In [ ]:
```python
import numpy as np
import math
import pandas as pd

import scipy

from sympy import *

import matplotlib.pyplot as plt
```

对于一个函数求解其积分问题,最直接的方式即是采用Newton-Leibniz公式,即:

$$\int_a^b f(x)dx = F(b) - F(a)$$

其中,$F(x)$为$f(x)$的原函数 而在某些情况下原函数$F(x)$很难求出,这时我们需要采用机械求积的方法来对函数进行积分求解

In [ ]:
```python
def f(x):
    return np.exp(x).astype(np.float64)

a,b = -1,1

N = 100
```

Newton-Cotes公式求解积分:\ 设将积分区间$[a,b]$均分为n份,每一份的长度$h = \frac{b-a}{n}$,选取等距节点$x_k = a + th, t = 0, 1, \dots, n$构造出的插值型求积公式称为Newton-Cotes公式.

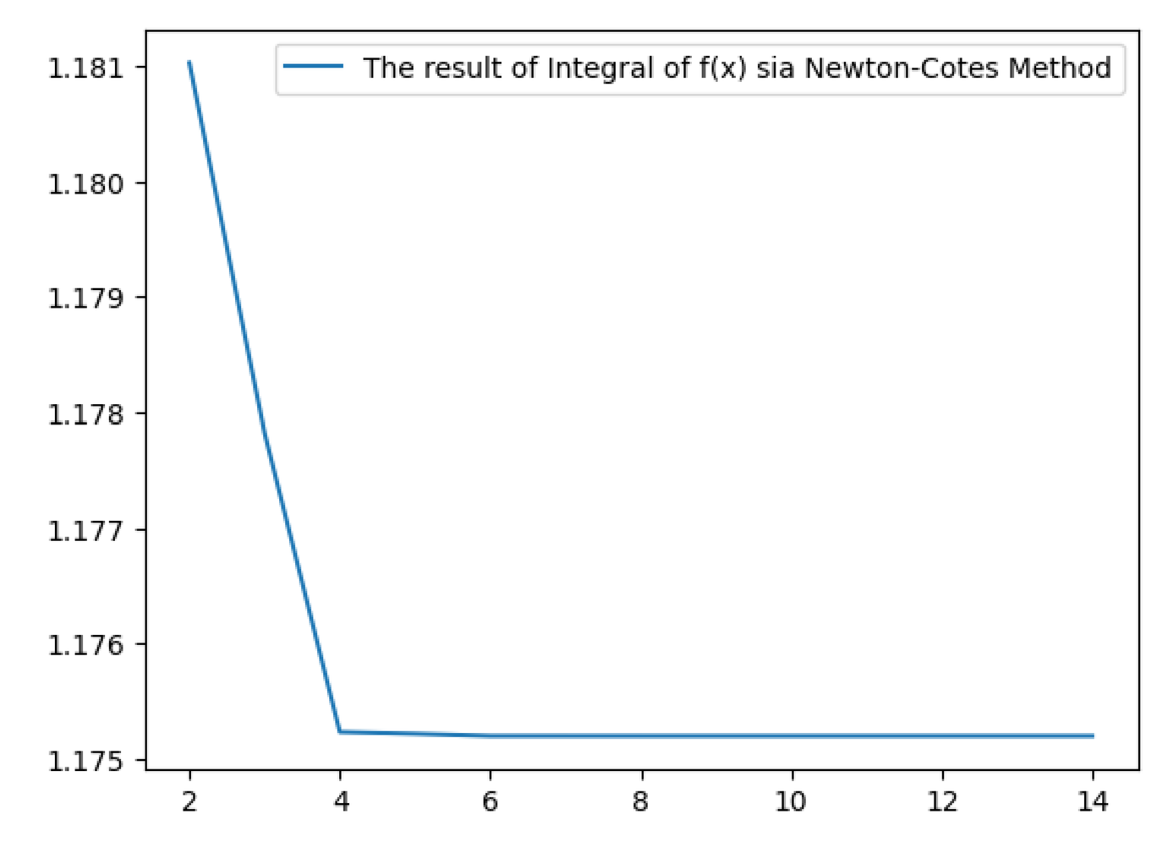
$$I_n = (b-a)\sum_{k=0}^{n} C_k^{(n)} f(x_k)$$

式中$C_k^{(n)}$称为Cotes系数,其取值如下:

$$C_k^{(n)} = \frac{h}{b-a}\int_0^n \prod_{j=0,k\neq j}^{n} \frac{t-j}{k-j}dt = \frac{(-1)^{n-k}}{nk!(n-k)!}\int_0^n \prod_{j=0,k\neq j}^{n}(t-j)dt$$

