

## 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 conséquence est la réécriture du polynome comme suit :

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

### Remarque

Le système de calcul de la différence divisée peut être visualiser comme une liste où on peut écraser l'élément  $k$  par sa différence divisée.  
Ceci nous sera utile lors de l'implémentation (utilisation de tableau et non de matrice).

### 0.1.3 Résolution Manuelle

Mettons en application la méthode de newton

$x_i$	2	6	4
$y_i$	4	1.5	-2

Calcul des différences divisées

$$\nabla^1 y_{(1)} = \frac{y_1 - y_0}{x_1 - x_0} = -0.625$$

$$\nabla^1 y_{(2)} = \frac{y_2 - y_0}{x_2 - x_0} = -3$$

$$\nabla^2 y_{(2)} = \frac{\nabla y_2 - \nabla y_1}{x_2 - x_1} = 1.1875$$

Tableau des différences divisées

$x$	$y$	$\nabla$	$\nabla^2$
2	$4 = b_0$		
		$-0.625 = b_1$	
6	1.5		
			$1.1875 = b_2$
		$-3$	
4	-2		

Calcul du polynôme

$$P_0(x) = b_2 = 1.1875$$

$$P_1(x) = -0.625 + (x - 6) \times P_0 = 1.1875x - 7.75$$

$$P_2(x) = 4 + (x_2)P_1 = 1.1875x^2 - 10.125x + 19.5$$

Le polynome interpolateur du groupement de points donné est donc le trinome :

$$P_2(x) = 1.1875x^2 - 10.125x + 19.5$$

### 0.1.4 Algorithme

Nous allons donc détailler les principales fonctions qui permettront par la suite l'implémentation de la méthode. Dans toute cette section,  $X$ ,  $Y$ ,  $DD$ ,  $E$ ,  $ne$ ,  $XN$  et  $P$  désigneront respectivement : les abscisses des points à interpoler, les ordonnées des points à interpoler, le tableau des différences divisées, Le tableau contenant l'évaluation du polynome sur un espace linéairement réparti, le nombre de nombre à générer de manière équitable sur un intervalle, un tableau contenant l'intervalle lineaire, un tableau contenant les coefficients du polynome

Listing 1 – "Divided Difference function"

```

Fonction DividedDifference(X, Y, DD):
    DD ← Y
    n ← X.length()
    for i from 0 to n-1:
        for j from n-1 to i+1 by step of -1:
            DD[j] ←  $\frac{D[j]-D[j-1]}{X[j]-X[j-i-1]}$ 
        end
    end
end

```

Listing 2 – "interpolate function"

```

Fonction double interpolate(DD,X,x):
    double eval ← 0
    n ← X.length()
    for i from n to 0 by step of -1:
        eval ← eval × (x - X[i]) + DD[i]
    end
    return eval

```

Listing 3 – "find coefficient function"

```

Fonction coef(P, DD, X):
    n ← X.length()
    P[0] ← DD[n-1]
    for i from n-2 to 0 by step of -1:
        for j from n-i-1 to j+1 by step of -1:
            P[j] ← P[j-1] - X[i] × P[j]
        end
        P[0] ← DD[0] - X[i] × P[0]
    end
end

```

Il s'agit uniquement de l'implémentation de la formule suivante :

$$P_{N-1}(x) = b_0 + (x - x_1)P_{N-2}(x)$$

Pour générer un espace linéairement peuplé en fonction de  $ne$ , on générera les nombres ainsi :

Listing 4 – "generate linear space"

```

some code:
for i from X[0] to X[X.length-1] by step of  $\frac{max(X)-min(X)}{ne}$ 

```

some code

### 0.1.5 Implémentation en C

Pour implémenter l'interpolation de newton, nous utiliserons ces préceptes :

- Les données seront stockées sous notation scientifique (pour ne pas avoir d'erreur d'arrondie lors de l'utilisation en python)
- Le type polynome qui est un composée d'un entier **deg** et d'un tableau de coefficient double.
- l'obtention du polynome d'interpolation sera comparé avec le resultat produit par sympy
- Le programme prend 3 paramètres : le fichier input, le fichier output et enfin un entier qui déterminera en combien de morceau équitable voulons nous ségmenter l'intervalle  $[X[0], X[X.length-1]]$

