数值分析第 12 章上机

By 211870125 陈睿硕

§1 问题

求 $f(x) = e^x$ 在区间 [0,1] 上的 n 次最佳平方逼近多项式:

$$\varphi(x) = \sum_{k=0}^{n} a_k \varphi_k(x)$$

Q1 $n = 5$，$\varphi(x) = x^k$;

Q2 $n = 5$，$\varphi(x)$ 为 k 次 Legendre 多项式;

Q3 $n = 10$，$\varphi(x) = x^k$;

Q4 $n = 10$，$\varphi(x)$ 为 k 次 Legendre 多项式;

Q5 实现 FFT，并应用于某个具体场景。如信号分析，图像，声音均可; 也可以用于偏微分方程求解 (谱方法); 也可以是自己找到的应用场景。

§2 算法思路

I 对Q1的解答

由书上的推导，只需解出线性方程组：

$$HX = b$$

其中，H 为 n+1 阶 Hilbert 矩阵，b 为由 $\int_0^1 x^k f(x)dx, k = 0, 1, \cdots, n$ 构成的 $(n + 1) \times 1$ 维向量。此方程的解 X 满足 $X = \begin{pmatrix} a_0 & a_1 & \cdots & a_n \end{pmatrix}^T$，如此可求得目标多项式的系数。
使用 Python 实现，代码见代码 1。如图 1。

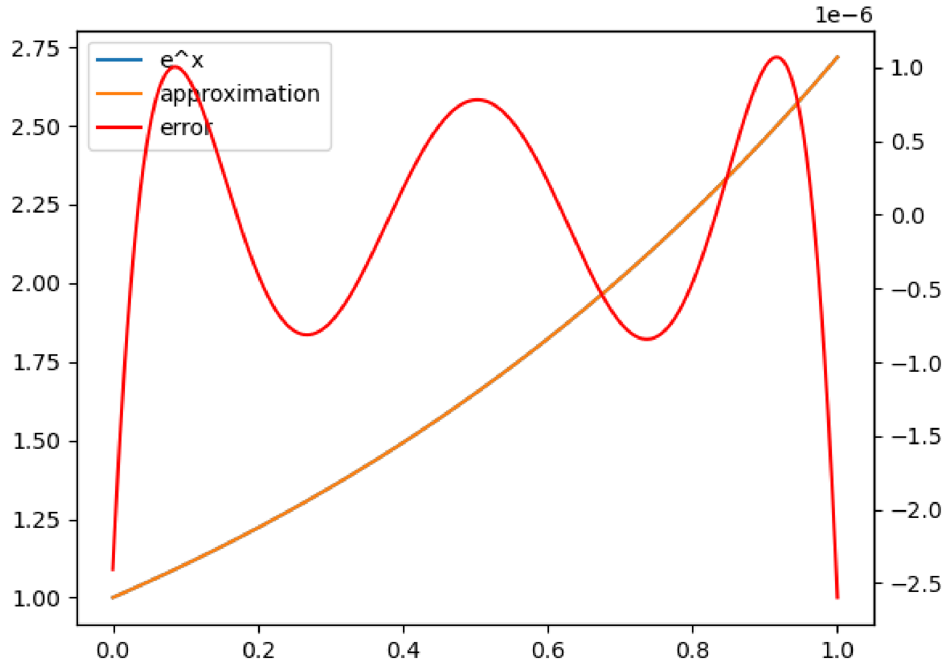

图 1: $n = 5$，$\varphi(x) = x^k$

II 对Q2的解答

Legendre 多项式是定义在 $[-1, 1]$ 上以 1 为权函数的直化多项式，故需做变换 $u = \frac{x+1}{2}$ 将其变换到区间 $[0, 1]$ 上。令 $g(x) = l(2x - 1)(l$ 为 Legendre 多项式)，则 $g(x)$ 为定义在 $[0, 1]$ 上的以 1 为权函数的直化多项式。类似于书上的推导，我们有：

$$a_j = \frac{\int_0^1 g(x)f(x)dx}{\int_0^1 g(x)^2dx} = (2j + 1)\int_0^1 g(x)f(x)dx, j = 0, 1, \cdots, n \tag{1}$$

