Pour la fonction $f(x) = \sin(\pi x)$
De même, nous prenons $l = 0, K = 1, h = \frac{6}{N}$. Nous avons $f^{(4)} = \pi^4 \sin(\pi x)$, donc $L = \pi^4$. Ce qui nous donne :

$$|f - s| \leq C_l(\pi \frac{6}{N})^4$$

En ajustant la constante C_l, nous concluons que l'erreur est bornée.

— Pour la fonction $f(x) = \sum_{i=1}^{10} cos(i\pi x) + sin(i\pi x)$
De même, nous avons $l = 0, K = 1, h = \frac{1}{N}$. Nous avons $f^{(4)} = \sum_{i=1}^{10} (i\pi)^4 cos(i\pi x) + (i\pi)^4 sin(i\pi x)$, d'où $L = (55\pi)^4$. Ce qui nous donne :

$$|f - s| \leq C_l(55\pi \frac{1}{N})^4$$

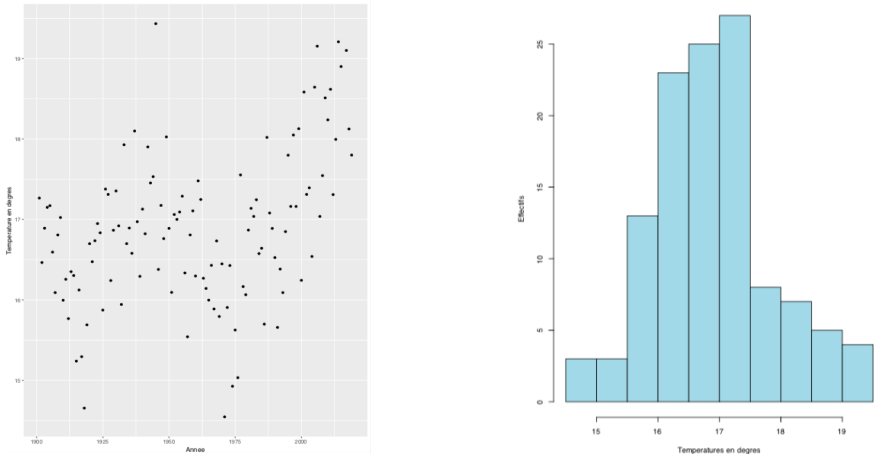
En ajustant la constante C_l, nous concluons que l'erreur est bornée.

Conclusion : Nous constatons à travers ces trois exemples que l'approximation par spline naturelle cubique est, en règle générale, la meilleure approximation. Suivie de près par l'interpolation linéaire. La spline quadratique naturelle quant à elle, n'est pas un modèle satisfaisant. En effet, son erreur est jusqu'à cinq fois plus importante que la cubique.

3 Application aux températures du Maroc entre 1901 et 2018

3.1 Présentation des données

Nous disposons des températures moyennes annuelles au Maroc de 1901 à 2018, disponible sur l'annexe A.5.
Dans un premier temps, nous avons tracé le nuage de points ainsi que l'histogramme associés à ces données avec le logiciel R.



(a) Nuage de points : températures annuelles moyennes au Maroc en fonction des années (b) Répartition de la température annuelle moyenne au Maroc

FIGURE 1.3.7 – Graphiques représentant la température annuelle moyenne au Maroc de 1901 à 2018

Nous avons ensuite effectué quelques statistiques simples sur la totalité de nos données que nous présentons ci-dessous.

Température minimale	14.55
Premier quartile	16.26
Médiane	16.85
Température moyenne	16.85
Troisième quartile	17.30
Température maximale	19.43

TABLE 1.1 – Description des données

Enfin, nous avons segmenté notre base de données afin d'obtenir quatre périodes. Le but est de voir l'évolution de la température en fonction du temps.

Température minimale	14.65
Premier quartile	16.09
Médiane	16.47
Température moyenne	16.43
Troisième quartile	16.89
Température maximale	17.38

(a) Première période : 1901 à 1930

Température minimale	15.54
Premier quartile	16.73
Médiane	16.97
Température moyenne	17.04
Troisième quartile	17.32
Température maximale	19.43

(b) Deuxième période : 1930 à 1959

Température minimale	14.55
Premier quartile	15.93
Médiane	16.43
Température moyenne	16.41
Troisième quartile	17.07
Température maximale	18.02

(c) Troisième période : 1959 à 1988

Température minimale	15.66
Premier quartile	16.88
Médiane	17.35
Température moyenne	17.54
Troisième quartile	18.31
Température maximale	19.21

(d) Quatrième période : 1988 à 2018

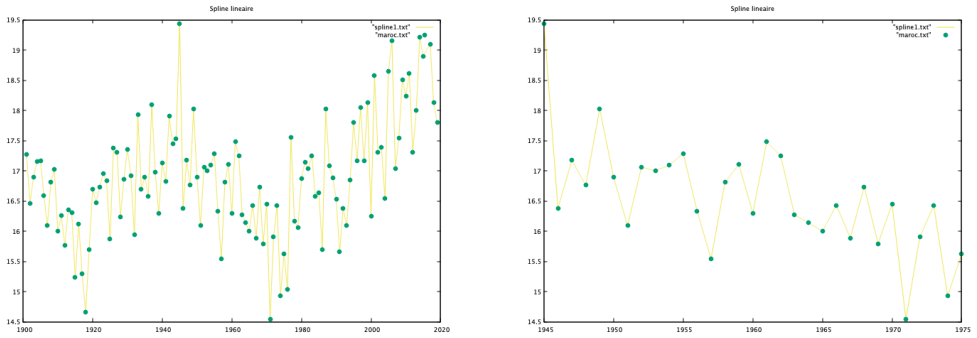
TABLE 1.2 – Description des données segmentées en 4 périodes

Nous constatons que la température est plus élevée sur les trente dernières années.

3.2 Tracé des splines

Dans cette partie, nous avons tracé les splines associées aux données de la température au Maroc de 1901 à 2018 en utilisant un programme que nous avons implémenté en C++. Pour les trois splines : linéaire, quadratique et cubique, nous avons dans premier temps tracé la spline sur l'ensemble global des années puis dans un second temps sur la période de 1945 à 1975 afin de mieux visualiser les courbes. Les résultats sont présentés ci-dessous.

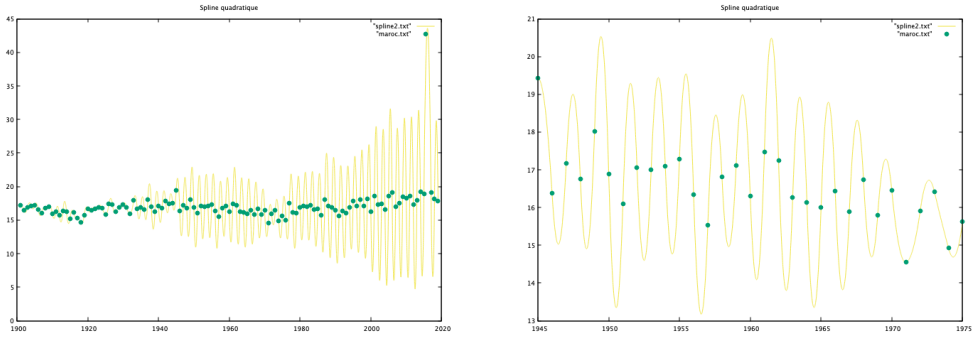
3.2.1 Interpolation linéaire



(a) Températures de 1901 à 2019 (b) Températures de 1945 à 1975

FIGURE 1.3.8 – Interpolation linéaire pour les températures

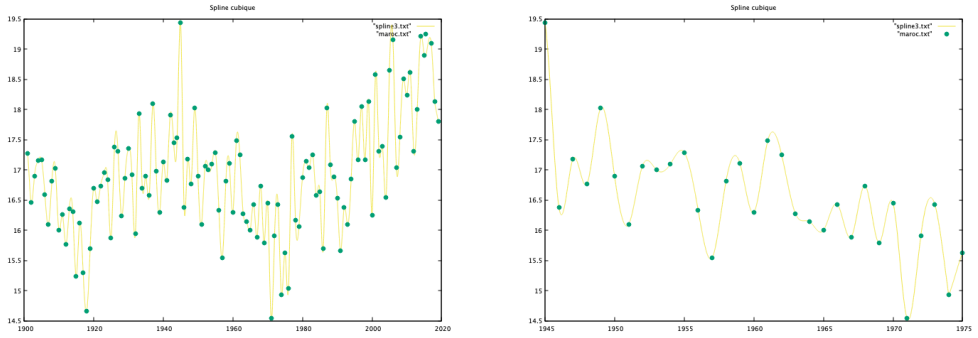
3.2.2 Spline quadratique



(a) Températures de 1901 à 2019 (b) Températures de 1945 à 1975

FIGURE 1.3.9 – Spline quadratique pour les températures

3.2.3 Spline cubique



(a) Températures de 1901 à 2019 (b) Températures de 1945 à 1975

FIGURE 1.3.10 – Spline cubique pour les températures

Chapitre 2

La prévision en théorie

Nous disposons des températures moyennes annuelles au Maroc de 1901 à 2018. Les noeuds $(t_i, p_i) \quad i = 1, \ldots, 118$, que nous avons définis au **chapitre 1** pour construire les coefficients des splines, ont donc les propriétés suivantes :

— les t_i représentent la partition de $[1901; 2018]$ telle que $t_{i+1} - t_i = 1$, nous avons donc $t_i = 1900 + i$

— les p_i représentent la température annuelle moyenne au Maroc pour l'année t_i

1 Calcul des matrices déterminant nos modèles

1.1 Interpolation linéaire

Le but de cette section est de définir la matrice $\mathbf{P^{(0)}}$ à l'aide de l'égalité

$$\int_{t_1}^{t_n} \mid f(t) \mid^2 dt = (p_1, \ldots, p_n)\mathbf{P^{(0)}}(p_1, \ldots, p_n)^\top.$$

Par identification nous obtenons :

$$\begin{aligned} \int_{t_1}^{t_n} \mid f(t) \mid^2 dt &= \sum_{i=1}^{n-1} \int_{t_i}^{t_i+1} \mid f(t) \mid^2 dt \\ &= \sum_{i=1}^{n-1} \int_{t_i}^{t_{i+1}} \mid \frac{t_{i+1} - t}{h_i} f(t_i) + \frac{t - t_i}{h_i} f(t_{i+1}) \mid^2 dt \end{aligned}$$

On pose $u = t - t_i$:

$$
\begin{aligned}
&= \sum_{i=1}^{n-1} \int_0^{h_i} \mid \frac{t_{i+1} - (u + t_i)}{h_i} f(t_i) + \frac{t = (u + t_i) - t_i}{h_i} f(t_{i+1}) \mid^2 du \\
&= \sum_{i=1}^{n-1} \int_0^{h_i} \mid \frac{h_i - u}{h_i} f(t_i) + \frac{u}{h_i} f(t_{i+1}) \mid^2 du \\
&= \sum_{i=1}^{n-1} \int_0^{h_i} \frac{(h_i - u)^2}{h_i^2} f(t_i) + \frac{u^2}{h_i^2} f(t_{i+1})^2 + 2\frac{(h_i - u)u}{h_i^2} f(t_i) f(t_{i+1}) du \\
&= \sum_{i=1}^{n-1} \int_0^{h_i} \frac{h_i^2 - 2h_i u + u^2}{h_i^2} f(t_i)^2 + \frac{u^2}{h_i^2} f(t_{i+1})^2 + 2(\frac{u}{h_i} - \frac{u^2}{h_i^2}) f(t_i) f(t_{i+1}) du \\
&= \sum_{i=1}^{n-1} [(u - \frac{u^2}{h_i} + \frac{u^3}{3h_i^2}) f(t_i)^2 + \frac{u^3}{3h_i^2} f(t_{i+1})^2 + 2(\frac{u^2}{2h_i} - \frac{u^3}{3h_i^2}) f(t_i) f(t_{i+1})]_0^{h_i} \\
&= \sum_{i=1}^{n-1} \frac{h_i}{3} f(t_i)^2 + \frac{h_i^2}{3} f(t_{i+1})^2 + 2(\frac{h_i}{2} - \frac{h_i}{3}) f(t_i) f(t_{i+1}) \\
&= \sum_{i=1}^{n-1} \frac{h_i}{3} p_i^2 + \frac{h_i}{3} p_{i+1}^2 + \frac{h_i}{3} p_i p_{i+1} \\
&= \sum_{i=1}^{n-1} \frac{1}{3} h_i (p_{i+1}^2 + p_i p_{i+1} + pi^2) \\
&= \frac{1}{3} [h_1 (p_1^2 + p_2^2 + p_2 p_1 + h_2 p_2^2 + p_3^2 + p_2 p_3) + \cdots \\
&+ h_{n-2} (p_{n-1}^2 + p_{n-2}^2 + p_{n-2} p_{n-1}) + h_{n-1} (p_n^2 + p_{n-1}^2 + p_{n-1} p_n)] \\
&= \mathbf{p}^\top \mathbf{P^{(0)}} \mathbf{p}
\end{aligned}
$$

où $\mathbf{P^{(0)}}$ est une matrice tridiagonale de dimension $(n \times n)$ et a pour coefficient

$$
\begin{aligned}
&\mathbf{P^{(0)}}_{1,1} = \frac{h_1}{3} \\
&\mathbf{P^{(0)}}_{n,n} = \frac{h_{n-1}}{3}, \\
&\mathbf{P^{(0)}}_{i,i} = \frac{h_{i-1} + h_i}{3}, \quad i = 2, \ldots, n-1, \\
&\mathbf{P^{(0)}}_{i,i+1} = \mathbf{P^{(0)}}_{i+1,i} = \frac{h_i}{6}, \quad i = 1, \ldots, n-1,
\end{aligned}
$$

1.2 Spline quadratique

Le but de cette section est de définir la matrice $\mathbf{P^{(1)}}$ à l'aide de l'égalité

$$
\int_{t_1}^{t_n} \mid f'(t) \mid^2 dt = (q_1, \ldots, q_n) \mathbf{P^{(1)}} (q_1, \ldots, q_n)^\top.
$$

Par identification nous obtenons :

$$\begin{aligned}
\int_{t_1}^{t_n} \mid f'(t) \mid^2 dt &= \sum_{i=1}^{n-1} \int_{t_i}^{t_i+1} \mid f'(t) \mid^2 dt \\
&= \sum_{i=1}^{n-1} \int_{t_i}^{t_{i+1}} \mid (q_i + u_i(t-t_i)) \mid^2 dt \\
&= \sum_{i=1}^{n-1} \int_{t_i}^{t_{i+1}} [q_i^2 + u_i^2(t-t_i)^2 + 2q_i u_i(t-t_i)]dt \\
&= \sum_{i=1}^{n-1} [tq_i^2 + \frac{u_i^2}{3}(t-t_i)^3 + q_i u_i(t-t_i)^2]_{t_i}^{t_{i+1}} \\
&= \sum_{i=1}^{n-1} (q_i^2 t_{i+1} + \frac{u_i^2}{3}h_i^3 + q_i u_i h_i^2) - (t_i q_i^2) \\
&= \sum_{i=1}^{n-1} q_i^2 h_i + \frac{u_i^2}{3}h_i^3 + q_i u_i h_i^2
\end{aligned}$$

Or $u_i = \dfrac{q_{i+1} - q_i}{h_i}$

$$\begin{aligned}
&= \sum_{i=1}^{n-1} q_i^2 h_i + \frac{1}{3}(\frac{q_{i+1}-q_i}{h_i})^2 h_i^3 + q_i(\frac{q_{i+1}-q_i}{h_i})h_i^2 \\
&= \sum_{i=1}^{n-1} q_i^2 h_i + \frac{1}{3}h_i q_{i+1}^2 + \frac{1}{3}h_i q_i^2 - \frac{2}{3}h_i q_i q_{i+1} + h_i q_i q_{i+1} - q_i^2 h_i \\
&= \sum_{i=1}^{n-1} \frac{1}{3}h_i(q_{i+1}^2 + q_i^2 + q_i q_{i+1}) \\
&= \frac{1}{3}[h_1(q_1^2 + q_2^2 + q_2 q_1) + h_2(q_2^2 + q_3^2 + q_2 q_3) + \cdots \\
&+ h_{n-2}(q_{n-1}^2 + q_{n-2}^2 + q_{n-2}q_{n-1}) + h_{n-1}(q_n^2 + q_{n-1}^2 + q_{n-1}q_n)] \\
&= \mathbf{q}^\top \mathbf{P^{(1)}} \mathbf{q}
\end{aligned}$$

où $\mathbf{q} = (q_1, q_2, \cdots, q_{n-1}, q_n)^\top$

où $\mathbf{P^{(1)}}$ est une matrice tridiagonale de dimension $(n \times n)$ et a pour coefficient

$$\begin{aligned}
&\mathbf{P^{(1)}}_{1,1} = \frac{h_1}{3} \\
&\mathbf{P^{(1)}}_{n,n} = \frac{h_{n-1}}{3}, \\
&\mathbf{P^{(1)}}_{i,i} = \frac{h_{i-1} + h_i}{3}, \quad i = 2, \ldots, n-1, \\
&\mathbf{P^{(1)}}_{i,i+1} = \mathbf{P^{(1)}}_{i+1,i} = \frac{h_i}{6}, \quad i = 1, \ldots, n-1
\end{aligned}$$

1.3 Spline cubique

Le but de cette section est de définir la matrice $\mathbf{P^{(2)}}$ à l'aide de l'égalité

$$\int_{t_1}^{t_n} \mid f''(t) \mid^2 dt = (u_1, \ldots, u_n)\mathbf{P^{(2)}}(u_1, \ldots, u_n)^\top.$$

Par identification nous obtenons :

$$\begin{aligned}
\int_{t_1}^{t_n} \mid f''(t) \mid^2 dt &= \sum_{i=1}^{n-1} \int_{t_i}^{t_i+1} \mid f''(t) \mid^2 dt \\
&= \sum_{i=1}^{n-1} \int_{t_i}^{t_{i+1}} \mid (u_i + (t-t_i)v_i) \mid^2 dt \\
&= \sum_{i=1}^{n-1} \int_{t_i}^{t_{i+1}} [u_i^2 + (t-t_i)^2 v_i^2 + 2u_i(t-t_i)v_i] dt \\
&= \sum_{i=1}^{n-1} [u_i^2 t + \frac{(t-t_i)^3}{3} v_i^2 + u_i(t-t_i)^2 v_i]_{t_i}^{t_{i+1}} \\
&= \sum_{i=1}^{n-1} (u_i^2 t_{i+1} + \frac{h_i^3}{3} v_i^2 + u_i h_i^2 v_i) - (t_i u_i^2) \\
&= \sum_{i=1}^{n-1} u_i^2 h_i + \frac{h_i^3}{3} v_i^2 + u_i v_i hi^2
\end{aligned}$$

Or $v_i = \dfrac{u_{i+1} - u_i}{h_i}$

$$\begin{aligned}
&= \sum_{i=1}^{n-1} u_i^2 h_i + \frac{h_i^3}{3} (\frac{u_{i+1} - u_i}{h_i})^2 + u_i \frac{u_{i+1} - u_i}{h_i} hi^2 \\
&= \sum_{i=1}^{n-1} u_i^2 h_i + \frac{h_i}{3} (u_{i+1}^2 + u_i^2 - 2u_i u_{i+1}) + u_i(u_{i+1} - u_i)h_i \\
&= \sum_{i=1}^{n-1} \frac{1}{3} h_i (u_{i+1}^2 + u_i^2 + u_i u_{i+1}) \\
&= \frac{1}{3}[h_1(u_1^2 + u_2^2 + u_2 u_1) + h_2(u_2^2 + u_3^2 + u_2 u_3) + \cdots \\
&+ h_{n-2}(u_{n-1}^2 + u_{n-2}^2 + u_{n-2}u_{n-1}) + h_{n-1}(u_n^2 + u_{n-1}^2 + u_{n-1}u_n)] \\
&= \mathbf{u}^\top \mathbf{P^{(2)}} \mathbf{u}
\end{aligned}$$

où $\mathbf{P^{(2)}}$ est une matrice tridiagonale de dimension $(n \times n)$ et a pour coefficient

$$\begin{aligned}
\mathbf{P^{(2)}}_{1,1} &= \frac{h_1}{3} \\
\mathbf{P^{(2)}}_{n,n} &= \frac{h_{n-1}}{3}, \\
\mathbf{P^{(2)}}_{i,i} &= \frac{h_{i-1} + h_i}{3}, \quad i = 2, \ldots, n-1, \\
\mathbf{P^{(2)}}_{i,i+1} &= \mathbf{P^{(2)}}_{i+1,i} = \frac{h_i}{6}, \quad i = 1, \ldots, n-1,
\end{aligned}$$

et $\mathbf{u}$= $(u_1, u_2, \cdots, u_{n-1}, u_n)^\top$

Nous remarquons que $\mathbf{P^{(0)}}$=$\mathbf{P^{(1)}}$=$\mathbf{P^{(2)}}$

2 Présentation des prévisions

Désormais, les h_i seront tous égaux à 1 (cela correspond à un intervalle de temps d'un an entre nos données). Le paramètre n correspond au nombre de données que nous avons en notre possession. Ici, $n = 118$. Le but de cette sous partie est, sachant les températures $p_1, \cdots, p_n$, de prévoir p_{n+1}.

