project2

20989977 Zhang Mingtao

2024/5/5

0.

```
library(reticulate)
```

2.

(a). $a=x_0 \quad x_1 \quad x_2 \quad \cdots \quad x_{N-1} \quad x_N=b \qquad x_k = a+k\cdot h$

$h \quad h \quad \cdots \quad h. \qquad h=\frac{b-a}{N}.$

$$f(x) = f\left(\frac{x_{k-1}+x_k}{2}\right) + \left(x-\frac{x_{k-1}+x_k}{2}\right) f'\left(\frac{x_{k-1}+x_k}{2}\right) + \frac{1}{2}\left(x-\frac{x_{k-1}+x_k}{2}\right)^2 \cdot f''\left(\frac{x_{k-1}+x_k}{2}\right) + \cdots$$

$$\Rightarrow \int_{x_{k-1}}^{x_k} f(x)dx = h\cdot f\left(\frac{x_{k-1}+x_k}{2}\right) + 0 + \frac{1}{24}h^3 f''\left(\frac{x_{k-1}+x_k}{2}\right) + 0 + O(h^5)$$

$$\Rightarrow \int_a^b f(x)dx = h\cdot \sum_{k=1}^{N} f\left(\frac{x_{k-1}+x_k}{2}\right) + \underline{\frac{1}{24}h^3 \sum_{k=1}^{N} f''\left(\frac{x_{k-1}+x_k}{2}\right) + O(h^5)}$$

$$\because \int_a^b f''(x)dx = h\cdot \sum_{k=1}^{N} f''\left(\frac{x_{k-1}+x_k}{2}\right) + O(h^3)$$

$$\Rightarrow \frac{1}{24}h^3 \sum_{k=1}^{N} f''\left(\frac{x_{k-1}+x_k}{2}\right) = \frac{h^2}{24}\int_a^b f''(x)dx + O(h^5)$$

$$= \frac{h^2}{24}(f'(b)-f'(a)) + O(h^5)$$

$\therefore$ leading-order error is $\underbrace{\frac{1}{24}[f'(b)-f'(a)]}_{c} \cdot h^2$.

(b) Let $x = \tan z$.

$$\Rightarrow \int_{-\infty}^{\infty} e^{-x^4}dx = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} e^{-\tan^4 z} dz \cdot \sec^2 z = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{e^{-\tan^4 z}}{\cos^2 z} dz.$$

$a=-\frac{\pi}{2}$, $b=\frac{\pi}{2}$. $g(z) = e^{-\tan^4 z}/\cos^2 z$

$$g(a) = \lim_{x\to a^+} g(x) = \lim_{z\to -\frac{\pi}{2}^+} \frac{e^{-\tan^4 z}}{\cos^2 z} = \lim_{z\to -\frac{\pi}{2}^+} \frac{e^{-\tan^4 z}\cdot(-4\tan^3 z)\cdot \sec^2 z}{-2\cos z \sin z \longrightarrow -\sin 2z}$$

$$= \lim_{z\to -\frac{\pi}{2}^+} \frac{16\tan^6 z \sec^4 z \cdot \underbrace{e^{-\tan^4 z}}_{\to 0} + \underbrace{e^{-\tan^4 z}}_{\to 0}(-12\tan^2 z \cdot \sec^4 z - 8\tan^3 z \cdot \sec^2 z)}{-2\cos(2z)}$$

$$= \frac{0}{-2\cdot(-1)} = 0.$$

$g(b) = \frac{0}{-2} = 0$. the same.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math

# from scipy.integrate import quad
# quad(f, a, b)

# (c)
def f(x):
    return math.exp(-(math.tan(x))**4) / (math.cos(x))**2

def composite_midpoint_rule(a, b, n):
    h = (b - a) / n
    integral = 0.0
    for i in range(n):
        xi = a + (i + 0.5) * h
        integral += f(xi)
    integral *= h
    return integral

def relative_error(fit, exact):
    return abs(fit - exact) / abs(exact)

def cmr_integral():
    a = -math.pi / 2
    b = math.pi / 2
    n = 1
    pre_fit = 0.0
    while True:
        fit = composite_midpoint_rule(a, b, n)
        if n > 1:
            error = relative_error(fit, pre_fit)
            if error <= 1e-6:
                break
        pre_fit = fit
        n *= 3
    return fit

cmr_integral()


