插值法

实验内容

在插值区间 [a, b] 上，对标准函数 f(x) = c * sin(d * x) + e * cos(f * x)，采集 n 个采样点，分别使用范德蒙德多项式插值、拉格朗日插值、牛顿插值、分段线性插值、分段三次 Hermite 插值方法进行插值，并完成各方法之间的对比。

编程实现

范德蒙德矩阵插值

```
    def interp(self):
        # 生成矩阵
        samples = self.samples
        n = len(samples)
        matrix = []
        y = np.transpose(np.array([samples[i].y for i in range(n)]))
        for i, sample in enumerate(samples):
            if i == 0:
                matrix = np.array([[pow(sample.x, i) for i in range(n)]])
            else:
                matrix = np.append(matrix, [[pow(sample.x, i) for i in
range(n)]], axis=0)
        # 计算范德蒙矩阵矩阵的结果
        res = list(reversed(np.linalg.solve(matrix, y)))
        # 获取对应的多项式函数
        fn = np.poly1d(res)
        self.fn = fn
        return fn
```

拉格朗日插值

```
    def interp(self):
        # 求拉格朗日每项的基函数
        n = len(self.samples)
        fn = np.poly1d([0])
        for i in range(0, n):
            item_fn = np.poly1d([1])
            div_num = 1
            for j in range(0, n):
                if i != j:
                    item_fn = item_fn * np.poly1d([1, -self.samples[j].x])
            for k in range(0, n):
                if i != k:
                    div_num = div_num * (self.samples[i].x - self.samples[k].x)
            item_fn = (item_fn / div_num) * self.samples[i].y
            fn = fn + item_fn
        self.fn = fn
        return fn
```

牛顿插值法

```
# 计算均差
def _div_diff(self, x1: float, x2: float, f1: float, f2: float):
    return (f2 - f1) / (x2 - x1)

# 计算均差表
def _div_table(self):
    n = len(self.samples)
    # 获得0阶均差，即所有采样点的函数值
    self.table.append([self.samples[i].y for i in range(n)])
    # 迭代 n - 1 次，获得所有均差值
    for i in range(1, n):
        # k 阶差商表
        k_table = [0 for _ in range(0, i)]
        for j in range(i, n):
            # 获得上一级的均差
            y1 = self.table[i - 1][j - 1]
            y2 = self.table[i - 1][j]
            # 获得对应的 x
            x1 = self.samples[j - i].x;
            x2 = self.samples[j].x;
            # 计算均差
            f = self._div_diff(x1, x2, y1, y2)
            k_table.append(f)
        self.table.append(k_table)
    # print(self.table)

# 牛顿插值法计算
def interp(self):
    # 获取均差表
    self._div_table()
    # 构造多项式参数
    n = len(self.samples)
    fn = np.poly1d([0])
    for i in range(0, n):
        # 对每一项都构造多项式最后迭代相加
        item = np.poly1d([self.table[i][i]])
        for j in range(0, i + 1):
            if j == 0:
                item = item * [1]
            elif j > 0:
                item = item * [1, -self.samples[j - 1].x]
        fn = fn + item
    self.fn = fn
    return fn
```

线性分段插值法

```
def interp(self):
    n = len(self.samples)
    for i in range(0, n-1):
        # 一阶拉格朗日插值算表达式
        x1 = self.samples[i].x
        x2 = self.samples[i + 1].x
        f1 = self.samples[i].y
```

```
            f2 = self.samples[i + 1].y
            fn = (f1 / (x1 - x2)) * np.poly1d([1, -x2]) + (f2 / (x2 - x1)) *
np.poly1d([1, -x1])
            self.poly.append(PolyFn(fn, x1, x2))

    def cal(self, x: float):
        for item in self.poly:
            if x >= item.a and x <= item.b:
                y = item.fn(x)
                return y
        print("[Debug] 要计算的值不在给定区间中")
        return 0

    def vector_cal(self, x):
        y = []
        for item_x in x:
            item_y = self.cal(item_x)
            y.append(item_y)
        return y
```