Définition 5. Une prévision linéaire est déterminée par les poids $\omega_1, \cdots, \omega_n$ et est de la forme : $\sum_{i=1}^{n} \omega_i p_i$

Nous allons définir la matrice $\mathbf{P}$ de taille $(n+1 \times n+1)$ qui représente la matrice de variance-covariance des $(p_1, \cdots, p_{n+1})$ pour l'interpolation linéaire, $(q_1, \cdots, q_{n+1})$ pour la spline quadratique et des $(u_1, \cdots, u_{n+1})$ pour la spline cubique. Par identification avec la matrice $\mathbf{P^{(0)}}$, la matrice $\mathbf{P}$ est une matrice tridiagonale qui a pour coefficient :

$$\begin{aligned} \mathbf{P}_{1,1} &= \mathbf{P}_{n+1,n+1} = \frac{1}{3} \\ \mathbf{P}_{i,i} &= \frac{2}{3}, i = 2, \ldots, n \\ \mathbf{P}_{i,i+1} &= \mathbf{P}_{i+1,i} = \frac{1}{6}, i = 1, \ldots, n \end{aligned} \tag{2.1}$$

Propriété 1. *La matrice* $\mathbf{P}$ *est inversible.*

Démonstration. En effet, elle est a diagonale strictement dominante :

$$| \frac{1}{3} | > | \frac{1}{6} | \text{ et } | \frac{2}{3} | > | \frac{1}{6} + \frac{1}{6} |$$

□

Définition 6. Comme le suggère l'article présenté en bibliographie, nous considérerons la matrice $\mathbf{P}$ ou $\mathbf{P}^{-1}$ comme la matrice de variance-covariance du vecteur $\mathbf{p}$ dans le cas de l'interpolation linéaire, du vecteur $\mathbf{q}$ dans le cas de la spline quadratique et du vecteur $\mathbf{u}$ dans le cas de la spline cubique.

Montrons que le choix de $\mathbf{P}$ ou $\mathbf{P}^{-1}$ comme matrice de variance-covariance est raisonnable.

Propriété 2. *Une matrice de variance-covariance est symétrique définie positive.*

Démonstration. • Par construction, la matrice est symétrique.

- Montrons maintenant qu'elle est définie positive. Soit $\mathbf{x} \in \mathbb{R}^n$ un vecteur non nul.

$$\begin{aligned}
\mathbf{x}^\top \mathbf{P}\mathbf{x} &= (x_1, \ldots, x_n) \begin{pmatrix} \frac{x_1}{3} + \frac{x_2}{6} \\ \frac{x_1}{6} + \frac{2x_2}{3} + \frac{x_3}{6} \\ \vdots \\ \frac{x_{i-1}}{6} + \frac{2x_i}{3} + \frac{x_{i+1}}{6} \\ \vdots \\ \frac{x_{n-2}}{6} + \frac{2x_{n-1}}{3} + \frac{x_n}{6} \\ \frac{x_{n-1}}{6} + \frac{x_n}{3} \end{pmatrix} \\
&= \frac{x_1^2}{3} + \frac{x_1x_2}{6} + \frac{x_1x_2}{6} + \frac{2x_2^2}{3} + \frac{x_2x_3}{6} + \frac{x_3x_2}{6} + \frac{2x_3^2}{3} + \frac{x_3x_4}{6} + \ldots \\
&+ \frac{x_ix_{i-1}}{6} + \frac{2x_i^2}{3} + \frac{x_ix_{i+1}}{3} + \ldots + \frac{x_nx_{n+1}}{6} + \frac{x_n}{3} \\
&= \frac{1}{6}(2x_1^2 + 2x_1x_2 + 4x_2^2 + 2x_2x_3 + 4x_3^2 \\
&+ \ldots + 2x_ix_{i-1} + 4x_i^2 + 2x_ix_{i+1} + \ldots + 4x_nx_{n-1} + 2x_n^2) \\
&= \frac{1}{6}(\sum_{i=1}^{n-1}(x_i + x_{i+1})^2) + (x_1^2 + x_n^2 + \sum_{i=2}^{n-1} 2x_i^2) \\
&\geq 0
\end{aligned}$$

Résolvons $\mathbf{x}^\top \mathbf{P}\mathbf{x} = 0$
Comme $(\sum_{i=1}^{n-1}(x_i + x_{i+1})^2)$, $(x_1^2 + x_n^2 + \sum_{i=2}^{n-1} 2x_i^2)$ sont positifs car ce sont des sommes de carrés alors pour que ce soit nul, il faut que $x_1 = x_2 = \ldots = x_n = 0$, soit $\mathbf{x} = 0$.
La matrice $\mathbf{P}$ est donc symétrique définie positive. □

Pour toutes les prévisions que nous allons définir dans les sections suivantes, nous utiliserons cette matrice $\mathbf{P}$ mais également son inverse $\mathbf{P^{-1}}$.

Propriété 3. *Si une matrice est symétrique définie positive alors son inverse est également symétrique définie positive.*

Démonstration. • Elle est symétrique :
Si $\mathbf{A}$ est symétrique, alors $\mathbf{A} = \mathbf{A}^\top$
On a :

$$\begin{aligned}
Id &= \mathbf{A}\mathbf{A}^{-1} \\
\iff Id &= (\mathbf{A}^{-1})^\top \mathbf{A}^\top \\
\iff (\mathbf{A}^\top)^\top &= (\mathbf{A}^{-1})^\top \\
\iff \mathbf{A}^{-1} &= (\mathbf{A}^{-1})^\top
\end{aligned}$$

Donc $\mathbf{A}^{-1}$ est symétrique

- Elle est définie positive :
Comme $\mathbf{A}$ est symétrique, ses valeurs propres $\lambda_1, \ldots, \lambda_n$ sont réelles. De plus, elle est définie positive, donc ses valeurs propres sont strictement positives. Les valeurs propres de $\mathbf{A}^{-1}$ sont $\frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_n}$, qui sont donc elles aussi strictement positives. La matrice $\mathbf{A}^{-1}$ est donc définie positive.

□

En pratique, pour inverser la matrice $\mathbf{P}$, nous utiliserons :

- En C++ : la fonction inverse que nous avons implémentée qui repose sur la décomposition LU
- En R : la fonction *solve.tridiag*
  En effet, nous définirons un vecteur de taille $n+1$ contenant la diagonale et un autre vecteur de taille n contenant les valeurs de la sous-diagonale et sur-diagonale qui sont égales étant donné que la matrice $\mathbf{P}$ est symétrique. Puis nous trouverons $\mathbf{P^{-1}}$ en appelant la fonction *solve.tridiag* avec comme second membre la matrice identité de taille $n+1 \times n+1$.

Interprétation stochastique

Définition 7. Si on suppose que $(y_1, ..., y_{n+1})$ est une réalisation d'un vecteur Gaussien $(Y_1, ..., Y_{n+1})$ centré de matrice de covariance $\mathbf{Q}$ de dimension $n+1 \times n+1$. Alors la meilleure prévision de Y_{n+1} sachant $Y_1, \cdots, Y_n$ (au sens L^2) est égale à $\mathbb{E}[Y_{n+1}|Y_1, \cdots, Y_n] = \hat{Y}_{n+1}$.

Introduisons la matrice $\bar{\mathbf{Q}}$ qui est la restriction de la matrice $\mathbf{Q}$ c'est-à-dire $\bar{\mathbf{Q}}_{i,j} = \mathbf{Q}_{i,j}$ pour $i, j = 1, \ldots, n$
Nous avons les égalités suivantes :

$$\begin{aligned}
\mathbb{E}[Y_{n+1}|Y_1, \cdots, Y_n] &= (\mathbf{Q}_{n+1,1}, \cdots, \mathbf{Q}_{n+1,n})\bar{\mathbf{Q}}^{-1} \begin{pmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_n \end{pmatrix} \\
&= (\omega_1, \cdots, \omega_n) \begin{pmatrix} Y_1 \\ Y_2 \\ \cdots \\ Y_n \end{pmatrix} \\
&= \omega_1 Y_1 + \omega_2 Y_2 + \cdots + \omega_n Y_n
\end{aligned}$$

En effet, regardons la propriété suivante :

Propriété 4. *Soit $(Y, X_1, \cdots, X_d)$ un vecteur gaussien. Alors $\mathbb{E}(Y|X_1, \cdots, X_d)$ est une fonction affine de $(X_1, \cdots, X_d)$.*

Nous avons donc :

$$\mathbb{E}[Y_{n+1}|Y_1, \ldots, Y_n] = \sum_{i=1}^{n} \omega_i Y_i$$

Multiplions cette égalité par X_j pour $j = 1, \ldots, n$, nous obtenons les n équations suivantes :

$$X_j \mathbb{E}[Y_{n+1}|Y_1, \ldots, Y_n] = \sum_{i=1}^{n} \omega_i Y_i Y_j, \quad \text{j} = 1, \ldots, n$$

Les Y_j sont des constantes car ils sont présents dans la partie conditionnelle (ils sont $Y_1, ..., Y_n$ mesurable), par la linéarité de l'espérance, nous obtenons :

$$\begin{aligned} \mathbb{E}[Y_j \mathbb{E}[Y_{n+1}|Y_1, \ldots, Y_n]] &= \sum_{i=1}^{n} \omega_i \mathbb{E}[Y_i Y_j] \\ \iff \mathbb{E}[Y_j Y_{n+1}] &= \sum_{i=1}^{n} w_i \mathbb{E}[X_i X_j] \end{aligned}$$

Or, $\mathbb{E}[Y_j Y_{n+1}] = \mathbf{Q}_{j,n+1}$ et $\mathbb{E}[X_i X_j] = \mathbf{Q}_{i,j}$ Nous obtenons donc :

$$\begin{aligned} \sum_{i=1}^{n} \omega_i \mathbf{Q}_{i,j} &= \mathbf{Q}_{n+1,j} \quad \text{j} = 1, \ldots, n \\ \text{Soit : } (\omega_1, \ldots, \omega_n)\tilde{\mathbf{Q}} &= (\mathbf{Q}_{n+1,1}, \mathbf{Q}_{n+1,2}, \ldots, \mathbf{Q}_{n+1,n}) \end{aligned}$$

Interprétation déterministe

On observe la série temporelle y_1, ..., y_n et nous voulons prévoir la valeur y_{n+1} à la prochaine date $n+1$. Une prévision linéaire est la forme $\sum_{i=1}^{n} w_i y_i$ et est déterminée par les poids w_1, ..., w_n.

Afin de contrôler l'erreur de cette prévision nous avons besoin d'une matrice $\mathbf{K}^{-1}$ symétrique définie positive d'ordre $n+1 \times n+1$ et de munir l'espace $\mathbb{R}^{n+1}$ du produit scalaire

$$< u, v >_K = u^\top \mathbf{K}^{-1} v$$

Montrons que $< u, v >_K = u^\top \mathbf{K}^{-1} v$ est un produit scalaire.

Démonstration. Soit $u, v, w \in \mathbb{R}^{n+1}$ et $\lambda \in \mathbb{R}$

— $< u, u > \geq 0$

$$< u, u >_K = u^\top \mathbf{K}^{-1} u \geq 0$$

car $\mathbf{K}$ est symétrique définie positive ainsi $\mathbf{K}^{-1}$ est également symétrique définie positive

— $< u, u >_K = 0 \Leftarrow x = 0$
Cette condition est vérifiée étant donnée que la matrice $\mathbf{K}^{-1}$ est symétrique définie positive.

— $< \lambda u, v >_K = < u, \lambda v >_K = \lambda < u, v >_K$

$$< \lambda u, v >_K = \lambda u^\top \mathbf{K}^{-1} v = u^\top \mathbf{K}^{-1} \lambda v = \lambda (u^\top \mathbf{K}^{-1} v)$$

— $< u, v + w >_K = < u, v >_K + < v, w >_K$

$$< u, v + w >_K = u^\top \mathbf{K}^{-1}(v + w) = u^\top \mathbf{K}^{-1} v + u^\top \mathbf{K}^{-1} w = < u, v >_K + < v, w >_K$$

— $< u, v >_K = < v, u >_K$

$$< u, v >_K = u^\top \mathbf{K}^{-1} v = (u^\top \mathbf{K}^{-1} v)^\top = v^\top \mathbf{K}^{-1} u = < v, u >_K$$

□

Introduisons les notations suivantes :

$$\mathbf{y}=(\mathrm{y}_1, \ldots, y_{n+1}) \qquad \tilde{\omega} = (\omega_1, \ldots, \omega_n, -1) \qquad \mathbf{z} = (z_1, \ldots, z_{n+1})$$

et $\|.\|_e$ pour la norme euclidienne, définie par : $\|(x_1, \ldots, x_n)\|_e = \sum_{i=1}^n x_i^2$.
Ecrivons le lien entre $\|.\|_e$ et $\|.\|_K$: Pour $u \in \mathbb{R}^n$, on a

$$\begin{aligned} \|u\|_K^2 &= u^\top \mathbf{K}^{-1} u \\ \|u\|_K &= \sqrt{u^\top \mathbf{K}^{-1} u} \end{aligned}$$

Or, comme $\mathbf{K}$ est symétrique définie positive, on a : $\mathbf{K}^{-1} = \mathbf{K}^{-1/2}\mathbf{K}^{-1/2}$. D'où :

$$\begin{aligned} \|u\|_K &= \sqrt{(u^\top \mathbf{K}^{-1/2})(\mathbf{K}^{-1/2} u)} \\ &= \sqrt{\|\mathbf{K}^{-1/2} u\|_e} \end{aligned}$$

Propriété 5. *En utilisant le produit scalaire $\|.\|_K$, nous obtenons :*

$$\sup\{|y_{n+1} - \sum_{i=1}^n \omega_i y_i|^2 : \quad \mathbf{y}\mathbf{K}^{-1}\mathbf{y}^\top \leq 1\} = \tilde{\omega}\mathbf{K}\tilde{\omega}^\top$$

Démonstration. La preuve repose sur le changement de variable suivant :

$$\mathbf{z}^\top = \mathbf{K}^{-1/2}\mathbf{y}^\top$$

et le fait que

$$|y_{n+1} - \sum_{i=1}^n \omega_i y_i|^2 = |\tilde{\omega}\mathbf{y}^\top|^2 = \mathbf{z}\mathbf{K}^{1/2}\tilde{\omega}^\top\tilde{\omega}\mathbf{K}^{1/2}\mathbf{z}^\top.$$

(2.3)

En effet, si le vecteur que nous cherchons vérifie $\mathbf{y}\mathbf{K}^{-1}\mathbf{y}^\top \leq 1$, alors il est dans la boule unité. De plus, $\sup\{|y_{n+1} - \sum_{i=1}^{n} \omega_i y_i|^2\}$ correspond à chercher la plus grande erreur commise au carré.

$$\begin{aligned} |y_{n+1} - \sum_{i=1}^{n} \omega_i y_i|^2 &= |\ \tilde{\omega}\mathbf{y}^\top\ |^2 \\ &= |\ \mathbf{y}\tilde{\omega}^\top\ |^2 \\ &= \mathbf{y}\tilde{\omega}^\top \mathbf{y}\tilde{\omega}^\top \end{aligned}$$

Nous savons que : $u^\top v = v^\top u$
donc : $|\ u^\top v\ | = u^\top v u^\top v = u^\top v v^\top u$, on a, avec le changement de variable $\mathbf{K}^{1/2}\mathbf{z}^\top = \mathbf{y}^\top$:

$$|y_{n+1} - \sum_{i=1}^{n} \omega_i y_i|^2 = \mathbf{z}\mathbf{K}^{1/2}\tilde{\omega}^\top\tilde{\omega}\mathbf{K}^{1/2}\mathbf{z}^\top$$

$\mathbf{K}^{1/2}\tilde{\omega}^\top\tilde{\omega}\mathbf{K}^{1/2}$ est une matrice symétrique définie positive, notons la $\mathbf{A}$. On a :

$$\sup\{\mathbf{z}\mathbf{A}\mathbf{z}^\top : \|\mathbf{z}\| \leq 1\}$$

Il s'agit de la plus grande valeur propre de la matrice $\mathbf{A}$. $\mathbf{A}$ est de la forme $\mathbf{u}\mathbf{u}^\top$ si on note $\mathbf{u} = \mathbf{K}^{-1/2}\tilde{\omega}^\top$.

Cherchons la plus grande valeur propre de $\mathbf{u}\mathbf{u}^\top$: Soit $\mathbf{v}$ un vecteur propre de la valeur propre λ on a

$$\mathbf{u}\mathbf{u}^\top\mathbf{v} = \lambda\mathbf{v}$$

or $\mathbf{u}^\top\mathbf{v}$ est un scalaire, on a donc :

$$(\mathbf{u}^\top\mathbf{v})\mathbf{u} = \lambda\mathbf{v}$$

On a : $\gamma\mathbf{u} = \lambda\mathbf{v}$, (avec $\gamma = \mathbf{u}^\top\mathbf{v}$) $\mathbf{u}$ et $\mathbf{v}$ sont dons colinéaires, la matrice $\mathbf{A}$ a donc pour unique direction propre $\mathbf{u}$. Les vecteurs propres sont donc multiples de $\mathbf{u}$ de norme 1 : $\frac{\mathbf{u}}{\|\mathbf{u}\|}$.
Il y a donc une seule valeur propre qui est :

$$\mathbf{u}\mathbf{u}^\top\frac{\mathbf{u}}{\|\mathbf{u}\|} = \frac{\mathbf{u}^\top\mathbf{u}}{\|\mathbf{u}\|}\mathbf{u} = \|\mathbf{u}\|^2\frac{\mathbf{u}}{\|\mathbf{u}\|}$$

on a $\lambda = \|\mathbf{u}\|^2$.
Bilan : Nous cherchions la plus grande valeur propre de la matrice. Nous avons trouvé qu'il n'y avait qu'une seule valeur propre qui est $\|\mathbf{u}\|^2$. Nous cherchons donc :

$$\begin{aligned} \|\mathbf{u}\|^2 &= \|\mathbf{K}^{1/2}\tilde{\omega}\|^2 \\ &= \tilde{\omega}\mathbf{K}^{1/2}\mathbf{K}^{1/2}\tilde{\omega}^\top \\ &= \tilde{\omega}\mathbf{K}\tilde{\omega}^\top \end{aligned}$$

□

Finalement nous avons

$$\sup\{\mathbf{z}\mathbf{A}\mathbf{z}^\top : \quad \|z\| \leq 1\} = \tilde{\omega}\mathbf{K}\tilde{\omega}^\top$$

Les meilleurs poids sont ceux qui minimisent cette forme quadratique. Trouvons le gradient de $\tilde{\omega}\mathbf{K}\tilde{\omega}^\top$. Pour tout i allant de 1 à n, nous avons :

$$\begin{aligned}\frac{\delta}{\delta\omega_i}(\tilde{\omega}\mathbf{K}\tilde{\omega}^\top) &= (\frac{\delta}{\tilde{\delta}\omega_i}\tilde{\omega})\mathbf{K}\tilde{\omega}^\top + \tilde{\omega}\mathbf{K}(\frac{\delta}{\delta\omega_i}\tilde{\omega}^\top) \\ &= \underbrace{(0,\ldots,0,1,0,\ldots,0)\mathbf{K}}_{\text{i-ème ligne de } \mathbf{K}}\tilde{\omega}^\top + \tilde{\omega}\underbrace{\mathbf{K}(0,\ldots,0,1,0,\ldots,0)^\top}_{\text{i-ème colonne de } \mathbf{K}}\end{aligned}$$

Nous savons que la matrice $\mathbf{K}$ est symétrique, donc pour tout i, sa colonne i est égale à sa ligne i. Afin de trouver le minimum, nous résolvons pour tout i allant de 1 à n :

$$\begin{aligned}\frac{\delta}{\delta\omega_i}(\tilde{\omega}\mathbf{K}\tilde{\omega}^\top) &= 0 \\ \iff 2(\sum_{j=1}^{n}\mathbf{K}_{i,j}\omega_j - \mathbf{K}_{i,n+1}) &= 0\end{aligned}$$

Nous retrouvons alors le système d'ordre $n \times n$ suivant :

$$\sum_{j=1}^{n}\mathbf{K}_{i,j}\omega_j = \mathbf{K}_{i,n+1}$$

Remarque : Si nous nous plaçons dans la boule de rayon $R > 0$, nous avons :

$$\begin{aligned}&\mathbf{y}\mathbf{K}^{-\mathbf{1}}\mathbf{y}^\top \leq R \\ \iff &\frac{\mathbf{y}\mathbf{K}^{-\mathbf{1}}\mathbf{y}^\top}{R} \leq 1\end{aligned}$$

Or, $\frac{\mathbf{K}^{-1}}{R} = (R\mathbf{K})^{-1}$. On se ramène au même cas. $R\mathbf{K}$ est une matrice symétrique définie positive, donc par le même raisonnement, nous pouvons affirmer que l'erreur maximale sera donc multipliée par R. De plus, les poids ainsi trouvés ne dépendent que de la matrice $\mathbf{K}$ et non du coefficient R.

