

0.1 Interpolation par la méthode de Newton

0.1.1 Introduction

La méthode de newton est une méthode d'interpolation qui permet de rendre le discret continu. C'est-à-dire que la méthode de newton peut établir, à partir d'un groupement de points, un polynome qui permet de tous les joindre.

Quelques observations

Pour comprendre au mieux cette méthode d'interpolation, nous remarquerons que le polynome d'interpolation peut s'écrire de la forme suivante :

$$P_{N-1}(x) = b_0 + b_1(x - x_1) + b_2(x - x_1)(x - x_2) + \dots + b_{N-1}(x - x_1) \dots (x - x_{N-1})$$

Où N est le nombre de points à interpoler.

$x_i, \forall i = 1, \dots, N-1$ désigne l'élément i de la matrice des abscisses des points à interpoler et $b_i, \forall i = 0, \dots, N-1$, la $i^{\text{ème}}$ différence divisée.

On déduira alors que la méthode de Newton produira un polynome de degré au plus $N-1$ pour N point.

0.1.2 Différence Divisée

Dans cette section, x_i (respect. y_i désigne l'abscisse (respect. l'ordonnée) du point i et b_i la $i^{\text{ème}}$ différence divisée. Comme mentionné dans 0.1.1, le polynome, pour exister, a besoin des **Différences Divisées**, notée b_i .

Elles s'obtiennent en cherchant les coefficients b_i tel que :

$$P_{N-1}(x_i) = y_i, \forall i = 1, \dots, N$$

Cela revient à résoudre le système linéaire suivant :

$$\begin{cases} y_1 &= P_{N-1}(x_1) = b_0 \\ y_2 &= P_{N-1}(x_2) = b_0 + (x_2 - x_1)b_1 \\ \dots &= \dots = \dots \\ y_N &= P_{N-1}(x_N) = b_0 + (x_N - x_1)b_1 + \dots + (x_N - x_1) \dots (x_N - x_{N-1})b_{N-1} \end{cases}$$

Notation

On synthétisera le système obtenu précédemment ainsi :

$$\text{La différence divisée } i \text{ de degrés } k : \nabla^k y_i = \frac{\nabla^{k-1} y_i - \nabla^{k-1} y_k}{x_i - x_k}, i = k+1, \dots, N.$$

Conséquences

Le coefficient b_i est donc calculable ainsi :

$$b_i = \begin{cases} y_1 & \text{si } i = 0 \\ \nabla^i y_{(i+1)} & \forall i = 1, \dots, N-1 \end{cases}$$

La seconde conséquences est la réécriture du polynome comme suit :

$$\begin{aligned} P_0(x) &= b_{N-1} \\ P_1(x) &= b_{N-2} + (x - x_{N-1})P_0(x) \\ &\dots = \dots \\ P_{N-1}(x) &= b_0 + (x - x_1)P_{N-2}(x) \end{aligned}$$

0.1.3 Résolution Manuelle

Soient les points donnés dans le tableau ci-dessous. Déterminer le polynôme interpolateur $p(x)$ de ces points.

x_i	2	6	4
y_i	4	1.5	-2

Calcul des polynômes pour $k=0$

$$\begin{aligned}p_0[x_0](x) &= y_0 = 4 \\p_0[x_1](x) &= y_0 = 1.5 \\p_0[x_2](x) &= y_0 = -2\end{aligned}$$

Calcul des polynômes pour $k=1$

$$\begin{aligned}p_1[x_0, x_1](x) &= \frac{(x - x_1)p_0[x_0](x) + (x_0 - x)p_0[x_1](x)}{x_0 - x_1} = \frac{(x - 6) \times 4 + (2 - x) \times 1.5}{2 - 6} \\&= \frac{2.5x - 21}{-4} = -0.625x + 5.25 \\p_1[x_1, x_2](x) &= \frac{(x - x_2)p_0[x_1](x) + (x_1 - x)p_0[x_2](x)}{x_1 - x_2} = \frac{(x - 4) \times 1.5 + (6 - x) \times (-2)}{6 - 4} \\&= \frac{3.5x - 18}{2} = 1.75x - 9\end{aligned}$$