In [ ]:
```python
def newton_cotes(f, a, b, n = 5):
    """采用Newton-Cotes公式对函数f在[a,b]区间进行数值积分，返回积分结果。

    Args:
        f (function): 待积分函数
        a (float): 区间下限
        b (float): 区间上限
        n (int): 阶数

    Returns:
        float: 积分结果
    """
    x = np.linspace(a, b, n + 1)

    def get_cotes_params(a,b,k,n):

        def get_polyminal(t):
            res = 1
            for i in range(n+1):
                if i != k:
                    res *= (t - i)
            return res

        res = (-1)**(n-k) * scipy.integrate.quad(get_polyminal, 0,n)[0] / (n * math.factorial(k) * math.factorial(n-k))

        return res

    cotes = [get_cotes_params(a,b,k,n) for k in range(n+1)]

    res = sum(cotes[k] * f(x[k]) for k in range(n+1))

    ##print(f"积分函数:{f.__name__},积分区间:[{a},{b}],阶数:{n},积分值:{res}")

    return res


n = [i for i in range(2,15)]

y_NC = [newton_cotes(f, a, b, i) for i in n]

plt.plot(n,y_NC)
plt.legend(["The result of Integral of f(x) sia Newton-Cotes Method"])
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x28b6b1aacd0>
```



而通过稳定性分析得知:\ 在n>=8时,Newton-Cotes方法的稳定性变得很差,出于稳定性考虑,我们选择使用划分区间分段求解的方法,即所谓的复合求积法

而我们容易发现,当我们需要更高的精度时,复合积分法可以做到的但是会显得很"吃力",例如:\ 复合梯形法的误差限估计大致为:$O(h^2)$,这意味着如果$\epsilon = 10^{-6}$时,$h = 10^{-3}$,我们需要对区间做约10次二分才能做到这一点。\ 而复合 Simpson 积分法误差限估计为:$O(h^4)$，这意味着如果$\epsilon = 10^{-6}$时,$h = 10^{-4}$,我们需要对区间做约5次二分才能做到这一点。\ 但是如果想要达到谱精度,在区间长度有限的情况下显然计算量较大.\ 在这种背景下,我们介绍Richard外推加速法:\ Difinition:我们定义如果对区间做一次二分称为计算加速一次\ 我们有: $$ T*m^{(k)} = \frac{4^m}{4^m - 1}T*{m-1}^{(k+1)} - \frac{1}{4^m - 1}T_{m-1}^{(k)} \ or \

T*m(h) = \frac{4^m}{4^m - 1}T*{m-1}(\frac{h}{2}) - \frac{1}{4^m - 1}T_{m-1}(h) $$

