上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

课程论文

COURSE PAPER

论文题目： 2020 年秋季学期计算方法大作业

学生姓名:	吕东旭
学生学号:	020039910004
课程名称:	计算方法
指导教师:	曾进
学 院 (系):	微纳电子学系

计算方法大作业：

用不同数值方法计算积分 $\int_0^1 \sqrt{x}\ln x\,dx = -\frac{4}{9}$

1. 利用复化梯形公式和复化辛普森公式计算积分

取不同的步长 h，分别用复化梯形公式和复化辛普森公式计算积分，给出误差中关于 h 的函数，并与积分精确值比较两个公式的精度，是否存在一个最小的 h，使得精度不能再被改善？

本问题可以分解为 4 个子问题：

1. 利用复化梯形公式求解给定积分问题，并给出误差中关于 h 的函数。
2. 利用复化辛普森公式求解给定积分问题，并给出误差中关于 h 的函数。
3. 与积分精确值比较复化梯形公式和复化辛普森公式的精度。
4. 探究：是否存在一个最小的 h，使得精度不能再被改善？

根据以上分类，我们用以下四小节来分别解答这四个问题。

1.1 利用复化梯形公式求解给定积分问题，并给出误差中关于 h 的函数

令 $f(x) = \sqrt{x}\ln x$，设将区间 $[a,b]$ 划分为 n 等份，等分点 $x_k = a + kh$，$h = \frac{b-a}{n}, k = 0,1,2,3,...,n$。由此得到复化梯形公式：

$$T_n = \frac{h}{2}[f(a) + 2\sum_{k=1}^{n-1} f(x_k) + f(b)] \tag{0–1}$$

复合梯形公式的余项为：

$$R_n(f) = -\frac{b-a}{12}h^2 f''(\eta), \eta \in (0,1) \tag{0–2}$$

在本题中，$a = 0, b = 1$，且因为 $f(0) \to -\infty$，我们这里令 $f(0) = \lim_{x\to 0}\sqrt{x}\ln x$，同时 $f(1) = 0$。等分点 x_k 也可以写作：$x_k = kh$，其中 $h = \frac{1}{n}, k = 0,1,2,3,...,n$。根据以上分析，$f(x)$ 的复化梯形公式可以写为：

$$T_n = \frac{h}{2}\left(\lim_{x\to 0}\sqrt{x}\ln x + 2\sum_{k=1}^{n-1}\sqrt{kh}\ln(kh)\right) \tag{0–3}$$

计算 $f(x)$ 的各阶导数：

$$\begin{aligned} f^{(1)}(x) &= \frac{1}{2}x^{-\frac{1}{2}}(\ln x + 2) \\ f^{(2)}(x) &= -\frac{1}{4}x^{-\frac{3}{2}}\ln x \\ f^{(3)}(x) &= \frac{1}{4}x^{-\frac{5}{2}}(\frac{3}{2}\ln x - 1) \\ f^{(4)}(x) &= x^{-\frac{7}{2}} - \frac{15}{16}x^{-\frac{7}{2}}\ln x \end{aligned} \tag{0–4}$$

将上式带入公式 (0–2) 中，得到 $f(x)$ 复化梯形公式的误差函数：

$$R_n(f) = \frac{h^2}{48}\eta^{-\frac{3}{2}}\ln\eta \tag{0–5}$$

1.2 利用复化辛普森公式求解给定积分问题，并给出误差中关于 h 的函数

若记 $x_{k+1/2} = x_k + \frac{1}{2}h$，则得到复化辛普森公式为：

$$\begin{aligned} S_n &= \frac{h}{6}\sum_{k=0}^{n-1}[f(x_k) + 4f(x_{k+1/2}) + f(x_{k+1})] \\ &= \frac{h}{6}[f(a) + 4\sum_{k=0}^{n-1} f(x_{k+1/2}) + 2\sum_{k=1}^{n-1} f(x_k) + f(b)] \end{aligned} \tag{0–6}$$

当 $f(x) \in C^4[a,b]$ 时，复化辛普森公式的余项为：

$$R_n(f) = -\frac{b-a}{180}\left(\frac{h}{2}\right) f^{(4)}(\eta), \eta \in (a,b) \tag{0–7}$$