Calcul du polynôme pour $k=2$

$$\begin{aligned}p_2[x_0, x_1, x_2](x) &= \frac{(x - x_2)p_1[x_0, x_1](x) + (x_0 - x)p_1[x_1, x_2](x)}{x_0 - x_2} = \frac{(x - 4)(-0.625x + 5.25) + (2 - x)(1.75x - 9)}{2 - 4} \\&= \frac{-2.375x^2 + 20.25x - 39}{-2} = 1.1875x^2 - 10.125x + 19.5\end{aligned}$$

Nous avons donc bien notre polynôme interpolateur de degré $n - 1 = 3 - 1 = 2$, qui passe par tous les points donnés :

$$p(x) = 1.1875x^2 - 10.125x + 19.5$$

Nous pouvons vérifier cela :

$$\begin{aligned} p(x_0) &= p(2) = 1.1875 \times 2^2 - 10.125 \times 2 + 19.5 = 4 \\ p(x_1) &= p(6) = 1.1875 \times 6^2 - 10.125 \times 6 + 19.5 = 1.5 \\ p(x_2) &= p(4) = 1.1875 \times 4^2 - 10.125 \times 4 + 19.5 = -2 \end{aligned}$$

0.1.4 Algorithme

Nous allons poser l'algorithme suivant, qui reprend simplement la relation de récurrence vue dans la section ??.

```

Fonction calculerPolynom(data, k, i, nbPoints):
    si k=0:
        renvoyer p0[xi](x)
    sinon:
        renvoyer  $\frac{(x-x_{i+k}) \times \text{calculerPolynom}(\text{data}, k-1, i, \text{nbPoints}) + (x_i - x) \times \text{calculerPolynom}(\text{data}, k-1, i+1, \text{nbPoints})}{x_i - x_{i+k}}$ 

```

0.1.5 Implémentation en C

Pour l'implémentation en C de cet algorithme, plusieurs contraintes nous font obstacle. Avant de commencer à implémenter notre code, nous devons les citer et trouver un moyen de les franchir.

Nous devons réfléchir à :

- De quelle manière stocker les points donnés ?
- Comment représenter un polynôme dans la mémoire ?
- Comment réaliser des opérations sur ces polynômes ?

Voici les décisions prises pour répondre à ces questions :

De quelle manière stocker les points donnés ?

Pour stocker les points fournis en entrée de notre programme, j'ai opté pour l'utilisation d'une matrice de flottants. La première ligne de cette matrice correspondra aux abscisses des points, tandis que la deuxième ligne contiendra leurs ordonnées. Sont alors mises en places des fonctions pour la création, le remplissage et l'affichage de la matrice, ainsi qu'une fonction pour libérer la mémoire allouée.

Ainsi, soient les points (1, 2), (2, 4), (5, 3), notre matrice contenant nos données sera alors $\begin{pmatrix} 1 & 2 & 5 \\ 2 & 4 & 3 \end{pmatrix}$.

Comment représenter un polynôme dans la mémoire ?

Pour représenter un polynôme en mémoire, j'ai choisi d'utiliser une nouvelle structure de données définissant le type **__polynom**. Chaque élément de cette structure se voit attribuer un entier **de-gree**, qui correspond au degré maximal du polynôme, ainsi qu'un tableau de nombres flottants **coefficients**. Les coefficients sont stockés dans l'ordre de la base canonique de l'espace vectoriel $\mathbb{R}_n[X]$. Autrement dit, le polynôme $3x^2 + 4x - 2$ serait représenté par le tableau $[-2, 4, 3]$.

