Rapport TP2 - Partie 1: Interpolation polynomiale

YMERAJ Eldis

10/2022

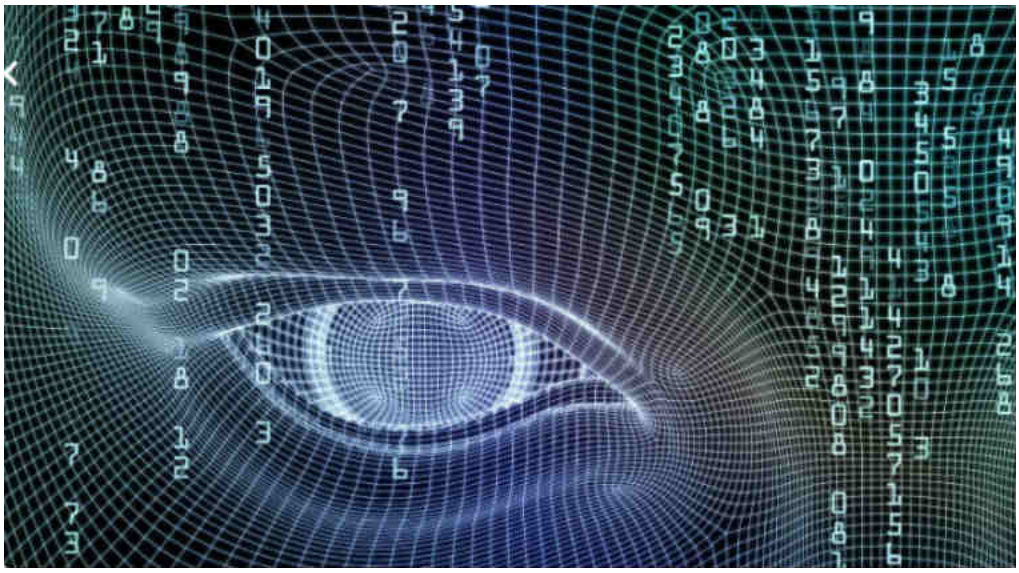

Figure 1 – Algorithmique Numérique

Table des matières

Introduction

Dans cette partie de rapport, nous allons voir l'implémentation des méthodes de Lagrange et Neville en langage C pour obtenir le polynôme définissant la relation entre un ensemble de données.

Mais qu'est ce que l'interpolation polynomiale ?

L'interpolation polynomiale est une technique d'interpolation d'un ensemble de données ou d'une fonction par un polynôme. En d'autres termes, étant donnée un ensemble de points obtenu, par exemple, à la suite d'une expérience, on cherche un polynôme qui passe par tous ces ponts et éventuellment vérifie d'autres conditions, de degré si possible le plus bas.
Nous considérons les polynômes comme outils pour résoudre des problèmes d'interpolation. Les algorithmes considérés deront donc uniquement des algorithmes d'évaluation de polynômes ou d'expression de polynômes dans différentes bases.

Interpolation de Lagrange

En analyse numérique, les polynômes de Lagrange, permettent d'interpoler une série de points par un polynôme qui passe exactement par ces points appelés aussi nœuds.
L'idée suivante est à la base des différentes méthodes d'interpolation polynômiale. Le polynôme P(N-1) est exprimé sous la forme :

$$P[N-1](x) = \sum_{i=0}^{N-1} y[i]l[i](x)$$

Où la fonction cardinale l[i](x) est déffinie par :

$$l[i](x) = \prod_{j=0, j!=i}^{N-1} \left(\frac{x - x[j]}{x[i] - x[j]} \right)$$

On peut remarquer que l[i](x[j]) vaut 0 pour j != i et 1 sinon. La fonction l[i](x) est le produit de N - 1 polynômes de degré 1, c'est donc un polynôme de degré N - 1.

Interpolation de Neville

Etant donnés N points (x[i], y[i]), il s'agit d'exprimer le polynôme P(N-1)=[x[1], x[2], . . . , x[N]] sur les points 1, 2, . . . , N en fonction des polynômes P(N-2)[x[1], x[2], . . . , x[N-1] et P(N-2)[x[2], x[3], . . . , x[N]] sur l'ensemble de points 1, 2, . . . , N-1 et 2, 3, . . . , N
On utilise la formule de Lagrange partant des polynômes d'interpolation P0[xi] de degré 0 pour chaque point i. La courbe associée à chacun de ces polynômes passe par le point (x[i],y[i]) i.e. quelque soit x, P0[xi](x)=y[i].
Methode :
Soient N points (x[i], y[i]) :

- Poser P0[xi](x) = y[i], quelque soit i = 1, . . . , N.
- Utiliser les valeurs précédentes 2 à 2 pour le calcul de P1[xi, xi+1], quelque soit i = 1, . . . , N 1.
- Poursuivre les calculs jusqu'à P(N-1)[x1, x2, . . . , xN].