在本题中，$a = 0, b = 1$，且因为 $f(0) \to -\infty$，我们这里令 $f(0) = \lim_{x\to 0}\sqrt{x}\ln x$，同时 $f(1) = 0$。等分点 x_k 也可以写作：$x_k = kh$，其中 $h = \frac{1}{n}, k = 0, 1, 2, 3, ..., n$。根据以上分析，$f(x)$ 的复化辛普森公式可以写为：

$$S_n = \frac{h}{6}\left(\lim_{x\to 0}\sqrt{x}\ln x + 4\sum_{k=0}^{n-1}\sqrt{\frac{(2k+1)h}{2}}\ln\frac{(2k+1)h}{2} + 2\sum_{k=1}^{n-1}\sqrt{kh}\ln(kh)\right) \tag{0–8}$$

将 $f(x)$ 的导数结果 (0–4) 带入公式 (0–7) 中，得到 $f(x)$ 复化辛普森公式的误差函数：

$$R_n(f) = \frac{h^4}{2880}\eta^{-\frac{7}{2}}(\frac{15}{16}\ln\eta - 1) \tag{0–9}$$

1.3 与积分精确值比较复化梯形公式和复化辛普森公式的精度

取区间等分份数 n 为 2 的指数倍，进行仿真，得到复化梯形公式和复化辛普森公式的精度对比表如下：

表 0–1 复化梯形公式和复化辛普森公式的精度比较表
(精确值 $\int_0^1 \sqrt{x}\ln x\, dx = -\frac{4}{9} = -0.444444\cdots$)

		复化梯形公式		复化辛普森公式	
区间等分份数 n	步长 h	求积结果	误差	求积结果	误差
1	1	-6.90775×10^{-9}	0.444444	-0.326752	0.117691
2	0.5	-0.245064	0.199379	-0.395783	0.048660
4	0.25	-0.358104	0.086340	-0.424752	0.019692
8	0.125	-0.408090	0.036354	-0.436602	7.841677×10^{-3}
16	6.25×10^{-2}	-0.429474	0.014969	-0.441361	3.083324×10^{-3}
32	3.125×10^{-2}	-0.438389	6.054958×10^{-3}	-0.443244	1.200028×10^{-3}
64	1.5625×10^{-2}	-0.442030	2.413760×10^{-3}	-0.443981	4.631324×10^{-4}
128	7.8125×10^{-3}	-0.443493	9.507894×10^{-4}	-0.444266	1.774805×10^{-4}
256	3.90625×10^{-3}	-0.444073	3.708077×10^{-4}	-0.444376	6.760613×10^{-5}
512	1.953125×10^{-3}	-0.444301	1.434065×10^{-4}	-0.444418	2.561969×10^{-5}
1024	9.765625×10^{-4}	-0.444389	5.506639×10^{-5}	-0.444434	9.665089×10^{-6}