In [ ]:
```python
class IntegralOfSteven:

    """
    Steven的积分类
    """

    def __init__(self,f,a,b,n):
        self.func = f
        self.a = a
        self.b = b
        self.n = n

    def compositeTrapzoid(self):
        """
        复合梯形法
        """
        h = (self.b - self.a) / self.n
        sum = 0
        for i in range(self.n):
            sum += self.func(self.a + i * h) + self.func(self.a + (i + 1) * h)
        return h / 2 * sum


    def compositeSimpson(self):
        """
        复合 Simpson 法
        """
        h = (self.b - self.a) / self.n
        sum = 0
        for i in range(self.n):
            sum += self.func(self.a + i * h) + 4 * self.func(self.a + (i + 1/2) * h) + self.func(self.a + (i + 1) * h)
        return h / 6 * sum


    def romberg(self,epsilon = 1e-10,output_key = False):
        """
        Romberg 方法

        Args:
            epsilon: 精度
        """
        k = 0
        n = self.n
        h = (self.b - self.a) / n


        R = list()
        R.append([sum([h * self.func(self.a + i * h) for i in range(n+1)])])

        while True:
            k += 1

            k_array = list()

            for j in range(k+1):

                if j == 0:

                    t = 0

                    for i in range(1,n+1,2):
                        t += self.func(self.a + i * h) * h

                    T_jk = R[k-1][0] / 2 + t

                    n,h = n*2,h / 2

                    k_array.append(T_jk)

                else:

                    T_jk = 4**k * k_array[-1] / (4**k - 1) - R[k-1][-1] / (4**k - 1)

                    k_array.append(T_jk)


            R.append(k_array)

            if k == 15 or abs(R[k][-1] - R[k-1][-1]) < epsilon:
                break

        if output_key:
            print(f"迭代次数为:{k},函数积分值为:{R[k][-1]}")

        return R[k][-1]


    def adaptivesIntegral(self,epsilon = 1e-8):
        """
        自适应梯形积分法
        """

        a = self.a
        b = self.b
        f = self.func

        # 计算初始粗略估计的积分值
        h = (b - a) / 2
        c = (a + b) / 2
        fa = f(a)
        fc = f(c)
        fb = f(b)
        integral_approximation = h / 3 * (fa + 4 * fc + fb)

        # 计算更精细估计的积分值
        def recursive_simpson(f, a, b, epsilon, whole_approximation):
            h = (b - a) / 2
            c = (a + b) / 2
            d = (a + c) / 2
            e = (c + b) / 2
            fd = f(d)
            fe = f(e)
            left_approximation = h / 6 * (fa + 4 * fd + fc)
            right_approximation = h / 6 * (fc + 4 * fe + fb)
            split_approximation = left_approximation + right_approximation

            if abs(split_approximation - whole_approximation) <= 15 * epsilon:
                ##这里取15倍误差是有原因的,详细可以参见论文证明

                return split_approximation + (split_approximation - whole_approximation) / 15

            return recursive_simpson(f, a, c, epsilon / 2, left_approximation) + recursive_simpson(f, c, b, epsilon / 2, right_approximation)

        return recursive_simpson(f, a, b, epsilon, integral_approximation)

    def gaussIntegral(self,deg = 3,output_key = False,param_out_key = False):
        """
        高斯积分法

        因为不得不用到多项式积分逻辑计算的内容,因此我们引入了符号计算库sympy来辅助我们计算积分
        """

        x = symbols("x")
        def f(x):
            return exp(x)

        #legendre多项式
        def L(deg):
            df = diff((x**2 - 1) ** (deg+1),x,deg+1)
            L = (1 / 2**(deg+1)) / factorial(deg+1) * df

            return L

        #高斯点获取
        def getGaussPoint(deg):
            x_k = solve(L(deg))
            return x_k

        #计算求积系数A_k
        def getQuadratureCoefficient(x_klist,deg):
            Alist = []
            for x_k in x_klist:
                A = 2/((1-x_k**2)*(diff(L(deg),x,1).subs(x,x_k))**2)
                ## Obeject(sympy).subs(x,y)表示将这个sympy物体中的"x"替换为另一个符号变量或者值
                Alist.append(A)
            return Alist


        res = 0
        x_klist = getGaussPoint(deg)

        Alist = getQuadratureCoefficient(x_klist,deg)

        length = len(Alist)

        if a == 1 and b == 1:
            for i in range(length):
                res += (Alist[i] * f(x_klist[i])).evalf()


        else:
            for i in range(length):
                x = (b-a)/2 * x_klist[i] + (b+a)/2
                res += (b-a)/2 * (Alist[i] * f(x)).evalf()



        if output_key:
            print(f"函数积分值为:{res}")

        if param_out_key:
            return [res,Alist]

        return res
```

In [ ]:
```python
inte = IntegralOfSteven(f,a,b,100)

y_CT = inte.compositeTrapzoid()
y_CS = inte.compositeSimpson()
y_RB = inte.romberg()
y_GL = inte.gaussIntegral()
y_CT,y_CS,y_RB,y_GL
```

Out[ ]:
```
(2.3504807335115387, 2.3504023874181783, 2.35040332193691, 2.35040209215638)
```

ex1:考察积分

$$\int_0^1 \sqrt{x}f(x)dx, \int_0^1 \frac{f(x)}{\sqrt{x}}dx$$

试建立2点,4点Gauss公式

In [ ]:
```python
def f(x):
    return x

ex_1 = IntegralOfSteven(f,0,1,100)

paramOfTwoPoints = ex_1.gaussIntegral(1,output_key=True,param_out_key=True)[1]
paramOfFourPoints = ex_1.gaussIntegral(3,output_key=True,param_out_key=True)[1]

print(paramOfTwoPoints)
print(paramOfFourPoints)
```

