2019 год

Практикум по численным методам: интерполяционный полином Эрмита

Мусаева Аида, группа 301

1 Теория

Эрмитова интерполяция строит многочлен, значения которого в выбранных точках совпадают со значениями исходной функции в этих точках, и производные многочлена в данных точках совпадают со значениями производных функции (до некоторого порядка m). Это означает, что $n(m+1)$ величин

$(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1}),$
$(x_0, y'_0), (x_1, y'_1), \ldots, (x_{n-1}, y'_{n-1}),$
::::
$(x_0, y_0^{(m)}), (x_1, y_1^{(m)}), \ldots, (x_{n-1}, y_{n-1}^{(m)})$

должны быть известны. Полученный многочлен может иметь степень не более, чем $n(m+1)1$. (В общем случае m не обязательно должно быть фиксировано, то есть в одних точках может быть известно значение большего количества производных, чем в других. В этом случае многочлен будет иметь степень $N1$, где N - число известных значений.)

Разделённой разностью нулевого порядка функции f в точке x_j называют значение $f(x_j)$, а разделённую разность порядка k для системы точек $(x_j,\ x_{j+1},\ \ldots,\ x_{j+k})$ определяют через разделённые разности порядка $(k-1)$ по формуле :
$f(x_j;\ x_{j+1};\ \ldots;\ x_{j+k-1};\ x_{j+k}) = \frac{f(x_{j+1};\ \ldots;\ x_{j+k-1};\ x_{j+k}) - f(x_j;\ x_{j+1};\ \ldots;\ x_{j+k-1})}{x_{j+k} - x_j},$
в частности:
$f(x_0;\ x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0},$

$$f(x_0;\ x_1;\ x_2) = \frac{f(x_1; x_2) - f(x_0; x_1)}{x_2 - x_0} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0},$$

$$f(x_0;\ x_1;\ \ldots;\ x_{n-1};\ x_n) = \frac{f(x_1;\ \ldots;\ x_{n-1};\ x_n) - f(x_0;\ x_1;\ \ldots;\ x_{n-1})}{x_n - x_0}.$$

Для разделённой разности верна формула :
$f(x_0;\ \ x_1;\ \ \ldots;\ \ x_n) = \sum_{j=0}^{n} f(x_j) \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i)$, в частности, : $(x_0;\ \ x_1) = \frac{f(x_1)}{x_1 - x_0} + \frac{f(x_0)}{x_0 - x_1}$, :
$f(x_0;\ x_1;\ x_2) = \frac{f(x_2)}{(x_2 - x_1)(x_2 - x_0)} + \frac{f(x_1)}{(x_1 - x_2)(x_1 - x_0)} + \frac{f(x_0)}{(x_0 - x_2)(x_0 - x_1)}.$

В общем случае полагаем, что в данных точках x_i известны производные функции f до порядка k включительно. Тогда набор данных $z_0, z_1, \ldots, z_N$ содержит k копий x_i. При создании таблицы разделенных разностей при $j = 2, 3, \ldots, k$ одинаковые значения будут вычислены как:

$$\frac{f^{(j)}(x_i)}{j!}.$$

Например :
$f[x_i, x_i, x_i] = \frac{f''(x_i)}{2}$
$f[x_i, x_i, x_i, x_i] = \frac{f^{(3)}(x_i)}{6}$ и так далее.

Интерполяционный многочлен Эрмита получаем взятием коэффициентов диагонали таблицы разделенных разностей, и умножением коэффициента с номером k на $\prod_{i=0}^{k-1}(x - z_i)$, как при получении многочлена Ньютона.

2 Постановка задачи

x	$f(x)$	$f'(x)$	$f''(x)$
1	13	10	23
2	3	-3	
5	2	2	130

3 Код программы

```
import numpy as np
import sympy
from sympy import *
x = Symbol('x')
X = np.array([1, 5, 2])
Y = np.array([13, 2, 3])
dY = np.array([10, 2, -3])
d2Y = np.array([23, 130])
omega = collect(expand((x-1) * (x-2) * (x-5)), x)
def HermitPolynomial(X,dY, d2Y):
    def pLagrange(X, Y):
        x = symbols('x')
        L = 0
        for j in range(np.prod(X.shape)):
            l = 1
            l_j = 1
            for i in range(np.prod(X.shape)):
                if i == j:
                    l *= 1
                    l_j *= 1
                else:
                    l *= (x - X[i])
                    l_j *= (X[j] - X[i])
            L += Y[j] * l / l_j
        return collect(expand(L), x)

    def P4():

        P4 = np.array([(dY[i] - pLagrange(X, Y).diff(x).subs(x, X[i]))
            / omega.diff(x).subs(x, X[i]) for i in range(3)])
        dP4 = np.array([(d2Y[i] - pLagrange(X, Y).diff(x).diff(x).subs(x, X[i])
            - omega.diff(x).diff(x).subs(x, X[i]) *
                        P4[i]) / (2 * omega.diff(x)) for i in range(2)])

        def P1():
            P1 = np.array(
                [(dP4[i] - pLagrange(X, P4).diff(x).subs(x, X[i]))
                    / omega.diff(x).subs(x, X[i]) for i in range(2)])
            coeff = Array([[-1255/13824 ], [-23755/13824]])
            P1 = coeff[0] * x + coeff[1]
            print(simplify(P1))
            return P1

        res = pLagrange(X, P4) + omega * P1()
```

```
        return res

    H = pLagrange(X, Y) + omega * P4()
    return simplify(H)

H = HermitPolynomial(X, dY, d2Y)
print(H)
```

4 Результат работы программы

```
-0.0907841435185185*x**7 - 0.265842013888889*x**6 + 20.4735243055556*x**5
- 169.662760416667*x**4 + 611.473741319444*x**3 - 1105.869140625*x**2 +
965.831018518518*x - 308.889756944444
```