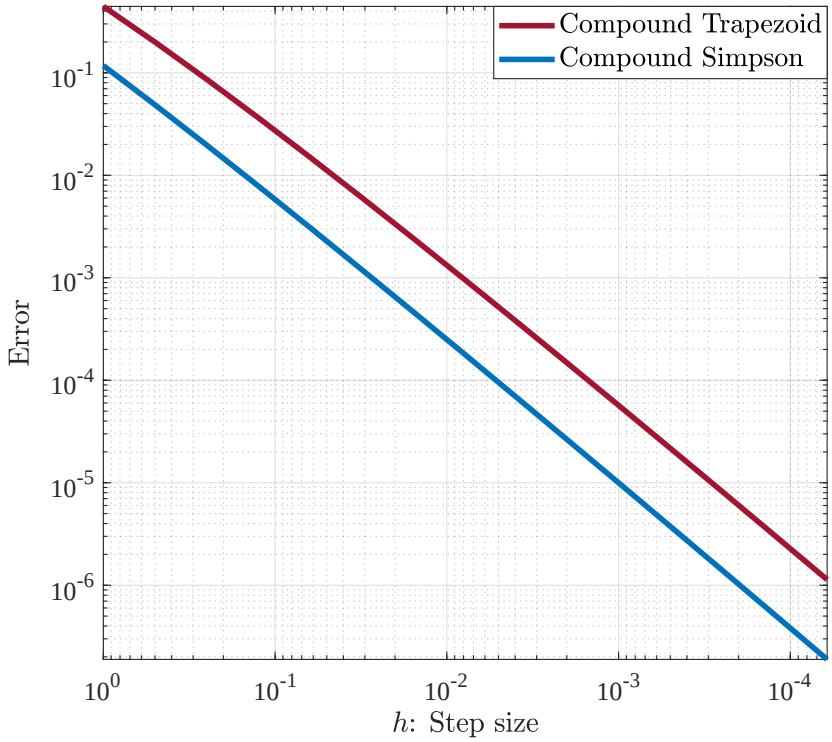

图 0–1 复化梯形公式和复化辛普森公式的误差随 h 变化表

为了更形象对比两种方法的误差，上表可以可视化为下图：

在 h 相等的条件下，复化辛普森公式的误差约为复化梯形公式的 $\frac{1}{5}$(仅从仿真结果看，不具有理论依据)，说明复化辛普森公式在计算积分 $\int_0^1 \sqrt{x}\ln x\,dx = -\frac{4}{9}$ 时精度相对更好。

1.4 探究：是否存在一个最小的 h，使得精度不能再被改善？

从表 (0–2) 和图 (0–1) 的结果来看，随着 h 的减小，复化梯形公式和复化辛普森公式的误差均可以有效降低。所以不存在一个最小的 h，使得精度不能再被改善。

然而，因为在计算机计算中，精度有限，所以在有限的精度下，复化梯形公式和复化辛普森公式的误差曲线均会与图 (0–1) 不同，因为舍入误差而出现拐点。此处，我们选择的精度限制方法如下：

$$Round(f(x)\times 100)/100$$

经过仿真获得的新的误差关于 h 的曲线如图 (0–2)。

根据图中曲线所示，我们得到的估计的 h' 的范围如下表：

表 0–2 复化梯形公式和复化辛普森公式的 h' 范围表

	复化梯形公式	复化辛普森公式
h' 范围	$[2.441406\times10^{-4}, 9.765625\times10^{-4}]$	$[3.051758\times10^{-5}, 4.882813\times10^{-4}]$