1 Implémentation des fonctions en langage C

1.1 Fonctions

Lagrange(...) :

- Génère un chiffre qui représente la maturité choisit aléatoirement.
- Valeur de retour de type double.
- Entrée : Deux tableaux de type double pour garder les valeurs de nos coordonnées(x, y), une variable de type double qui garde la valeur de coordonné qu'on va tester et la taille de tableaux.
- En sortie : La fonction va nous retourner la valeur de f(x) pour le X qu'on a entré en suivant la méthode d'interpolation de Lagrange.

```
1  double Lagrange(double *x, double *y, double X, int n){
2
3      double L;
4      double P = 0;
5      //calcul de Li(x)
6      for (int i = 0; i < n; i++){
7          L = 1;
8          for (int j = 0; j < n; j++){
9              if (j != i)
10             {
11                 L = L*((X - x[j]) / (x[i] - x[j]));
12
13             }
14         }
15         //calcul du polynome
16         P = P + y[i]* L;
17
18     }
19     printf("Par la methode d'interpolation de Lagrange f(x) = %.5f\n\n",P);
20     return (P);
21 }
```

Notre fonction accepte comme paramétre un tableau pour la coordonnée xi ; un autre tableau pour la coordonnée yi, une valeur de type double qui est une point dans la tableau xi qu'o va tester. La fonction est tres facile, premièrement à l'aide d'une boucle imbriqué for on a calculé Li(x) et après la valeur de cette polynome. La fonction nous retourne la valeur de P(x) pour le point qu'on teste.

Neville(...) :

- Fonction qui implemente la methode d'interpolation de Neville.
- Valeur de retour de type double.
- Entrée : Deux tableaux de type double pour garder les valeurs de nos coordonnées(x, y), une variable de type double qui garde la valeur de coordonné qu'on va tester et la taille de tableaux.
- En sortie : La fonction va nous retourner la valeur de f(x) pour le X qu'on a entré en suivant la méthode d'interpolation de Lagrange.

```
1  double  Neville (double *x, double *y, double X, int n)
2  {
3
4      double  * P1=malloc(n*sizeof(double)),
5              * P0=malloc(n*sizeof(double));
6      P0=y;
7      for(int k=1;k<n-1;k++)
8      {
9          for(int i=0;i<n-k;i++)
10         {
11             P1[i]=((X-x[i+k])*P0[i]+(x[i]-X)*P0[i+1])/(x[i]-x[i+k]);
12
13         }
14
15         P0=P1;
16     }
17     printf("Par la methode d'interpolation de Neville, f(x) = %.5f\n", P1[0]);
18     return P1[0];
19 }
```

Notre fonction accepte comme paramétre un tableau pour la coordonnée xi ; un autre tableau pour la coordonnée yi, une valeur de type double qui est une point dans la tableau xi qu'o va tester. On a utilisé la formule de Lagrange partant les polynômes d'interpolation P0[xi] de degré 0 pour chaque point i. Pour chaque x, P0[xi](x) = yi. Dans la boucle imbriqué qu'on a calculé :

$$P1[xi, x(i+1)](x) = \left(\frac{(x - x[i+1])P0[xi](x) + (xi - x)P0[x[i+1]](x)}{xi - x[i+1]} \right)$$

$$Pk[xi, ..., x(x(i+k)](x) = \left(\frac{(x - x[i+k])P(k-1)[xi, ..., x(i+k-1)](x) + (xi - x)P(k-1)[x(i+1, ...x(i+k)](x)}{xi - x[i+k]} \right)$$

La fonction nous retourne la valeur de P(x) pour le point qu'on teste.

1.1.1 Main

Tout d'abord dans le main, on a initialisé les tableaux avec qu'on va faire les jeux d'essais. Le programme demande à l'utilisateur d'entrer la valeur de x d'un point qu'il veut tester et il retourne la valeur de f(x) par la méthode de Lagrange et Neville.

2 Jeux d'essais

Observons les résultats de quelques tests. Lagrange sera représenté en rouge, et Neville en bleu.

```
coordonnees (x, y):
|-001.00000, -001.50000|
|-000.50000, +000.00000|
|+000.00000, +000.25000|
|+000.50000, +000.00000|
|+001.00000, +000.00000|
```

La plupart des yi étant nuls, la méthode de Lagrange est particulièrement rapide ici.