Dans la suite de ce chapitre, nous présenterons différentes prévisions : une pour l'interpolation linéaire et deux pour les splines quadratiques et cubiques. Nous serons donc amenées à résoudre le système linéaire suivant :

$$\sum_{i=1}^{n}\omega_i\mathbf{Q}_{i,j} = \mathbf{Q}_{n+1,j}, \quad j = 1,\ldots,n,$$

avec $\mathbf{Q}$ une matrice de dimension $n+1 \times n+1$ définie en fonction de $\tilde{\mathbf{P}}$ et $\omega_1,\ldots,\omega_n$ les inconnues.
La matrice $\tilde{\mathbf{P}}$ sera soit $\mathbf{P}$, soit $\mathbf{P}^{(-\mathbf{1})}$. Ainsi, chaque prévision que nous présenterons dans ce chapitre permettra de définir deux modèles :

- premier modèle : $\tilde{\mathbf{P}} = \mathbf{P}$
- deuxième modèle : $\tilde{\mathbf{P}} = \mathbf{P}^{-\mathbf{1}}$

Après avoir calculé ces inconnues, nous pourrons prédire la température à l'année $n+1$ que l'on notera $\hat{p}_{n+1}$ de la manière suivante :

$$\hat{p}_{n+1} = \sum_{i=1}^{n} \omega_i p_i.$$

Nous allons construire des modèles de prévision pour l'interpolation linéaire et la spline quadratique et cubique. Pour les splines, nous proposerons un modèle sans contrainte puis un modèle sous contrainte.

2.1 Interpolation linéaire

Pour les prévisions par interpolation linéaire, nous utilisons la matrice $\mathbf{Q} = \tilde{\mathbf{P}}$ définie précédemment.
Dans un premier temps, nous résoudrons le système suivant afin de déterminer le vecteur des poids ω de dimension n

$$\sum_{i=1}^{n} \omega_i \tilde{\mathbf{P}}_{i,j} = \tilde{\mathbf{P}}_{n+1,j}, \quad j = 1, \cdots, n \tag{2.4}$$

Ensuite, nous déterminerons $\hat{p}_{n+1}$:

$$\hat{p}_{n+1} = \sum_{i=1}^{n} \omega_i \mathbf{p}_i \tag{2.5}$$

Lors de l'application, nous comparerons ces deux modèles de prévision (le premier avec $\tilde{\mathbf{P}}= \mathbf{P}$ et le second avec $\tilde{\mathbf{P}}= \mathbf{P^{-1}}$). L'algorithme reprenant ces deux modèles est disponible en annexe B.1

2.2 Spline quadratique

2.2.1 Méthode avec contrainte

Dans un premier temps, exprimons le vecteur $(q_1, q_2, \ldots, q_{n+1})$, en fonction du vecteur $(p_1, p_2, \ldots, p_{n+1})$.
D'après la relation, 1.2 nous avons :

$$p_{i+1} - p_i = \frac{1}{2}(q_{i+1} + q_i)$$

En évaluant la relation précédente de 1 à i, nous obtenons le système suivant :

$$\begin{aligned} p_2 - p_1 &= \frac{1}{2}(q_2 - q_1) \\ p_3 - p_2 &= \frac{1}{2}(q_3 - q_2) \\ &\vdots \\ p_{i+1} - p_i &= \frac{1}{2}(q_{i+1} - q_i) \end{aligned}$$

En additionnant les i égalités précédentes, nous avons donc :

$$p_{i+1} - p_1 = \frac{1}{2}(q_1 - q_{i+1}) + \sum_{j=2}^{i} q_i$$

$$p_{i+1} = p_1 + \frac{1}{2}(q_1 - q_{i+1}) + \sum_{j=2}^{i} q_i$$

Pour obtenir un système à $n+1$ équations et $n+1$ variables, nous ajoutons la relation $p_1 = p_1$ et nous obtenons donc :

$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \\ p_{n+1} \end{pmatrix} = \begin{pmatrix} p_1 \\ p_1 + \frac{q_1}{2} + \frac{q_2}{2} \\ p_1 + \frac{q_1}{2} + q_2 + \frac{q_3}{2} \\ p_1 + \frac{q_1}{2} + q_2 + q_3 + \frac{q_4}{2} \\ \vdots \\ p_1 + \frac{q_1}{2} + q_2 + \cdots + q_n + \frac{q_{n+1}}{2} \end{pmatrix}$$

$$= p_1 \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} + \mathbf{R} \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \\ q_{n+1} \end{pmatrix}$$

Avec

$$\mathbf{R} = \begin{pmatrix} 0 & & \cdots & & & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \cdots & & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & \\ \frac{1}{2} & 1 & \cdots & 1 & & \frac{1}{2} \end{pmatrix}$$

Cherchons la moyenne du vecteur $(p_1, \ldots, p_{n+1})^\top$, sachant que le vecteur $(q_1, \ldots, q_{n+1})^\top$ est une réalisation d'un vecteur aléatoire centré de matrice de covariance $\tilde{\mathbf{P}}$ et grace à la linéarité de l'espérance et :

$$\mathbb{E}((p_1, \ldots, p_{n+1})^\top) = \begin{pmatrix} \mathbb{E}(p_1) \\ \mathbb{E}(p_1 + \frac{q_1}{2} + \frac{q_2}{2}) \\ \mathbb{E}(p_1 + \frac{q_1}{2} + q_2 + \frac{q_3}{2}) \\ \mathbb{E}(p_1 + \frac{q_1}{2} + q_2 + q_3 + \frac{q_4}{2}) \\ \vdots \\ \mathbb{E}(p_1 + \frac{q_1}{2} + q_2 + \cdots + q_n + \frac{q_{n+1}}{2}) \end{pmatrix}$$

$$= (q_1, q_1, \ldots, q_1)^\top$$

$(p_1, \ldots, p_{n+1})^\top$ devient vecteur aléatoire de moyenne $p_1(1, \ldots, 1)^\top$ et de covariance $\mathbf{Q} = \mathbf{R}\tilde{\mathbf{P}}^{-1}\mathbf{R}^\top$.
On résout le système

$$\begin{cases} \sum_{i=1}^{n} \omega_i \mathbf{Q}_{i,j} + \lambda = \mathbf{Q}_{n+1,j}, \quad j = 1, \ldots, n, \\ \\ \sum_{i=1}^{n} \omega_i = 1 \end{cases}$$

où $\omega_1, \ldots, \omega_n, \lambda$ sont les inconnues et $\mathbf{Q}_{i,j}$ les coefficients de $\mathbf{Q}$.

Finalement nous calculons :

$$\hat{p}_{n+1} = \sum_{i=1}^{n} \omega_i p_i \tag{2.6}$$

Pour le code en annexe B.2 : Nous avons codé cette méthode comme résolution du système suivant :

$$\mathbf{Dx} = \mathbf{b}$$

$$\text{avec } \mathbf{D} = \begin{pmatrix} & & & 1 \\ & \mathbf{Q}_{i,j} & & 1 \\ & & & 1 \\ & & & \vdots \\ & & & 1 \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix} \text{ avec } i, j = 1, \ldots, n, \quad \text{x} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \\ \lambda_1 \end{pmatrix} \text{ et b} = \begin{pmatrix} \mathbf{Q}_{1,n+1} \\ \mathbf{Q}_{2,n+1} \\ \vdots \\ \mathbf{Q}_{n,n+1} \\ 1 \end{pmatrix}$$

2.2.2 Méthode sans contrainte : par prévision de q_{n+1}

Nous allons donc d'abord calculer une estimation de q_{n+1}. Pour ça, résolvons le système linéaire suivant :

$$\sum_{i=1}^{n} \omega_i \tilde{\mathbf{P}}_{i,j} = \tilde{\mathbf{P}}_{n+1,j}, \quad j = 1, \ldots, n,$$

avec $\omega_1, \ldots, \omega_n$ les inconnues et $\tilde{\mathbf{P}}_{i,j}$ les coefficients de $\mathbf{P}$ ou de son inverse $\mathbf{P^{-1}}$ que nous avons défini ci-dessus 2.1. Ayant calculé ces inconnues, nous obtenons la prévision :

$$\hat{q}_{n+1} = \sum_{i=1}^{n} \omega_i q_i.$$

Avec cette prévision $\hat{q}_{n+1}$, nous allons déterminer une prévision de p_{n+1}.

Grace à la relation (1.2), nous avons donc la prévision de p_{n+1} par la relation :

$$\hat{p}_{n+1} = \frac{1}{2} q_n + \frac{1}{2} \hat{q}_{n+1} + p_n \tag{2.7}$$

Cet algorithme est disponible en annexe B.3

2.3 Spline cubique

2.3.1 Méthode avec contraintes

Exprimons $(p_1, \ldots, p_{n+1})$ en fonction de $(p_1, p_2, u_1, \ldots, u_{n+1})$. Reprenons les relations de continuités définies en 1.4. Nous avons :

$$\begin{aligned} p_{i+1} - p_i &= q_i + \frac{1}{2}u_i + \frac{1}{6}v_i, \quad i = 1, \cdots, n \\ \text{ou encore : } p_{i+2} - p_{i+1} &= q_{i+1} + \frac{1}{2}u_{i+1} + \frac{1}{6}v_{i+1} \quad i = 0, \cdots, n-1 \end{aligned}$$

En soustrayant, nous obtenons pour $i = 1, \cdots, n-1$:

$$\begin{aligned} p_{i+2} - 2p_{i+1} + p_i &= q_{i+1} - q_i + \frac{1}{2}(u_{i+1} - u_i) + \frac{1}{6}(v_{i+1} - v_i) \\ &= \frac{1}{2}(u_{i+1} + u_i) - \frac{1}{2}(u_{i+1} - u_i) + \frac{1}{6}(u_{i+2} - 2u_{i+1} + u_i) \\ &= \frac{1}{6}u_{i+2} + \frac{2}{3}u_{i+1} + \frac{1}{6}u_i. \end{aligned} \tag{2.8}$$

C'est un système à $n-1$ équations et $n+1$ inconnues $p_1, \ldots, p_{n+1}$. Pour avoir le même nombre d'équations et d'inconnues, nous ajoutons les deux équations suivantes :

$$p_1 = p_1, \quad p_2 = p_2$$

Nous obtenons maintenant un système de la forme :

$$\mathbf{A}(p_1, \ldots, p_{n+1})^\top = (p_1, p_2, \frac{u_1}{6} + \frac{u_2}{3} + \frac{u_3}{6}, \ldots, \frac{u_{n-1}}{6} + \frac{u_n}{3} + \frac{u_{n+1}}{6})^\top$$

où $\mathbf{A}$ est une matrice de dimension $n+1 \times n+1$ ayant pour coefficients non nuls :

$$\begin{aligned} &\mathbf{A}_{1,1} = \mathbf{A}_{2,2} = 1 \\ &\mathbf{A}_{i,i-1} = -2, \quad i = 3, \ldots, n+1, \\ &\mathbf{A}_{i,i} = \mathbf{A}_{i,i-2} = 1, \quad i = 3, \ldots, n+1, \end{aligned}$$

Le second membre $(p_1, p_2, \frac{u_1}{6} + \frac{u_2}{3} + \frac{u_3}{6}, \ldots, \frac{u_{n-1}}{6} + \frac{u_n}{3} + \frac{u_{n+1}}{6})^\top$ s'écrit sous la forme :

$$\mathbf{B}(p_1, p_2, u_1, \ldots, u_{n+1})$$

où $\mathbf{B}$ est une matrice de dimension $n+1 \times n+3$ ayant pour coefficients non nuls :

$$\begin{aligned} &\mathbf{B}_{1,1} = \mathbf{B}_{2,2} = 1 \\ &\mathbf{B}_{i,i+1} = \frac{1}{3}, \quad i = 3, \ldots, n+1, \\ &\mathbf{B}_{i,i} = \mathbf{B}_{i,i+2} = \frac{1}{6}, \quad i = 3, \ldots, n+1 \end{aligned}$$

Finalement, nous obtenons

$$\begin{aligned} (p_1, \ldots, p_{n+1})^\top &= \mathbf{A}^{-1}\mathbf{B}(p_1, p_2, u_1, \ldots, u_{n+1})^\top \\ &= p_1\mathbf{A}^{-1}\mathbf{B}(1, 0, , \ldots, 0)^\top + p_2\mathbf{A}^{-1}\mathbf{B}(0, 1, 0, , \ldots, 0)^\top \\ &+ \mathbf{A}^{-1}\mathbf{B}(0, 0, u_1, \ldots, u_{n+1})^\top. \end{aligned}$$

Nous définissons la matrice $\mathbf{P^{invbig}}$ de taille $n+3 \times n+3$ telle que :

$$\mathbf{P^{invbig}}_{i+2,j+2} = \tilde{\mathbf{P}}_{i,j} \quad \text{pour } i = 1, \ldots, n+1$$

Autrement dit $\mathbf{P^{invbig}}$ est de la forme :

$$\mathbf{P^{invbig}} = \begin{pmatrix} 0 & 0 & \cdots & & \cdots & 0 \\ 0 & 0 & \cdots & & \cdots & 0 \\ \vdots & \vdots & & \tilde{\mathbf{P}} & & \\ 0 & 0 & & & & \end{pmatrix}$$

Ensuite, nous définissons la matrice $\mathbf{Q}$ de taille $n+1 \times n+1$ par la produit suivant :

$$\mathbf{Q} = \mathbf{A^{-1}} \times \mathbf{B} \times \mathbf{P^{invbig}} \times \mathbf{B}^\top \times \mathbf{A^{-1}}^\top$$

On résout le système :

$$\begin{cases} \sum_{i=1}^{n} \omega_i \mathbf{Q}_{i,j} + \lambda_1 + \lambda_2 j = \mathbf{Q}_{n+1,j}, \quad j = 1, \ldots, n, \\ \sum_{i=1}^{n} \omega_i = 1 \\ \sum_{i=1}^{n} i\omega_i = n+1 \end{cases}$$

Finalement nous calculons :

$$\hat{p}_{n+1} = \sum_{i=1}^{n} \omega_i p_i \tag{2.9}$$

Pour le code en annexe B.4 : Nous avons codé cette méthode comme résolution du système suivant :

$$\mathbf{Dx} = \mathbf{b}$$

avec $\mathbf{D}=\begin{pmatrix} & & & & 1 & 1 \\ & \mathbf{Q}_{i,j} & & & 1 & 2 \\ & & & & 1 & 3 \\ & & & & \vdots & \vdots \\ & & & & 1 & N \\ 1 & 1 & \cdots & 1 & 0 & 0 \\ 1 & 2 & \cdots & N & 0 & 0 \end{pmatrix}$ avec $i, j = 1, \ldots, n$, $\quad \text{x}= \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \\ \lambda_1 \\ \lambda_2 \end{pmatrix}$ et $\text{b} = \begin{pmatrix} \mathbf{Q}_{1,n+1} \\ \mathbf{Q}_{2,n+1} \\ \vdots \\ \mathbf{Q}_{n,n+1} \\ 1 \\ N+1 \end{pmatrix}$

2.3.2 Méthode sans contrainte

Dans un premier temps, nous posons :

$$\tilde{p_i} = p_{i+2} - 2p_{i+1} + p_i, \quad i = 1, \ldots, n-1,$$

En reprenant 2.8, nous définissons le système :

$$\tilde{p_i} = \frac{u_i}{6} + \frac{u_{i+1}}{3} + \frac{u_{i+2}}{6}, \quad i = 1, \ldots, n-1,$$

Nous pouvons l'écrire sous forme matricielle :

$$\tilde{\mathbf{p}} = \mathbf{M}\mathbf{u}$$

où

$$\tilde{\mathbf{p}} = (\tilde{p}_1, \ldots, \tilde{p}_{n-1})^\top \quad \text{et} \quad \mathbf{u} = (u_1, \ldots, u_{n+1})^\top$$

et $\mathbf{M}$ est une matrice d'ordre $n-1 \times n+1$ définie pour $i = 1, \cdots, n-1$ et pour $j = 1, \cdots, n+1$ par :

$$\begin{aligned} \mathbf{M_{i,i}} = \mathbf{M_{i,i+2}} &= \frac{1}{6} \\ \mathbf{M_{i,i+1}} &= \frac{1}{3} \\ \mathbf{M_{i,j}} &= 0 \quad \text{sinon} \end{aligned}$$

En supposant que $\mathbf{u}$ est gaussien centré de matrice de covariance $\tilde{\mathbf{P}}$, alors nous en déduisons que $\tilde{\mathbf{p}}$ est également gaussien centré de matrice de covariance $\mathbf{Q} = \mathbf{M}\tilde{\mathbf{P}}\mathbf{M}^\top$.

Montrons que $\tilde{\mathbf{p}}$ est bien un vecteur gaussien. En effet, $\tilde{\mathbf{p}}$ est une application linéaire du $\mathbf{u}$.

Lemme 1. *La fonction caractéristique d'un vecteur gaussien $X \simeq \mathcal{N}(m, \Sigma)$ est :*

$$\varphi_X(t) = e^{-\frac{1}{2}(t\Sigma t)^t} \times e^{i\langle t, m\rangle}$$

Démonstration. $\forall t = \begin{pmatrix} t_1 \\ \vdots \\ t_d \end{pmatrix} \in \mathbb{R}^d$, on a :

$$\begin{aligned} \varphi_X(t) &= E\left(e^{i\langle t, \Sigma^{\frac{1}{2}} Z + m\rangle}\right) \\ &= E\left(e^{i\langle t, \Sigma^{\frac{1}{2}} Z\rangle}\right) e^{i\langle t, m\rangle} \\ &= \underbrace{E\left(e^{i\langle (\Sigma^{\frac{1}{2}})^t t, Z\rangle}\right)}_{\varphi_Z((\Sigma^{\frac{1}{2}})^t t)} e^{i\langle t, m\rangle} \\ &= e^{-\frac{1}{2}(t\Sigma t)^t + i\langle t, m\rangle} \\ &= e^{-\frac{1}{2}(t\Sigma t)^t} e^{i\langle t, m\rangle} \end{aligned}$$

□

Propriété 6. *Si $X \hookrightarrow \mathcal{N}(m, \Sigma)$ et A matrice quelconque de même taille que Σ, alors $AX \hookrightarrow \mathcal{N}(Am, A\Sigma A^t)$.*

Démonstration. On fait le calcul :

$$\begin{aligned}\varphi_{AX}(t) &= E\left(e^{i\langle t, AX\rangle}\right) \\ &= \varphi_X(A^t t) \\ &= e^{i\langle A^t t, m\rangle - t^t A\Sigma A^t t\frac{1}{2}} \\ &= e^{i\langle t, Am\rangle - t^t A\Sigma A^t\frac{1}{2}}\end{aligned}$$

Et la propriété est obtenue par le lemme précédent.