分段三次埃米尔特插值法

```
    def interp(self):
        # 根据公式计算区间内的三次插值多项式
        n = len(self.samples)
        for i in range(0, n-1):
            item = np.poly1d([0])
            x0 = self.samples[i].x
            x1 = self.samples[i + 1].x
            y0 = self.samples[i].y
            y1 = self.samples[i + 1].y
            d0 = self.samples[i].d
            d1 = self.samples[i + 1].d
            item += y0 * (1 + (2 / (x1 - x0)) * np.poly1d([1, -x0])) *
(np.poly1d([1, -x1]) / (x0 - x1)) * (np.poly1d([1, -x1]) / (x0 - x1))
            item += y1 * (1 + (2 / (x0 - x1)) * np.poly1d([1, -x1])) *
(np.poly1d([1, -x0]) / (x1 - x0)) * (np.poly1d([1, -x0]) / (x1 - x0))
            item += d0 * (np.poly1d([1, -x0])) * (np.poly1d([1, -x1]) / (x0 -
x1)) * (np.poly1d([1, -x1]) / (x0 - x1))
            item += d1 * (np.poly1d([1, -x1])) * (np.poly1d([1, -x0]) / (x1 -
x0)) * (np.poly1d([1, -x0]) / (x1 - x0))
            self.poly.append(PolyFn(item, x0, x1))
```

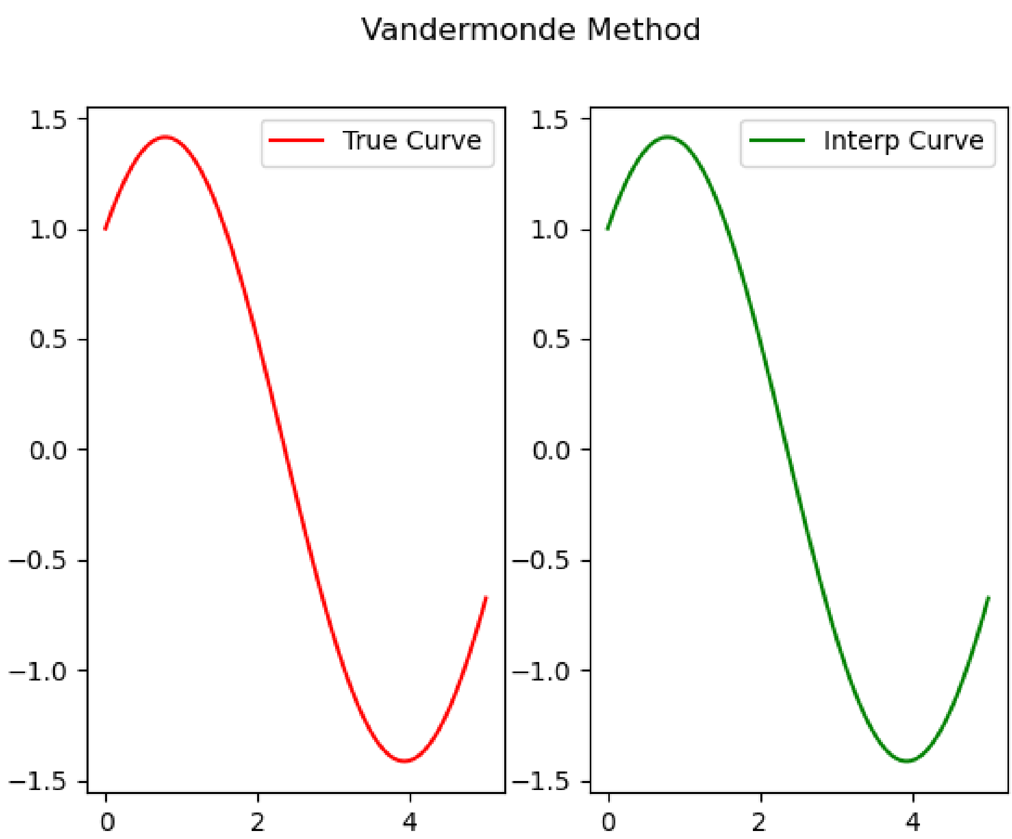
实验结果

我们在插值之后在区间内随机生成5个点进行演算，结果如下:

