E06c 编程作业解答

姓名：范舟　　学号：516030910574

注意：

1. 程序在文档中也要粘贴，同时把代码和该文档放在同一个文件夹中打包发给我.（建议多个同学或整个班级一起打包；邮箱：terenceyuyue@sjtu.edu.cn）
2. 该文档不需打印，只收电子版.

问题：　用不同数值方法计算积分 $\int_0^1 \sqrt{x}\ln x\mathrm{d}x = -\frac{4}{9}$.

1　复化求积方法

1.1　编写复化梯形公式的函数文件，命名为 Trapezoidal.m，画出最大模误差与步长 h 的函数图像.

注：计算中涉及到的求和可通过 sum 命令完成，避免循环.

下面是 Trapezoidal 函数的实现，其中参数 f 是被积函数的函数句柄，a 与 b 为积分上下界，n 为等分的小区间数目.

```matlab
function ret = Trapezoidal(f, a, b, n)
    x = linspace(a, b, n + 1);
    h = (b - a) / n;
    fx = arrayfun(f, x);
    ret = h / 2. * (fx(1) + 2. * sum(fx(2 : n)) + fx(n + 1));
end
```

下面的程序（test_room.m）调用 Trapezoidal 函数计算题目中的积分，并计算步长 h 取不同值时的最大模误差，并画出图像. 这段程序也用于调用后面编写的其他积分函数进行测试与绘图.

```matlab
a = 0; b = 1;
n_arr = 1 : 300;
I = -4 / 9;

% 调用 Trapezoidal, Simpson, Romberg 函数进行测试
d = abs(arrayfun(@(n_) Trapezoidal(@ex1_fun, a, b, n_), n_arr) - I);
% d = abs(arrayfun(@(n_) Simpson(@ex1_fun, a, b, n_), n_arr) - I);
% d = abs(arrayfun(@(n_) Romberg(@ex1_fun, a, b, n_), n_arr) - I);
h = (b - a) ./ n_arr;

```

```
% 画出最大模误差与步长 h 的函数图像
plot(h, d, 'm-o');
% plot(n_arr, d, 'b-x');

% 调用 AdaptSimpson 函数进行测试
% approx_I = AdaptSimpson(@ex1_fun, a, b, 10^(-4));
% fprintf("%.8f %.8f\n", approx_I, abs(approx_I - I));

% 题目中的被积函数
function y = ex1_fun(x)
    if x == 0
        y = 0;
    else
        y = sqrt(x) * log(x);
    end
end
```

