hw4-task2

1. 习题4.8-4.14

2. 习题4.16,4.19-4.22

```python
dec = 6 # 设置每一步计算保留小数点后位数（精度，可以自己调整）
import numpy as np
np.set_printoptions(formatter={'float': ('{: 0.' + str(dec) + 'f}').format})
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['text.usetex'] = True
import sympy as sp
```

1 习题4.8

对于一般的 `Lagrange` 插值而言有如下关系：

$$f(x) = L(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega(x)$$

当 $f(x)$ 是小于等于 n 次的多项式函数时，有 $f^{(n+1)}(\xi) = 0$ ，因此有 $f(x) = L(x)$ 。

2 `class LagrangeInterpolation` 编写

```python
class LagrangeInterpolation:

    def __init__(self, x_in: np.ndarray, y_in: np.ndarray) -> None:
        self.x_in = x_in
        self.y_in = y_in
        self.N = len(self.x_in)
        self.getInterpolationCoeff()
        pass

    def getInterpolationCoeff(self) -> None:
        X = np.zeros([self.N, self.N])
        for i in range(self.N):
            X[i] = self.x_in ** i
            pass
        self.coeff = np.linalg.inv(X.T) @ self.y_in
        pass

    def value(self, x: float) -> float:
        x_ = x ** np.arange(0, self.N, 1)
        return self.coeff @ x_
        pass

    def values(self, x: np.ndarray) -> np.ndarray:
        n = len(x)
        y = np.zeros(n)
        for i in range(n):
            y[i] = self.value(x[i])
            pass
        return y
        pass

    def expr(self) -> str:
        s = ""
```

```python
        for i in range(self.N):
            if i == 0:
                s += str(self.coeff[i]) + " + "
                pass
            else:
                s += str(self.coeff[i]) + "x^" + str(i) + " + "
                pass
            pass
        return s[:-2]
        pass

    pass
```

3 习题4.9

3.1 线性插值

```python
x = np.array([45, 60]) * np.pi / 180
y = np.sin(x)
i = LagrangeInterpolation(x, y)
i.expr()
```

```
'0.23035091339287384 + 0.6070244240594347x^1 '
```

```python
print("插值值为: ", i.value(50 * np.pi / 180))
print("精确值为: ", np.sin(50 * np.pi / 180))
print("error = ", np.abs(i.value(50 * np.pi / 180) - np.sin(50*np.pi/180)))
```

```
插值值为:  0.7600796553858448
精确值为:  0.766044443118978
error =  0.005964787733133248
```

3.2 二次插值

```python
x = np.array([30, 45, 60]) * np.pi / 180
y = np.sin(x)
i = LagrangeInterpolation(x, y)
i.expr()
```

```
'-0.05877803813906546 + 1.251252649641426x^1 + -0.35153865113380967x^2 '
```

```python
print("插值值为: ", i.value(50 * np.pi / 180))
print("精确值为: ", np.sin(50 * np.pi / 180))
print("error = ", np.abs(i.value(50 * np.pi / 180) - np.sin(50*np.pi/180)))
```

```
插值值为:  0.7654338952290286
精确值为:  0.766044443118978
error =  0.0006105478899494088
```

4 习题4.10

5 `function AitkenInterpolation` 编写

```
def AitkenInterpolation(x: float, x_in: np.ndarray, y_in: np.ndarray) ->float:
    for i in range(len(x_in) - 1):
        for j in range(len(x_in) - i - 1):
            y_in[j] = (y_in[j+1] - y_in[j]) / (x_in[j+i+1] - x_in[j]) * (x - x_in[j]) + y_in[j]
            pass
        pass
    return y_in[0]
    pass
```

```
x = np.arange(-2, 3, 1)
y = np.power(3, x+2) / np.power(3, 2)
y
```

```
array([ 0.111111,  0.333333,  1.000000,  3.000000,  9.000000])
```

```
print("0.5处插值值为：", AitkenInterpolation(0.5, x, y))
```

```
0.5处插值值为： 1.7083333333333333
```

6 习题4.11

```
x = np.array([
    0.46, 0.47, 0.48, 0.49
])
y = np.array([
    0.4846555, 0.4937452, 0.5027498, 0.5116683
])
i = LagrangeInterpolation(x[0:-1], y[0:-1])
i.expr()
```

```
'-0.02546380000035242 + 1.3046850000009727x^1 + -0.42549999999937427x^2 '
```

```
print("x = 0.472时，插值值为：", i.value(0.472))
```

```
x = 0.472时，插值值为： 0.495552928000246
```

```
i = LagrangeInterpolation(y[1:], x[1:])
i.expr()
```

```
'0.07016152590608726 + 0.514457522638622 3x^1 + 0.5981826073884804x^2 '
```

```
print("f(x)=0.5 时值为（反插值）：", i.value(0.5))
```

```
f(x)=0.5 时值为（反插值）： 0.4769359390725185
```

7 习题4.12

$$f[x_0, x_0] = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} = f'(x_0)$$