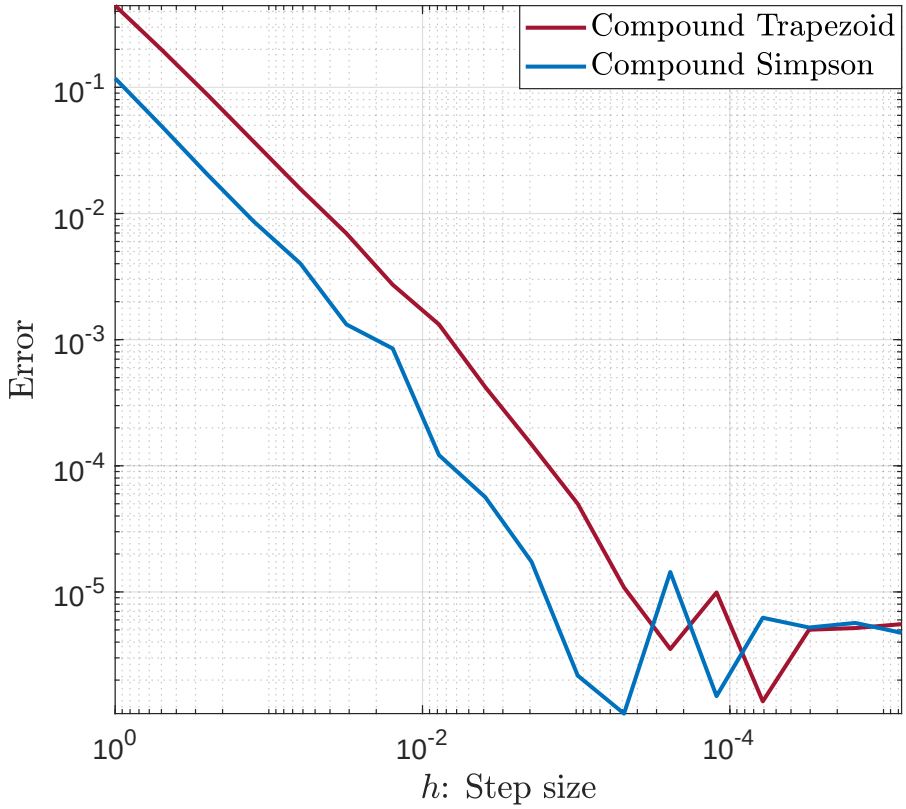

图 0–2 考虑舍入误差后复化梯形公式和复化辛普森公式的误差随 h 变化表

2. 用变步长梯形公式计算积分，使其误差小于 10^{-4}

1.1 节介绍的梯形公式可以有效提高求积精度，但是在实际计算时若精度不够可以将步长逐次分半。设将区间 $[a,b]$ 分为 n 等份，共有 $n+1$ 个分点，如果将求积区间再二分一次，则分店增至 $2n+1$ 个，我们将二分前后两个积分值联系起来加以考察，注意到每个子区间 $[x_k, x_{k+1}]$ 经过二分只增加了一个分点 $x_{k+\frac{1}{2}} = \frac{1}{2}(x_k + x_{k+1})$。所以 T_{2n} 的表达式为：

$$T_{2n} = \frac{h}{4}\sum_{k=0}^{n-1}[f(x_k) + f(x_{k+1})] + \frac{h}{2}\sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) \tag{0–10}$$

其中 $h = \frac{b-a}{n}$。由此可以得到递推公式为

$$T_{2n} = \frac{1}{2}T_n + \frac{h}{2}\sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) \tag{0–11}$$

仿真误差设置为 10^{-4}，得到的仿真结果为-0.444389378040032，误差为 $5.50663999154821 \times 10^{-5}$，此时 $n = 1024$，$h = 9.765625 \times 10^{-4}$。此结果与表 (0–2) 保持一致。

3. 用 Romberg 公式计算积分，使其误差小于 10^{-4}

龙贝格求积公式也称为逐次分半加速法。是数值计算方法之一，用以求解数值积分。是在梯形公式、辛普森公式和柯特斯公式之间关系的基础上，构造出一种加速计算积分的方法。作为一种外推算法，在不增加计算量的前提下提高了误差的精度。

设以 $T_0^{(k)}$ 表示二分 k 次后求得的梯形值，且以 $T_m^{(k)}$ 表示序列 $\{T_0^{(k)}\}$ 的 m 次加速值，则递推公式可得：

$$T_m^{(k)} = \frac{4^m}{4^m-1}T_{m-1}^{(k+1)} - \frac{1}{4^m-1}T_{m-1}^{(k)}, \quad k = 1, 2, \cdots \tag{0–12}$$

龙贝格求积算法计算过程如下：

1. 取 $k = 0$，$h = b - a$，求 $T_0^{(0)} = \frac{h}{2}[f(a) + f(b)]$。
2. 令 $1 \longrightarrow k$(k 记为区间 $[a, b]$ 的二分次数)。
3. 求梯形值 $T_0\left(\frac{b-a}{2^k}\right)$，即按递推公式 (0–11) 计算 $T_0^{(k)}$。
4. 求加速值，按公式 (0–12) 逐个求出 T 表的第 k 行其余各元素 $T_j^{(k-j)}(j = 1, 2, \cdots, k)$。
5. 若 $|T_k^{(0)} - T_{k-1}^{(0)}| < \varepsilon$，则终止计算，并取 $T_k^{(0)} \simeq I$，否则令 $k + 1 \simeq k$ 转第三步继续计算。

在仿真中，我们设置误差需低于 10^{-4}，得到下面的 Romberg 算法的 T 表。

表 0–3 Romberg 算法的 T 表 (限于页面宽度，保留 5 位有效数字)

k	$T_0^{(k)}$	$T_1^{(k)}$	$T_2^{(k)}$	$T_3^{(k)}$	$T_4^{(k)}$	$T_5^{(k)}$	$T_6^{(k)}$	$T_7^{(k)}$	$T_8^{(k)}$	$T_9^{(k)}$	$T_{10}^{(k)}$
0	-0.00000										
1	-0.24506	-0.32675									
2	-0.35810	-0.36564	-0.36625								
3	-0.40809	-0.41142	-0.41214	-0.41232							
4	-0.42947	-0.43090	-0.43120	-0.43128	-0.43130						
5	-0.43838	-0.43898	-0.43911	-0.43914	-0.43915	-0.43915					
6	-0.44203	-0.44227	-0.44232	-0.44233	-0.44234	-0.44234	-0.44234				
7	-0.44349	-0.44359	-0.44361	-0.44361	-0.44361	-0.44361	-0.44361	-0.44361			
8	-0.44407	-0.44411	-0.44412	-0.44412	-0.44412	-0.44412	-0.44412	-0.44412	-0.44412		
9	-0.44430	-0.44432	-0.44431	-0.44432	-0.44432	-0.44432	-0.44432	-0.44432	-0.44432	-0.44432	
10	-0.44439	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440	-0.44440