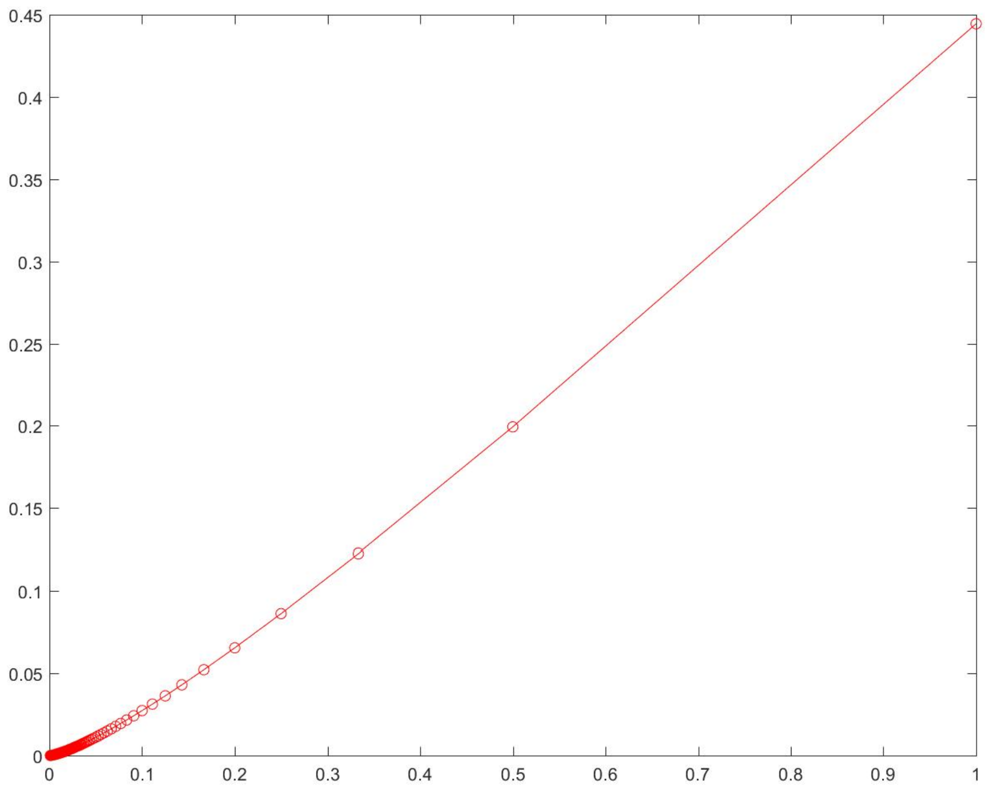

图 1: 使用复化梯形公式的最大模误差与步长 h 的函数图像

1.2 编写复化 Simpson 公式的函数文件，命名为 Simpson.m，画出最大模误差与步长 h 的函数图像.

```
function ret = Simpson(f, a, b, n)
    x = linspace(a, b, 2 * n + 1);
```

```
    h = (b - a) / n;
    fx = arrayfun(f, x);
    ret = h / 6. * (sum(fx(1 : 2 : 2 * n - 1)) + sum(fx(3 : 2 : 2 * n + 1)) ...
        + 4. * sum(fx(2 : 2 : 2 * n)));
end
```

使用 test_room.m 调用 Simpson 函数计算题目中的积分，并计算步长 h 取不同值时的最大模误差，画出图像如下.

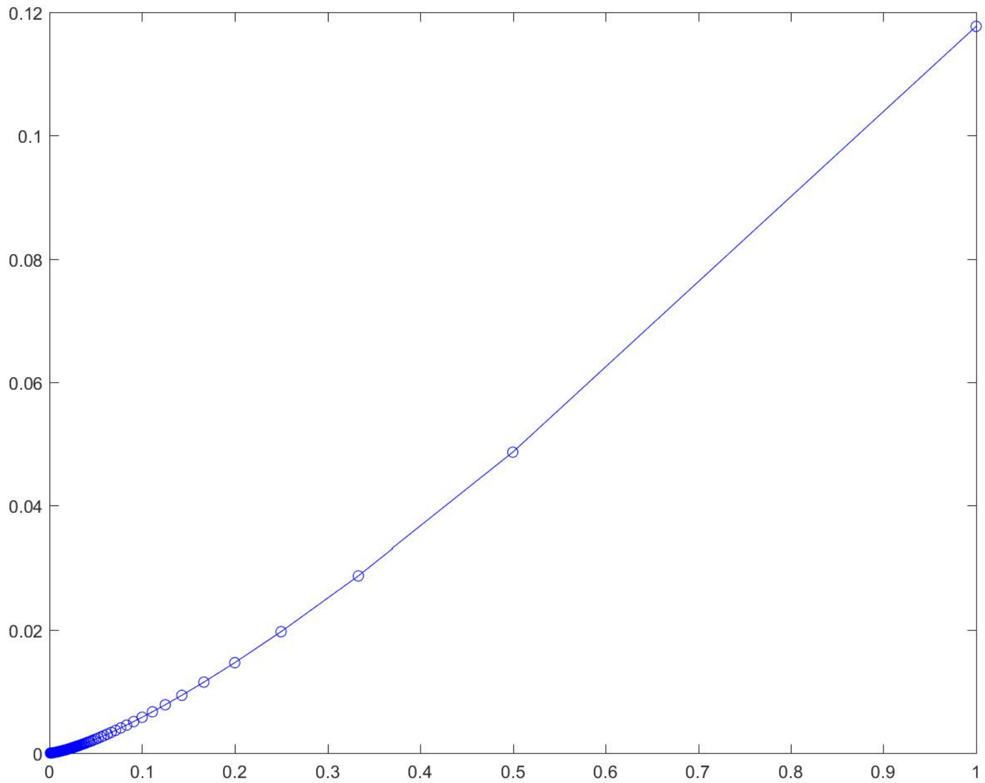

图 2: 使用复化 Simpson 公式的最大模误差与步长 h 的函数图像

1.3 比较两个公式的精度，回答问题：是否存在一个最小的 h，使得精度不能再被改善？

答： 由上面两个公式的最大模误差与步长 h 的函数图像可以明显看出，复化 Simpson 公式的精度比复化梯形公式高很多，这是由于复化梯形公式的余项为

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

而复化 Simpson 公式的余项为

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

根据余项公式，可知误差随 h 减小而减小，精度随 h 减小而提高，但在实际的计算中由于浮点数表示的精度有限，精度不会无限度提升. 当精度已经达到机器精度时，继续减小 h 也不能再改善精度.