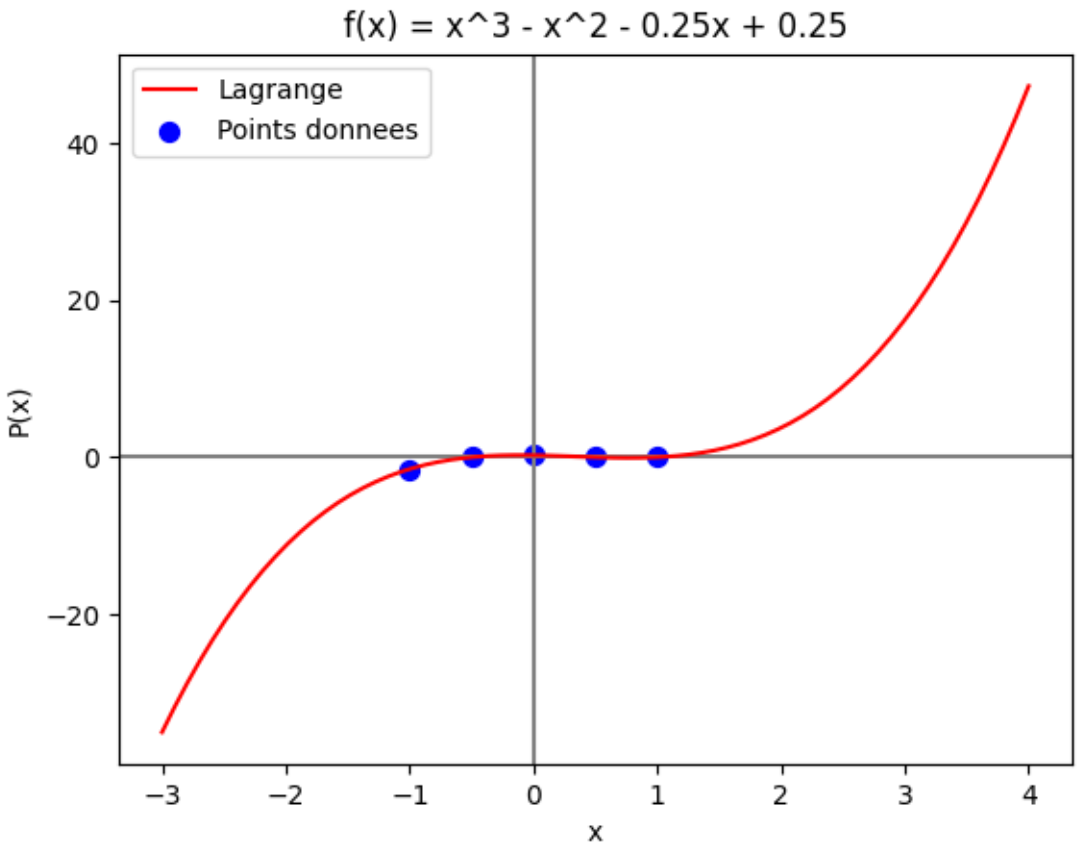

Figure 2 – Lagrange test 1

Polynôme pas simplifiée
P(x) = (x-(-0.50000))*(x)*(x-(0.50000))*(x-(1.00000))*(-1.000000000000000) +(x-(-1.00000))*(x-(-0.50000))*(x-(0.50000))*(x-(1.00000))*(+1.000000000000000)
On observe aussi que chaque xi et yi est un facteur d'une puissance de 2, ce qui explique qu'il n'y ait aucune perte de précision pendant la compilation. On observe aussi que chaque xi et yi est un facteur d'une puissance de 2, ce qui explique qu'il n'y ait aucune perte de précision pendant la compilation.

Neville, il nous donne un resultat plus simplifié que celui de Lagrange. Mais à la fain on voit que les deux méthodes donnent la même solution.

