Interpolation de Lagrange

Lors d'un suivi temporel d'une grandeur g, on procède généralement à une numérisation du signal. On dispose alors d'un ensemble de $n+1$ points de coordonnées (t_i, y_i) (avec $0 \leqslant i \leqslant n$ et les t_i deux à deux distincts), où pour tout $i \in [\![0\,;n]\!]$, $y_i = g(t_i)$ est la valeur mesurée à l'instant t_i.

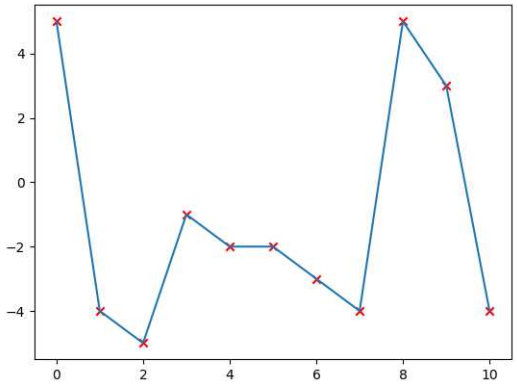
Pour exploiter ces données, il va souvent être utile de « reconstruire » le signal pour tout temps (compris entre t_0 et t_n), pas seulement aux temps t_i. C'est l'objet de l'*interpolation*.

Pour tout le chapitre, on suppose que l'on dispose de deux listes `T` et `Y` contenant respectivement les t_i (triés par ordre croissant) et les y_i correspondant, pour $i \in [\![0\,;n]\!]$.

1 Interpolation affine par morceaux

Une première façon d'interpoler consiste à traiter chaque morceau de la forme $[t_i\,;t_{i+1}]$ indépendamment des autres, d'où le nom d'*interpolation par morceaux*.

L'idée consiste à simplement relier les points (t_i, y_i) et (t_{i+1}, y_{i+1}) par un segment comme illustré ci-contre.



Proposition 1 – Équation de la droite passant par deux points donnés

Soient $A(x_A, y_A)$ et $B(x_B, y_B)$ deux points d'abscisses distinctes.
La droite (AB) a pour équation $y = \dfrac{y_B - y_A}{x_B - x_A}x + \dfrac{x_B y_A - x_A y_B}{x_B - x_A}$.

Démonstration. Comme A et B sont d'abscisses distinctes, la droite (AB) n'est pas verticale, notons donc $y = \alpha x + \beta$ une équation de celle-ci. Comme A et B sont deux points de la droite, on a

$$\begin{cases} y_A = \alpha x_A + \beta \\ y_B = \alpha x_B + \beta \end{cases} \underset{\mathrm{L}_1 \leftarrow x_B\mathrm{L}_1 - x_A\mathrm{L}_2}{\overset{\mathrm{L}_2 \leftarrow \mathrm{L}_2 - \mathrm{L}_1}{\Longleftrightarrow}} \begin{cases} x_B y_A - x_A y_B = \beta(x_B - x_A) \\ y_B - y_A = \alpha(x_B - x_A) \end{cases} \overset{x_A \neq x_B}{\Longleftrightarrow} \begin{cases} \beta = \dfrac{x_B y_A - x_A y_B}{x_B - x_A} \\ \alpha = \dfrac{y_B - y_A}{x_B - x_A} \end{cases}.$$

❑

On peut alors se servir de cette équation pour notre interpolation : on cherche d'abord sur quel intervalle de la forme $]t_i\,;t_{i+1}]$ se trouve notre temps t (supposé compris entre t_0 et t_n), puis on approxime la grandeur g sur cet intervalle par la droite reliant les points (t_i, y_i) et (t_{i+1}, y_{i+1}).

```python
def intervalle(t: float, T: [float]) -> int:
    """
    Renvoie l'indice i pour lequel T[i] < t <= T[i+1].
    On suppose T triée par ordre croissant et T[0] <= t <= T[len(T)-1].
    """
    i = 0
    while t > T[i+1]:
        i = i + 1
    return i

def droite(xA, yA, xB, yB, x):  # cf prop 1
    return (yB - yA) / (xB - xA) * x + (xB*yA - xA*yB) / (xB - xA)

```

```python
def g_affine(T, Y, t):
    """
    Valeur au point d'absisse t de l'approximation affine par morceaux
    des points d'abscisses T et d'ordonnées Y.
    """
    i = intervalle(t, T)
    return droite(T[i], Y[i], T[i+1], Y[i+1], t)
```

Cette interpolation affine par morceaux est facile à mettre en œuvre mais n'est pas vraiment satisfaisante. En effet, le signal reconstitué est certes continu mais pas dérivable, ce qui est rarement le cas dans le monde physique. L'erreur entre la valeur réelle et l'approximation proposée peut ainsi être importante.