□

Par conséquent, comme $\tilde{\mathbf{p}} = \mathbf{Mu}$, $\tilde{\mathbf{p}} \hookrightarrow \mathcal{N}(0, \mathbf{M}\tilde{\mathbf{P}}\mathbf{M}^t)$

Pour cette seconde prévision, il s'agit donc de résoudre le système suivant :

$$\sum_{i=1}^{n-2} \omega_i \mathbf{Q_{i,j}} = \mathbf{Q_{n-1,j}}, \quad j = 1, \ldots, n-1.$$

Ayant trouvé les poids ω_i, nous pouvons ensuite calculer la prévision à l'année $n+1$:

$$\hat{p}_{n+1} = 2p_n - p_{n-1} + \sum_{i=1}^{n-2} \omega_i \tilde{p}_i, \tag{2.10}$$

L'algorithme en annexe B.5 reprend ce modèle.

Chapitre 3

Prévisions

Dans ce chapitre, nous présentons les résultats des prévisions obtenues lorsque nous avons implémenté les algorithmes des diverses prévisions présentées au chapitre précédent. Nous avons d'abord codé toutes les prévisions en C++, puis au vu des résultats, nous avons utilisé le logiciel R pour étudier les matrices. En effet, nous souhaitions avoir les valeurs propres, le conditionnement des matrices de nos systèmes. Nous avons donc poursuivi la partie programmation sur R car ce logiciel met à notre disposition une multitude de fonctions permettant d'étudier nos matrices contrairement au logiciel C++ où tout nécessitait d'être programmé.

1 Interpolation linéaire

1.1 Prévision avec la matrice $\mathbf{P^{-1}}$

Pour cette première prévision de l'interpolation linéaire, nous utilisons la matrice P^{-1}. Après avoir résolu le système 2.5, nous obtenons une prévision de -8.90011°. Ce résultat semble très mauvais. Afin de comprendre cette erreur importante, nous avons étudié la matrice de ce modèle.
Dans un premier temps, nous avons tracé la répartition des poids en fonction des années et obtenons le graphique suivant.

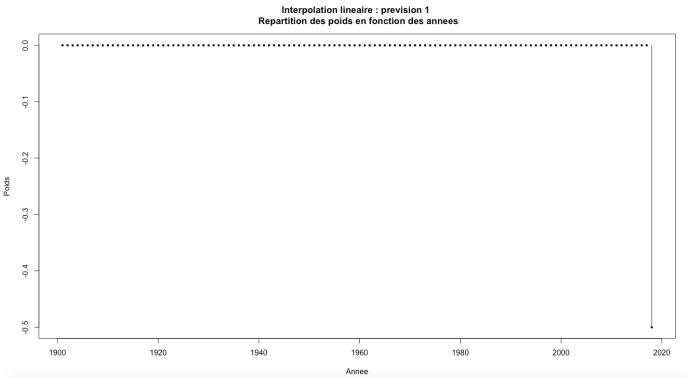


FIGURE 3.1.1 – Répartition des poids ω_i en fonction des années

Nous constatons que seule la dernière année contribue à notre prévision. En effet, les poids représentants les températures de 1901 à 2017 sont quasiment nuls. Seul le poids associé à l'année 2018 est significatif. Ce dernier vaut -0.5 c'est pourquoi nous obtenons une prévision négative.
Afin de comprendre l'erreur engendrée par ce modèle, nous nous sommes intéressées aux propriétés de la matrice $\mathbf{P^{-1}}$ du système que nous avons résolu pour déterminer les poids. En effet, nous avons calculé son conditionnement, sa valeur propre minimale en module avec le logiciel R.

Définition 8. Soit $\|.\|$ une norme matricielle définie sur $\mathbb{M}_n(\mathbb{K})$ et $\mathbf{A}\in \mathbb{M}_n(\mathbb{K})$ une matrice inversible. On appelle conditionnement de $\mathbf{A}$ par rapport à la norme $\|.\|$, le nombre $cond(\mathbf{A}) = \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$. Si $\|.\| = \|.\|_p$ avec $p = 1, 2, \ldots$, on note $cond_p(\mathbf{A}) = \|\mathbf{A}\|_p\|\mathbf{A}^{-1}\|_p$

Le tableau ci-dessous présente les résultats que nous avons obtenus :

Conditionnement	3.999527
Valeur propre minimale en module	1.000118

TABLE 3.1 – Propriétés de la matrice P^{-1}

Nous constatons que cette matrice est bien conditionnée. Le modèle est donc correct mais est mauvais étant donné qu'il ne donne pas une prévision réaliste.

Afin de corriger ce modèle, nous avons essayé de diviser la prévision par la somme des poids, ce qui revient à modifier la matrice de covariance. En effet en divisant chaque coté de l'égalité 2.4 nous avons finalement le système suivant :

$$\sum_{i=1}^{n} \frac{\omega_i}{\sum_{l=1}^{n} \omega_l} \mathbf{K}_{i,j} = \frac{\mathbf{K}_{n+1,i}}{\sum_{l=1}^{n} \omega_l}$$

De manière générale, si nous multiplions par λ, nous avons :

$$\lambda \sum_{i=1}^{n} \omega_i \mathbf{K}_{i,j} = \lambda \mathbf{K}_{n+1,i}$$

Nous avons le vecteur $(\lambda\omega_1, \ldots, \lambda\omega_n)$ qui représente les poids associés à la matrice $\tilde{\mathbf{K}}$ pour i et j allant de 1 à n :

$$\tilde{\mathbf{K}} = \begin{pmatrix} & & & \lambda\mathbf{K}_{1,n+1} \\ & \mathbf{K}_{i,j} & & \lambda K_{2,n+1} \\ & & & \lambda K_{3,n+1} \\ & & & \vdots \\ & & & \lambda K_{n,n+1} \\ \lambda K_{n+1,1} & \lambda K_{n+1,2} & \cdots & \lambda K_{n+1,n} \quad \lambda^2 K_{n+1,n+1} \end{pmatrix}$$

Soit :

$$\tilde{\mathbf{K}}_{i,j} = \begin{cases} \mathbf{K} \quad \text{si } \{i,j\} = 1, \ldots, n \\ \lambda\mathbf{K} \quad \text{si } \{i \text{ ou } j\} = n+1 \\ \lambda^2 \mathbf{K}_{n+1,n+1} \quad \text{si } i = j = n+1. \end{cases}$$

Minimiser l'erreur maximale revient à :

$$\min_{\omega_1,\ldots,\omega_n} \sup\{|y_{n+1} - \sum_{i=1}^{n} \omega_i y_i|^2 : \quad \mathbf{y}\tilde{K}^{-1}\mathbf{y}^\top \leq 1\} = \min_{\omega_1,\ldots,\omega_n} \tilde{\omega}\tilde{\mathbf{K}}\tilde{\omega}^\top$$

$$= \lambda^2 K_{n+1,n+1} - (\lambda K_{n+1,1}, \ldots, \lambda K_{n+1,n})\mathbf{K}^{-1} \begin{pmatrix} \lambda K_{1,n+1} \\ \vdots \\ \lambda K_{n,n+1} \end{pmatrix}$$

$$= \lambda^2 (K_{n+1,n+1} - (K_{n+1,1}, \ldots, K_{n+1,n})\mathbf{K}^{-1} \begin{pmatrix} K_{1,n+1} \\ \vdots \\ K_{n,n+1} \end{pmatrix})$$

$$= \lambda^2 (\min_{\omega_1,\ldots,\omega_n} \tilde{\omega}\mathbf{K}\tilde{\omega}^\top)$$

Cela revient donc à multiplier l'erreur par λ^2

Finalement, la nouvelle prévision obtenue est bien meilleure : 17.8°. Nous pouvons conjecturer que ce modèle modifié donne une prévision réaliste. Afin de valider notre conjecture, vérifions ce modèle ajusté sur des températures déjà connues. Le tableau ci-dessous résume les résultats :

Année	Température estimée	Température mesurée
2018	18.12502	17.80022
2017	19.10102	18.12502
2016	18.9008	19.10102
2015	19.20845	18.9008

TABLE 3.2 – Test du modèle ajusté sur quelques années

Conclusion : Les prévisions pour les années étudiées étant assez proches des valeurs mesurées, ce modèle, qui nous donne une prévision de 17.80°, nous semble pertinent pour prédire la température au Maroc en 2019.

1.2 Prévision avec la matrice P

La seconde prévision de l'interpolation linéaire repose sur un raisonnement analogue comme nous l'avons explicité au second chapitre, cependant, nous utilisons la matrice **P**. Ce modèle nous donne une prévision pour 2019 de 3.759465°. Cette prévision ne semble pas correcte, nous avons donc repris le même raisonnement que pour la prévision précédente. Tout d'abord, le graphique ci-dessous représente la répartition des poids en fonction des années.

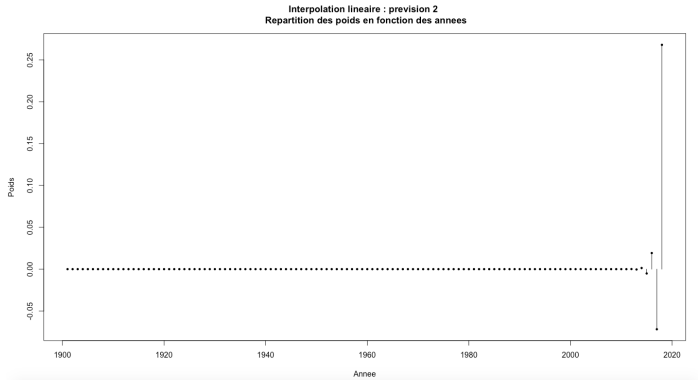


FIGURE 3.1.2 – Répartition des poids ω_i en fonction des années

Nous constatons sur ce graphique que seuls les poids associés aux quatre dernières années semblent être les plus importants. Ils valent pour les années 2015, 2016, 2017 et 2018 respectivement environ -0.005, 0.019, -0.072 et 0.27.
Comme ci-dessus, nous avons calculé le conditionnement et la valeur propre minimale en module de la matrice du système et obtenons les résultats suivants :

Conditionnement	3.99953
Valeur propre minimale en module	0.25

TABLE 3.3 – Propriétés de la matrice P

La matrice **P** est bien conditionnée. Ce modèle semble donc mauvais.

Par le même raisonnement que ci-dessus, c'est-à-dire, en divisant la prévision obtenue par la somme des poids, nous obtenons une prévision très correcte : 17.79°. Nous avons également calculé la prévision pour les années de 2015 à 2018 que nous avons à notre disposition afin d'évaluer l'efficacité de ce modèle corrigé.

Année	Température estimée	Température mesurée
2018	17.83844	17.80022
2017	19.19455	18.12502
2016	18.75194	19.10102
2015	19.45634	18.9008

TABLE 3.4 – Test du modèle ajusté sur quelques années

Conclusion : Les températures estimées sont assez proches de nos données. Nous pouvons donc penser que ce modèle corrigé est un bon modèle de prédiction. Nous obtenons alors une prévision de 17.79°.

2 Spline quadratique

2.1 Prévision avec contrainte

2.1.1 Prévision avec la matrice $\mathbf{P^{-1}}$

Cette première prévision, en résolvant le système 2.6 avec la matrice P^{-1}, nous donne une température au Maroc en 2019 de 10.55478°. Nous trouvons ci-dessous la répartition des poids en fonction des années.

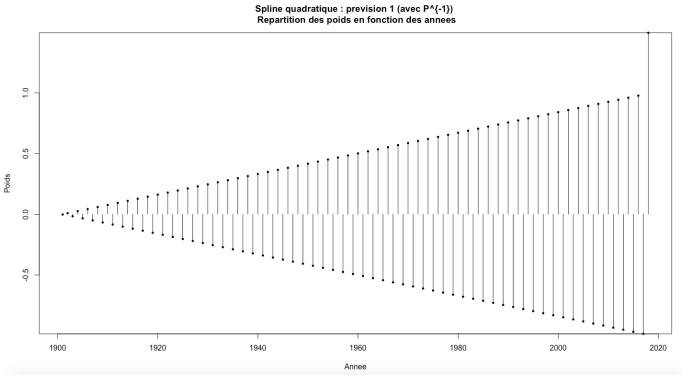


FIGURE 3.2.3 – Répartition des poids ω_i en fonction des années

Nous constatons que plus les années augmentent, plus les poids semblent contribuer de manière importante à la prévision. Cependant, le dernier poids correspondant à l'année 2018 est plus important que les autres et vaut environ 1.5. Cette prévision n'étant pas réaliste, nous avons étudié la matrice **D** de notre système final permettant de trouver les poids.

Conditionnement	41933451
Valeur propre minimale en module	0.0001334371

TABLE 3.5 – Propriétés de la matrice $\mathbf{P^{-1}}$

Ce tableau nous montre que la matrice **D** est mal conditionnée.

Remarque : Soit le système **Ax=b** où $\mathbf{x} \in \mathbb{R}^n$ est le vecteur des inconnues, **A** est une matrice réelle inversible de dimension $n \times n$ et $\mathbf{b} \in \mathbb{R}^n$ un vecteur. Si **A** est mal conditionnée, une petite variation sur le vecteur **b** entraîne une grande variation sur notre vecteur solution **x**. Ces erreurs peuvent provenir par exemple d'erreurs de mesures ou d'erreurs dans la représentation en virgule flottante des nombres.
Par ailleurs, elle a au moins une valeur propre très proche de 0. Ainsi, cette matrice semble entraîner de mauvaises prévisions. Avant de juger ce modèle, nous devons donc résoudre les problèmes liés à ce mauvais conditionnement.
Pour résoudre ce problème auquel nous serons confrontées plusieurs fois dans ce chapitre, nous mettons en place deux méthodes que nous présentons ci-dessous.

Présentation des méthodes

- **Méthode 1 : utiliser l'inverse généralisée**

  L'inverse généralisée sous le logiciel R est définie par la fonction *ginv*. En utilisant l'élimination de Gauss, la fonction *ginv* prend une matrice réelle $\mathbf{A}$ en paramètre et renvoie une matrice $\mathbf{A}^\dagger$ telle que : $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$. *ginv* peut donc prendre en paramètre des matrices rectangulaires, non inversibles, que l'on ne pouvait pas prendre avec d'autres fonctions telles que *solve*. L'intérêt d'utiliser *ginv* est de créer une nouvelle matrice que l'on considérera comme l'inverse de la matrice entrée en paramètre. Cette nouvelle inverse remplacera celle que l'on trouvait avec la fonction *solve*.

- **Méthode 2 : ajouter un paramètre de lissage**

  La méthode de lissage appliquée à une matrice $\mathbf{A}$ consiste à introduire un paramètre λ réel positif afin de créer une nouvelle matrice $\tilde{\mathbf{A}} = \mathbf{A} + \lambda Id$
  L'intérêt du paramètre de lissage est de construire, à partir d'une matrice initiale $\mathbf{A}$ ayant pour valeurs propres $\nu_1, \ldots, \nu_n$ avec au moins une valeur propre très proche de zéro, une nouvelle matrice $\tilde{\mathbf{A}}$ où les valeurs propres sont $\nu_1 + \lambda, \ldots, \nu_n + \lambda$.

  Remarque : Le réel λ doit être à la fois suffisamment grand pour éloigner la valeur propre de zéro et suffisamment petit pour ne pas s'éloigner notre modèle.

En appliquant la méthode 1, nous n'obtenons pas un résultat satisfaisant puisque nous retrouvons une prévision de 10.55478°. Nous allons donc appliquer la méthode 2. Pour cela, nous devons sélectionner le paramètre λ optimal. Pour ce faire, nous traçons l'évolution du conditionnement de la matrice $\mathbf{D}$ en fonction de λ qui varie ici entre 0 et 5.

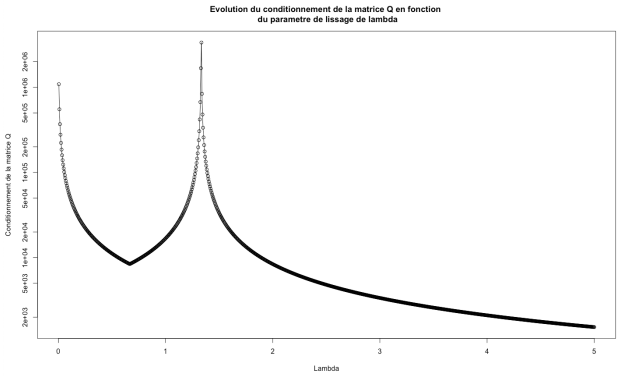


FIGURE 3.2.4 – Conditionnement en fonction de lambda variant entre 0 et 5

Nous constatons que pour $\lambda \in [0, 0.665]$, le conditionnement de la matrice diminue fortement, puis augmente brutalement jusqu'à environ 1.5. Ensuite, le conditionnement chute jusque 1527. Nous souhaitons une valeur de λ assez petite pour diminuer le conditionnement de manière significative sans trop s'éloigner de notre matrice initiale. C'est pourquoi nous nous concentrerons sur $\lambda \in [0; 1]$

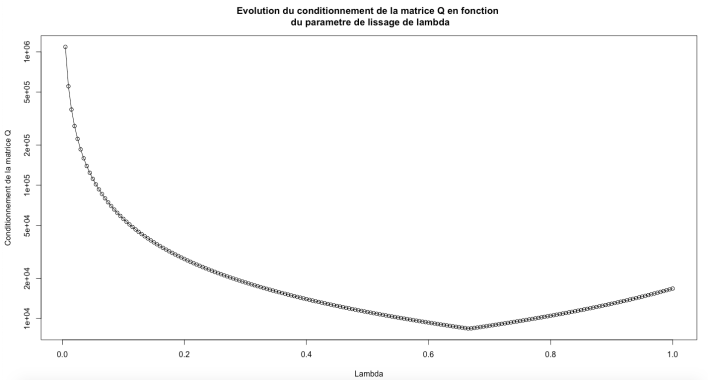


FIGURE 3.2.5 – Conditionnement en fonction de lambda

Il semblerait que le conditionnement soit minimal pour une valeur de λ située aux alentours de 0.65.

Finalement, nous avons tracé l'évolution de la prévision en fonction de λ afin de choisir celui qui nous donnera la prévision la plus réaliste.

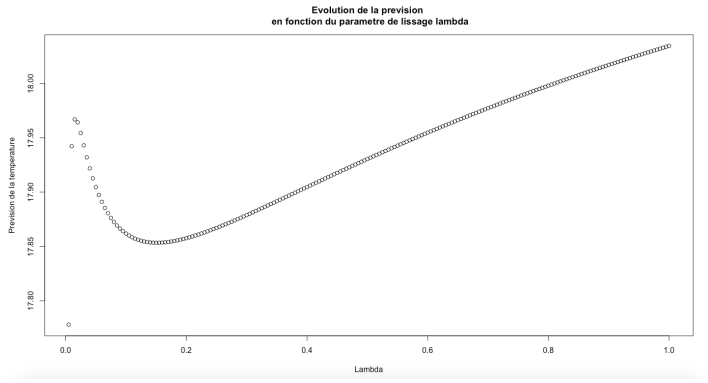


FIGURE 3.2.6 – Prévision en fonction de lambda variant entre 0 et 1

Sur ce graphique, nous constatons que la température varie entre 17,75° et 18,05° pour λ variant entre 0 et 1. Pour des valeurs de λ très petites, la température croit. Ensuite, elle décroît avant de croître à nouveau pour λ proche de 0.15.