J'ai donc implémenté la structure `__polynom` ainsi que les fonctions suivantes :

- La fonction `createPolynom` crée un nouveau polynôme en initialisant le degré et les coefficients.
- La fonction `printPolynom` permet d'afficher le polynôme en console.
- La fonction `freePolynom` libère la mémoire allouée pour stocker le polynôme, évitant ainsi les fuites de mémoire.

Comment réaliser des opérations sur ces polynômes ?

Dans le but d'implémenter la méthode de Neville, nous avons besoin de pouvoir manipuler des polynômes. Pour cela, j'ai codé les opérations nécessaires : l'addition de polynômes, la multiplication de polynômes, et la division d'un polynôme par un flottant.

Addition de polynôme

L'opération d'addition de deux polynômes est relativement simple. Pour ce faire, il suffit d'additionner un à un les coefficients du polynôme de degré minimum, avec les coefficients de l'autre polynôme. Les résultats de ces additions sont ensuite stockés dans un nouveau polynôme, que nous renvoyons en sortie de la fonction d'addition.

Multiplication de polynôme

La multiplication de polynôme, bien qu'aisée à effectuer manuellement, s'avère plus ardue lorsqu'il s'agit de l'implémenter de manière efficace. Heureusement ici, nous remarquons que la méthode de Neville n'utilise que la multiplication d'un polynôme de degré 1 avec un polynôme de degré k . Cela facilitera grandement notre travail.

Pour ce faire, nous prenons en paramètre de la fonction un nouveau polynôme de degré $k + 1$. Ensuite, nous remplissons ce polynôme en multipliant chaque coefficient du polynôme de degré k par le coefficient associé à X de l'autre polynôme. Enfin, nous additionnons ces résultats avec les produits des coefficients du polynôme de degré k avec le coefficient restant du polynôme de degré 1. Nous stockons cela dans le nouveau polynôme.

Pour éclaircir tout cela, voici un exemple :

Supposons que l'on veuille réaliser le produit $(2X - 1)(3X^2 + 2X + 1)$.

Les polynômes sont respectivement représentés par les tableaux $[-1, 2]$ et $[1, 2, 3]$.

1. Nous avons notre nouveau polynôme de degré 3 : $[., ., ., .]$
2. Nous multiplions les coefficients $[1, 2, 3]$ par le coefficient devant X de l'autre polynôme, soit 2. On obtient : $[., 2, 4, 6]$.
3. Enfin, on multiplie les coefficients $[1, 2, 3]$ par l'autre coefficient de l'autre polynôme, soit -1 . On ajoute ces produits dans le nouveau polynôme, ce qui donne : $[-1, 0, 1, 6]$.

Le produit de $(2X - 1)(3X^2 + 2X + 1)$ est donc $6X^3 + X^2 - 1$.

Division de polynôme

Pour effectuer la division d'un polynôme par un flottant, nous stockons simplement, dans un nouveau polynôme, les quotients résultants de la division de chaque coefficient du polynôme par le diviseur fourni en paramètre.

0.1.6 Exemples d'exécution

Voici les différentes sorties du programme pour l'interpolation des jeux de données présents en annexe de ce document. Vous trouverez également un graphe représentant les points donnés, ainsi que le polynôme trouvé.

Listing 1 – Annexe 1 data results

```
Polynom
0.000000x^19 - 0.000000x^18 + 0.000000x^17 - 0.000000x^16 +
0.000000x^15 - 0.000000x^14 + 0.000000x^13 - 0.000000x^12 +
0.000001x^11 - 0.000025x^10 + 0.000361x^9 - 0.004044x^8 +
0.035004x^7 - 0.229974x^6 + 1.116527x^5 - 3.842630x^4 +
8.760551x^3 - 11.683978x^2 + 6.758180x^1 + 0.999870
```