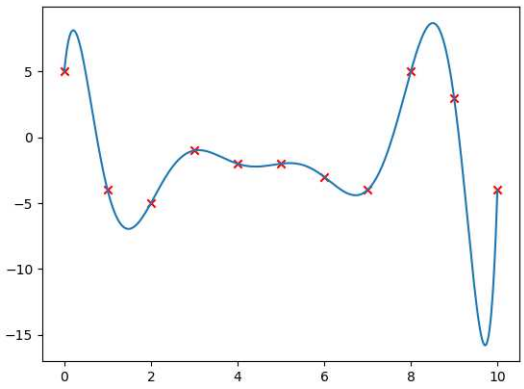
Remarque. Une façon d'améliorer cela est de regrouper les points non plus par deux mais par trois ou quatre et chercher une courbe (une parabole ou une cubique) passant par ces points. C'est la très efficace idée des splines [1].

2 Interpolation de Lagrange

2.1 Principe

Une seconde méthode d'interpolation consiste à chercher un polynôme P de degré minimal passant par les $n+1$ points (t_i, y_i), *i.e.* vérifiant $P(t_i) = y_i$ pour tout $i \in [\![0\,;n]\!]$. La fonction obtenue étant polynomiale, elle sera de classe $\mathcal{C}^\infty$.

Remarque. Comme il y a $n+1$ équations à vérifier, on pourrait se servir de l'algorithme du pivot de Gauss pour résoudre ce système linéaire et déterminer les coefficients d'un polynôme qui convient mais cela peut s'avérer lent si le nombre de points est trop important.



On dispose plus précisément du théorème suivant [2].

Théorème 2 – Interpolation de Lagrange

Soient $n+1$ points $(x_0, y_0), \dots, (x_n, y_n)$ avec les x_i deux à deux distincts.
Il existe un unique polynôme P de degré au plus n vérifiant, pour tout $i \in [\![0\,;n]\!]$, $P(x_i) = y_i$.
Celui-ci est donné par $P(X) = \sum_{i=0}^{n} y_i L_i(X)$ où, pour tout $0 \leqslant i \leqslant n$, $L_i(X) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{X - x_j}{x_i - x_j}$.
Les polynômes L_i sont appelés *polynômes de Lagrange* associés aux abscisses $x_0, \dots, x_n$, et P est appelé *polynôme interpolateur de Lagrange* de ces $n+1$ points.

Démonstration.
Voir le sujet de modélisation 2024 Q4-Q8.
Idées : on montre que :

- Les L_i sont de degré n ;
- Ils vérifient $L_i(x_i) = 1$ et, si $k \neq i$, $L_i(x_k) = 0$;
- On en déduit que $(L_0, \dots, L_n)$ est une base de $\mathbb{R}_n[X]$;
- On vérifie que P a les propriétés recherchées.

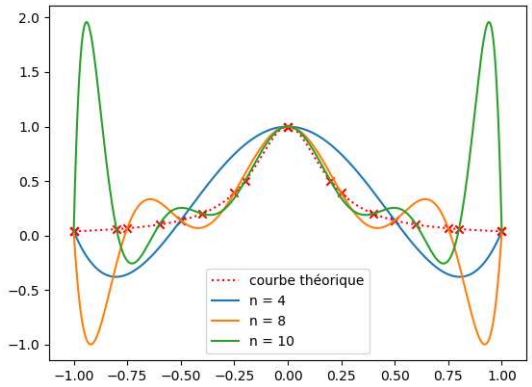
❑

1. Voir `https://fr.wikipedia.org/wiki/Spline`.
2. Cette méthode a été publiée par Lagrange en 1795 mais en fait déjà découverte indépendamment par Waring en 1779 et par Euler en 1783.

2.2 Limite : le phénomène de Runge

Comme on peut le voir sur la figure du paragraphe précédent, la courbe d'interpolation obtenue peut se mettre à varier très fortement entre deux points (par exemple les deux plus à droite), ce qui rend le résultat discutable.

On peut montrer qu'augmenter le nombre de points n'est alors pas une solution. Au contraire, il existe même des cas, voir ci-contre, pour lesquels l'écart entre la courbe théorique et celle obtenue par interpolation de Lagrange augmente indéfiniment avec le nombre de points. Ce comportement est appelé *phénomène de Runge*.



Remarque. Pour remédier à ce problème, il faut jouer sur le choix des abscisses des points d'interpolation. Au lieu de les prendre équirépartis sur l'intervalle d'étude, il faut plutôt choisir les abscisses de Tchebychev, c'est-à-dire les zéros des polynômes[3] définis par $T_n(x) = \cos(n \arccos(x))$.

3 Application culturelle : partage de clé secrète de Shamir

Un professeur d'informatique possède un coffre-fort (contenant je ne sais quel secret !), dont le code est composé de quatre chiffres, disons 2025. Il souhaite partager l'accès à ce coffre à cinq personnes de sorte que :

— chaque personne possède une partie du code ;

— une personne seule ne peut pas ouvrir le coffre.