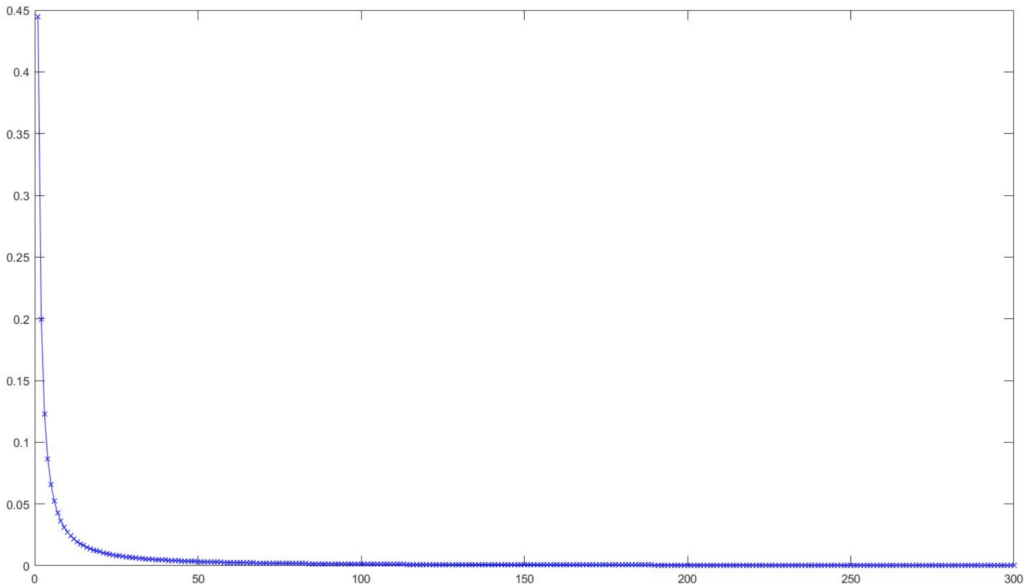

图 3: 使用复化梯形公式的最大模误差与二分次数 n 的函数图像

2 Romberg 求积方法

2.1 编写 Romberg 公式的函数文件，命名为 Romberg.m，画出最大模误差与步长的函数图像.

下面实现的 Romberg 函数同时接受二分次数 n 和所要求的精度 eps 两个参数（其中 eps 的缺省值为 0），如果二分次数达到 n 或所得结果的误差已经小于 eps，则终止计算.

```
function ret = Romberg(f, a, b, n, eps)
   if nargin < 5
       eps = 0;
   end
   T = zeros(n + 1, n + 1);
   h = b - a;
   T(1, 1) = h / 2. * [f(a) + f(b)];
   for k = 1 : n
       h = h / 2.;
       T(1, k + 1) = T(1, k) / 2.;
       for x = a + h : h * 2. : b - h
           T(1, k + 1) = T(1, k + 1) + h * f(x);
       end
       for i = 1 : k
           T(i + 1, k - i + 1) = (4^i * T(i, k - i + 2) - T(i, k - i + 1)) / (4^i - 1);
       end
```

```
        if abs(T(k + 1, 1) - T(k, 1)) < eps
            ret = T(k + 1, 1);
            break;
        end
        if k == n
            ret = T(k + 1, 1);
        end
    end
end
```

使用 test_room.m 调用 Romberg 函数计算题目中的积分，并计算步长 h 取不同值时的最大模误差，画出图像如下.

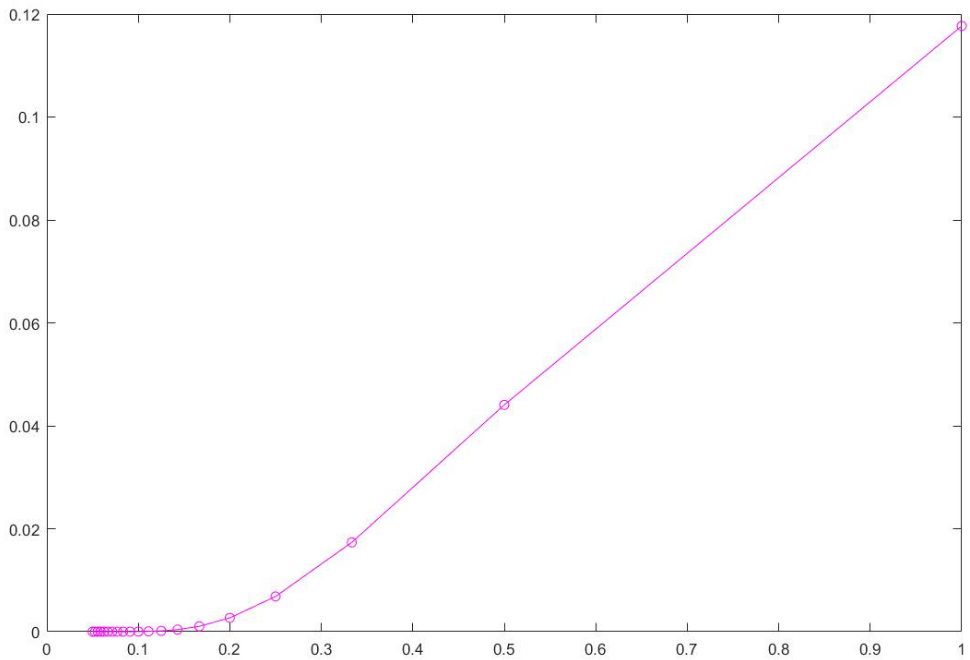

图 4: 使用 Romberg 公式的最大模误差与步长 h 的函数图像

3 自适应 Simpson 求积方法

3.1 说明如何处理二分过程中树状图扩展的问题.