Au vu de ces deux graphiques, nous allons sélectionner plusieurs valeurs de λ proche de celle donnant le conditionnement minimal ainsi que quelques valeurs extrêmes. Nous calculerons ensuite les prévisions que nous obtenons pour l'année 2019 mais également celles pour l'année 2018. En effet, nous avons à notre disposition la température au Maroc en 2018, nous choisirons ainsi une valeur de lambda réunissant quatre critères : un conditionnement raisonnable, une valeur de λ assez faible, une prévision pour 2018 proche de la température mesurée et enfin une prévision pour 2019 réaliste.
Nous étudions donc les paramètres de lissage λ suivants :

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.01	552180.8	17.80381	17.9422
0.1	55881.22	17.99501	17.86181
0.5	11188.97	18.31635	17.93042
0.6	9324.725	18.3499	17.95471
0.665	8413.566	18.36814	17.96956
0.7	8836.072	18.37702	17.97723
0.8	10493.02	18.39941	17.99797
3	3359.087	18.5349	18.2127
5	1527.403	18.53619	18.27079

TABLE 3.6 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

Nous constatons que tous les paramètres de lissage donnent des prévisions cohérentes pour les années 2018 et 2019. La meilleure prévision pour l'année 2018 est celle de λ=0.01. Cependant, choisir $\lambda = 0.1$ permet de diviser le conditionnement par 10. Nous choisirons cette valeur.

Avec ce choix de lambda, la répartition des poids est la suivante et nous obtenons donc comme prévision pour 2019 environ 17.86°.

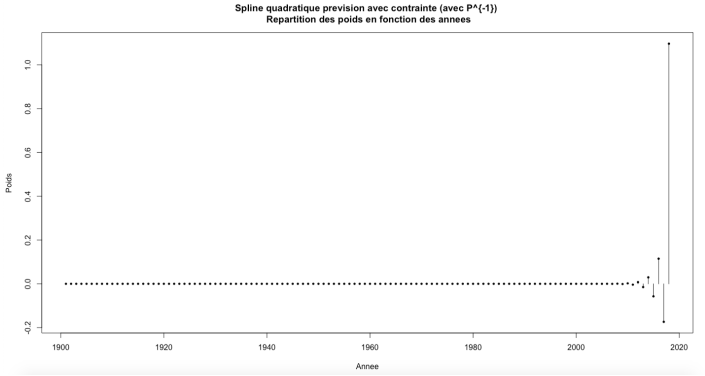


FIGURE 3.2.7 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.1$

Nous observons que les années qui contribuent le plus à notre prévision sont les années de 2013 à 2018, notamment, l'année 2018 avec une contribution au minimum cinq fois supérieure aux précédentes (il vaut 1.09).

Conclusion : Numériquement, le système 2.6 défini au chapitre 2 ne nous donne finalement pas une prévision acceptable. Cependant, avec un paramètre de lissage, nous obtenons une température de 17.86° qui est très réaliste.

2.1.2 Prévision avec la matrice P

Ce modèle nous donne la prévision suivante : -2.230029. Cette prévision n'est pas du tout admissible. Nous pouvons observer la répartition des poids pour cette prévision ci-dessous.

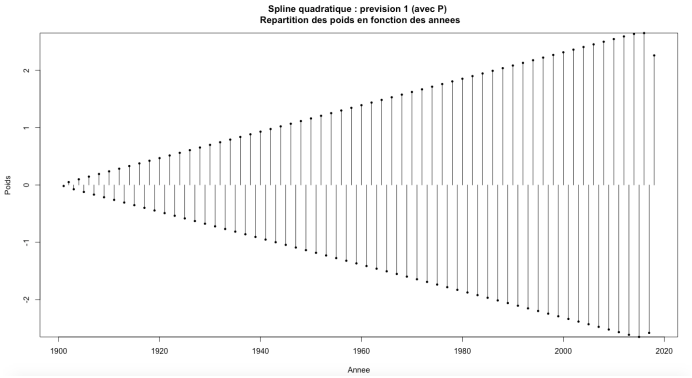


FIGURE 3.2.8 – Répartition des poids ω_i en fonction des années

Nous constatons que plus les années avancent, plus les poids sont importants et donc plus les données associées à ces années contribuent à la prévision. A l'exception de la dernière année qui contribue légèrement moins que les précedentes puisque son poids est d'environ (le poids de l'année 2018 vaut 2.26 contre -2.58, 2.65, -2.65, en 2017, 2016 et 2015).

Obtenant une prévision qui n'est pas réaliste, nous avons raisonné de manière analogue comme précédemment et obtenons les résultats suivants :

Conditionnement	378725215
Valeur propre minimale en module	1.466904e-05

TABLE 3.7 – Propriétés sur la matrice P

La matrice de notre système final est donc mal conditionnée et cela pourrait expliquer une prévision non réaliste. Nous allons donc appliquer les méthodes présentées ci-dessous afin de résoudre ce problème.
La méthode 1 utilisant la fonction *ginv* nous permet d'obtenir une prévision de 14.54581° bien meilleure que la prévision précédente. Néanmoins, nous souhaitons avoir une prévision plus réaliste, nous décidons donc d'appliquer la seconde méthode.

En raisonnant de manière analogue à la section précédente, nous traçons les graphiques représentant l'évolution du conditionnement dans un premier temps pour λ allant de 0 à 5, et de la prévision en fonction du paramètre de lissage que nous introduisons. Nous faisons varier ce paramètre λ de 0 à 1.

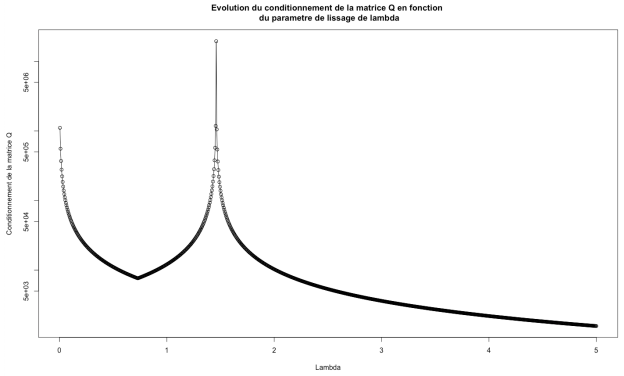


FIGURE 3.2.9 – Conditionnement en fonction de lambda variant entre 0 et 5

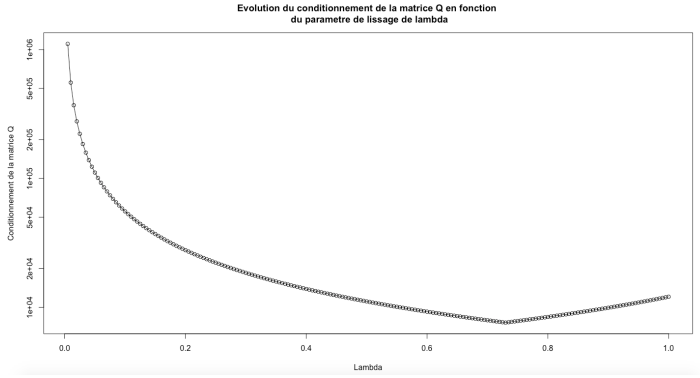


FIGURE 3.2.10 – Conditionnement en fonction de λ variant entre 0 et 1

Le conditionnement suit les mêmes variations que pour le cas $\tilde{\mathbf{P}} = \mathbf{P^1}$, et sa valeur minimale est atteinte en $\lambda = 0.73$ sur $[0\,;1]$.

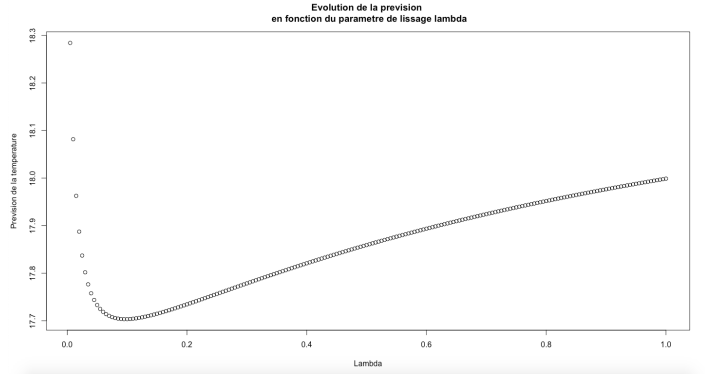


FIGURE 3.2.11 – Prévision en fonction de λ

La prévision varie entre 17.7° et 18.3°. Elle chute pour $\lambda \in [0, 0.1]$ puis augmente. Regardons à présent quelques prévisions et le conditionnement pour les années 2018 et 2019 pour des valeurs de λ proches du minimum ainsi que des valeurs extrêmes.

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.01	554740.7	17.15506	18.08185
0.1	55548.19	17.90431	17.70346
0.6	9259.997	18.33736	17.89363
0.7	7937.311	18.36968	17.92436
0.73	7614.298	18.37814	17.93293
0.8	8422.331	18.39605	17.95187
0.9	9927.265	18.41788	17.9766
3	3608.769	18.54554	18.20875
5	1570.646	18.54509	18.27183

TABLE 3.8 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

Nous choisissons le paramètre $\lambda = 0.1$ car c'est celui qui correspond le plus aux critères énoncés en **2.1.1**.

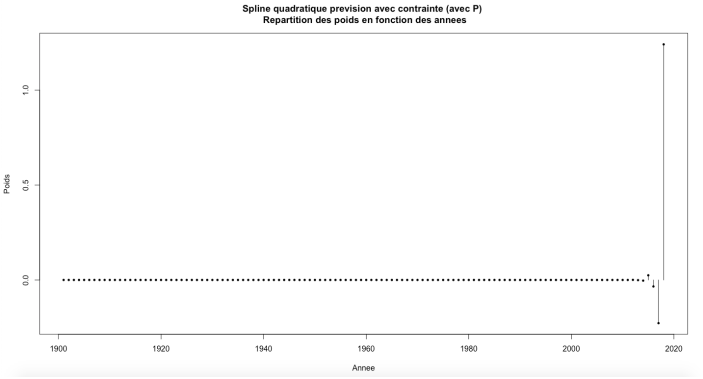


FIGURE 3.2.12 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.1$

De nouveau, ce sont les dernières années de 2015 à 2018 qui contribuent à la création de notre prévision et plus particulièrement l'année 2018 avec un poids de 1.24.

Conclusion : Le paramètre de lissage nous a permis d'obtenir une prévision recevable de 17.70°.

2.2 Prévision sans contrainte

2.2.1 Prévision avec la matrice $\mathbf{P^{-1}}$

Pour le modèle sans contrainte, nous résolvons le système 2.7 avec la matrice $\mathbf{P}^{-1}$, nous obtenons 5.901255° comme prévision de la température moyenne annuelle au Maroc en 2019. La répartition des poids des q_i est la suivante :

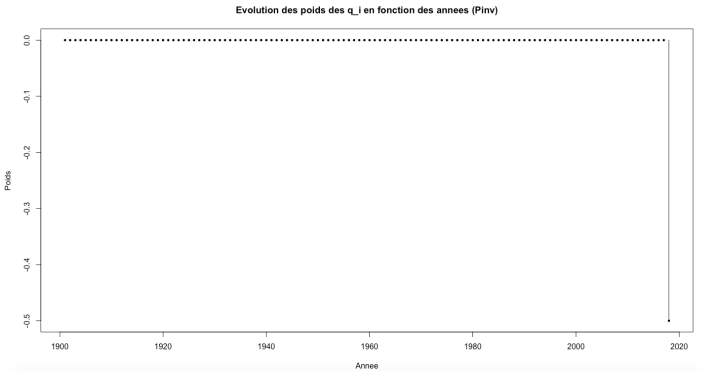


FIGURE 3.2.13 – Répartition des poids des q_i en fonction des années

Nous observons qu'il n'y a uniquement la dernière année (2018) avec un poids de -0.5 qui contribue à la prévision q_{n+1}. Cette prévision ne nous satisfaisant pas, regardons quelques propriétés de la matrice.

Conditionnement	3.99
Valeur propre minimale en module	1

TABLE 3.9 – Propriétés de la matrice P^{-1}

Nous constatons que la matrice est bien conditionnée et a une valeur propre minimale en module qui vaut 1 et qui n'est donc pas proche de 0. Ainsi le système semble bien conditionné. La prévision n'étant pas réaliste mais le système étant bien posé, nous pouvons donc conclure que le modèle est mauvais.
Comme pour pour le modèle précédent, nous avons essayé d'utiliser la fonction *ginv* et nous obtenons exactement la même prévision. Néanmoins, introduire un paramètre de lissage n'était pas nécessaire car la matrice est bien conditionnée et la valeur en propre minimale en module n'est pas proche de 0.
En utilisant la commande *ginv*, la prévision est identique à celle obtenue au départ.
Afin de comprendre pourquoi ce modèle n'est pas concluant, nous avons décidé de tracer l'évolution du conditionnement ainsi que l'évolution de la prévision en fonction de N. En effet, nous nous demandons si ce modèle fonctionne pour de plus petites valeurs de N. Nous faisons donc varier N de 10 à 118 par pas de 10.

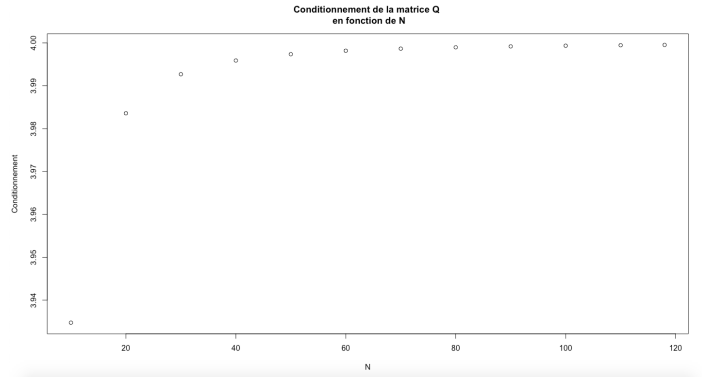


FIGURE 3.2.14 – Conditionnement en fonction de N

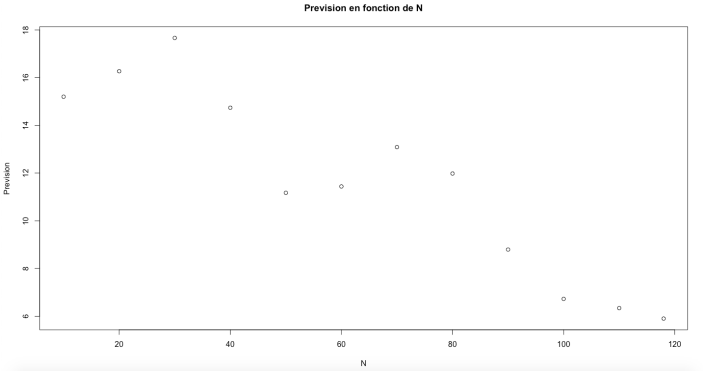


FIGURE 3.2.15 – Prévision en fonction de N

Le conditionnement augmente mais reste relativement faible suivant les valeurs de N, puisqu'il varie entre 3.935 et 4. Cependant, les prévisions varient de 18° à 6°. Globalement, nous constatons une baisse des prévisions lorsque N augmente. Il n'y a que pour N=10, N=20 et N=30 que les prévisions semblent correctes puisqu'elles valent respectivement 15.2°, 16.27° et 17.66°. En effet, dans le tableau 1.2a, nous avons vu que les températures moyenne et médiane annuelles valent environ 16,4° sur les trente premières années.

Conclusion : La prévisions de ce modèle est très aberrante : 5.90°. Ceci n'est pas dû à un mauvais conditionnement. Ce modèle est donc à rejeter.

2.2.2 Prévision avec la matrice P

Pour ce nouveau modèle sans contrainte qui est basé sur la spline quadratique et sur la matrice $\tilde{P} = P$, nous obtenons une prévision de -14.70 qui n'est pas du tout réaliste. Nous observons ci-dessous la répartition des poids des q_i.

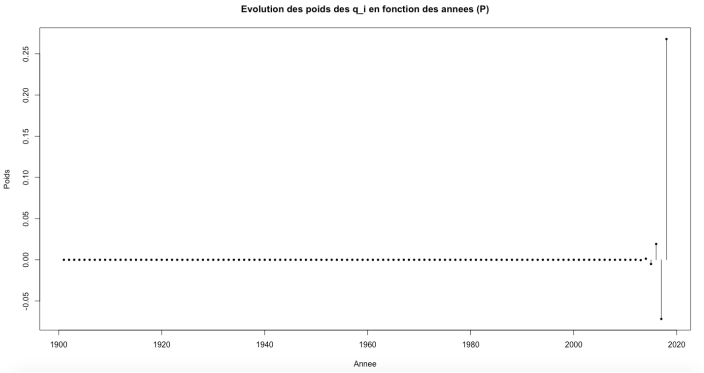


FIGURE 3.2.16 – Répartition des poids des q_i en fonction des années

Les années 2014 à 2018 contribuent au calcul de notre prévision de $q_n + 1$, particulièrement l'année 2018 avec un poids de 0.27. Comme nous l'avons fait auparavant, nous regardons le conditionnement de la matrice du système et sa valeur propre minimale en module. Nous obtenons les résultats suivants :

Conditionnement	3.99
Valeur propre minimale en module	0.25

TABLE 3.10 – Propriétés sur la matrice P

La matrice semble bien conditionnée ainsi le système final à résoudre n'est pas perturbé. Nous pouvons donc penser que le modèle est mauvais. Nous avons de nouveau fait appel à la fonction *ginv*, cependant, pour le cas $\tilde{\mathbf{P}} = \mathbf{P^{-1}}$, la prévision est identique et comme précédemment le paramètre de lissage n'était pas nécessaire. Regardons à présent l'influence du nombre de données N sur le conditionnement et la prévision.

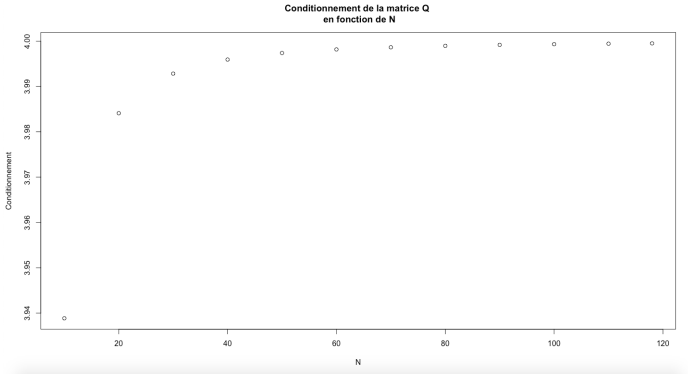


FIGURE 3.2.17 – Conditionnement en fonction de N

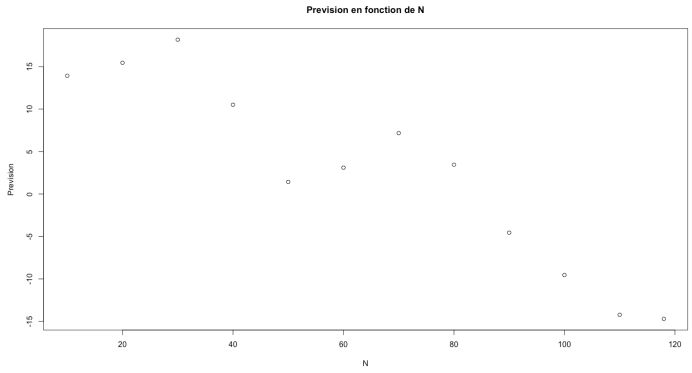


FIGURE 3.2.18 – Prévision en fonction de N

Le conditionnement augmente mais reste très faible car il varie de 3.94 à 4. Pour les prévisions, les variations sont les mêmes que précédemment mais pour N supérieur à 80 les températures deviennent négatives. En partie 1, sur le tableau 1.2a, nous constatons que sur la première période, c'est-à-dire, de 1901 à 1930, les températures moyenne et médiane annuelles valent toutes deux environ 16,4. Il semblerait donc que la meilleure prévision est réalisée pour N=20 et vaut 15.45°.

