Computional Homework5 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Introduction to Least-Squares Fitting

For more background on the least-squares method, see the notes

From_Data_Fitting_to_Integral_Transforms.pdf.

Given a polynomial basis $\{1, x, x^2, \dots, x^n\}$ and a set of data points $\{(x_i, y_i)\}_{i=0}^{m}$, we can use an orthogonal basis to fit the data. In particular, for the polynomial basis, we can construct orthogonal polynomials using the Gram-Schmidt process.

The recursive formula for generating the $(k+1)$-th orthogonal polynomial from the previously constructed orthogonal polynomials is:

$$S_{k+1}(x) = xS_k(x) - \sum_{j=0}^{k} b_{k,j} S_j(x)$$

Here, the first term increases the degree by one, and the sum uses the Gram-Schmidt process to maintain orthogonality.

The coefficients $b_{k,j}$ are given by:

$$b_{k,j} = \frac{\langle xS_k(x), S_j(x)\rangle}{\langle S_j(x), S_j(x)\rangle}, \qquad j = 0, \dots, k$$

We define the inner product as

$$\langle S_p(x), S_q(x)\rangle = \sum_{i=0}^{m} S_p(x_i) S_q(x_i) \omega(x_i)$$

where $\omega(x_i)$ is a weight function.

Then,

$$b_{k,j} = \frac{\langle xS_k(x), S_j(x)\rangle}{\langle S_j(x), S_j(x)\rangle}$$

with

$$\begin{aligned}\langle xS_k(x), S_j(x)\rangle &= \sum_{i=0}^{m} x_i S_k(x_i) S_j(x_i)\omega(x_i)\\ &= \sum_{i=0}^{m} S_k(x_i) x_i S_j(x_i)\omega(x_i)\\ &= \langle S_k(x), xS_j(x)\rangle.\end{aligned}$$

Thus,

$$b_{k,j} = \frac{\langle S_k(x), xS_j(x)\rangle}{\langle S_j(x), S_j(x)\rangle}$$

Expanding $xS_j(x)$ using the same recursion:

$$xS_j(x) = S_{j+1}(x) + \sum_{i=0}^{j} b_{j,i} S_i(x)$$

Thus,

$$b_{k,j} = \frac{\langle S_k(x), S_{j+1}(x) + \sum_{i=0}^{j} b_{j,i} S_i(x)\rangle}{\langle S_j(x), S_j(x)\rangle}$$

$$= \frac{\langle S_k(x), S_{j+1}(x)\rangle + \sum_{i=0}^{j} b_{j,i}\langle S_k(x), S_i(x)\rangle}{\langle S_j(x), S_j(x)\rangle}$$

Special cases:

- For $j = 0, \dots, k-2$, $j+1 \neq k$ and $i \leq j$, the orthogonality implies $b_{k,j} = 0$.
- For $j = k-1$ and $i \leq k-1$, the last term vanishes and

$$b_{k,k-1} = \frac{\langle S_k(x), S_k(x)\rangle}{\langle S_{k-1}(x), S_{k-1}(x)\rangle}$$

For $j = k$, we have

$$\begin{aligned}
b_{k,k} &= \frac{\langle S_k(x), S_{k+1}(x)\rangle + b_{k,k}\langle S_k(x), S_k(x)\rangle}{\langle S_k(x), S_k(x)\rangle} \\
&= \frac{\langle S_k(x), xS_k(x) - \sum_{j=0}^{k} b_{k,j}S_j(x)\rangle + b_{k,k}\langle S_k(x), S_k(x)\rangle}{\langle S_k(x), S_k(x)\rangle} \\
&= \frac{\langle S_k(x), xS_k(x)\rangle + \langle S_k(x), -\sum_{j=0}^{k} b_{k,j}S_j(x)\rangle + b_{k,k}\langle S_k(x), S_k(x)\rangle}{\langle S_k(x), S_k(x)\rangle} \\
&= \frac{\langle S_k(x), xS_k(x)\rangle + \langle S_k(x), -b_{k,k}S_k(x)\rangle + b_{k,k}\langle S_k(x), S_k(x)\rangle}{\langle S_k(x), S_k(x)\rangle} \\
&= \frac{\langle S_k(x), xS_k(x)\rangle}{\langle S_k(x), S_k(x)\rangle}
\end{aligned}$$

Therefore,

$$S_{k+1}(x) = xS_k(x) - b_{k,k}S_k(x) - b_{k,k-1}S_{k-1}(x)$$

$$b_{k,k} = \frac{\sum_{i=0}^{m} S_k(x_i)\, x_i\, S_k(x_i)\, \omega(x_i)}{\sum_{i=0}^{m} S_k(x_i)S_k(x_i)\, \omega(x_i)}$$

```python
def cal_lam(self, ii):
    numerator = np.dot(self.xn * self.S[ii - 1](self.xn), self.S[ii - 1](self.xn))
    denominator = np.dot(self.S[ii - 1](self.xn), self.S[ii - 1](self.xn))
    return numerator / denominator
```

$$b_{k,k-1} = \frac{\sum_{i=0}^{m} S_k(x_i)\, S_k(x_i)\, \omega(x_i)}{\sum_{i=0}^{m} S_{k-1}(x_i)S_{k-1}(x_i)\, \omega(x_i)}$$

```python
def cal_miu(self, ii):
    if ii == 1:
        return 0.0
    else:
        numerator = np.dot(self.S[ii - 1](self.xn), self.S[ii - 1](self.xn))
        denominator = np.dot(self.S[ii - 2](self.xn), self.S[ii - 2](self.xn))
    return numerator / denominator
```