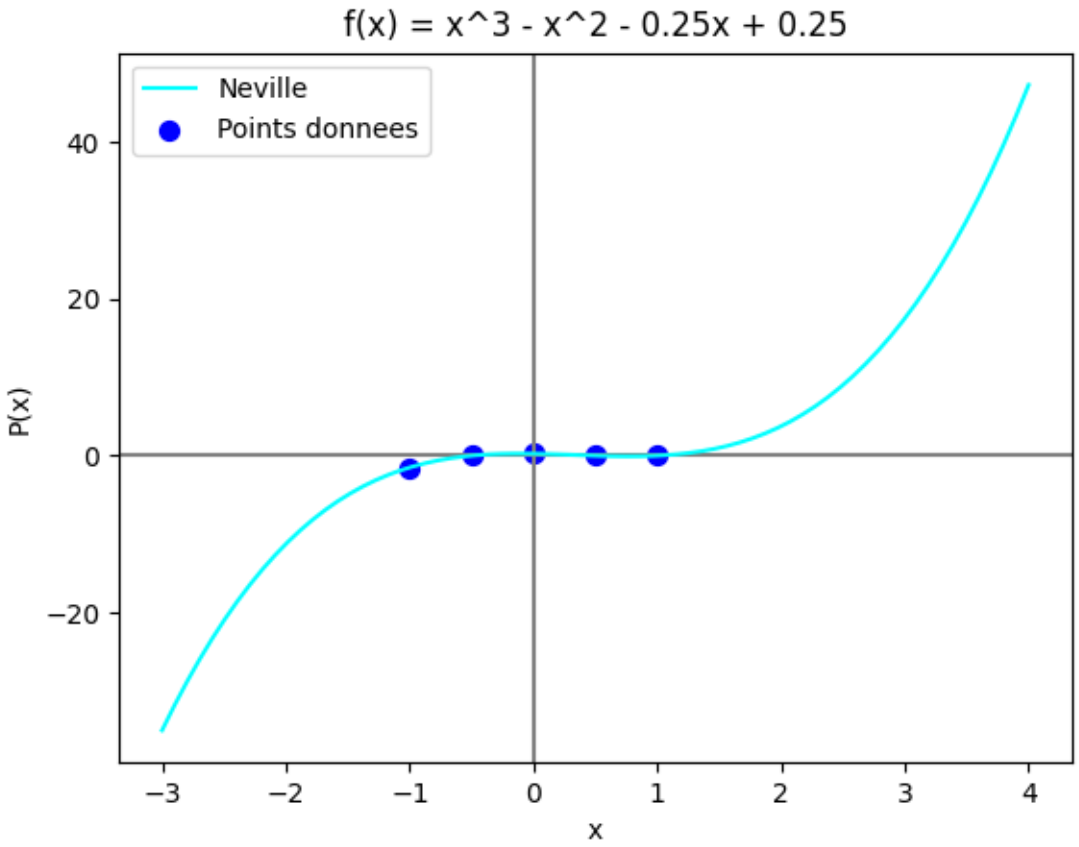

Figure 3 – Neville test 1

Polynôme pas simplifiée
P(x) = (x+1)*((x-1)*(x-1)+0.75)-1.5
Le test suivant illustre certaines conséquences de la perte de précision des floats, lorsqu'on s'éloigne des facteurs de puissances de 2.

```
Coordonnées (x, y):
|−005.00000, −002.00000|
|−001.00000, +006.00000|
|+000.00000, +001.00000|
|+002.00000, +003.00000|
```

P(x) = (x-(-1.0))*(x)*(x-(2.0))*(0.014285714285714)+(x-(-5.0))*(x)*(x-(200000))*(0.5)+ (x-(-5.0))*(x-(-1.0))*(x-(2.0))*(-0.1)+(x-(-5.0))*(x-(-1.00000))*(x)*(+0.071428571428571)

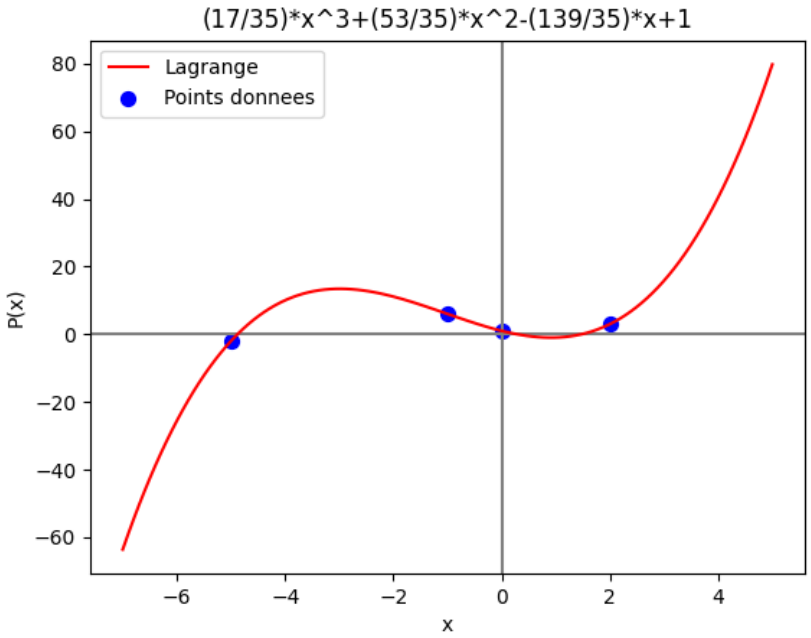

Figure 4 – Lagrange test 2

Non seulement perd-on de la précision avec Lagrange et Neville, mais les différences dans les étapes de calcul font que ces différences, entre le polynôme trouvé et le polynôme idéal, divergent entre elles.