Conclusion : La matrice définissant le modèle est bien conditionnée mais nous obtenons une prévision non réaliste : 5.90°. Nous concluons donc qu'il s'agit ici d'un très mauvais modèle.

3 Splines cubiques

3.1 Prévision avec contraintes

3.1.1 Prévision avec la matrice $\mathbf{P^{-1}}$

Le système 2.9 nous permet d'obtenir cette prévision et nous donne un résultat de -297.6931°. Sur le graphique ci-dessous, nous pouvons observer la répartition des poids.

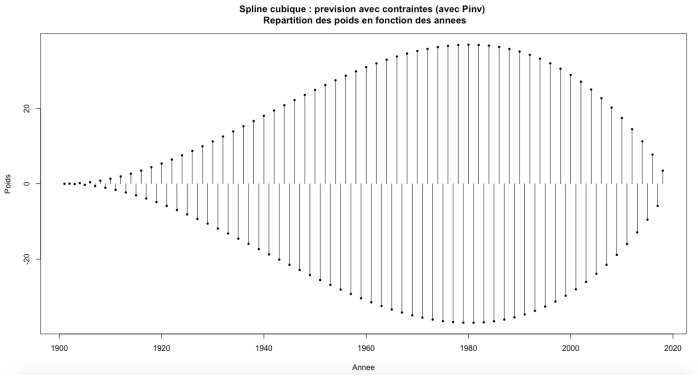


FIGURE 3.3.19 – Répartition des poids ω_i en fonction des années

Les premières années contribuent faiblement à notre prévision. Nous constatons que plus nous nous approchons de l'année 1980, plus l'année a un poids important.
Cette prévision n'est pas du tout réaliste. Nous allons donc étudier la matrice $\mathbf{D}$, nous retrouvons les résultats ci-dessous.

Conditionnement	4.941801e+14
Valeur propre minimale en module	1.348731e-08

TABLE 3.11 – Propriétés de la matrice D

Nous constatons que cette matrice est très mal conditionnée et a au moins une valeur propre quasiment nulle. Ainsi, avant de juger ce modèle, nous devons corriger ces problèmes.

Le conditionnement étant particulièrement grand, nous avons cherché à savoir à partir de quand le conditionnement de la matrice du système explosait. Pour ce faire, nous avons tracé l'évolution du conditionnement de cette matrice en fonction de N qui représente le nombre d'années pris en compte dans notre prévision ainsi que l'évolution de la prévision en fonction de N. Nous obtenons ainsi les graphiques suivants :

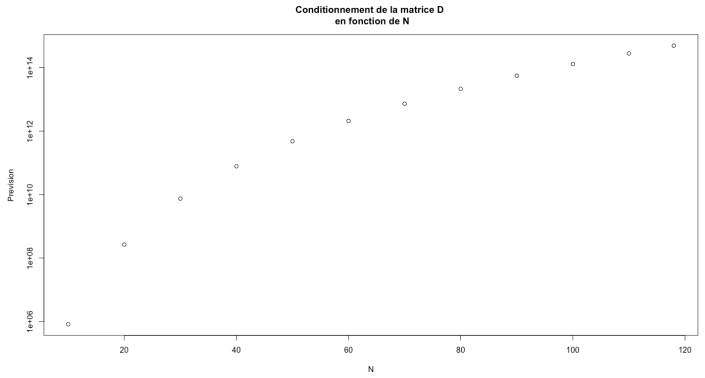


FIGURE 3.3.20 – Conditionnement en fonction de N

Nous remarquons que plus nous prenons de données, plus le conditionnement augmente. En prenant très peu de données (N=10), nous avons un conditionnement de l'ordre de 10^6, ce qui est déjà grand. Nous pouvons donc penser que plus nous prenons de données, plus l'erreur sur la prévision sera importante.

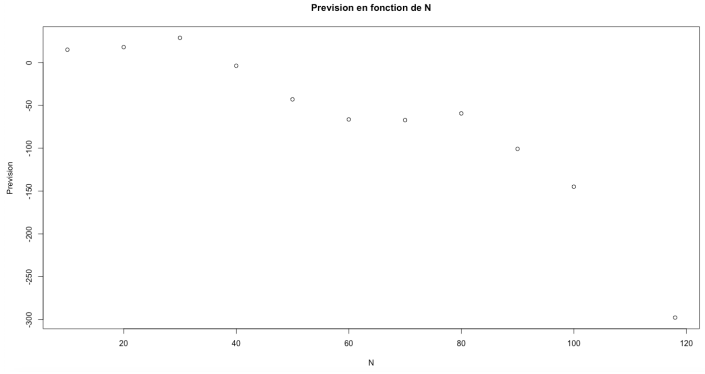


FIGURE 3.3.21 – Prévision en fonction de N

Les prévisions affichées sur le graphique ci-dessus varient d'environ 28.8° à -297.69°. Les seules prévisions réalisables sont pour N=10 et N=20 où nous obtenons respectivement 15° et 17.99° qui correspondent à un conditionnement de 10^6 et 10^9.

Nous pouvons supposer que dans ce cas que le conditionnement de la matrice a un impact sur les prévisions. En effet, plus le conditionnement augmente, moins la prévision est proche de la réalité.

Maintenant que nous avons étudié l'origine de ce mauvais conditionnement et donc de cette prévision non réaliste, nous allons introduire un paramètre de lissage comme nous l'avons expliqué précédemment.

Dans un premier, nous étudions le conditionnement de la matrice $\mathbf{D}$ pour $\lambda \in [0, 5]$.

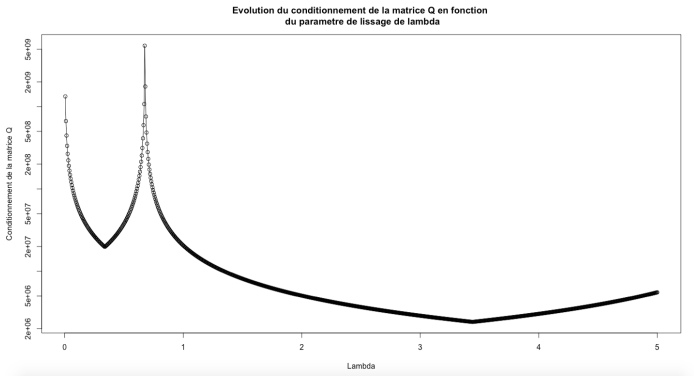


FIGURE 3.3.22 – Conditionnement en fonction de lambda variant entre 0 et 5

Nous observons que le conditionnement diminue pour $\lambda \in [0; 0,34]$ puis augmente jusqu'à atteindre sa valeur maximale de l'ordre de 10^9 pour λ=0.675. Il décroît à nouveau jusqu'à $\lambda \simeq 3.4$. Il finit par légèrement augmenter à partir de cette valeur.

Comme nous avons pu le faire précédemment, nous étudions désormais le conditionnement et la prévision pour $\lambda \in [0, 1]$.

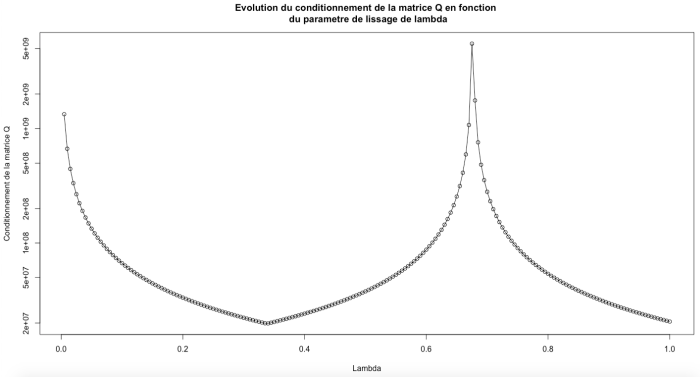


FIGURE 3.3.23 – Conditionnement en fonction de λ

Nous allons donc sélectionner le λ pour lequel le conditionnement, quelques valeurs voisines ainsi que des valeurs extrêmes.
Afin de voir comment ce paramètre de lissage agit sur notre prévision en fonction de sa valeur nous avons tracé le graphique ci-dessous.

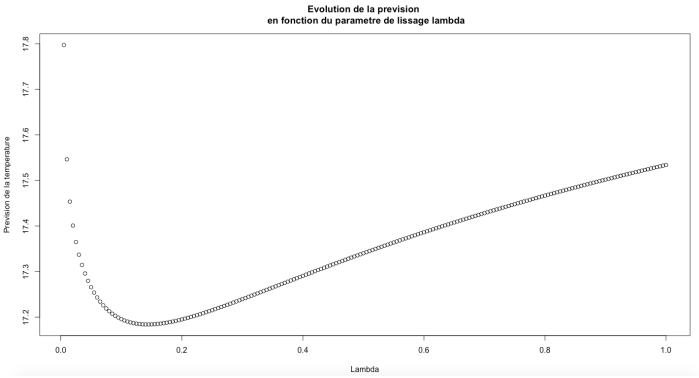


FIGURE 3.3.24 – Evolution de la prévision en fonction du paramètre λ

Nous observons que la prévision varie entre 17.2° et 17.8°. Dans un premier, elle diminue jusqu'à $\lambda \simeq 0.125$ puis ensuite augmente jusqu'à $\simeq 17.5°$.
Pour les valeurs de λ suivantes, nous allons analyser la température prédite en les introduisant dans notre modèle. Ensuite, nous regarderons la température qu'ils permettent de prédire en 2018 sachant que nous disposons de cette température : 17.80022.

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.01	667119827	16.93747	17.54639
0.1	66712065	17.49609	17.19562
0.2	33356035	17.72081	17.19499
0.3	22237358	17.87382	17.23941
0.34	19842537	17.9223	17.25982
0.4	24152893	17.98529	17.29066
0.5	37859976	18.07081	17.34029
3	2870829	18.59568	17.87841
3.44	2413788	18.62002	17.91863
5	5533921	18.67587	18.02226

TABLE 3.12 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

Au vu de ces résultats, nous choisissons $\lambda = 0.34$ qui nous donne une prévision de 17.92° pour 2019. En effet, il nous donne une prévision pour 2018 proche de la température mesurée tout en minimisant le conditionnement pour $\lambda \in [0, 1]$. Néanmoins, le conditionnement reste extrêmement élevé.
Avec l'introduction du paramètre de lissage $\lambda = 0.34$, la répartition des poids est la suivante.

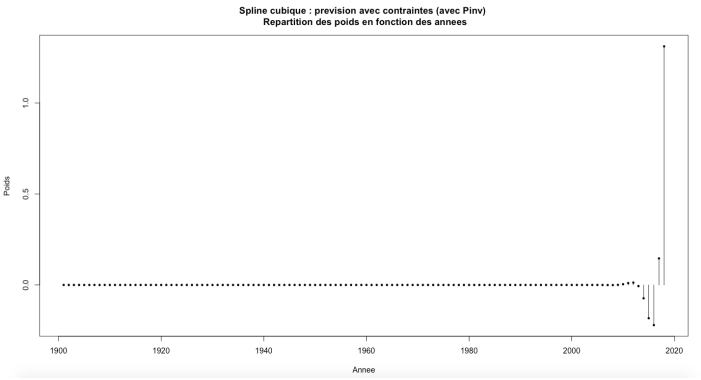


FIGURE 3.3.25 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.34$

Nous notons que seuls les poids à partir de 2014 sont significatifs et particulièrement celui de 2018 qui vaut $\simeq 1.4$.
Dans un second temps, nous avons également calculé la prévision de ce modèle en utilisant *ginv*. Nous obtenons alors une prévision de 16.17347°.

Conclusion : Nous retenons alors comme prévision celle obtenue avec l'introduction du paramètre de lissage qui est de 17.26°. Néanmoins, le conditionnement, restant trop important, nous oblige à utiliser ce modèle avec précaution.

3.1.2 Prévision avec la matrice P

Après avoir implémenté l'algorithme correspondant au système 2.9, nous constatons que nous ne pouvons obtenir une prévision car le déterminant de la matrice **D** est nul, ce qui nous oblige à travailler avec la fonction *ginv* qui nous donne la prévision : 16.12501. La répartition des poids est la suivante :

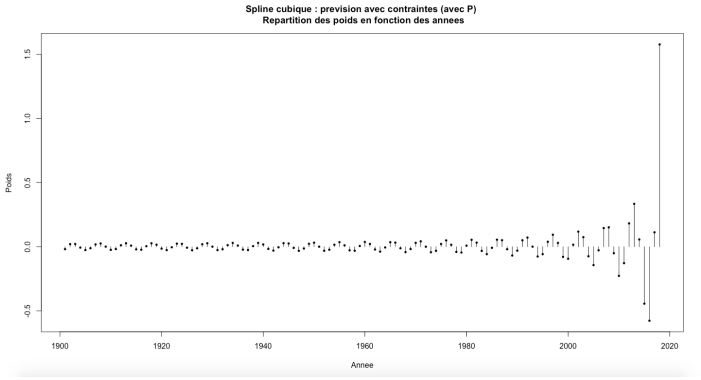


FIGURE 3.3.26 – Répartition des poids ω_i en fonction des années

Nous observons que les poids sont plutôt faibles de 1901 à 1980. A partir de cette date, les poids deviennent de plus en plus importants jusqu'en 2018 où il vaut environ 1.6.

Analysons de plus près la matrice D

Conditionnement	4.537217e+15
Valeur propre minimale en module	1.454124e-09

TABLE 3.13 – Informations sur la matrice **D**

Au vu du conditionnement énorme et de la valeur propre minimale en module quasiment nulle, nous allons introduire un paramètre de lissage.

Cependant, nous allons tout d'abord étudier l'évolution du conditionnement et de la prévision en fonction de N comme nous l'avons fait précédemment à la section **3.1.1** puisque le conditionnement est de l'ordre de 10^{15}.

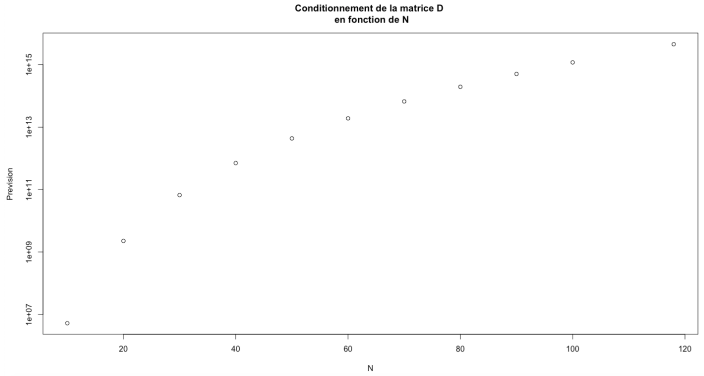


FIGURE 3.3.27 – Conditionnement en fonction de N

A nouveau, nous constatons que plus nous prenons de données, plus le conditionnement augmente. Il varie d'environ 10^7 à environ 10^{15} pour N allant de 5 à 118. Quelque soit le nombre de données que nous choisissons de conserver, le conditionnement reste très grand.

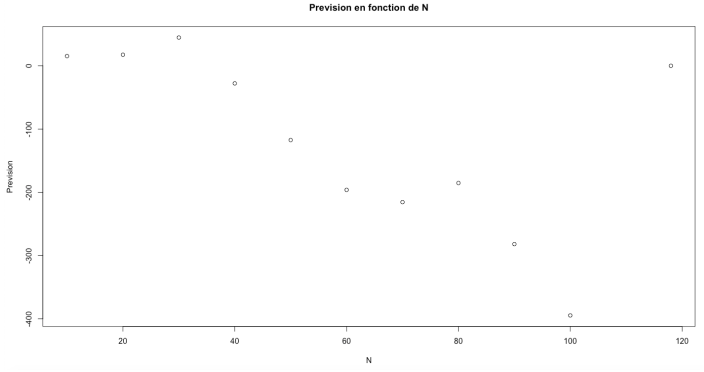


FIGURE 3.3.28 – Prévision en fonction de N

Les prévisions affichées sur le graphique ci-dessus varient d'environ 45° à -395°. Les seules prévisions réalisables sont pour N=10 et N=20 où nous obtenons respectivement 15.33° et 17.48° qui correspondent à un conditionnement de 10^6 et 10^9. La dernière valeur (N=118) n'est absolument pas significative car la matrice est devenue non inversible, et par construction de l'algorithme, cela nous donne une prévision égale à 0°.

Par un raisonnement analogue, nous arrivons aux mêmes conclusions que pour les figures 3.3.20 et 3.3.21.

Afin de déterminer le paramètre de lissage, nous avons comme précédemment tracé les graphiques suivants.

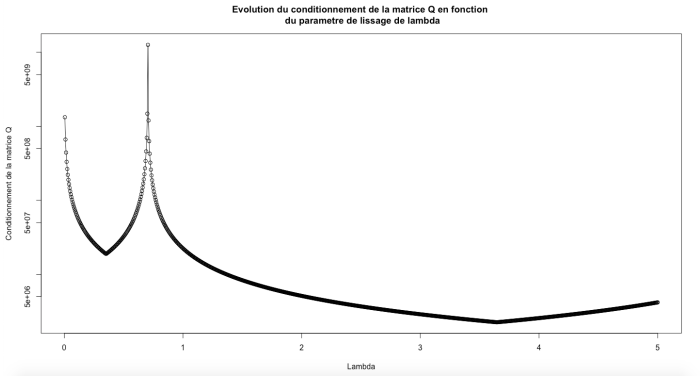


FIGURE 3.3.29 – Conditionnement en fonction de lambda variant entre 0 et 5

Nous pouvons faire les mêmes remarques que pour la figure 3.3.22. En effet, les variations du conditionnement sont les mêmes. Tout d'abord, il décroît jusqu'à atteindre le premier minimum local en $\lambda = 0.35$ et vaut 18946291. Ensuite, il augmente brutalement puis atteint sa valeur maximale 12176371483 pour λ=0.705 avant de décroître à nouveau pour atteindre le minimum global sur $[0, 5]$ 3.645 où il vaut 2179305. Finalement, il croît légèrement jusque λ =5 et vaut 4184515.

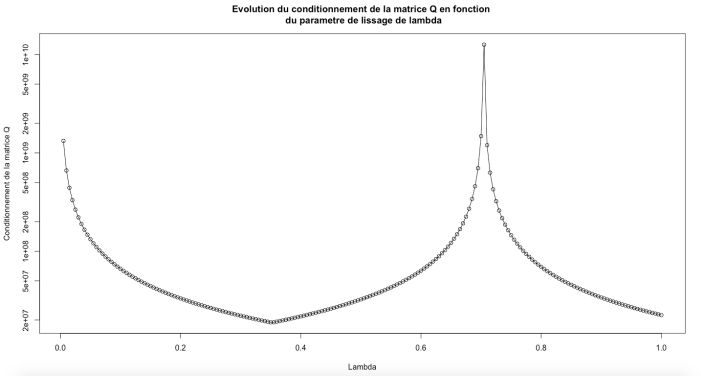


FIGURE 3.3.30 – Conditionnement en fonction de λ

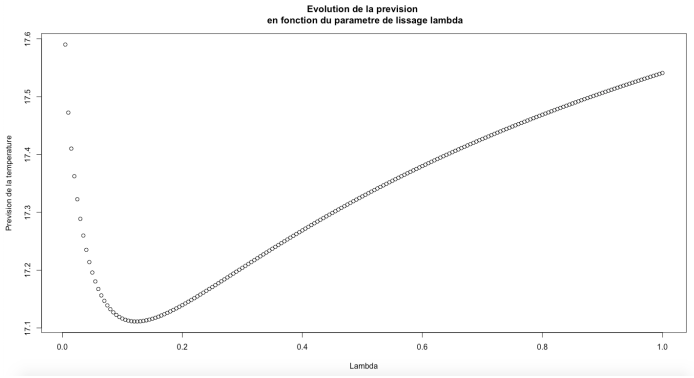


FIGURE 3.3.31 – Evolution de la prévision en fonction du paramètre λ

Sur les deux graphiques ci-dessus, nous nous sommes concentrées sur des valeurs de λ variant entre 0 et 1 afin de sélectionner notre paramètre de lissage. Les prévisions varient entre 17.1° et 17.6°, elles sont donc toutes réalistes. Pour déterminer le paramètre λ, regardons le tableau ci-dessous :

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.01	663120056	17.07611	17.47243
0.1	66312015	17.46508	17.11628
0.2	33156008	17.73077	17.13953
0.3	22104006	17.89905	17.20407
0.35	18946291	17.96246	17.23702
0.4	21779216	18.01652	17.26877
0.5	32430561	18.10435	17.32757
3	2888751	18.61387	17.89289
3.645	2255741	18.64539	17.94927
5	4184515	18.68804	18.03491

TABLE 3.14 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

Nous choisissons le paramètre λ =0.35 car il permet d'avoir le conditionnement minimal pour $\lambda \in [0, 1]$ tout en donnant une prédiction pour 2018 cohérente. En effet, $\lambda = 3.645$ nous donne un meilleur conditionnement, cependant, la prévision pour 2018 est moins proche de la température mesurée que celle obtenue avec $\lambda = 0.35$. Par ailleurs, $\lambda = 3.645$ nous éloigne beaucoup de notre matrice initiale. Il ne serait donc pas raisonnable de choisir ce paramètre.