Une façon naïve de le faire est de donner un chiffre à chaque personne (et la cinquième reçoit par exemple l'ordre des chiffres). Ceci n'étant pas satisfaisant, nous souhaitons que le partage vérifie deux propriétés supplémentaires :

— la perte de sa partie par une personne ne compromet pas l'accès (dans la méthode naïve, la perte d'un chiffre est irrémédiable) ;

— la connaissance d'une partie ne facilite pas la découverte du code[4].

Plus précisément, on souhaiterait par exemple que la connaissance de trois parties sur les cinq du code soit suffisante pour ouvrir le coffre, mais qu'en connaître seulement deux parties n'apporte aucune information sur le code. C'est ce que propose la méthode de *partage de clé secrète de Shamir* qui repose sur le principe d'interpolation de Lagrange.

Décrivons brièvement sa mise en œuvre. Tout d'abord le partage du secret :

① Le secret est 2025.

② On veut que trois parties suffisent à retrouver le code donc on choisit au hasard[5] deux nombres supplémentaires, disons 137 et 104. On considère alors le polynôme $P = 2025 + 137X + 104X^2$.

③ On veut partager le secret entre cinq personnes donc on crée cinq points de la forme $(a, P(a))$: $(1, 2266)$, $(2, 2715)$, $(3, 3372)$, $(4, 4237)$, $(5, 5310)$.

④ Chaque personne reçoit un de ces points.

Et maintenant la reconstitution du secret à partir de ces points :

① Si trois personnes mettent en commun leurs parties du code, d'après le théorème 2, il existe un unique polynôme de degré 2 passant par leurs trois points : c'est P.

② Il suffit alors de calculer $P(0)$ pour obtenir le secret.

On peut remarquer que la connaissance de seulement un ou deux points ne livre aucune information sur le secret car il existe une infinité de polynôme de degré 2 passant par deux points donnés.

3. Voir le DS4 de maths.

4. Avec quatre chiffres au départ, il y a 10 000 possibilités, c'est un peu long à tester. En connaissant deux chiffres via la répartition naïve, il n'en reste plus que 100, c'est relativement rapide à tester.

5. Là se cache une petite subtilité pour la mise en pratique effective.

TD n° 10 : Interpolation de Lagrange

Exercice 1. *Calcul du polynôme de Lagrange*

1. Écrire une fonction d'entête

```
def poly_Lagrange(i: int, T: [float], x: float) -> float:
```

qui renvoie la valeur de $L_i(x)$ du polynôme de Lagrange, défini dans le théorème 2, associé aux abscisses données dans la liste `T`.

2. Écrire une fonction d'entête

```
def interpol_Lagrange(T: [float], Y:[float], x: float) -> float:
```

donnant la valeur de $P(x)$ où P est le polynôme d'interpolation de Lagrange associé aux points dont les abscisses et les ordonnées se trouvent respectivement dans les listes `T` et `Y`.

Exercice 2. *Erreur commise*

On reprend les notations de l'exercice précédent. On suppose que le polynôme d'interpolation de Lagrange nous a servi à estimer la valeur d'un signal g (fonction définie par ailleurs, prenant en paramètre un flottant `x`).

Écrire une fonction d'entête

```
def erreur(T: [float], Y: [float]) -> float:
```

qui renvoie l'erreur maximale commise par cette estimation. Comme on ne peut pas calculer pour tout réel, on choisira 1 000 points équirépartis entre `T[0]` et `T[len(T)-1]`.

Exercice 3. *Méthode de Hörner (d'après CCP 2020)*

On considère un polynôme $P = \sum_{k=0}^{n} a_k X^k \in \mathbb{R}_n[X]$. Celui-ci est représenté en Python par la liste des coefficients `coeff = [` a_0`, ...,` a_n`]`.

1. Écrire une fonction d'entête

```
def eval(coeff: [float], b: float) -> float:
```

où la liste `coeff` est celle des coefficients de P, qui renvoie la valeur de $P(b)$ en la calculant via la formule naturelle $P(b) = \sum_{k=0}^{n} a_k b^k$.

2. En admettant que le calcul de $b^k = b \times \ldots \times b$ utilise $k-1$ multiplications, quelle est la complexité de `eval` en fonction du degré n du polynôme ?

On peut être plus astucieux en utilisant l'algorithme de Hörner qui se base sur l'égalité suivante :

$$P(X) = ((\cdots((a_n X + a_{n-1})\,X + a_{n-2})\,X + \ldots)\,X + a_1)\,X + a_0.$$

Autrement dit, pour évaluer P en b, on commence par calculer $a_n \times b + a_{n-1}$ puis on multiplie le résultat par b et on ajoute a_{n-2}, ensuite on multiplie ce résultat par b et on ajoute a_{n-3}, etc.

3. ★ Écrire une fonction[6] d'entête

```
def horner(coeff: [float], b: float) -> float:
```

où `coeff` est la liste des coefficients de P, qui renvoie la valeur de $P(b)$ en utilisant l'algorithme de Hörner.

4. Quelle est la complexité de `horner` en fonction du degré n du polynôme ?

6. Vous pouvez même en proposer deux : une version itérative et une version récursive.