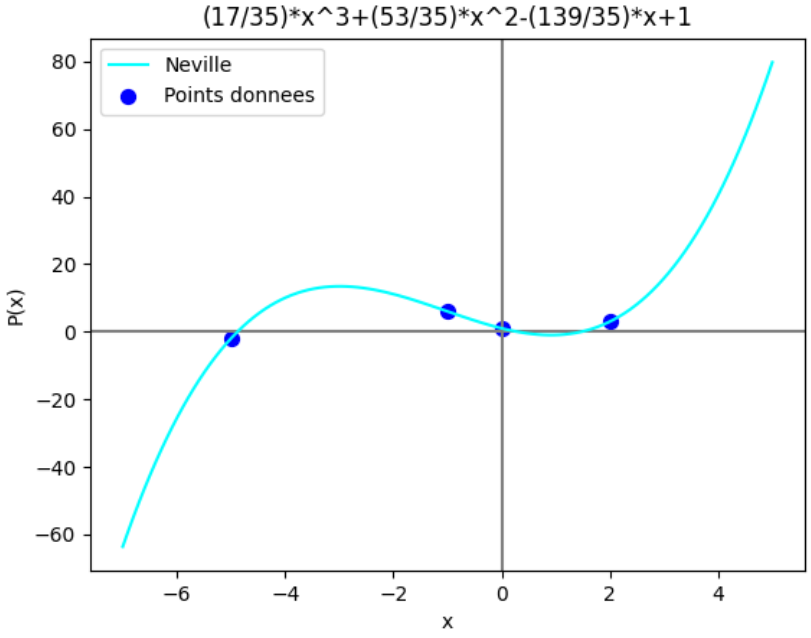

Figure 5 – Nevillee test 2

2.1 Jeu d'essai 4.1

Densité (D) de l'eau en fonction de la température (T)
Quand on examine un jeu de données avec des yi très proches les uns des autres,notre implémentation de Lagrange échoue.

```
Coordonnées (x, y):
|+000.00000, +000.99987|
|+002.00000, +000.99997|
|+004.00000, +001.00000|
|+006.00000, +000.99997|
|+008.00000, +000.99988|
|+010.00000, +000.99973|
|+012.00000, +000.99953|
|+014.00000, +000.99927|
|+016.00000, +000.99897|
|+018.00000, +000.99846|
|+020.00000, +000.99805|
|+022.00000, +000.99751|
|+024.00000, +000.99705|
...
```

P(x) = (x-(2.0))*(x-(4.0))*(x-(6.0))*(x-(8.0))*(x-(10.0))*(x-(12.0))*(x-(14.0))*(x-(16.0))*(x-(18.0))*(x-(20.0))*(x-(22.0))*(x-(24.0))*(x-(26.0))*(x-(28.0))*(x-(30.0))*(x-(32.0))*(x-(34.0))*(x-(36.0))*(x-(38.0))*(-0.0)+(x)*(x-(4.0))*(x-(6.0))....

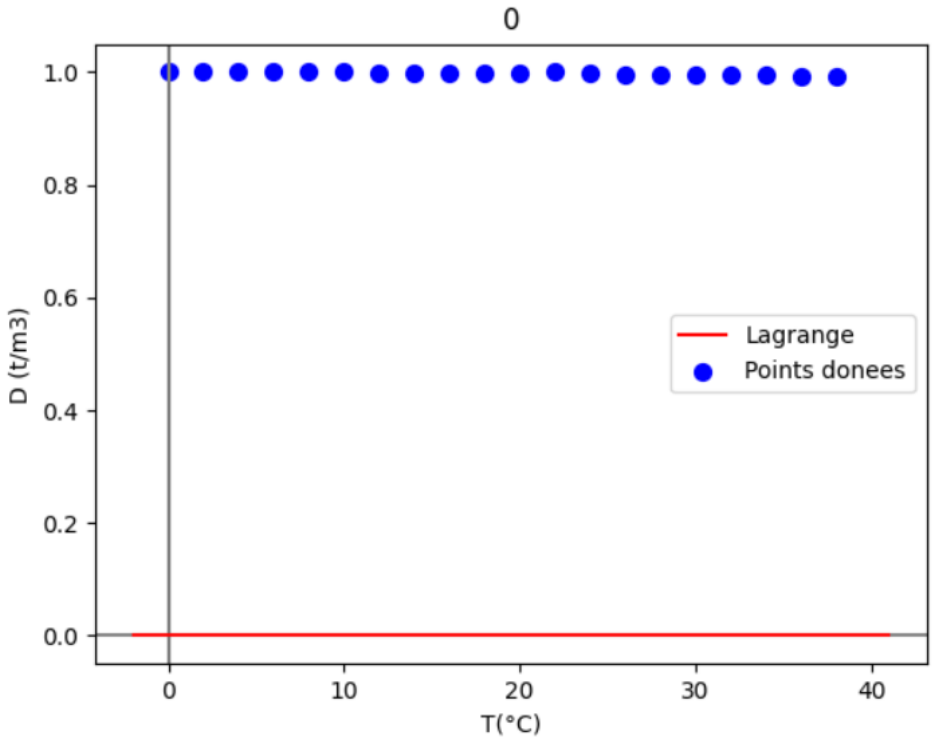

Figure 6 – Lagrange - (jeu d'essai 4.1)