因为 Romberg 的 T 表中，每一列元素及对角线元素均收敛到所求的积分值 I，所以我们将上表的第一列元素和对角线元素与准确值的误差可视化，得到图 (0–3)。

根据 T 表和可视化图的显示，我们可以看到，Romberg 算法可以有效收敛到准确值。在第 11 次递推之后，Romberg 算法结果 $T_{10}^{(10)}$ 与准确值 $-\frac{4}{9} = -0.44444\cdots$ 的误差小于 10^{-4}，达到预期效果，且第一列至第九列结果均收敛至准确值，验证了课本 P113 页结论：如果 $f(x)$ 充分光滑，那么 T 表的每一列的元素及对角线元素均收敛到所求的积分值 I，即：

$$\lim_{k\to\infty} T_m^{(k)} = I$$

$$\lim_{m\to\infty} T_m^{(0)} = I$$

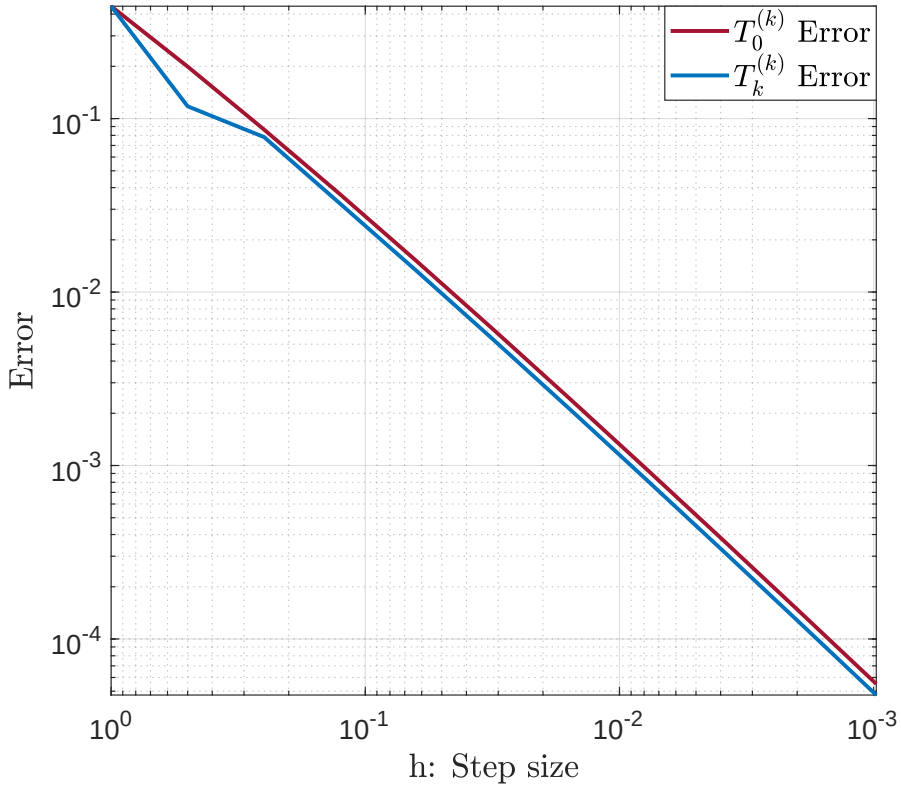

图 0–3 Romberg 算法第一列和对角线值与准确值的误差随 h 变化图

4. 用自适应辛普森公式计算积分，使其误差小于 10^{-4}

复化求积算法通常适用于被积函数变化不太大的积分，如果在求积区间中被积函数变化很大，这时统计将区间等分，用复合求积公式计算积分工作量大，因为要达到误差要求对变化剧烈部分必须将区间细分。针对此类问题的算法技巧是在不同区间上预测被积函数变化的剧烈程度确定相应步长，这种方法称为自适应方法。自适应方法应用于复化辛普森公式中得到的称为自适应辛普森算法。