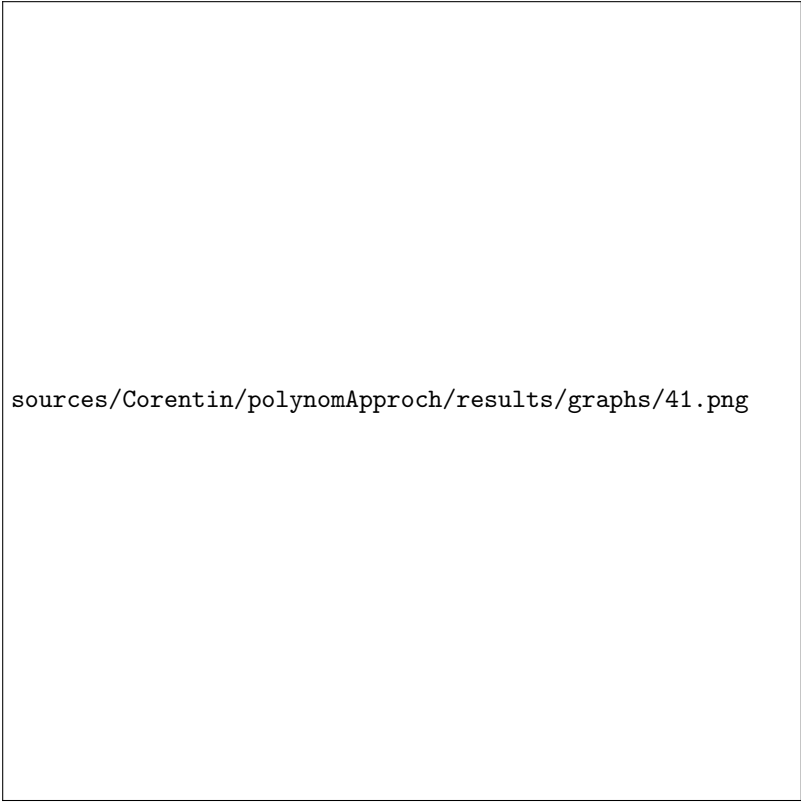
### 0.1.6 Exemples d'exécution

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 5 – 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



sources/Corentin/polynomApproch/results/graphs/41.png

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

# Listing 6 – 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

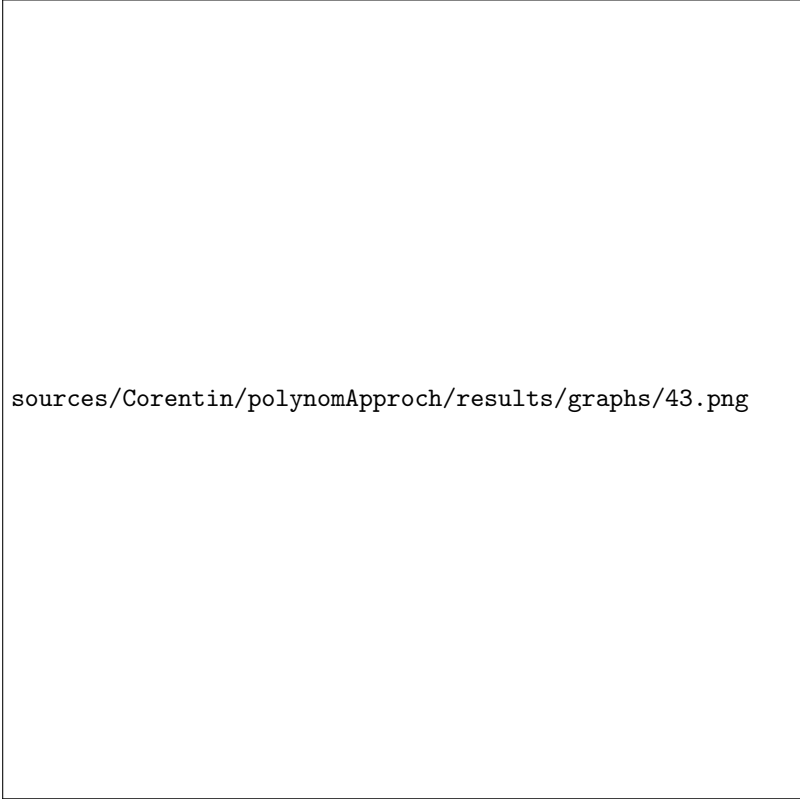
<sources/Corentin/polynomApproch/results/graphs/42.png>

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

### Listing 7 – 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