On note au passage que la courbe qui décrit la densité de l'eau en fonction de la température devrait en fait ressembler à une parabole inversée définie dans R+. Avec des données plus proches de la perfection, et plusoptimales pour notre implémentation, on aurait probablement pu obtenir un polynôme de degré 2. Avec une dizaine de coordonnées, l'algorithme fonctionne. C'est-à-dire qu'on s'approche de l'approximation qu'est le polynôme unique que l'on recherche. On remarque que la solution n'est pas exactement la même entre nos fonctions Lagrange et Neville.

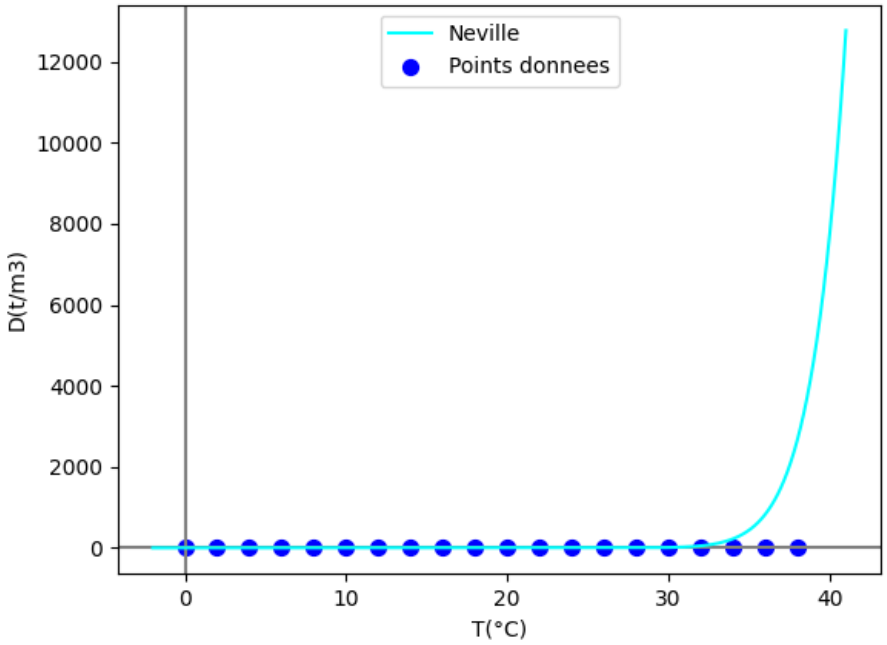

Figure 7 – Neville - (jeu d'essai 4.1)

2.2 Jeu d'essai 4.2

Dépenses mensuelles et revenus

```
Coordonnées (x, y):
|+752.00000, +085.00000|
|+855.00000, +083.00000|
|+871.00000, +162.00000|
|+734.00000, +079.00000|
|+610.00000, +081.00000|
|+582.00000, +083.00000|
|+921.00000, +281.00000|
|+492.00000, +081.00000|
|+569.00000, +081.00000|
...
```

Le nombre d'operations est auprès pres 1700. On a un grand nombre d'opérations, plus que 1500 opérations. Avec un large jeu de données représentées par des points plus espacés, Notre Lagrange, lui, n'est pas assez sensible aux facteurs minuscules que la méthode génère.

