Interpolation et Approximation

par Léo Peyronnet

Novembre 2022

Compte rendu du TP consistant à programmer et comparer certaines méthodes d'interpolation et d'approximation.

1 Rappel des méthodes

1.1 Méthodes d'interpolations

L'interpolation est une opération mathématique visant à déterminer une fonction passant par des points donnés du plan. Plus précisément, soient $x_1, ..., x_n$ des réels distincts, $y_1, ..., y_n$ des réels, $n \in \mathbb{N}^*$. Alors l'interpolation consiste à déterminer une fonction telle que $\forall i \in [1, n], f(x_i) = y_i$; ce qui correspond à passer par l'ensemble des points d'interpolations (x_i, y_i).

Les méthodes détaillés ci-dessous interpolent des fonctions polynomiales de degré au plus $n - 1$.

1.1.1 Méthode de Lagrange

La méthode de Lagrange se base sur le principe de superposition, c'est à dire que les points d'interpolation vont être traités un par un.

Soit $L_1, ..., L_n \in \mathbb{R}_{(n-1)}[X]$ tels que $\forall a, b \in [1, n]$, $L_a(x_b) = 1$ si $a = b$, 0 sinon, alors le polynôme $P_{(n-1)}$ est exprimé sous la forme :

$$P_{(n-1)}(x) = \sum_{i=0}^{(n-1)} y_i L_i(x)$$

avec $L_i(x)$:

$$L_i(x) = \prod_{j=0, j\neq i}^{(n-1)} \frac{x - x_j}{x_i - x_j}$$

1.1.2 Méthode de Neville

Quant à elle, la méthode de Neville se base sur la décomposition du polynome $P_{(n-1)}[x_1, x_2, ..., x_n]$ en $P_{(n-2)}[x_1, x_2, ..., x_{n-1}]$ et $P_{(n-2)}[x_2, x_3, ..., x_n]$.

Alors, en admettant $P_0[x_i], \forall x, i = 1, ..., n; P_0[x_i](x) = y_i$:

$$\forall x,\ P_k[x_i, \ldots, x_{i+k}](x) = \frac{(x - x_{i+k})P_{k-1}[x_i, \ldots, x_{i+k-1}](x) + (x_i - x)P_{k-1}[x_i + 1, \ldots, x_{i+k}](x)}{x_i - x_{i+k}},$$

$$\forall k = 1, \ldots, n - 1$$

2 Présentation des programmes

2.1 lagrange()

```c
float lagrange(float * X, float * Y, float xentree, int taille){
    float result=0;
    for (int i=0;i<taille;i++){
        float Li=1;
        for (int j=0;j<taille;j++){
            if (j!=i){
                Li*=(xentree-X[j])/(X[i]-X[j]);
            }
        }
        result+=Y[i]*Li;
    }
    return result;
}
```

La boucle j correspond à l'opérateur produit, la boucle i correspond à l'opérateur somme.

2.2 neville()

```c
double neville(float * X, float * Y, float xentree, int n){
    double * Pk=(double *)malloc(n * sizeof(double));
    if (Pk==NULL){return 0;}
    for (int j=0;j<n;j++){
        Pk[j]=Y[j];
    }

    for(int k=1;k<n;k++){
        for(int i=0;i<n;i++){
            Pk[i]=((xentree-X[i+k])*Pk[i]+(X[i]-xentree)*Pk[i+1])/(X[i]-X[i+k]);
        }
    }
    return Pk[0];
}
```

3 Observations sur les jeux d'essais

4 Conclusion