Avec l'introduction du paramètre de lissage $\lambda = 0.35$, la répartition des poids est la suivante.

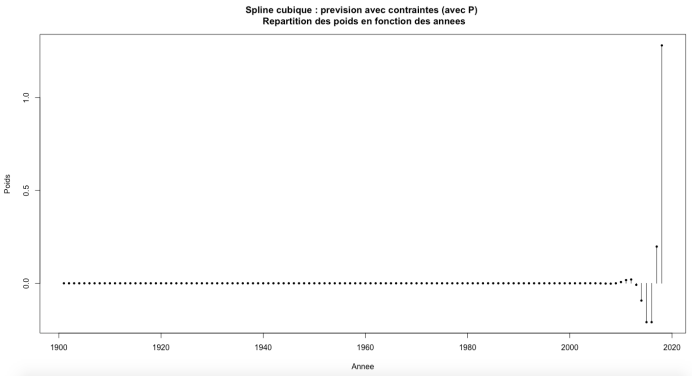


FIGURE 3.3.32 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.35$

Nous observons que les poids qui contribuent le plus sont ceux correspondant aux années de 2014 à 2018. A nouveau, celui de 2018 est plus important que les autres et vaut 1.28.

Conclusion : Pour ce modèle, la fonction *ginv* nous donnait une prévision acceptable. Nous avons tout de même décidé d'introduire un paramètre de lissage qui nous donne une prévision de 17.24°. Avec le choix de ce paramètre, le conditionnement diminue mais reste tout de même trop important pour que l'on valide avec certitude la fiabilité de ce modèle.

3.2 Prévision sans contrainte

3.2.1 Prévision avec la matrice $\mathbf{P^{-1}}$

Le système 2.10 nous donne une prévision de : -296.674°, ce qui est très mauvais. La fonction *ginv* nous donne encore exactement le même résultat. Les poids sont ainsi répartis :

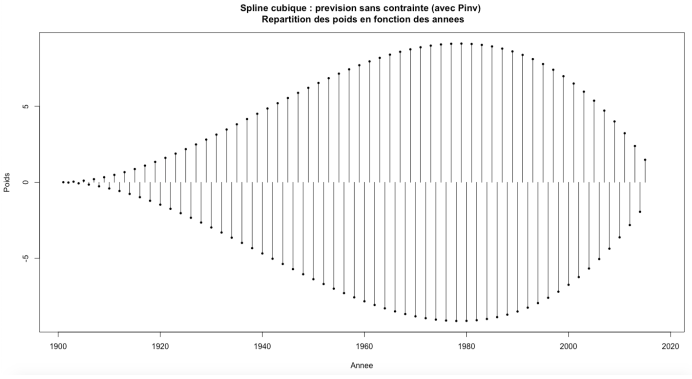


FIGURE 3.3.33 – Répartition des poids ω_i en fonction des années

Comme pour la figure 3.3.19, plus nous nous approchons de l'année 1980, plus le poids est important. Calculons à présent le conditionnement et la valeur propre minimale en module de la matrice du modèle.

Conditionnement	2051063
Valeur propre minimale en module	2.166379e-07

TABLE 3.15 – Propriétés de la matrice P^{-1}

Le conditionnement est grand (de l'odre de 10^6) et nous avons au moins une valeur propre très proche de zéro. Ces deux valeurs étant très mauvaises, introduisons un paramètre de lissage.

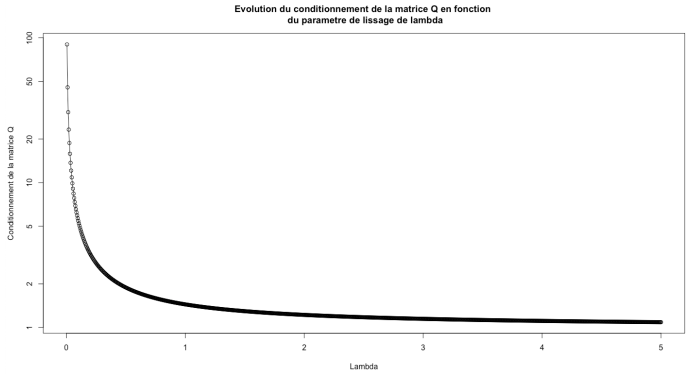


FIGURE 3.3.34 – Conditionnement en fonction de lambda variant entre 0 et 5

Sur l'intervalle $[0; 5]$, nous observons une baisse significative du conditionnement de la matrice jusqu'à avoir un conditionnement très proche de 1, nous pouvons donc obtenir une matrice bien conditionnée tout en gardant une valeur de λ petite. Regardons le conditionnement et les prévisions pour l'année 2019 sur l'intervalle $[0; 1]$:

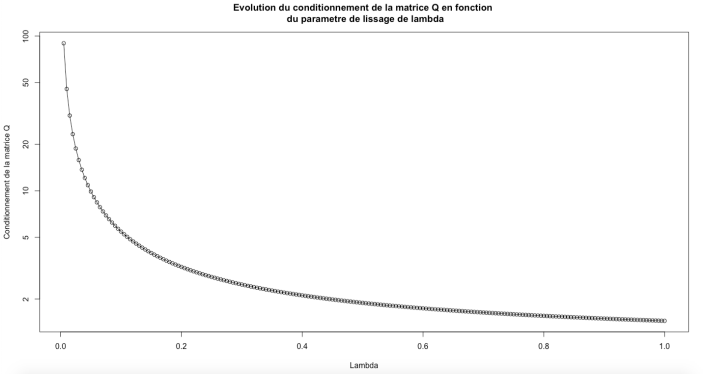


FIGURE 3.3.35 – Evolution conditionnement en fonction de λ pour le cas classique $\mathbf{P^{-1}}$

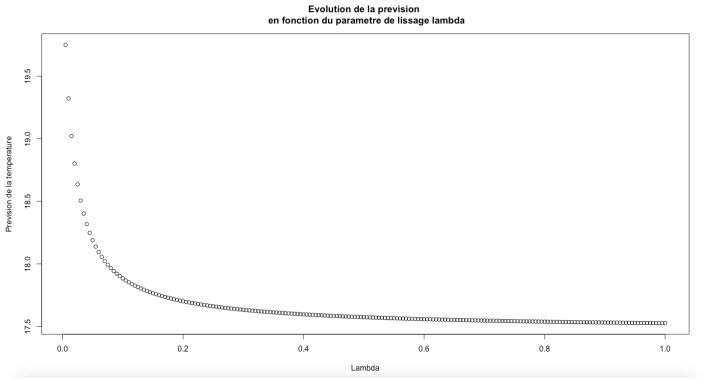


FIGURE 3.3.36 – Evolution prévision en fonction de λ pour le cas $\mathbf{P^{-1}}$ (échelle logarithmique pour l'axe des ordonnées)

Les prévisions se situent entre 17.5° et 19.8°, ce qui est recevable. Comme pour le conditionnement, plus λ augmente, plus la prévision baisse. Nous avons sélectionné quelques paramètres pour lesquelles nous avons regardé le conditionnement et la prévision en 2018 et 2019 :

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.0001	4434.771	22.55959	14.89392
0.001	445.2414	15.1636	20.42436
0.01	45.4328	15.30706	19.32238
0.1	5.443367	16.54213	17.88462
0.3	2.481124	16.88059	17.63353
0.5	1.888675	16.96997	17.5743
0.7	1.634768	17.01337	17.54732
0.9	1.493708	17.53185	17.03944
1	1.444338	17.04898	17.52636

TABLE 3.16 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

La température mesurée en 2018 étant de 17.80022, nous choisissons $\lambda = 0.9$ comme paramètre de lissage, avec une prévision pour 2019 de 17.03°. Ce paramètre nous permet d'avoir un conditionnement très proche de 1 et de bonnes prévisions pour les années 2018 et 2019.

Avec l'introduction du paramètre de lissage $\lambda = 0.9$, la répartition des poids est la suivante.

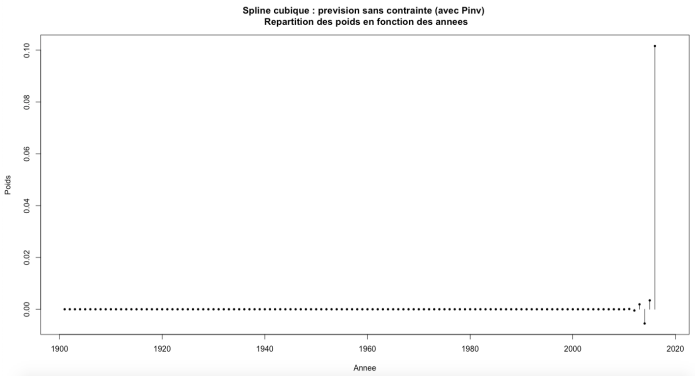


FIGURE 3.3.37 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.9$

Les années contribuant le plus à la prévision de la température en 2019 sont les années de 2015 à 2018. Le poids observé en 2018 est 10 fois plus important que les précédents et vaut 0.1.

Conclusion : L'introduction d'un paramètre de lissage dans ce modèle a permis d'obtenir une prévision acceptable et une matrice très bien conditionnée. Ainsi, nous pouvons retenir ce modèle et nous avons comme prévision 17.04°.

3.2.2 Prévision avec la matrice P

En prenant la matrice **P**, nous obtenons la prévision : -811.5265, *ginv* nous donne une nouvelle fois la même prévision. Nous trouvons ci-dessous le graphique de la répartition des poids :

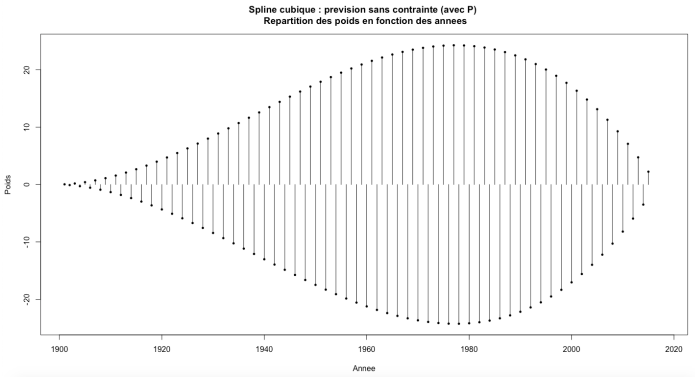


FIGURE 3.3.38 – Répartition des poids ω_i en fonction des années

Comme pour les figures 3.3.19 et 3.3.33 qui ont la même forme, plus les années s'approchent de 1978, plus les poids associés sont importants. Cette prévision est extrêmement mauvaise, regardons les propriétés de la matrice **P** :

Conditionnement	18781252
Valeur propre minimale en module	2.365297e-08

TABLE 3.17 – Propriétés de la matrice P

Le conditionnement est de l'ordre de 10^7 et nous avons au moins une valeur propre quasiment nulle. Au vu des mauvais résultats, introduisons un paramètre de lissage.

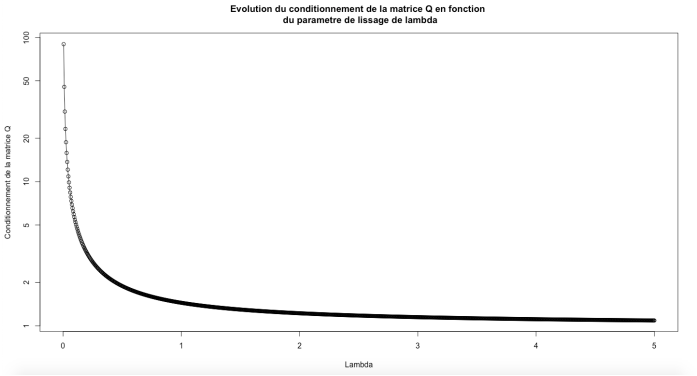


FIGURE 3.3.39 – Conditionnement en fonction de lambda variant entre 0 et 5

Comme pour la figure 3.3.35, le conditionnement décroît très rapidement et tend vers 1. Nous pouvons donc avoir un bon conditionnement tout en ayant une valeur propre petite. Regardons le conditionnement et la prévision pour $\lambda \in [0; 1]$

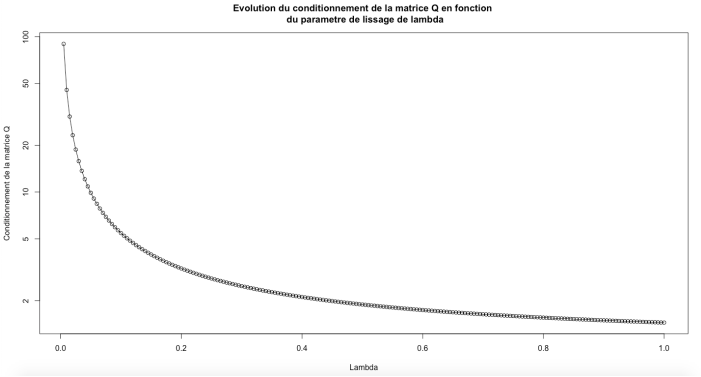


FIGURE 3.3.40 – Evolution conditionnement en fonction de λ pour le cas **P**

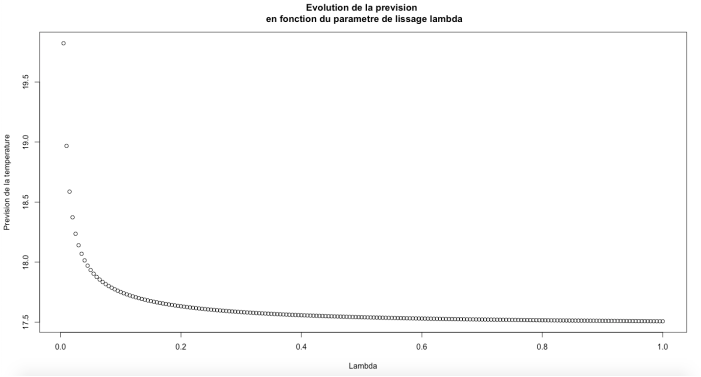


FIGURE 3.3.41 – Evolution prévision en fonction de λ pour le cas **P**

Au même titre que le conditionnement, quand λ augmente la prévision diminue. Elle reste

tout de même entre 17.5° et 19.9°, ce qui est acceptable. Testons plusieurs valeurs de λ en comparant le conditionnement et les prévision en 2018 et 2019.

λ	Conditionnement	Prévision en 2018	Prévision en 2019
0.0001	4442.273	14.96625	22.43592
0.001	445.2219	13.76499	21.72051
0.01	45.42313	15.1819	18.96763
0.1	5.442323	16.69078	17.75081
0.3	2.480774	16.91913	17.58399
0.5	1.888465	17.54118	16.98825
0.7	1.634618	17.02452	17.5219
0.9	1.493592	17.04722	17.5111
1	1.444232	17.05569	17.50734

TABLE 3.18 – Prévisions de la température 2018 et 2019 en fonction du paramètre λ

La température réelle en 2018 étant de 17.80022, nous choisissons $\lambda = 0.5$ qui nous donne une prévision de 17.54° en 2018. Le conditionnement associé est très bon (1.888).

Avec l'introduction du paramètre de lissage $\lambda = 0.5$, la répartition des poids est la suivante.

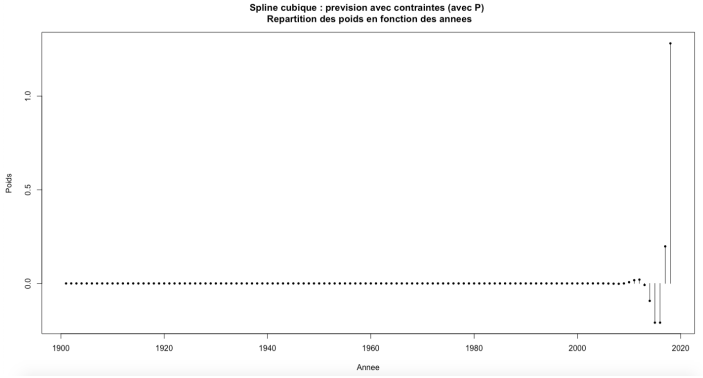


FIGURE 3.3.42 – Répartition des poids ω_i en fonction des années avec $\lambda = 0.5$

Comme sur les précédentes répartitions des poids après lissage, ce sont les dernières années (de 2014 à 2018) qui contribuent le plus à la création de la prévision. L'année qui contribue le plus est de nouveau la dernière année avec un poids d'environ 1.4, soit cinq fois plus grand que les poids précédents.

Conclusion : Pour ce dernier modèle, nous constatons à nouveau que sans l'introduction d'un paramètre de lissage, la prévision n'est pas acceptable. Nous choisissons le paramètre $\lambda = 0.5$. Retenons donc comme prévision pour l'année 2019 16.99°.

Conclusion

A travers ce travail de recherche, nous avons pu constater que, parmi les trois méthodes d'approximation que nous avons étudiées : interpolation linéaire, splines quadratiques et cubiques, les splines cubiques semblent être la meilleure méthode permettant d'approcher une fonction. Nous avons pu l'observer à plusieurs reprises, en calculant les erreurs à partir de fonctions théoriques données. Ceci justifie que leur utilisation soit plus récurrente que les autres. Nous avons construit dix modèles de prévision : quatre sous contrainte et six sans contrainte. En passant à la partie pratique, nous nous sommes aperçues que le numérique n'est pas aussi parfait que la théorie. En effet, en implémentant nos modèles de prévision en langage de programmation C++, nous avons obtenu des prévisions aberrantes. C'est pourquoi nous avons dû poursuivre notre étude sur le logiciel R qui met à notre disposition de nombreux outils appropriés et directement utilisables. A travers deux méthodes, nous avons pu minimiser les erreurs numériques et ainsi obtenir des prévisions pertinentes. Finalement, le tableau ci-dessous reprend toutes les prévisions que nous avons obtenues grâce à nos différents modèles.

Spline utilisée	Prévision obtenue
Linéaire	17.80°
Quadratique	5.90°
Cubique	17.04°

(a) Modèle sans contrainte - $\tilde{\mathbf{P}} = \mathbf{P^{-1}}$

Spline utilisée	Prévision obtenue
Linéaire	17.79°
Quadratique	-14.70°
Cubique	16.99°

(b) Modèle sans contrainte - $\tilde{\mathbf{P}} = \mathbf{P}$

Spline utilisée	Prévision obtenue
Quadratique	17.86°
Cubique	17.26°

(c) Modèle sous contrainte - $\tilde{\mathbf{P}} = \mathbf{P^{-1}}$

Spline utilisée	Prévision obtenue
Quadratique	17.70°
Cubique	17.24°

(d) Modèle sous contrainte - $\tilde{\mathbf{P}} = \mathbf{P}$

TABLE 3.19 – Conclusion : Prévisions de la température au Maroc en 2019

Suite à l'obtention de ces prévisions, nous allons déterminer la plus pertinente d'entre elles. Au vu de ce tableau récapitulatif, nous obtenons 8 prévisions réalistes sur 10. Néanmoins, comme nous avons pu le constater au chapitre 3 aux sous-sections 3.1.1 et 3.1.2, où nous étudions les modèles de spline cubique sous contraintes, l'introduction d'un paramètre de lissage nous permet d'obtenir de bonnes prévisions mais les conditionnements restent trop importants, nous ne pouvons donc les adopter.
Parmi, les six prévisions restantes, nous décidons de nous concentrer sur celles obtenues par spline cubique sans contrainte. En effet, d'après les études que nous avons menées au chapitre 1, les splines cubiques semblent meilleures. Finalement, nous devons choisir l'une des deux prévisions suivantes : 17.04° et 16.99°. Au chapitre 1 à la sous-section 3.1 dans

laquelle nous avons étudié nos données, nous avons constaté que la température semble croître selon que le temps passe. C'est pourquoi nous avons comparé les deux dernières prévisions restantes avec les températures médiane et moyenne annuelles au Maroc sur les trente dernières années qui valent respectivement 17.35° et 17.54°. Finalement, notre choix se porte donc sur la prévision **17.04**°. Ainsi, si nous devions sélectionner un seul modèle, ce serait celui construit par **spline cubique, sans contrainte**, avec comme matrice $\tilde{\mathbf{P}} = \mathbf{P}^{-1}$

Remerciements

Nous tenons sincèrement à remercier nos encadrants, Madame Calgaro et Monsieur Dermoune qui nous ont accompagnées tout au long de ce travail, ont su se rendre disponible pour répondre à nos questions, nous guider dans nos recherches et nous apporter de nombreux conseils.