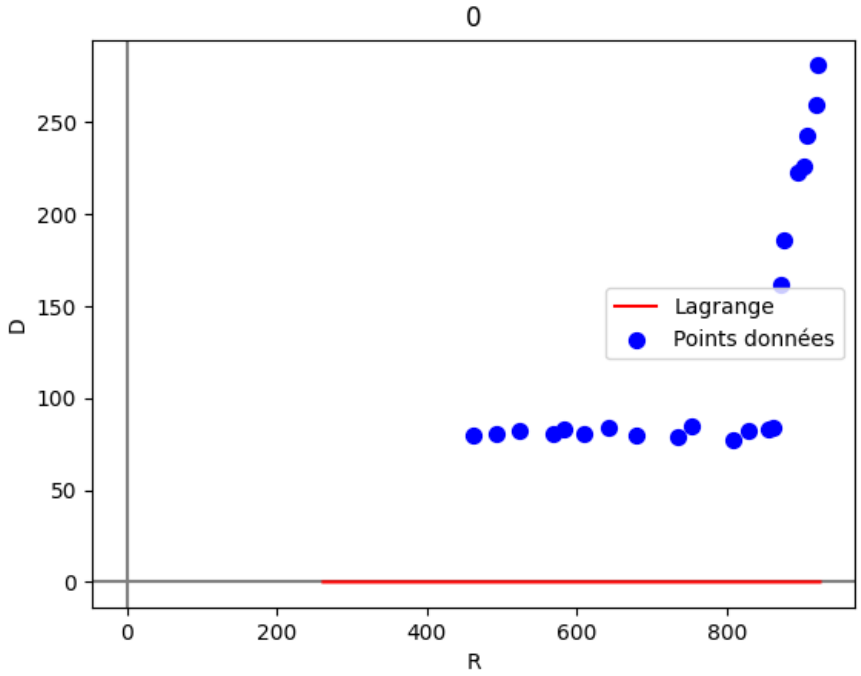

Figure 8 – Lagrange - (jeu d'essai 4.2)

Avec un large jeu de données représentées par des points plus espacés, Newton nous donne une courbe, mais qui peine encore à passer par tous les points. Notre Lagrange, lui, n'est pas assez sensible aux facteurs minuscules que la méthode génère (inférieurs à 2 -52).

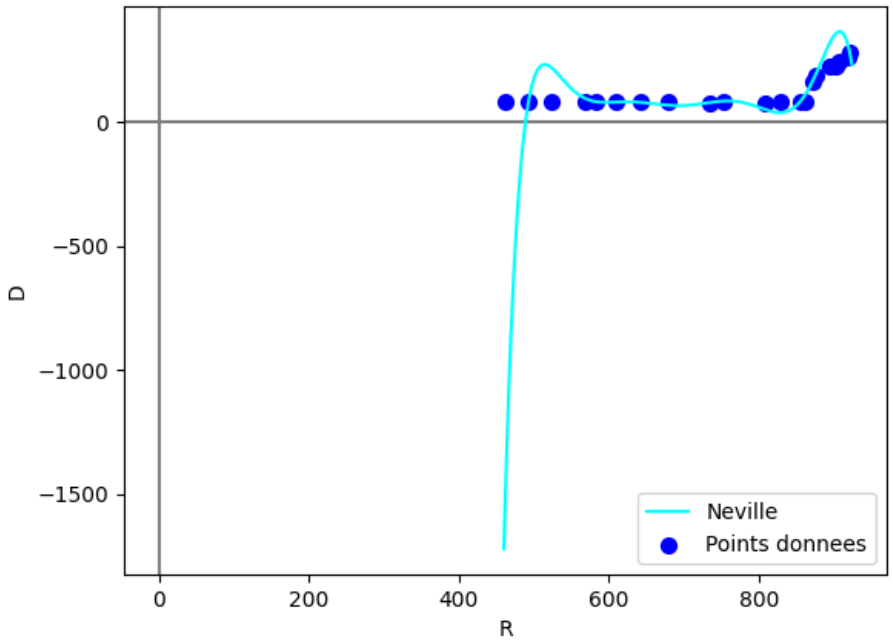

Figure 9 – Neville - (jeu d'essai 4.2)

2.3 Jeu d'essai 4.3

Série S dûe à Anscombe
Quand on examine un jeu de données avec des yi très proches les uns des autres,notre implémentation de Lagrange échoue.

```
Coordonnées (x, y):
|+010.00000, +008.04000|
|+008.00000, +006.95000|
|+013.00000, +007.58000|
|+009.00000, +008.81000|
|+011.00000, +008.33000|
|+014.00000, +009.96000|
|+006.00000, +007.24000|
|+004.00000, +004.26000|
|+012.00000, +010.84000|
|+007.00000, +004.82000|
|+005.00000, +005.68000|
```

P(x) = 8.039999999999999+0.544999999999999*(x-(10.0))-0.139666666666666*(x-(10.0))*(x-(8.0))+0.293833333333334*(x-(10.0))*(x-(8.0))*(x-(13.0))-0.160583333333334*(x-(10.0))*(x-(8.0))*(x-(13.0))*(x-(9.0))+...

Avec une dizaine de coordonnées, l'algorithme fonctionne. C'est-à-dire qu'on s'approche de l'approximation qu'est le polynôme unique que l'on recherche. On remarque que la solution est presque la même entre nos fonctions Lagrange et Neville, et que Neville est généralement bien plus rapide.