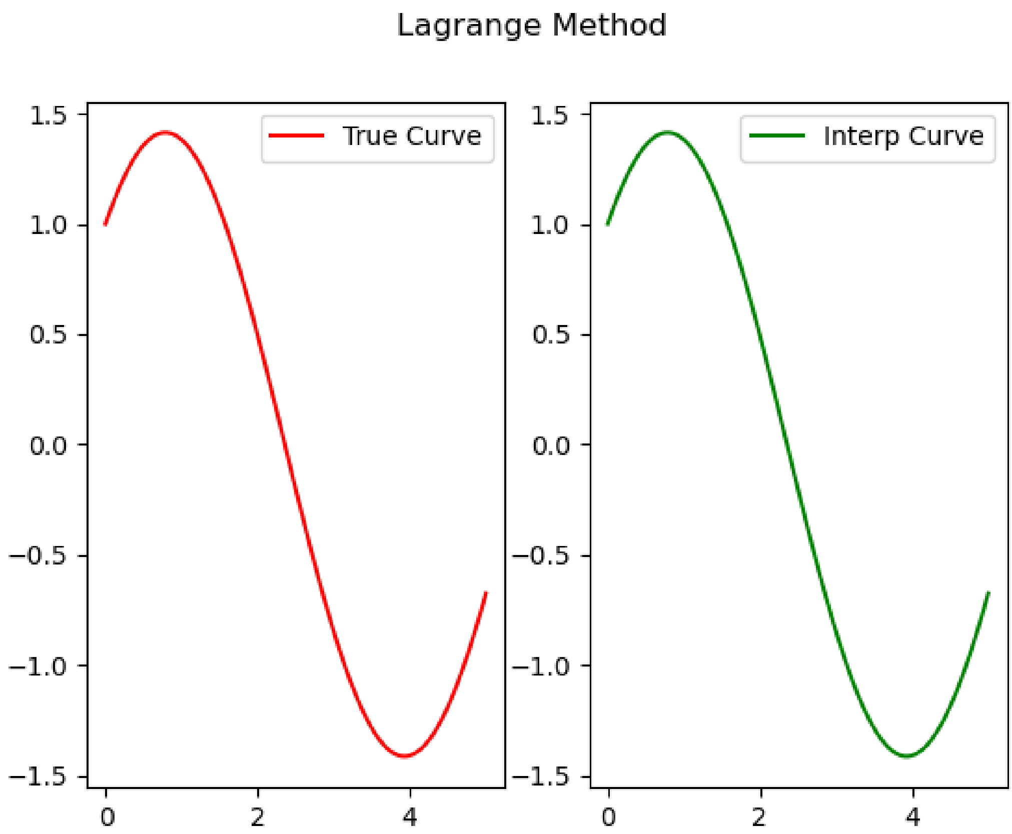
```
PS F:\KuangjuX\作业\数值计算\Numerical-Analysis> python .\example.py
范德蒙德插值法
插值法计算的结果为: -1.3584366492355384, 原函数计算的结果为: -1.3584366521472226, 误差为: 2.911684227058231e-09
插值法计算的结果为: 1.1825637267631854, 原函数计算的结果为: 1.1825637396363868, 误差为: 1.287320139375936e-08
插值法计算的结果为: 1.2584162462819908, 原函数计算的结果为: 1.2584564240806269, 误差为: 4.017779863607629e-05
插值法计算的结果为: 1.0588038612431065, 原函数计算的结果为: 1.058910914183432, 误差为: 0.00010705294032553603
插值法计算的结果为: -1.3219010069236368, 原函数计算的结果为: -1.3219020468718785, 误差为: 1.0399482417433603e-06
拉格朗日插值法
插值法计算的结果为: -0.39308358234148977, 原函数计算的结果为: -0.3930837401236802, 误差为: 1.5778219042417163e-07
插值法计算的结果为: 1.1330627273516636, 原函数计算的结果为: 1.1330627289873256, 误差为: 1.635662050247788e-09
插值法计算的结果为: -0.8953752164557434, 原函数计算的结果为: -0.8953755532416646, 误差为: 3.3678592126218376e-07
插值法计算的结果为: -1.2092235225454884, 原函数计算的结果为: -1.2092524615569846, 误差为: 2.8939011496253997e-05
插值法计算的结果为: 1.074088895295506, 原函数计算的结果为: 1.074088897610537, 误差为: 2.3150308336994385e-09
牛顿插值法
插值法计算的结果为: 0.25004255967062294, 原函数计算的结果为: 0.2500423931429431, 误差为: 1.6652767986791162e-07
插值法计算的结果为: -0.34118242515928077, 原函数计算的结果为: -0.34118226356358117, 误差为: 1.6159569959928177e-07