将自适应 Simpson 积分实现为递归函数 AdaptSimpson，在函数中计算当前区间上的积分，按照停机准则判断，若需要对当前区间进一步二分，则分别对二分得到的两个小区间调用这个递归函数，便可实现自适应 Simpson 积分中的树状二分扩展过程. 使用时只需要对整个区间 $[a, b]$ 调用一次 AdaptSimpson 函数即可.

3.2 编写自适应的程序，停止准则为精度达到 10^{-4}，程序命名为 AdaptSimpson.m. 注：可能的话，“打印”出区间二分的详细情形.

下面实现的 AdaptSimpson 函数即为上面所述的递归函数，计算 $[a, b]$ 区间上函数 f 的积分，要求精度为 eps.

```
function ret = AdaptSimpson(f, a, b, eps)
    mid = (a + b) / 2.;
    s1 = SimpleSimpson(f, a, b);
    s2 = SimpleSimpson(f, a, mid) + SimpleSimpson(f, mid, b);
    if abs(s1 - s2) < 15. * eps
        ret = s2 + 1. / 15. * (s2 - s1);
    else
        % 输出区间二分的情形
        fprintf("Interval Split: [%.8f, %.8f] -> [%.8f, %.8f], [%.8f, %.8f] \n",...
            a, b, a, mid, mid, b);
        ret = AdaptSimpson(f, a, mid, eps / 2.) + ...
            AdaptSimpson(f, mid, b, eps / 2.);
    end
end

function ret = SimpleSimpson(f, a, b)
    ret = (b - a) / 6 * (f(a) + 4. * f((a + b) / 2.) + f(b));
end
```

在 test_room.m 中用下面的代码调用 AdaptSimpson 函数计算题目中的积分，得到积分结果为 -0.44444323，最大模误差为 0.00000121，符合精度要求.

```
approx_I = AdaptSimpson(@ex1_fun, a, b, 10^(-4));
fprintf("%.8f %.8f\n", approx_I, abs(approx_I - I));
```

在 AdaptSimpson 函数中输出每次区间划分的情形，得到如下输出结果：

```
Interval Split: [0.00000000, 1.00000000] -> [0.00000000, 0.50000000], [0.50000000, 1.00000000]
Interval Split: [0.00000000, 0.50000000] -> [0.00000000, 0.25000000], [0.25000000, 0.50000000]
Interval Split: [0.00000000, 0.25000000] -> [0.00000000, 0.12500000], [0.12500000, 0.25000000]
Interval Split: [0.00000000, 0.12500000] -> [0.00000000, 0.06250000], [0.06250000, 0.12500000]
Interval Split: [0.00000000, 0.06250000] -> [0.00000000, 0.03125000], [0.03125000, 0.06250000]
Interval Split: [0.00000000, 0.03125000] -> [0.00000000, 0.01562500], [0.01562500, 0.03125000]
Interval Split: [0.00000000, 0.01562500] -> [0.00000000, 0.00781250], [0.00781250, 0.01562500]
Interval Split: [0.00000000, 0.00781250] -> [0.00000000, 0.00390625], [0.00390625, 0.00781250]
Interval Split: [0.00000000, 0.00390625] -> [0.00000000, 0.00195313], [0.00195313, 0.00390625]
Interval Split: [0.00000000, 0.00195313] -> [0.00000000, 0.00097656], [0.00097656, 0.00195313]
Interval Split: [0.00000000, 0.00097656] -> [0.00000000, 0.00048828], [0.00048828, 0.00097656]
Interval Split: [0.00000000, 0.00048828] -> [0.00000000, 0.00024414], [0.00024414, 0.00048828]
Interval Split: [0.00000000, 0.00024414] -> [0.00000000, 0.00012207], [0.00012207, 0.00024414]
Interval Split: [0.00000000, 0.00012207] -> [0.00000000, 0.00006104], [0.00006104, 0.00012207]
Interval Split: [0.00000000, 0.00006104] -> [0.00000000, 0.00003052], [0.00003052, 0.00006104]
```