Temps d'execution : 0.096877 secondes

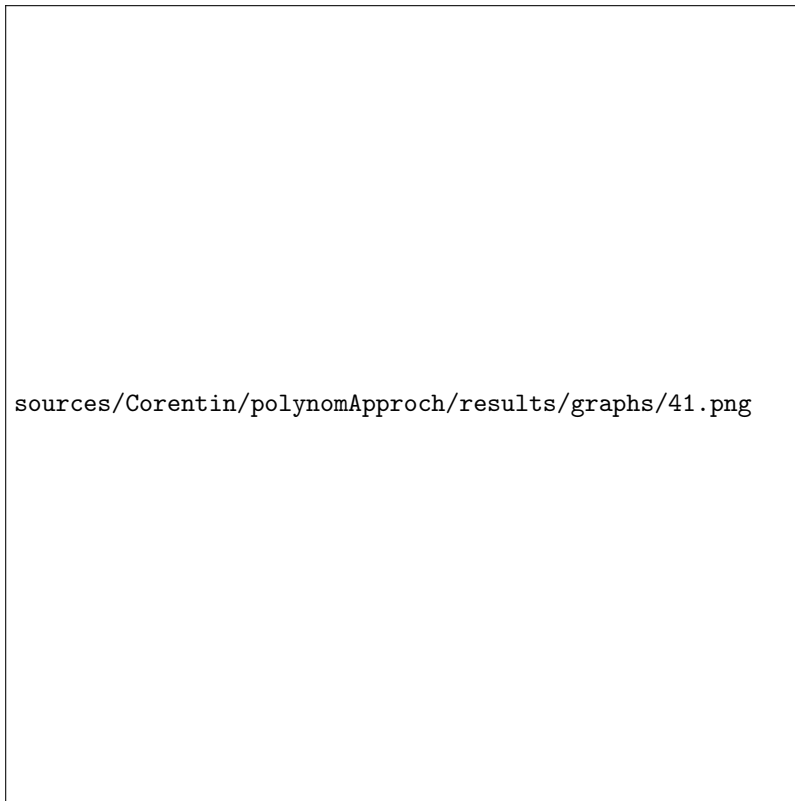
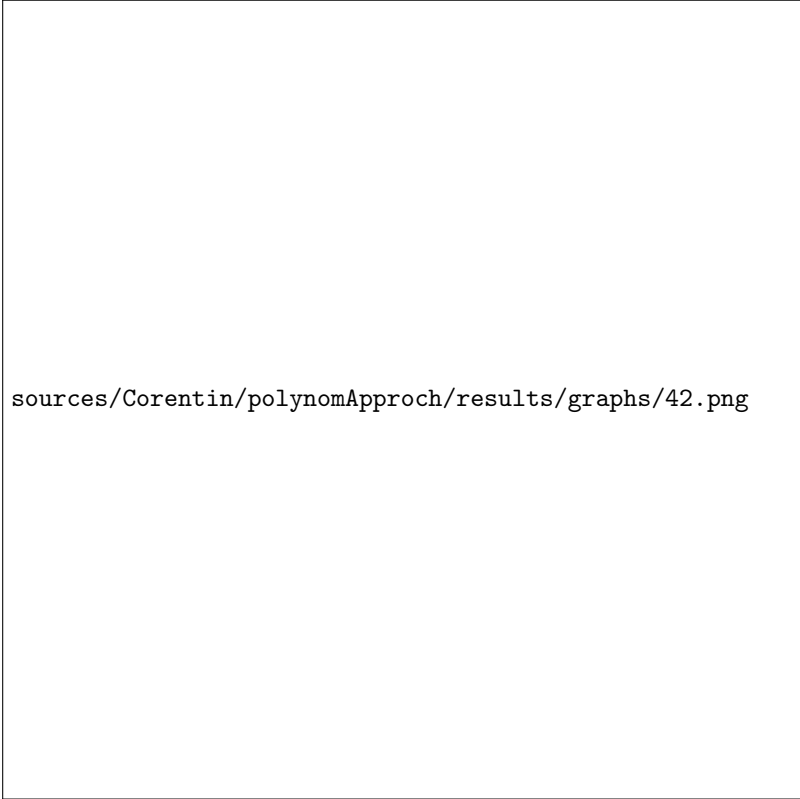