使用 Python 实现，代码见代码 2。如图 2。

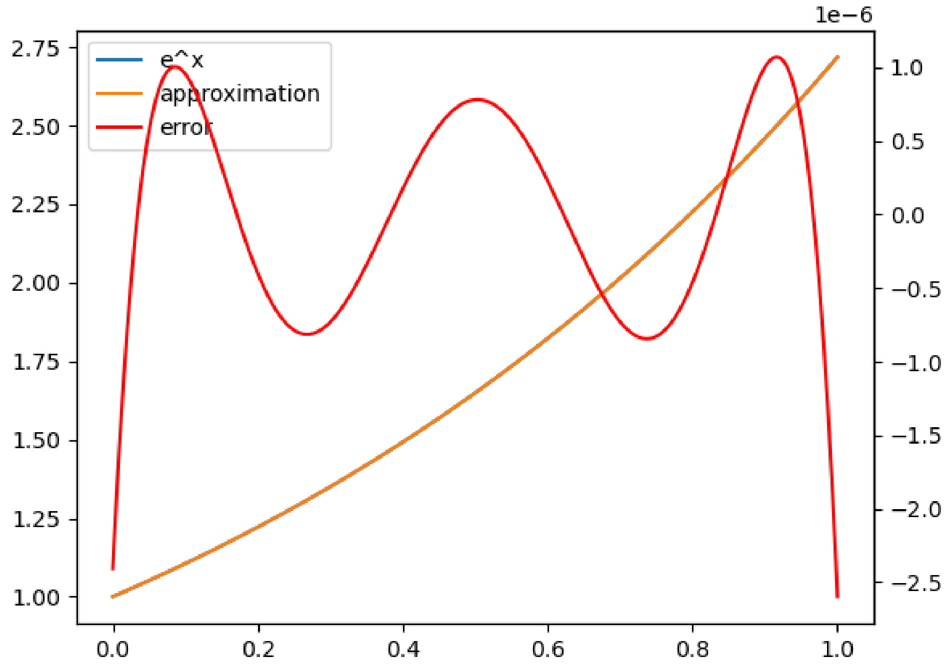

图 2: $n = 5$，$\varphi(x)$ 为 k 次 Legendre 多项式

III 对Q3的解答

方法同**Q1**，使用 Python 实现，代码见代码 3。如图 3。

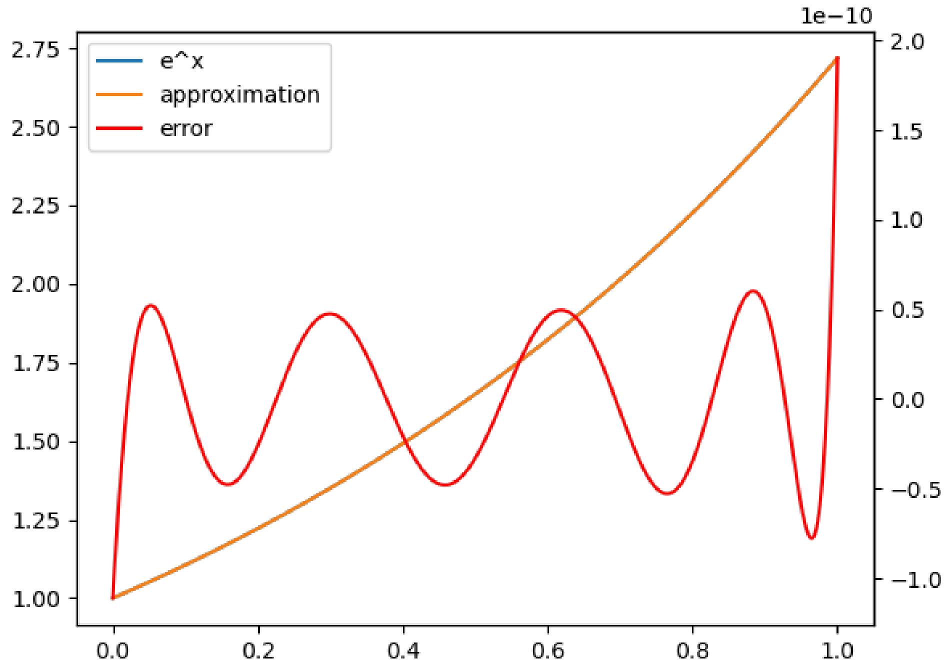

图 3: $n = 10$，$\varphi(x) = x^k$

IV 对Q4的解答

方法同**Q2**，使用 Python 实现，代码见代码 4。如图 4。

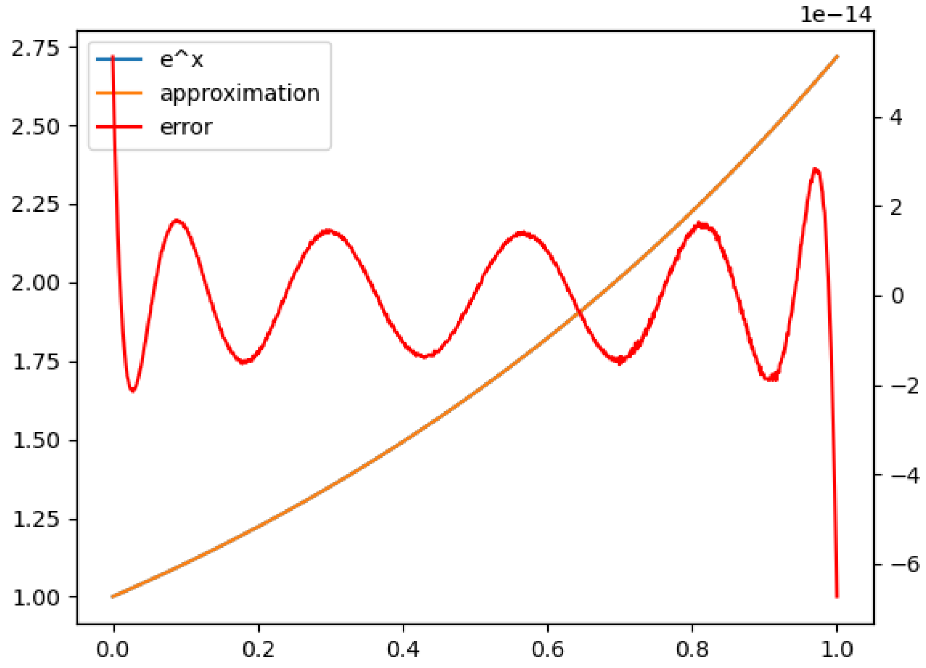

图 4: $n = 10$，$\varphi(x)$ 为 k 次 Legendre 多项式

V 对Q5的解答

我们使用 Cooley-Tukey FFT 算法实现了 FFT（见代码 5），并使用 FFT 对正弦信号进行压缩，保留大于 100 的频率成分，截断其他成分（见代码 5）。

如图 5。

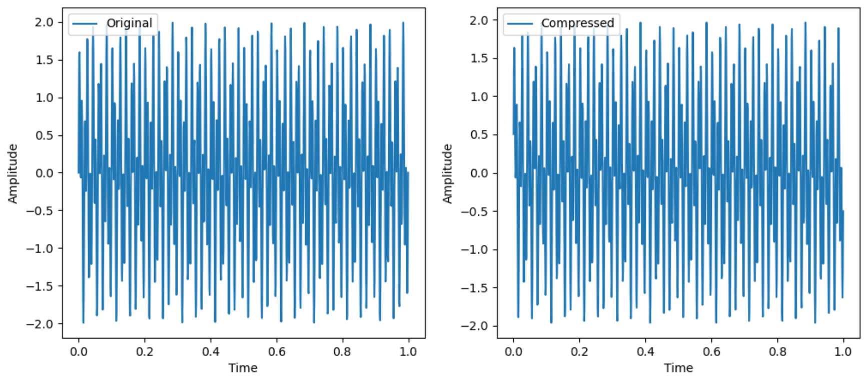

图 5: 压缩正弦信号

§3 结果分析

I 对Q1，Q2，Q3，Q4的分析

程序计算出相应多项式系数的运行时间如下：(我们取运行 50 次的时间平均值)

Run time(s)	$\varphi(x) = x^k$	$\varphi(x)$ 为 k 次 Legendre 多项式
$n = 5$	0.11888313293457031	0.12630319595336914
$n = 10$	0.12168169021606445	0.14491581916809082

可以看到时间差距并不大，于是我们取 n=100、1000，再做观察：

Run time(s)	$\varphi(x) = x^k$	$\varphi(x)$ 为 k 次 Legendre 多项式
$n = 100$	0.13881516456604004	0.35213565826416016
$n = 1000$	22.1142737865448	60+

可以看到 n 增大后，由于计算 Legendre 多项式的负担，使用 Legendre 多项式作为基函数反而用时更长，且估计可知 Legendre 多项式系数会发生上溢。

但观察图片可知，使用 Legendre 多项式作为基函数的误差随着 n 的增大会以数量级小于使用 x^k 作为基函数的误差，这和我们理论分析的结论是一致的：计算含 Hilbert 矩阵的线性方程组时出现的舍入误差会产生干扰。

§4 附录：程序代码

代码 1: Seventh Week 1A.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import quad
from scipy.linalg import hilbert
import time

def f(x, k):
    return x**k * np.exp(x)

def integral(k,f):
    result, error = quad(f, 0, 1, args=(k,))
    return result

def approx(x, a):
    return sum(a[i] * x**i for i in range(len(a)))
start=time.time()
n=5
b=np.array([integral(i,f=f) for i in range(n+1)])
A=hilbert(n+1)
```