8 习题4.13

这里比较好的证明方法是数学归纳法，因为比较显然，就不再说明了。

9 习题4.14

同样比较显然，这里就不在多说明。

10 习题4.16

```
def DifferenceQuotient(x_in: np.ndarray, y_in: np.ndarray) -> float:
    x_ = x_in.copy()
    y_ = y_in.copy()
    for i in range(len(x_in) - 1):
        for j in range(len(x_in) - i - 1):
            y_[j] = (y_[j+1] - y_[j]) / (x_[j+i+1] - x_[j])
            pass
        pass
    return y_[0]
    pass
```

```
f = lambda x: x**5 - 3*x**3 + x - 1
for i in range(1, 7):
    x = 3 ** np.arange(0, i+1, 1)
    y = f(x)
    print("f[3^0,... 3^"+str(i)+"] = ", DifferenceQuotient(x, y))
    pass
```

```
f[3^0,... 3^1] =  83
f[3^0,... 3^2] =  1171
f[3^0,... 3^3] =  1207
f[3^0,... 3^4] =  121
f[3^0,... 3^5] =  1
f[3^0,... 3^6] =  0
```

11 习题4.18

$$f[x_0, x_1, \cdots, x_p] = \begin{cases} 0 & p < n+1 \\ 1 & p = n+1 \end{cases}$$

12 习题4.19

```
class NewtonInterpolation:

    def __init__(self, x_in: np.ndarray, y_in: np.ndarray) -> None:
        self.x_in = x_in
        self.y_in = y_in
        self.N = len(self.x_in)
```

```python
        self.getNewtonInterpolationCoeff()
        pass

    def differenceQuotient(self, x: np.ndarray, y: np.ndarray) -> float:
        N = len(x)
        sum = 0
        for k in range(N):
            prod = 1
            for i in range(N):
                if i != k:
                    prod *= (x[k] - x[i])
                    pass
                pass
            sum += y[k] / prod
            pass
        return sum
        pass

    def getNewtonInterpolationCoeff(self) -> None:
        self.coeff = np.zeros(self.N)
        for i in range(0, self.N):
            self.coeff[i] = self.differenceQuotient(self.x_in[0:i+1], self.y_in[0:i+1])
            pass
        pass

    def value(self, x: float) -> float:
        sum = 0
        for i in range(self.N):
            prod = 1
            for j in range(i):
                prod *= (x - self.x_in[j])
                pass
            sum += self.coeff[i] * prod
            pass
        return sum
        pass

    def expr(self, n = 100000) -> str:
        s = ""
        s += str(self.coeff[0]) + " + "
        for i in range(1, min(n, self.N)):
            prod = ""
            for j in range(i):
                prod += "(x - " + str(self.x_in[j]) + ")"
                pass
            s += str(self.coeff[i]) + prod + " + "
            pass
        return s[:-2]
        pass

    pass
```

```python
x = np.array([
    0, 0.2, 0.3, 0.5
])
y = np.sinh(x)
i = NewtonInterpolation(x[:-1], y[:-1])
print("x = 0.23时，二次插值值为：", i.value(0.23), " error = ", np.abs(i.value(0.23) - np.sinh(0.23)))
```

```
x = 0.23时，二次插值值为： 0.2321151495384235  error =  8.194582535159256e-05
```

```python
i = NewtonInterpolation(x, y)
print("x = 0.23时，三次插值值为：", i.value(0.23), " error = ", np.abs(i.value(0.23) - np.sinh(0.23)))
```

```
x = 0.23时，三次插值值为： 0.23203185053791106  error =  1.3531751608397702e-06
```

13 习题4.20

```python
x = np.array([
    -1, 0, 2, 3
])
y = np.array([
    -4, -1, 0, 3
])
i = NewtonInterpolation(x, y)
print("x = 1.5时，y插值值为：", i.value(1.5))
```

```
x = 1.5时，y插值值为： -0.40624999999999944
```

```python
i = NewtonInterpolation(y, x)
print("y = 0.5时，x插值值为：", i.value(0.5))
```

```
y = 0.5时，x插值值为： 2.9107142857142856
```

14 习题4.21

$$l_0(x_0) = 1 \quad l_0(x_i) = 0(i \neq 0)$$

运用牛顿插值，可以直接导出：

$$l_0[x_0, x_1, \cdots, x_n] = \sum_{k=0}^{n} \frac{l_0(x_k)}{\prod_{i=0,i\neq k}^{n}(x_k - x_i)} = \frac{1}{\prod_{i=1}^{n}(x_0 - x_i)}$$

因此显然有：

$$l_0(x) = \sum_{j=0}^{n-1} \prod_{k=0}^{j} \frac{x - x_k}{x_0 - x_{k+1}}$$

证毕。

15 习题4.22

```
x = np.arange(0, 0.6, 0.1)
y = np.array([
    1, 1.32, 1.68, 2.08, 2.52, 3
])
iN = NewtonInterpolation(x, y)
iN.expr(3)
```

```
'1.0 + 3.1999999999999993(x - 0.0) + 2.000000000000014(x - 0.0)(x - 0.1) '
```

```
iL = LagrangeInterpolation(x, y)
iL.expr()
```

```
'1.0 + 3.000000000000199x^1 + 1.9999999999995453x^2 + -1.8189894035458565e-12x^3 +
0.0x^4 + 0.0x^5 '
```