具体地，假设 $S(a,b)$ 表示 $\int_a^b f(x)\,dx = \int_a^b \sqrt{x}\ln x\,dx$，用 $I(f)$ 表示精确值。则自适应辛普森算法的流程如下：

1. 如果 $|I(f) - S(a,b)| < \varepsilon$，则算法结束；否则算法继续。
2. 用复化辛普森公式近似计算 $S(a, \frac{a+b}{2})$ 和 $S(\frac{a+b}{2}, b)$，进行递归。

经过仿真计算得到，自适应辛普森公式得到的计算结果为：-0.444440799763923，误差为 3.644680×10^{-6}。因为自适应辛普森算法的误差是近似计算的，所以设定 $\varepsilon = 10^{-4}$ 并不能保证一定可以搜索到误差最接近 ε 的结果，因此这里的误差要小于 10^{-4}。

5. 比较上述 5 种求积公式的计算效率

为比较 5 种求积公式的计算效率，我们设定相等的计算误差，即 $\varepsilon < 10^{-4}$，统计五种方法达到该计算精度下的计算复杂度。计算复杂度的统计方式，按照 $f(x)$ 参与计算的次数为量度。下面是在 MATLAB 程序中的复杂度统计结果：

表 0–4 5 种求积公式计算复杂度统计表

复化梯形公式	复化辛普森公式	变步长梯形公式	Romberg 算法	自适应辛普森算法
389	667	1024	512	147

由上表可以得知，

1. 求积效率顺序：自适应辛普森算法 > 复化梯形公式 >Romberg 算法 > 复化辛普森公式 > 变步长梯形公式。
2. Romberg 算法是基于变步长梯形公式的外推公式，有效提高收敛速率。
3. 虽然复化梯形公式和复化辛普森公式相对复杂度较低，但是由于无法在一开始得知步长与误差的关系，所以不够灵活。
4. 自适应辛普森算法因为可以自适应地在不同的区间上采用不同的步长，可以有效提高计算效率。

代码附录

Main 函数

```
%% ----- Definition ----- %
syms x;
f      = sqrt(x).*log(x);
result = -4/9                    ;
eps    = 1e-20                   ;

%% ----- Test of Simpson and Trapezoid formula ----- %
num                    = 15           ;
n_arr                  = 2 .^ (0:num-1) ;
h_arr                  = 1 ./ n_arr   ;
compound_simpson_res   = zeros(1, num);
compound_simpson_err   = zeros(1, num);
compound_trapezoid_res = zeros(1, num);
compound_trapezoid_err = zeros(1, num);

for i = 1 : num
    n                          = n_arr(i)
;
    % ----- compound simpson results ----- %
    compound_simpson_res(i)    = compound_simpson(f, x, eps, 1, n)
;
    compound_simpson_err(i)    = abs(result - compound_simpson_res(i))
;
    % ----- compound trapezoid results ----- %
    compound_trapezoid_res(i) = compound_trapezoid(f, x, eps, 1, n)
;
    compound_trapezoid_err(i) = abs(result - compound_trapezoid_res(i));
end
figure;
loglog(h_arr, compound_trapezoid_err, 'ro-');
hold on
loglog(h_arr, compound_simpson_err, 'ko-')  ;
axis on;
grid on;
axis tight;
```

```
set(gca,'XDir','reverse');
xlabel('h:_Step_size');
ylabel('Error');
legend( ...
        'Compound_Trapezoid', ...
        'Compound_Simpson', ...
        'Times_New_Roman', ...
        'Southwest' ...
);

%% ----- Step-changing Trapezoid ----- %
[step_changing_trapezoid_T, step_changing_trapezoid_step] = step_changing_trapez
step_num_step_changing_trapezoid = 1 ./ step_changing_trapezoid_step;

%% ----- Romberg ----- %
[T_ii, T] = Romberg(f, x, eps, 1, 1e-4, result);
iter                            = size(T, 1);
compound_trapezoid_res_Romberg = T(:,1)     ;
Romberg_res                     = diag(T)    ;
Trapezoid_err                   = abs(result .* ones(iter, 1) - compound_trapezoi
Romberg_err                     = abs(result .* ones(iter, 1) - Romberg_res);

figure;
x_label = 1:iter;
loglog(x_label, Trapezoid_err, 'ro-');
hold on
loglog(x_label, Romberg_err, 'ko-')  ;
axis on;
grid on;
axis tight;
% set(gca,'XDir','reverse');
xlabel('h:_Step_size');
ylabel('Error');
legend( ...
        'T_0^{(k)}_Error', ...
        'T_k^{(k)}_Error'...
);

%% ----- Self-adaptive Simpson ----- %
[complexity_simpson, y] = self_adaptive_simpson(f, x, eps, 1, 1e-4, 0);
```

