Interpolation polynomiale : algorithme de Neville

1. Introduction

On considère une fonction $f(x)$ dont on connaît les valeurs aux N points

$$x_0, x_1, ...x_{N-1}$$

On pose

$$y_i = f(x_i) \tag{1}$$

Pour calculer $f(x)$ pour x quelconque, on peut effectuer une interpolation polynomiale, qui consiste à calculer $f(x)$ en utilisant l'unique polynôme de degré $N-1$ qui passe par les points

$$(x_0, y_0), (x_1, y_1), ...(x_{N-1}, y_{N-1}) \tag{2}$$

On présente ici l'algorithme de Neville, qui permet de faire efficacement une interpolation polynomiale lorsque le nombre de valeurs de x est faible.

2. Algorithme de Neville

L'algorithme de Neville ([1]) construit les polynômes par récurrence. On commence par les polynômes de degré 0. Le polynôme de degré 0 qui passe par le point (x_i, y_i) est :

$$P_i(x) = y_i \tag{3}$$

Partant de ces N polynômes de degré 0, on construit les polynômes de degré 1. Le polynôme de degré 1 passant par les points $(x_i, y_i), (x_{i+1}, y_{i+1})$ est :

$$P_{i,i+1}(x) = \frac{(x - x_{i+1})P_i(x) + (x_i - x)P_{i+1}(x)}{x_i - x_{i+1}} \tag{4}$$

On a en effet :

$$P_{i,i+1}(x_i) = P_i(x_i) = y_i \tag{5}$$
$$P_{i,i+1}(x_{i+1}) = P_{i+1}(x_{i+1}) = y_{i+1} \tag{6}$$

La relation de récurrence permettant de passer de deux polynômes de degré $m-1$ à un polynôme de degré m est :

$$P_{i,i+1,...i+m}(x) = \frac{(x - x_{i+m})P_{i,i+1,...,i+m-1}(x) + (x_i - x)P_{i+1,i+2,...i+m}(x)}{x_i - x_{i+m}} \tag{7}$$

Pour le démontrer, on raisonne par récurrence, en supposant que les deux polynômes de degré m-1 apparaissant dans la relation ci-dessus passent par les m points figurant en indice. On vérifie alors que le polynôme de degré m passe par les m+1 points d'indices i,i+1,...i+m. Pour le premier point, on a en effet :

$$P_{i,i+1,\dots i+m}(x_i) = \frac{(x_i - x_{i+m})P_{i,i+1,\dots,i+m-1}(x_i)}{x_i - x_{i+m}} = y_i \tag{8}$$

La vérification est similaire pour le point x_{i+m}. Pour un point x_{i+k} (commun aux deux polynômes de degré m-1), on a :

$$P_{i,i+1,\dots i+m}(x_{i+k}) = \frac{(x_{i+k} - x_{i+m})y_{i+k} + (x_i - x_{i+k})y_{i+k}}{x_i - x_{i+m}} = y_{i+k} \tag{9}$$

Le tableau suivant montre l'enchaînement des calculs jusqu'au polynôme de degré $N-1$, dans le cas $N = 4$:

$$\begin{array}{lllll} P_0 & & & & \\ & P_{01} & & & \\ P_1 & & P_{012} & & \\ & P_{12} & & P_{0123} & \\ P_2 & & P_{123} & & \\ & P_{23} & & & \\ P_3 & & & & \end{array} \tag{10}$$

3. Implémentation récursive

L'algorithme de Neville est aisément programmé de manière récursive. Pour cela, on définit une fonction $r(i, m, x)$ qui calcule la valeur du polynôme de degré m en fonction des valeurs des deux polynômes de degré $m-1$

Si $m = 0$, la fonction renvoie y_i. Sinon, elle renvoie :

$$\frac{(x - x_{i+m})r(i, m-1, x) + (x_i - x)r(i+1, m-1, x)}{x_i - x_{i+m}} \tag{11}$$

On définit une classe dont le constructeur prend en argument les valeurs de x et y sous forme de listes.

neville.py

```
class Neville:
    def __init__(self,x,y):
        self.x = x
        self.y = y
        self.N = len(x)
```

La fonction suivante définit la récurrence, en s'appelant récursivement deux fois :

```
    def recurrence(self,i,m,x):
        if m==0:
            return self.y[i]
        else:
            return ((x-self.x[i+m])*self.recurrence(i,m-1,x)+\
            (self.x[i]-x)*self.recurrence(i+1,m-1,x))/(self.x[i]-self.x[i+m])
```

Voici la fonction qui effectue l'interpolation :

```
def interpoler(self,x):
    return self.recurrence(0,self.N-1,x)
```

Pour tester l'algorithme, la fonction suivante effectue l'évaluation d'un polynôme dont les coefficients sont donnés dans une liste `c` (rangés par degré croissant) :

```
def eval_polynome(self,c,x):
    p = len(c)
    i = p-1
    y = c[i]
    while i>0:
        i -= 1
        y = y*x+c[i]
    return y
```

La fonction suivante calcule les valeurs des listes x et y à partir d'un polynôme donné, sur un intervalle donné :

```
def polynome(self,c,xmin,xmax):
    p = len(c)
    dx = (xmax-xmin)/p
    for k in range(p):
        self.x[k] = xmin+k*dx
        self.y[k] = self.eval_polynome(c,self.x[k])
```

Faisons un test avec un polynôme de degré 4 :

```
import neville
x=[0,0,0,0]
y=[0,0,0,0]
nev = neville.Neville(x,y)
c = [1,2,-3,2]
nev.polynome(c,0.0,10.0)
x0 = 3.564
y0 = nev.interpoler(x0)


print(y0)
--> 60.562252287999996

print(y0-nev.eval_polynome(c,x0))
--> -1.4210854715202004e-14
```

Références

[1] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical recipes, the art of scientific computing*, (Cambridge University Press, 2007)