插值法计算的结果为: -0.8894143197958497, 原函数计算的结果为: -0.8894227570465145, 误差为: 8.437250664794504e-06
插值法计算的结果为: -0.25524131967320596, 原函数计算的结果为: -0.25524115381827805, 误差为: 1.6585492790266443e-07
插值法计算的结果为: -1.230605340671628, 原函数计算的结果为: -1.2306085130722832, 误差为: 3.172400655282459e-06
分段线性插值法
插值法计算的结果为: -1.0819274749200787, 原函数计算的结果为: -1.0819304935650091, 误差为: 3.0186449304636653e-06
插值法计算的结果为: 0.9064279636695469, 原函数计算的结果为: 0.9064289262728152, 误差为: 9.626032683174301e-07
插值法计算的结果为: 1.0572879772045543, 原函数计算的结果为: 1.0572909639292707, 误差为: 2.986724716436129e-06
插值法计算的结果为: -1.3035043369958808, 原函数计算的结果为: -1.3035074958998023, 误差为: 3.158903921551825e-06
插值法计算的结果为: 0.3991798634774466, 原函数计算的结果为: 0.39917990526238745, 误差为: 4.1784940840727813e-08
分段三次 Hermite 插值法
插值法计算的结果为: 1.389400386975467, 原函数计算的结果为: 1.3255186718621292, 误差为: 0.06388171511333773
插值法计算的结果为: 1.4084549556137063, 原函数计算的结果为: 1.3320664845097658, 误差为: 0.07638847110394043
插值法计算的结果为: 1.1392707028862787, 原函数计算的结果为: 1.096005942846971, 误差为: 0.04326476003930768
插值法计算的结果为: -0.8185434332117438, 原函数计算的结果为: -0.7315364002751107, 误差为: 0.08700703293663314
插值法计算的结果为: -0.9302627022843808, 原函数计算的结果为: -0.5543343429435403, 误差为: 0.37592835934084046
```