Then, we can use the orthogonality of this basis to get the fitting coefficients for the least-squares approximation. The coefficients are given by

$$b_k = \frac{\langle S_k(x), y\rangle}{\langle S_k(x), S_k(x)\rangle} = \frac{\sum_{i=0}^{m} S_k(x_i)\, y_i\, \omega(x_i)}{\sum_{i=0}^{m} S_k(x_i) S_k(x_i)\, \omega(x_i)}$$

```
    def cal_b(self, ii):
        numerator = np.dot(self.S[ii](self.xn), self.yn)
        denominator = np.dot(self.S[ii](self.xn), self.S[ii](self.xn))
        return numerator / denominator
```

Finally, we can fit the data points using the orthogonal basis:

$$\hat{y}(x) = \sum_{k=0}^{n} b_k\, S_k(x)$$

```
def cal_S(self, ii):
    lam, miu = self.lam[ii], self.miu[ii]
    S1, S2 = self.S[ii - 1], self.S[ii - 2]
    def S_ii(x):
        return (x - lam) * S1(x) - miu * S2(x)
    return S_ii

def cal_ortho_basis(self):
    for ii in range(1, self.N):
        self.lam[ii] = self.cal_lam(ii)
        self.miu[ii] = self.cal_miu(ii)
        self.S[ii] = self.cal_S(ii)
    self.b = np.array([self.cal_b(ii) for ii in range(self.N)])
    return self.S

def predict(self, x):
    y = np.zeros_like(x)
    for ii in range(self.N):
        y += self.b[ii] * self.S[ii](x)
    return y
```

2 Result:

We compare the effect of different polynomial degrees in least-squares fitting using the orthogonal basis. Figure 1 shows the fitting curves for select degrees. As the degree increases, the fit approaches the true function. When **the degree is equal to the number of data points**, the fitting curve passes through all points but oscillates strongly, illustrating overfitting.

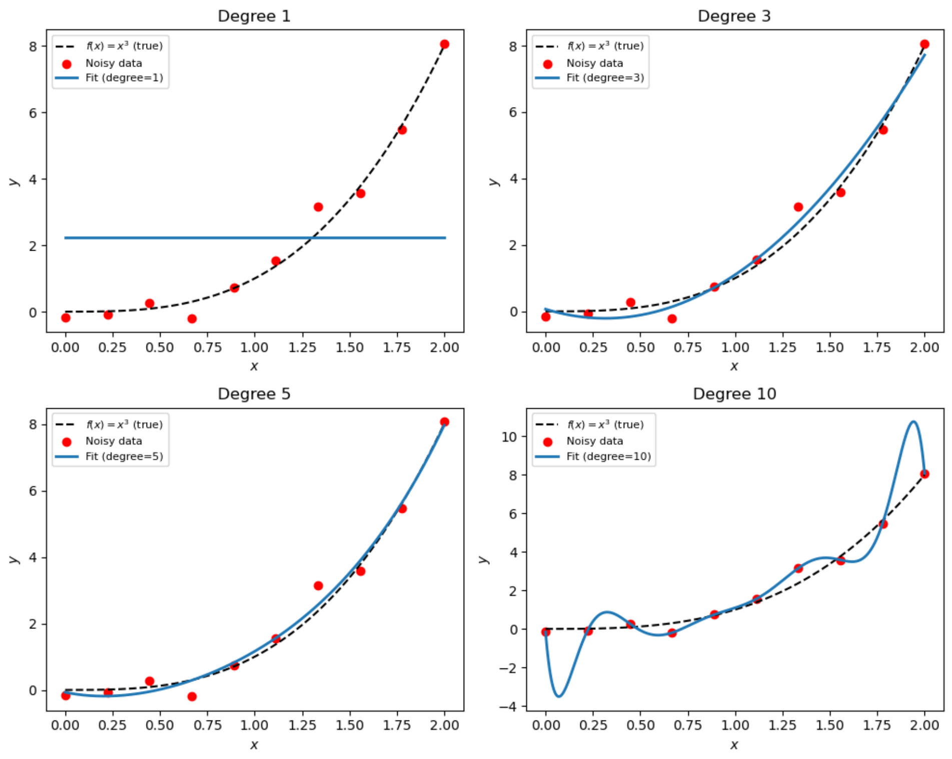

Figure 1 Orthogonal polynomial fitting results for degree 1, 3, 5, and 10. When the degree is equal to the number of data points, the fit exactly passes through all data but oscillates severely (overfitting).