复化梯形公式: compound_trapezoid.m

```
function Tn = compound_trapezoid(f, x, a, b, n)
    h = (b - a) ./ n;
    k = 0 : n-1;
    xk = a + k .* h;
    Tn = h/2 * (subs(f, x, a) + subs(f, x, b)) + h .* sum(round(subs(f, x, xk)*1
end
```

复化辛普森公式: compound_simpson.m

```
function Sn = compound_simpson(f, x, a, b, n)
    h = (b - a) ./ n;
    k = 0 : n-1;
    f1 = subs(f, x, a + k*h + 1/2*h);
    f2 = subs(f, x, a + k*h);
    Sn = h/6 .* (subs(f, x, a) + subs(f, x, b) + 4 * sum(round(f1*100)/100) + 2
end
```

变步长梯形公式: step_changing_trapezoid.m

```
function [T,h] = step_changing_trapezoid(f, x, a, b, e)
    h = b - a;
    T1 = h * (subs(f, x, a) + subs(f, x, b)) / 2;
    T2 = T1 / 2 + h / 2 * subs(f, x, a + h / 2);
    while abs(T2 - T1) >= e
        h = h / 2;
        T1 = T2;
        S = 0;
        x_i = a + h / 2;
        while x_i < b
            S = S + subs(f, x, x_i);
            x_i = x_i + h;
        end
        T2 = T1 / 2 + S * h / 2;
    end
    T = T2;
end
```

龙贝格算法: Romberg.m

```
function [y, T] = Romberg(f, x, a, b, e, res)
    T = 0;
    h = b - a;
    T(1,1) = h/2 * (subs(f, x, a) + subs(f, x, b));
    T(2,1) = compound_trapezoid(f, x, a, b, 2);
    T(2,2) = 4/3 * T(2,1) - 1/3 * T(1,1);
    i = 2;
    % while (abs(T(i,i) - T(i-1, i-1)) >= e)
    while (abs(T(i,i) - res) >= e)
        i = i + 1;
        T(i,1) = compound_trapezoid(f, x, a, b, 2^(i-1));
        for j = 2 : i
            T(i,j) = 4^j / (4^j-1) * T(i,j-1) - 1/(4^j-1) * T(i-1, j-1);
        end

    end
    y = T(end,end);
end
```

自适应辛普森算法: self_adaptive_simpson.m

```
function [complexity_cur,y] = self_adaptive_simpson(f, x, a, b, e, complexity)
  h = (b - a);
  S = h / 6 * (subs(f, x, a) + subs(f, x, a) + 4 * subs(f, x, (a+b)/2));
  h = h / 2;
  S1 = h / 6 * (subs(f, x, a) + subs(f, x, a+h) + 4 * subs(f, x, a+h/2));
  S2 = h / 6 * (subs(f, x, a+h) + subs(f, x, b) + 4 * subs(f, x, a+h*3/2));
  complexity_cur = complexity + 3;
  if abs(S - S1 - S2) < e
    y = S1 + S2 + (S1 + S2 - S) / 15;
  else
    [complexity_cur_1, y1] = self_adaptive_simpson(f, x, a, (a+b)/2, e/2, comple
    [complexity_cur_2, y2] = self_adaptive_simpson(f, x, (a+b)/2, b, e/2, comple
    y = y1 + y2;
    complexity_cur = complexity_cur_1 + complexity_cur_2;
  end
end
```

致　谢

在本次大作业中，感谢助教学长对于我代码和数学公式方面的帮助，还有一起学习的小伙伴们的讨论与探索。

非常荣幸能够选择本学期计算方法课程，曾老师的讲课浅显易懂，深入浅出，对我的科研和学习生活帮助很大。再次感谢曾老师的谆谆教导。

最后，辛苦助教学长审阅作业。