可以看到我们的插值函数与标准函数的误差较小，能够较好地模拟出标准函数的特性。 除此之外我们还生成了插值函数与标准函数的对比图像如下:

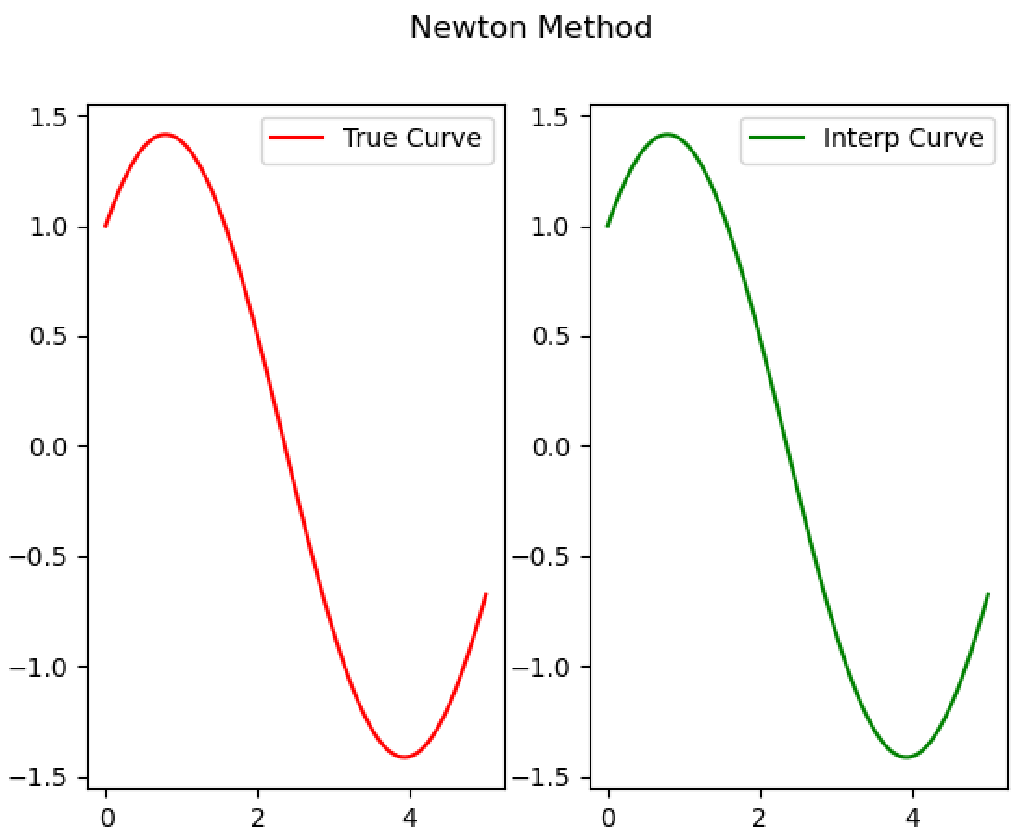
- 范德蒙德插值法



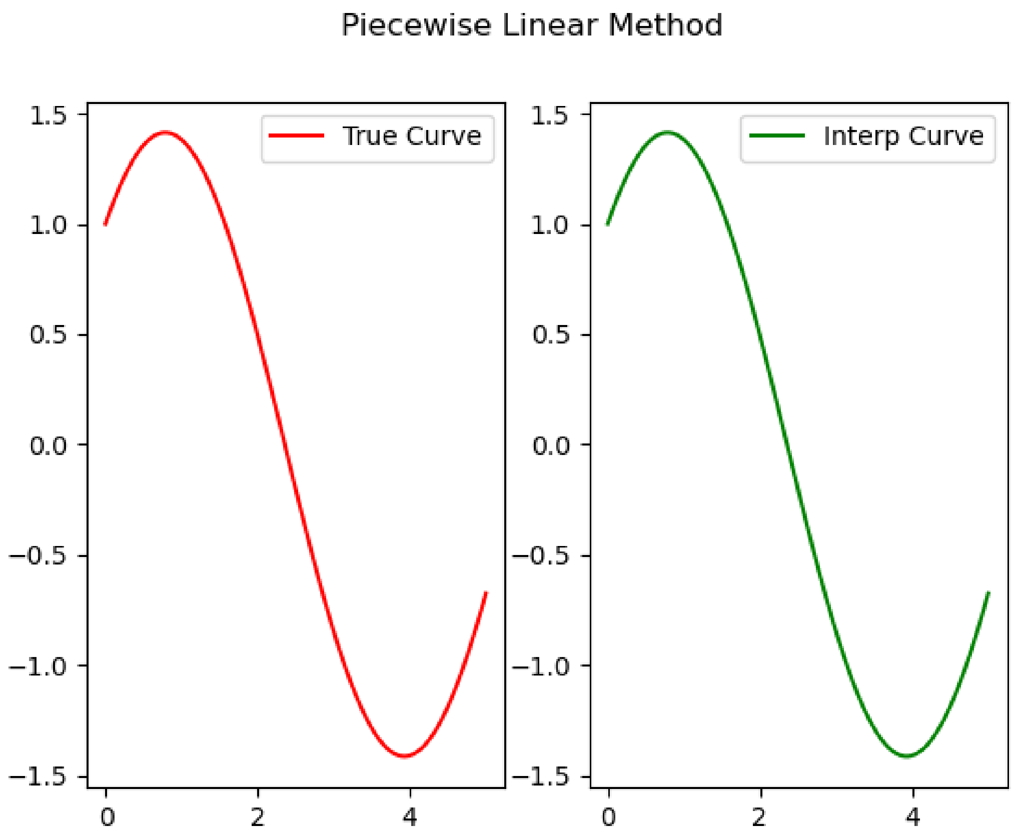
- 拉格朗日插值法



- 牛顿插值法



- 分段线性插值法



- 分段三次 Hermite 插值法