Bibliographie

[Article] *Interpolations, courbes de Bézier et B-Splines*, Thomas Guillod, pages 8-11.

[Article] *Interpolation and prediction of data-three kernel selection criteria*, Azzouz Dermoune, Khalifa Es-Sebaiy, Mohammed Es.Sebaiy, Jabrane Moustaaid, page 16.

[Cours] *Modélisation et analyse numérique*, Calgaro Caterina. Cours de licence de mathématiques - Semestre 6, pages 23-26.

[Documentation] Documentation de R - à propos de la fonction *ginv*, https://www.rdocumentation.org/packages/matlib/versions/0.9.1/topics/Ginv

Annexe A

Annexes : Splines

A.1 Code C++ : calcul des coefficients a_i et b_i de l'interpolation linéaire

```
matrice spline1(vecteur x,vecteur y)
{
    int n=x.dim();
    matrice M(2,n-1);
    for(int i=0;i<n-1;i++)
    {
    M(0,i)=(y(i+1)-y(i))/(x(i+1)-x(i));
    M(1,i)=(y(i)*x(i+1)-y(i+1)*x(i))/(x(i+1)-x(i));
    }
return M;
}
```

A.2 Code C++ : calcul des coefficients p_i, q_i et u_i d'une spline quadratique naturelle

```
matrice spline2(vecteur t,vecteur p)
{
    int n=t.dim();
    vecteur q(n), u(n-1), h(n-1);
    matrice M(3,n-1);
    // Definition du vecteur h
    for (int i=0;i<n-1;i++){
    h(i)=t(i+1)-t(i);}
    // Condition spline naturelle : f'(t_1)=q_1=0
    q(0)=0;
    // Calcul du vecteur q et du vecteur u
    for(int i=1;i<n;i++){
        q(i)=(2/h(i-1))*(p(i)-p(i-1))-q(i-1);
        u(i-1)=(q(i)-q(i-1))/h(i-1);}
```

```
    // Stockage des vecteurs p, q, u dans la matrice M
    for(int i=0;i<n-1;i++){
        M(0,i)=p(i);
        M(1,i)=q(i);
        M(2,i)=u(i);}
    return M;
}
```

A.3 Code C++ : Calcul des coefficients p_i, q_i, u_i et v_i d'une spline cubique naturelle

```
matrice spline3(vecteur t,vecteur p)
{
    int n=t.dim();
    vecteur h(n-1), q(n-1), v(n-1), b(n), u(n), piv(n);
    matrice L(n,n), decompo(n,n), M(4,n-1);
    // Definition du vecteur h
    for (int i=0;i<n-1;i++){
        h(i)=t(i+1)-t(i);}
    //Definition de la matrice permettant de determiner les u_i
    L(0,0)=L(n-1,n-1)=1;
    for(int i=0;i<n-1;i++){
        L(i,i+1)=L(i+1,i)=h(i);}
    for(int i=1;i<n-1;i++){
        L(i,i)=2*(h(i-1)+h(i));}
    //Definition du second membre du systeme
    for (int i=1;i<n-1;i++){
        b(i)=6*((p(i+1)-p(i))/h(i)-(p(i)-p(i-1))/h(i-1));}
    //Factorisation LU pour resoudre le systeme
    decompo=L.lu(piv);
    u=decompo.solvelu(b,piv);
    // Condition spline naturelle : f''(t_1)=u_1=0 & f''(t_n)=u_n=0
    u(0)=u(n-1)=0;
    // Creation des vecteurs contenant les coefficients
    for(int i=0; i<n-1;i++){
        q(i)=(p(i+1)-p(i))/(h(i))-(h(i)/6)*(2*u(i)+u(i+1));
v(i)=(u(i+1)-u(i))/h(i);}
    // Stockage des vecteurs p, q, u dans la matrice M
    for(int i=0;i<n-1;i++){
        M(0,i)=p(i);
        M(1,i)=q(i);
        M(2,i)=u(i);
        M(3,i)=v(i);}
    return M;
}
```

A.4 Code C++ : Calcul de la norme l_2 de la différence du vecteur de la fonction théorique et du vecteur de l'approximation par spline

```
double norme(int choix_spline)
{
    vecteur serie(100000000);
    vecteur spline(10000000);
    vecteur abs(10000000);
    read_vect("ordonnee_serie.txt",serie);
    read_vect("abscisse_serie.txt",abs);
    string nom_erreur;
    switch(choix_spline)
    {
        case 1:{
            read_vect("ordonnee_spline1.txt",spline);
            nom_erreur=("erreur_spline_l.txt");
            break;}
        case 2:{
            read_vect("ordonnee_spline2.txt",spline);
            nom_erreur=("erreur_spline_q.txt");
            break;}
        case 3:{
            read_vect("ordonnee_spline3.txt",spline);
            nom_erreur=("erreur_spline_c.txt");
            break;}
        default:
        {
            break;}
    }
    vecteur nrm(serie.dim());
    int k=0;
    double erreur=0;
    for(int i=0;i<serie.dim();i++)
    {
        nrm(i)=spline(i)-serie(i);
    }
    erreur=nrm.norme_l2();
    return erreur;
}
```

A.5 Données : Températures annuelles moyennes au Maroc de 1901 à 2018

Année	Température
1901	17,26595
1902	16,46445
1903	16,89110
1904	17,15045
1905	17,17115
1906	16,59410
1907	16,09055
1908	16,80720
1909	17,02400
1910	15,99565
1911	16,25630
1912	15,76695
1913	16,35055
1914	16,30280
1915	15,23885
1916	16,12285
1917	15,29395
1918	14,65460
1919	15,68985
1920	16,69855
1921	16,47455
1922	16,73505
1923	16,94875
1924	16,83405
1925	15,87310
1926	17,37765
1927	17,31120
1928	16,24045
1929	16,86560
1930	17,35415
1931	16,92065
1932	15,94330
1933	17,92855
1934	16,69940
1935	16,89310
1936	16,57905
1937	18,09945
1938	16,97220
1939	16,29280

Année	Température
1940	17,12830
1941	16,82185
1942	17,90155
1943	17,45370
1944	17,53185
1945	19,43390
1946	16,37885
1947	17,17460
1948	16,76200
1949	18,02585
1950	16,88955
1951	16,09340
1952	17,06150
1953	17,00045
1954	17,09255
1955	17,28915
1956	16,33480
1957	15,54125
1958	16,80745
1959	17,10650
1960	16,29705
1961	17,47970
1962	17,24790
1963	16,26875
1964	16,14225
1965	15,99710
1966	16,42955
1967	15,88720
1968	16,73295
1969	15,79380
1970	16,44775
1971	14,54600
1972	15,90555
1973	16,42725
1974	14,92625
1975	15,62540
1976	15,03310
1977	17,55335
1978	16,16550
1979	16,06435

Année	Température
1980	16,86770
1981	17,13790
1982	17,03770
1983	17,24490
1984	16,57530
1985	16,64185
1986	15,69875
1987	18,02020
1988	17,08010
1989	16,88845
1990	16,52590
1991	15,65780
1992	16,38310
1993	16,09055
1994	16,84880
1995	17,79865
1996	17,16175
1997	18,04950
1998	17,16170
1999	18,12820
2000	16,24315
2001	18,58385
2002	17,31125
2003	17,39330
2004	16,53990
2005	18,64455
2006	19,15395
2007	17,03765
2008	17,54505
2009	18,51130
2010	18,23800
2011	18,61795
2012	17,30885
2013	17,99565
2014	19,20845
2015	18,90080
2016	19,10102
2017	18,12502
2018	17,80022

Annexe B

Annexes : Prévisions

B.1 Code R : Prévisions interpolation linéaire avec utilisation de la commande *solve*

```
# Definition de la matrice P diag=vector("double",N+1)
diag[1]=1/3
diag[N+1]=1/3
for(i in seq(2,N))
{
  diag[i]=2/3
}
sdiag=vector("double",N)
for(i in seq(1,N))
{
  sdiag[i]=1/6
}
P=matrix(nrow=N+1,ncol=N+1,0)
P=triDiag(diag, sdiag, sdiag)

# Definition de la matrice P^{-1}
Pinv=matrix(nrow=N+1,ncol=N+1,0)
Pinv=Solve.tridiag(sdiag,diag,sdiag,diag(N+1))

# Creation des systemes a resoudre pour les previsions 1
(avec Pinv) et 2 (avec P)
b1=vector("double",N)
b2=vector("double",N)
for(i in seq(1,N))
{
  b1[i]=Pinv[N+1,i];
  b2[i]=P[N+1,i];
}
Q1=matrix(nrow=N,ncol=N,0)
Q2=matrix(nrow=N,ncol=N,0)
for(i in seq(1,N))
```

```
{
  for(j in seq(1,N))
  {
    Q1[i,j]=Pinv[i,j];
    Q2[i,j]=P[i,j];
  }
}
poids1=vector("double",N)
poids2=vector("double",N)
poids1=solve(resol1,b1)
poids2=solve(resol2,b2)
prevision1=0
prevision2=0
for(i in seq(1,N))
{
 prevision1=prevision1+poids1[i]*temperature[i]
 prevision2=prevision2+poids2[i]*temperature[i]
}
prevision1=prevision1/sum(poids1[1:N])
prevision2=prevision2/sum(poids2[1:N])

graph_poids=data.frame(annee, poids1)
plot(graph_poids,type='p',pch=20,lwd=0.0001)
lines(graph_poids,type='h')

graph_poids2=data.frame(annee, poids2)
plot(graph_poids2,type='p',pch=20,lwd=0.0001)
lines(graph_poids2,type='h')

# PROPRIETES MATRICES
cond1=kappa(resol1,exact=TRUE)
vp1=eigen(resol1)
valmin1=min(abs(vp1$values))
cond2=kappa(resol2,exact=TRUE)
vp2=eigen(resol2)
valmin2=min(abs(vp2$values))
```

B.2 Code R : Prévisions spline quadratique avec contrainte avec introduction d'un paramètre de lissage

```
# Definition de la matrice P
diag=vector("double",N+1)
diag[1]=1/3
diag[N+1]=1/3
for(i in seq(2,N))
{
  diag[i]=2/3
}
sdiag=vector("double",N)
```

```
for(i in seq(1,N))
{
  sdiag[i]=1/6
}
P=matrix(nrow=N+1,ncol=N+1,0)
P=triDiag(diag, sdiag, sdiag)
# Definition de la matrice P^{-1}
Pinv=matrix(nrow=N+1,ncol=N+1,0)
Pinv=Solve.tridiag(sdiag,diag,sdiag,diag(N+1))
R=matrix(nrow = N+1, ncol = N+1, 0)
for(i in seq(2,N+1))
{
  for(j in seq(1,i))
  {
    R[i,j]=1
    if (j==1){
      R[i,j]=0.5
    }
    if (i==j)
    {
      R[i,j]=0.5
    }
  }
}
Q1=matrix(ncol = N+1, nrow = N+1, 0)
Q2=matrix(ncol = N+1, nrow = N+1, 0)
Q1=R%*%Pinv%*%t(R)
Q2=R%*%P%*%t(R)
b1=vector("double",N+1)
b2=vector("double",N+1)
for(i in seq(1,N))
{
  b1[i]=Q1[i,N+1]
  b2[i]=Q2[i,N+1]
}
b1[N+1]=1
b2[N+1]=1
for(i in seq(1,N))
{
  Q1[N+1,i]=1
  Q1[i,N+1]=1
  Q2[N+1,i]=1
  Q2[i,N+1]=1
}
Q1[N+1,N+1]=0
Q2[N+1,N+1]=0
lambda1=0.02
lambda2=0.025
Q1=Q1+lambda1*diag(N+1)
```

```
Q2=Q2+lambda2*diag(N+1)
poids1=vector("double",N+1)
poids1=solve(Q1,b1)
#poids1=ginv(Q1)%*%b1
poids2=vector("double",N+1)
poids2=solve(Q2,b2)
#poids2=ginv(Q2)%*%b2
prevision1=0
prevision2=0
for(i in seq(1,N))
{
  prevision1=prevision1+poids1[i]*temperature[i]
  prevision2=prevision2+poids2[i]*temperature[i]
}
```

B.3 Code C++ : Prévisions spline quadratique sans contrainte

```
vecteur prev_quad_sans_cont(vecteur x, vecteur y)
{
    int N=x.dim();
    matrice P(N+1,N+1);
    P(0,0)=P(N,N)=1./3;
    P(0,1)=P(N,N-1)=1./6;
    for(int i=1;i<N;i++)
    {
        P(i,i)=2./3;
        P(i,i-1)=P(i,i+1)=1./6;
    }
    matrice Pinv(N+1,N+1);
    Pinv=inverse(P);
    matrice resol1(N,N), resol2(N,N);
    for (int i=0;i<N;i++){
        for (int j=0; j<N; j++){
            resol1(i,j)=Pinv(i,j);
            resol2(i,j)=Pinv(i,j);
        }
    }
    matrice LU1(N,N),LU2(N,N);
    vecteur piv1(N),piv2(N),b1(N),b2(N);
    for (int i=0;i<N;i++){
        b1(i)=Pinv(N,i);
        b1(i)=P(N,i);
    }
    LU1=resol1.lu(piv);
    LU2=resol2.lu(piv);
    vecteur poids1(N), poids2(N);
    poids1=LU1.solvelu(b1,piv1);
    poids2=LU1.solvelu(b2,piv2);
```

```
    double prevision_q1=0, prevision_q2=0;
    vecteur q(N);
    q(0)=0;
    for(int i=1;i<N;i++)
    {
        q(i)=2*(y(i)-y(i-1))-q(i-1);
    }

    for(int i=0;i<N;i++)
    {
        prevision_q1=prevision_q1+poids1(i)*q(i);
        prevision_q2=prevision_q2+poids1(i)*q(i);
    }
    double prevision1=0;
    double prevision2=0;
    prevision1=0.5*q(N-1)+0.5*prevision_q1+y(N-1);
    prevision2=0.5*q(N-1)+0.5*prevision_q2+y(N-1);
    vecteur prevision(2);
    prevision(0)=prevision1;
    prevision(1)=prevision2;
    return prevision;
}
```

B.4 Code C++ : Prévision spline cubique avec contraintes

```
double prevision_1_spline_cubique(vecteur donnee_x, vecteur donnee_y,
                                  double annee)
{
    int N=donnee_x.dim();
    // Creation de la matrice A
    matrice A(N+1,N+1);
    A(0,0)=1.0;
    A(1,1)=1.0;
    for(int i=2;i<N+1;i++){
      A(i,i)=1.0;
      A(i,i-1)=-2.0;
      A(i,i-2)=1.0;}
    A=inverse(A);
    // Creation de la matrice P
    matrice P(N+1,N+1);
    P(0,0)=1./3;
    P(N,N)=1./3;
    P(0,1)=1./6;
    P(N,N-1)=1./6;

    for(int i=1;i<N;i++){
        P(i,i)=2./3;
        P(i,i-1)=1./6;
        P(i,i+1)=1./6;}
```

```
    // Inversion de la matrice P : On a la matrice de covariance
    matrice Pinv(N+1,N+1);
    Pinv=inverse(P);
    // Creation de la matrice B
    matrice B(N+1,N+1);
    for (int i=2;i<N+1;i++){
      B(i,i)=1./6;
      B(i,i-1)=1./3;
      B(i,i-2)=1./6;
    }
    //T1 et T2 contiennent les transposees de A et B
    matrice T1(N+1,N+1);
    matrice T2(N+1,N+1);
    matrice K(N+1,N+1);
    T1=T1.tr(A);
    T2=T2.tr(B);
    //K est le produit defini au chapitre 2
    K=A*B*Pinv*T2*T1;
    //Creation du second membre
    vecteur b(N+2);
    for (int i=0;i<N;i++){
      b(i)=K(i,N-1);}
    b(N)=1;
    b(N+1)=N+1;
    // Creation de la matrice appelee D dans notre chapitre 2
    matrice resol(N+2,N+2);
    for (int i=0;i<N;i++){
      resol(N,i)=1.0;
      resol(i,N)=1.0;
      resol(i,N+1)=i+1;
      resol(N+1,i)=i+1;
      for (int j=0;j<N;j++){
        resol(i,j)=K(i,j);}
    }
    // Resolution du systeme pour trouver les poids
    matrice LU(N+2,N+2);
    vecteur piv(N+2);
    LU=resol.lu(piv);
    vecteur poids(N+2);
    poids=LU.solvelu(b,piv);
    // Calcul de notre prevision
    double prevision=0;
    for(int i=0;i<N;i++){
        prevision=prevision+poids(i)*donnee_y(i);}
    return prevision;
}
```

B.5 Code R : Prévisions spline cubique sans contrainte avec utilisation de la fonction ginv

```
# Definition de la matrice P
diag=vector("double",N+1)
diag[1]=1/3
diag[N+1]=1/3
for(i in seq(2,N))
{
  diag[i]=2/3
}
sdiag=vector("double",N)
for(i in seq(1,N))
{
  sdiag[i]=1/6
}
P=matrix(nrow=N+1,ncol=N+1,0)
P=triDiag(diag, sdiag, sdiag)
# Definition de la matrice P^{-1}
Pinv=matrix(nrow=N+1,ncol=N+1,0)
Pinv=Solve.tridiag(sdiag,diag,sdiag,diag(N+1))
M=matrix(nrow=N-1,ncol=N+1,0)
for(i in seq(1,N-1))
{
  M[i,i]=1/6;
  M[i,i+1]=1/3;
  M[i,i+2]=1/6;
}
Q1=matrix(nrow=N-1,ncol=N-1,0)
Q2=matrix(nrow=N-1,ncol=N-1,0)
Q1=M%*%Pinv%*%t(M)
Q2=M%*%P%*%t(M)
b1=vector("double",N-2)
b2=vector("double",N-2)
for(i in seq(1,N-2))
{
  b1[i]=Q1[N-1,i];
  b2[i]=Q2[N-1,i];
}
D1=matrix(nrow=N-2,ncol=N-2,0)
D2=matrix(nrow=N-2,ncol=N-2,0)
for(i in seq(1,N-2))
{
  for(j in seq(1,N-2))
  {
    D1[i,j]=Q1[i,j]
    D2[i,j]=Q2[i,j]
  }
}
```

```
poids1=vector("double",N-2)
poids2=vector("double",N-2)
poids1=ginv(D1)%*%b1
poids2=ginv(D2)%*%b2

T_tilde=vector("double",N-2)
for(i in seq(1,N-2))
{
  T_tilde[i]=temperature[i+2]-2*temperature[i+1]+temperature[i];
}

prevision1=0
prevision2=0
for(i in seq(1,N-2))
{
  prevision1=prevision1+poids1[i]*T_tilde[i];
  prevision2=prevision2+poids2[i]*T_tilde[i];
}
prevision1=prevision1+2*temperature[N]-temperature[N-1]
prevision2=prevision2+2*temperature[N]-temperature[N-1]
```