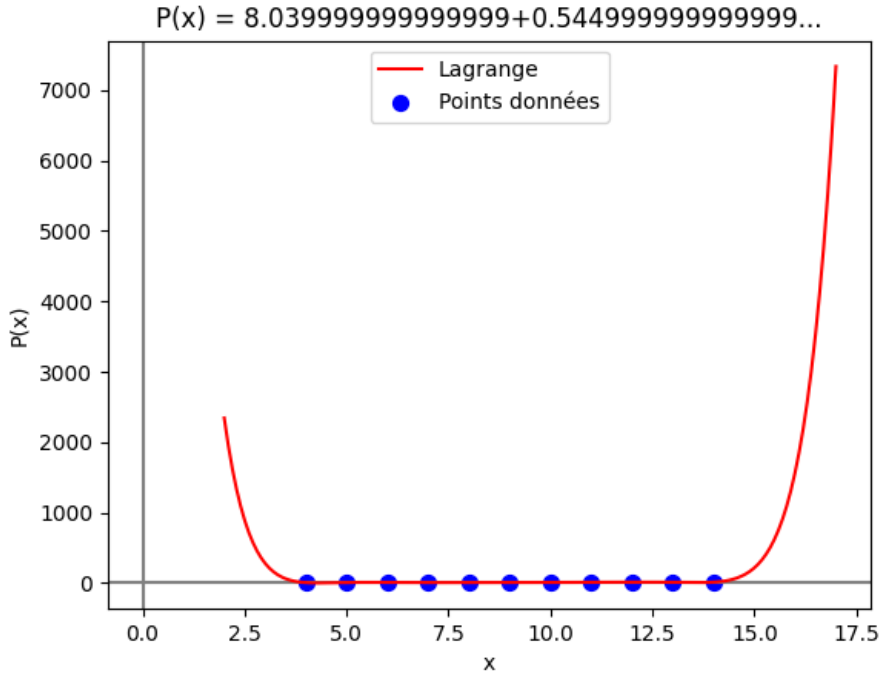

Figure 10 – Lagrange - (jeu d'essai 4.3)

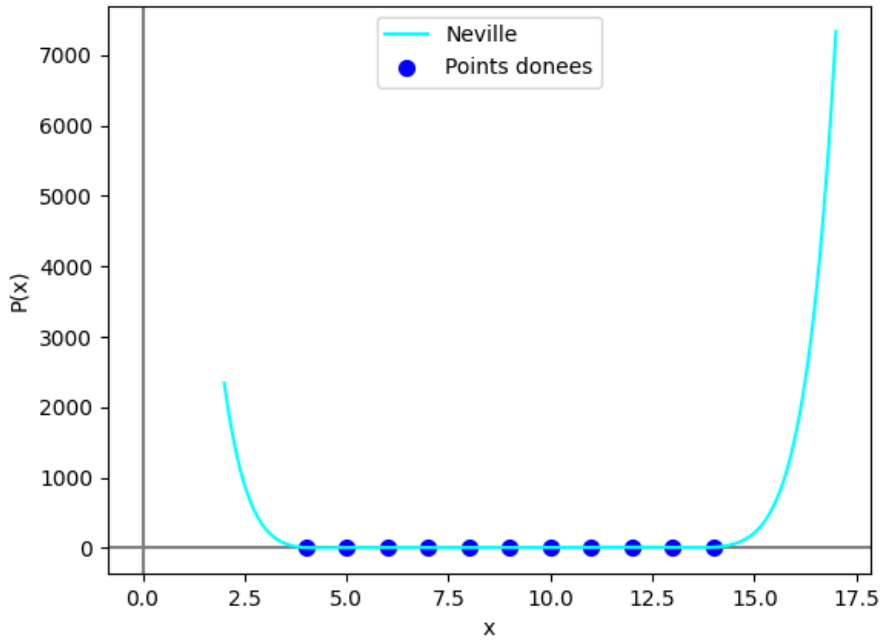

Figure 11 – Neville - (jeu d'essai 4.3)

Conclusion

Souvent, on se trouve avec des données qu'on sait liées les unes aux autres, sans connaître de formule qui les lie. Ces données, de plus, sont généralement légèrement imprécises. L'interpolation est alors une démarche pleinement justifiée : on acquiert plus de données sur le sujet-problème, et leur manque de précision absolue est relativement négligeable vu d'où on part, si l'on prend garde de ne pas oublier que ce sont des approximations.

L'algorithme de Neville est une méthode récursive du calcul de la valeur du polynôme d'interpolation en un point donné, avec lequel il est aisé d'ajouter des points d'interpolation au fur et à mesure. Il est moins adapté pour fournir une expression du polynôme d'interpolation.

Si le polynome à beaucoup de racines, Lagrange peut être très rapide. Autrement, Neville offre des solutions plus rapidement que Lagrange, notamment quand le nombre de points connus est élevé. Neville tend à manipuler des fractions moins petites que Lagrange, car Lagrange peut plus facilement accumuler des fractions de fractions à chaque coefficient ; si ces nombres ont des décimales au delà de $1 * 10^{-15}$, la solution diverge de celle qu'on pourrait trouver à la main ou avec Neville en C. En fait, si ces nombres sont inférieurs à 2^{-52}, ils seront considérés comme indistincts de 0, et notre "solution" pourra facilement perdre des degrés ou être le polynôme nul.

Table des figures