```
X=np.linalg.solve(A,b)
fig,ax1=plt.subplots()
x = np.linspace(0, 1, 1000)
y = np.e**x
y_approx = approx(x, X)
end=time.time()
print("run time:",end-start)
line1,=ax1.plot(x, y, label='e^x')
line2,=ax1.plot(x, y_approx, label='approximation')
ax2=ax1.twinx()
line3,=ax2.plot(x,y_approx-y,label='error',color='r')
lines = [line1, line2, line3]
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels,loc='best')
plt.show()
```

代码 2: Seventh Week 1B.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import quad
from scipy.special import legendre
import time

def f(x, k):
    return legendre(k)(2*x-1) * np.exp(x)

def integral(k, func):
    result, error = quad(func, 0, 1, args=(k,))
    return result

def approx(x, a):
    return sum(a[i] * legendre(i)(2*x-1) for i in range(len(a)))

start=time.time()
n = 5
coefficient = np.array([(2*i+1)*integral(i, f) for i in range(n+1)])

fig, ax1 = plt.subplots()
x = np.linspace(0, 1, 1000)
y = np.e**x
y_approx = approx(x, coefficient)
```

```
end=time.time()
print("run time:",end-start)
line1, = ax1.plot(x, y, label='e^x')
line2, = ax1.plot(x, y_approx, label='approximation')
ax2 = ax1.twinx()
line3, = ax2.plot(x, y_approx-y, label='error', color='r')
lines = [line1, line2, line3]
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='best')
plt.show()
```

代码 3: Seventh Week 1C.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import quad
from scipy.linalg import hilbert
import time

def f(x, k):
    return x**k * np.exp(x)

def integral(k,f):
    result, error = quad(f, 0, 1, args=(k,))
    return result

def approx(x, a):
    return sum(a[i] * x**i for i in range(len(a)))
start=time.time()
n=10
b=np.array([integral(i,f=f) for i in range(n+1)])
A=hilbert(n+1)
X=np.linalg.solve(A,b)
fig,ax1=plt.subplots()
x = np.linspace(0, 1, 1000)
y = np.e**x
y_approx = approx(x, X)
end=time.time()
print("run time:",end-start)
line1,=ax1.plot(x, y, label='e^x')
line2,=ax1.plot(x, y_approx, label='approximation')
ax2=ax1.twinx()
```

```
line3,=ax2.plot(x,y_approx-y,label='error',color='r')
lines = [line1, line2, line3]
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels,loc='best')
plt.show()
```

代码 4: Seventh Week 1D.py

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.integrate import quad
from scipy.special import legendre
import time

def f(x, k):
    return legendre(k)(2*x-1) * np.exp(x)

def integral(k, func):
    result, error = quad(func, 0, 1, args=(k,))
    return result

def approx(x, a):
    return sum(a[i] * legendre(i)(2*x-1) for i in range(len(a)))

start=time.time()
n = 10
coefficient = np.array([(2*i+1)*integral(i, f) for i in range(n+1)])

fig, ax1 = plt.subplots()
x = np.linspace(0, 1, 1000)
y = np.e**x
y_approx = approx(x, coefficient)
end=time.time()
print("run time:",end-start)
line1, = ax1.plot(x, y, label='e^x')
line2, = ax1.plot(x, y_approx, label='approximation')
ax2 = ax1.twinx()
line3, = ax2.plot(x, y_approx-y, label='error', color='r')
lines = [line1, line2, line3]
labels = [l.get_label() for l in lines]
ax1.legend(lines, labels, loc='best')
plt.show()
```

代码 5: Seventh Week 2.py

```
import numpy as np
import matplotlib.pyplot as plt

def fft(x):
    N = len(x)
    if N == 1:
        return x
    even = fft(x[0::2])
    odd = fft(x[1::2])
    freq = np.zeros(N, dtype=complex)
    for k in range(N//2):
        t = np.exp(-2j * np.pi * k / N) * odd[k]
        freq[k] = even[k] + t
        freq[k + N//2] = even[k] - t
    return freq

def ifft(x):
    N = len(x)
    if N == 1:
        return x
    even = ifft(x[0::2])
    odd = ifft(x[1::2])
    time = np.zeros(N, dtype=complex)
    for k in range(N//2):
        t = np.exp(2j * np.pi * k / N) * odd[k]
        time[k] = even[k] + t
        time[k + N//2] = even[k] - t
    return time

N = 1024
t = np.linspace(0, 1, N)
x = np.sin(2 * np.pi * 50 * t) + np.sin(2 * np.pi * 120 * t)
X = fft(x)

X_mag = np.abs(X)
X_mag[X_mag < 100] = 0
X2 = X_mag * np.exp(1j * np.angle(X))

x2 = np.real(ifft(X2)/N)
```

```
plt.figure(figsize=(12, 5))
plt.subplot(121)
plt.plot(t, x, label='Original')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.subplot(122)
plt.plot(t, x2, label='Compressed')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.show()
```