FIGURE 1 – Interpolation du jeu de données 1 de l'annexe

Listing 2 – Annexe 2 data results

Polynom
 $0.000000x^{20} - 0.000000x^{19} + 0.000000x^{18} - 0.000000x^{17} +$
 $0.000000x^{16} - 0.000000x^{15} + 0.000000x^{14} - 0.000000x^{13} +$
 $0.000000x^{12} - 0.000002x^{11} + 0.001309x^{10} - 0.860809x^9 +$
 $465.832642x^8 - 206286.906250x^7 + 74020648.000000x^6 -$
 $21189636096.000000x^5 + 4725708685312.000000x^4$
 $- 791294909612032.000000x^3 + 93583403189796864.000000x^2 -$
 $6969840492355780608.000000x^1 + 245843147369784279040.000000$

Temps d'exécution : 0.196364 secondes



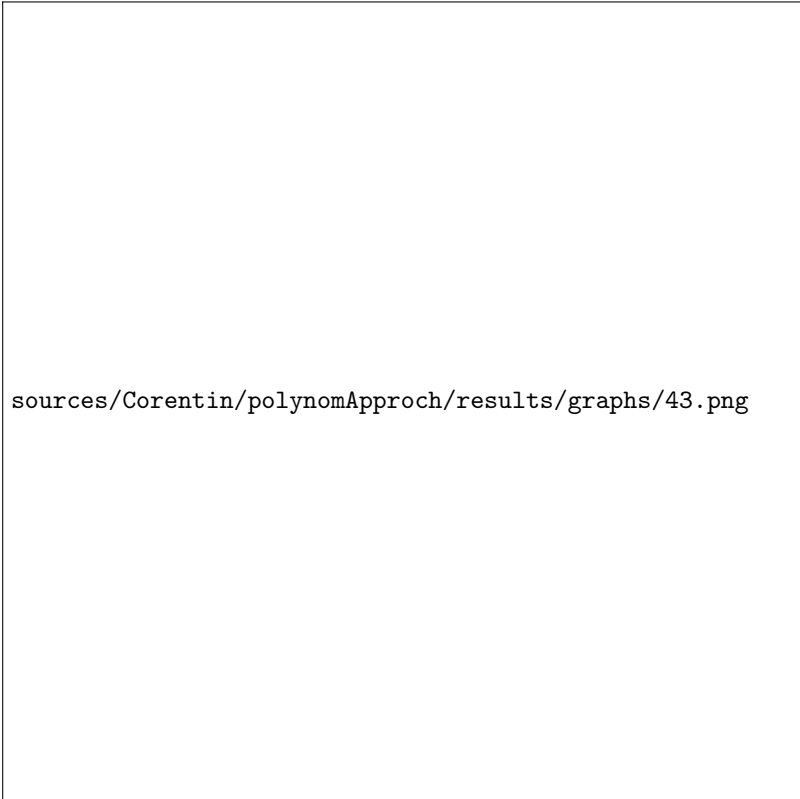
sources/Corentin/polynomApproch/results/graphs/42.png

FIGURE 2 – Interpolation du jeu de données 2 de l'annexe

Listing 3 – Annexe 3 data results

```
Polynom  
0.000012x^10 - 0.001164x^9 + 0.049084x^8 - 1.220944x^7 +  
19.789038x^6 - 217.668701x^5 + 1639.865601x^4 -  
8326.726562x^3 + 27183.572266x^2 - 51370.457031x^1 + 42569.960938
```

Temps d'execution : 0.000282 secondes



sources/Corentin/polynomApproch/results/graphs/43.png

FIGURE 3 – Interpolation du jeu de données 2 de l'annexe