# (d)
```

```
## 1.8128049541109559
```

```
def tent(R, R_pre, m):
    return R + 1/(4**m-1)*(R-R_pre)

def romberg_integration():
    a = -math.pi / 2
    b = math.pi / 2
    n = 200
    h = (b-a)/n
    fit = 0
    R_matrix = []
    for i in range(n):
        if i%2 == 1:
            fit += f(a+i*h)
    pre_fit = fit
    fit *= h
    R_matrix.append([fit])

    iteration=1

    while len(R_matrix)<=1 or abs(R_matrix[-1][-1] - R_matrix[-2][-1])/((2*(len(R_matrix)))**2
-1) > 1e-6:
        iteration += 1
        n *= 3
        h = (b-a)/n
        fit = 0
        for i in range(n):
            if (i/3) % 2 == 1:
                continue
            if i%2 ==1:
                fit += f(a+i*h)
        fit += pre_fit
        pre_fit = fit
        fit *= h
        R_matrix.append([fit])
        while len(R_matrix[-1]) < len(R_matrix):
            R_matrix[-1].append(tent(R_matrix[-1][-1], R_matrix[-2][len(R_matrix[-1])-1], len(R
_matrix[-1])))
        return R_matrix[-1][-1]*2

romberg_integration()


# (e)
```

```
## 1.8128049541109563
```

```
def composite_trapezoidal_rule(a, b, n):
    h = (b - a) / n
    integral = 0.0
    for i in range(n + 1):
        xi = a + i * h
        if i == 0 or i == n:
            integral += 0.5 * f(xi)
        else:
            integral += f(xi)
    integral *= h
    return integral

def ctr_integral():
    a = -math.pi / 2
    b = math.pi / 2
    n = 2
    pre_fit = 1.57
    while True:
        fit = composite_trapezoidal_rule(a, b, n)
        if n > 1:
            error = relative_error(fit, pre_fit)
            if error <= 1e-6:
                break
        pre_fit = fit
        n *= 2
    return fit

ctr_integral()

# (f)
```

```
## 1.8128049541109548
```

```
def gaussxw(N):
    # Initial approximation to roots of the Legendre polynomial
    a = np.linspace(3,4*N-1,N)/(4*N+2)
    x = np.cos(np.pi*a+1/(8*N*N*np.tan(a)))

    # Find roots using Newton's method
    epsilon = 1e-15
    delta = 1.0
    while delta>epsilon:
        p0 = np.ones(N,float)
        p1 = np.copy(x)
        for k in range(1,N):
            p0,p1 = p1,((2*k+1)*x*p1-k*p0)/(k+1)
        dp = (N+1)*(p0-x*p1)/(1-x*x)
        dx = p1/dp
        x -= dx
        delta = max(abs(dx))

    # Calculate the weights
    w = 2*(N+1)*(N+1)/(N*N*(1-x*x)*dp*dp)

    return x,w

def G_integral():
    a = -math.pi / 2
    b = math.pi / 2
    n = 1
    pre_fit = 0
    while True:
        x,w = gaussxw(n)
        xp = 0.5*(b-a)*x + 0.5*(b+a)
        wp = 0.5*(b-a)*w
        fit = 0
        for k in range(n):
            fit += wp[k]*f(xp[k])
        if n > 1:
            error = relative_error(fit, pre_fit)
            if error <= 1e-6:
                break
        pre_fit = fit
        n += 1
    return fit

G_integral()

# (g)
```

```
## 1.812802553397727
```

```
def f(x):
    numerator = np.exp(-(x[0]**4 + x[1]**4 + x[2]**4 + x[3]**4))
    denominator = 1 + x[0]**2 + x[1]**2 + x[2]**2 + x[3]**2
    # Avoid overflow and division by zero
    if denominator == 0:
        return 0
    else:
        return numerator / denominator

def metropolis_algorithm(n, step_size):
    x = np.zeros(4)  # Initial point
    integral_sum = 0.0
    integral_squared_sum = 0.0
    count = 0

    for _ in range(n):
        y = x + step_size * np.random.randn(4)
        pa = min(1, f(y) / f(x))

        if np.random.rand() < pa:
            x = y
            count += 1

        weight = np.exp(-(x[0]**4 + x[1]**4 + x[2]**4 + x[3]**4))
        value = f(x)
        integral_sum += value / weight
        integral_squared_sum += (value / weight)**2

    integral_mean = integral_sum / n
    integral_variance = (integral_squared_sum / n - integral_mean**2) / n
    integral_error = np.sqrt(integral_variance / n)

    a_rate = count / n

    return integral_mean, integral_error, a_rate

n = int(1e6)
step_size = 0.1
integral_mean, integral_error, a_rate = metropolis_algorithm(n, step_size)
integral_value = integral_mean

print("Approximate integral:", integral_value)
```

```
## Approximate integral: 0.5121431620842426
```

```
print("Error estimate:", integral_error)
```

```
## Error estimate: 1.545525665587253e-07
```

```
print("Acceptance rate:", a_rate)
```

```
## Acceptance rate: 0.861609
```