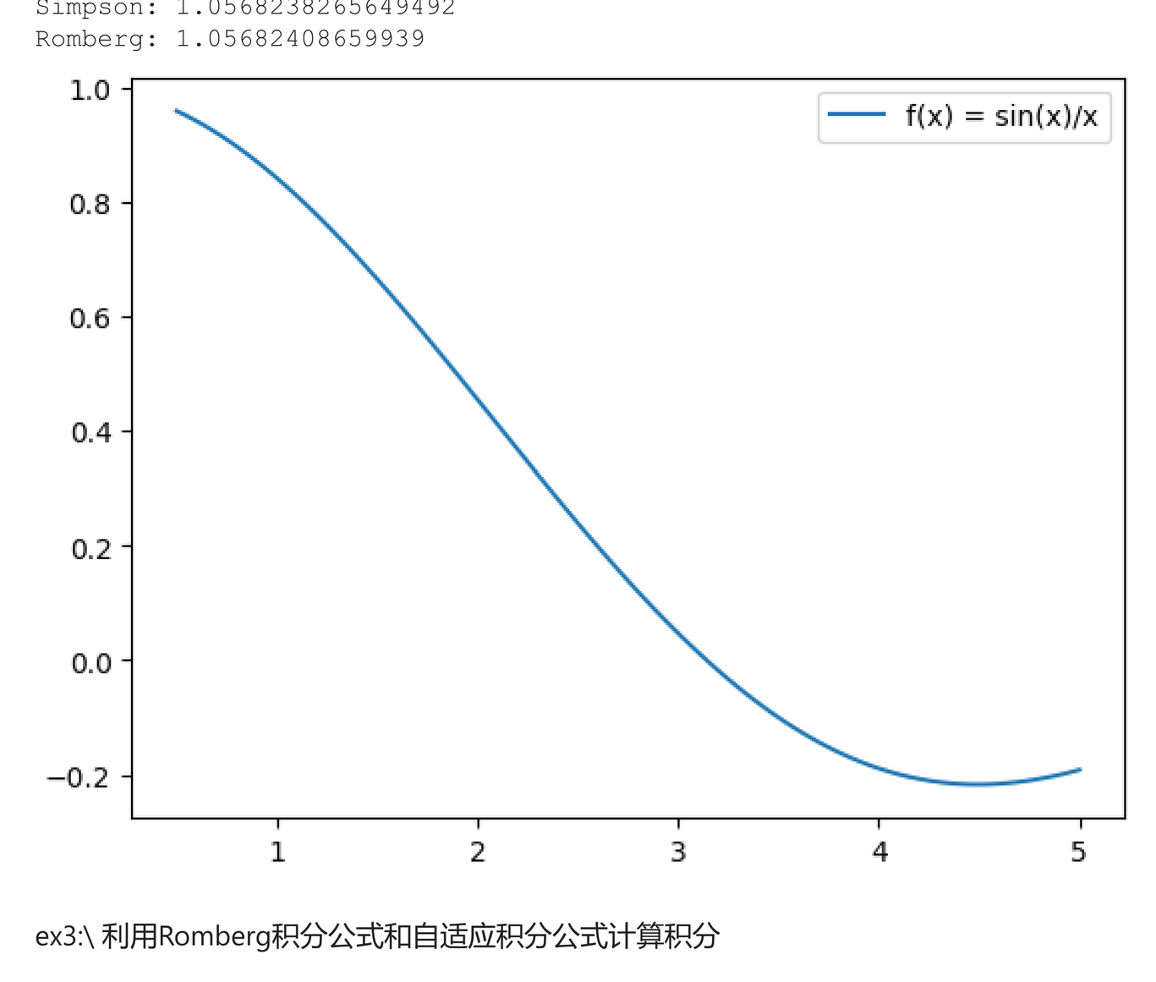
```
函数积分值为:2.34269608790973
函数积分值为:2.35040209215638
[1.00000000000000, 1.00000000000000]
[0.347854845137454, 0.652145154862546, 0.652145154862546, 0.347854845137454]
```

ex2:取n = 8,16;用复合梯形公式,simpson公式,Romberg积分计算

$$\int_{0.5}^5 \frac{\sin x}{x}dx$$

In [ ]:
```python
def ex_2(x):
    return np.sin(x) / x


resOfEx_2 = IntegralOfSteven(ex_2,0.5,5,101)

x = np.linspace(0.5,5,101)

y = ex_2(x)

plt.plot(x,y)
plt.legend(["f(x) = sin(x)/x"])


Trap = resOfEx_2.compositeTrapzoid()
Simp = resOfEx_2.compositeSimpson()
Romb = resOfEx_2.romberg()
print("Trapzoid:",Trap)
print("Simpson:",Simp)
print("Romberg:",Romb)
```

```
Trapzoid: 1.0568664460911164
Simpson: 1.0568238265649492
Romberg: 1.05682408659939
```



ex3:\ 利用Romberg积分公式和自适应积分公式计算积分

$$\int_{0.5}^5 \frac{100}{x^2}\sin\frac{10}{x}dx$$

可选择自适应Simpson,自适应中点公式,或者自适应两点Gauss公式

In [ ]:
```python
def ex_3(x):
    return 100 / x**2 * np.sin(10 / x)

resOfEx_3 = IntegralOfSteven(ex_3,0.5,5,101)

x = np.linspace(0.5,5,101)
y = ex_3(x)

plt.plot(x,y)
plt.legend(["f(x) = 100 / x**2 * np.sin(10 / x)"])

romberg = resOfEx_3.romberg(output_key=True)
```

```
迭代次数为:15,函数积分值为:-8.242237103561967
```