

# 第五章作业

Anonymity 3220100000\*

\* College of Control Science and Engineering, Robotics Engineering

July 8, 2024

## 1 问题描述

假设通过某电阻器的电流可以表示为函数：

$$i(t) = (60 - t)^2 + (60 - t) \sin(\sqrt{t})$$

并且电阻是电流的函数，

$$R(t) = 10i(t) + 2i(t)^{2/3}$$

采用不同的数值积分方法计算  $t = 0$  到  $60$  之间的平均电压，并分析误差。

## 2 思路分析

要计算  $t = 0$  到  $t = 60$  之间的平均电压，使用以下积分表达式：

$$V_{avg} = \frac{1}{T} \int_0^T V(t) dt$$

其中  $T$  是电流的周期，即  $T = 60$ 。

电压  $V(t)$  由以下公式给出：

$$V(t) = i(t) \cdot R(t)$$

将  $i(t)$  和  $R(t)$  的表达式代入，得：

$$V(t) = [(60-t)^2 + (60-t) \sin(\sqrt{t})] \cdot \left[ 10((60-t)^2 + (60-t) \sin(\sqrt{t})) + 2((60-t)^2 + (60-t) \sin(\sqrt{t}))^{2/3} \right]$$

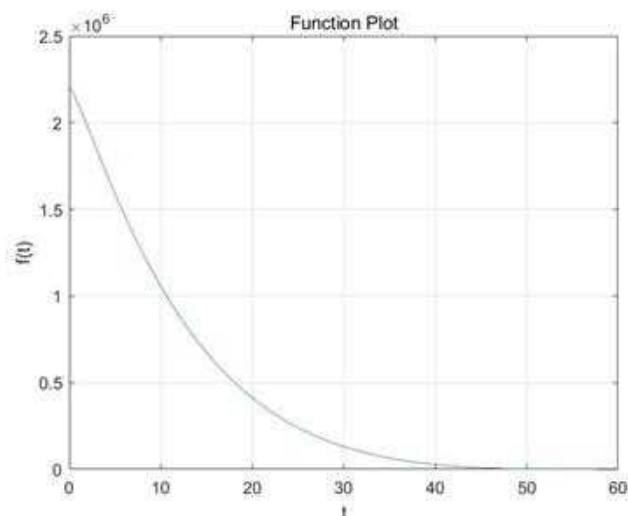
将  $V(t)$  代入平均电压的积分表达式，然后计算积分：

$$V_{avg} = \frac{1}{60} \int_0^{60} [(60-t)^2 + (60-t) \sin(\sqrt{t})] \cdot \left[ 10((60-t)^2 + (60-t) \sin(\sqrt{t})) + 2((60-t)^2 + (60-t) \sin(\sqrt{t}))^{2/3} \right] dt$$

首先绘制函数

$$f(t) = \frac{1}{60} [(60-t)^2 + (60-t) \sin(\sqrt{t})] \cdot \left[ 10((60-t)^2 + (60-t) \sin(\sqrt{t})) + 2((60-t)^2 + (60-t) \sin(\sqrt{t}))^{2/3} \right]$$

在积分区间  $[0, 60]$  内的函数图像：



再采用 *matlab* 内置函数 *integral* 先求解准确解为 26361215.183948，再将其与其他方法求到的解进行对比。

#### 命令行窗口

```
% 求解积分
V_avg = 1/60 * integral(fun, 0, 60);

% 将结果格式化为6位小数
result_str = sprintf('%0.6f', V_avg);

disp(['平均电压准确值是: ', result_str]);
平均电压准确值是: 26361215.183948
fx >>
```

## 2.1 梯形公式

最简单的梯形公式是仅取原函数的积分区间的积分上限和积分下限进行求解，表达式为：

$$I_1(f) = (b-a)\frac{1}{2}f(a) + (b-a)\frac{1}{2}f(b) = (b-a)\frac{f(a)+f(b)}{2}$$

这种方式仅具有一阶精度，即当原函数为线性时无误差，或者函数图像接近线性时误差较小。

## 2.2 复合梯形公式

复合梯形公式又称 *Newton - Cotes* 积分，是在梯形公式的基础上，将积分区间多等分，而后进行多组梯形公式的计算相加，表达式为：

$$T_n = \int_a^b f(x)dx \approx \frac{h}{2} \left[ f(a) + f(b) + 2 \sum_{k=1}^{n-1} f(x_k) \right] \quad h = \frac{b-a}{n}, x_k = a + kh, (k = 0, 1, \dots, n)$$

原理是由于梯形公式仅具有一阶精度，因此在等分足够精度后，函数图像会近似于线性，从而减小误差。

### 2.3 Simpson 公式

Simpson 公式是将区间端点和区间中点三个点近似看成抛物线上对应的三个点，以二次曲线逼近的方式取代矩形或梯形积分公式，以求得定积分的数值近似解，表达式为：

$$I(f) = \int_a^b f(x)dx \approx (b-a) \left[ \frac{1}{6}f(a) + \frac{4}{6}f\left(\frac{a+b}{2}\right) + \frac{1}{6}f(b) \right]$$

其原理是在每一段的两端和中点处的值近似为抛物线，逐段积分后加起来，即得到原定积分的数值解。

### 2.4 复合 Simpson 公式

复合 Simpson 公式之于 Simpson 公式，类似于复合梯形公式之于梯形公式，只是用同样的积分方式，但是将积分区间细分化，总而提高精度的一种方式，其表达式为：

$$S_n = \sum_{k=0}^{n-1} = \frac{1}{6}H \left[ f(a) + 4 \sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right) + 2 \sum_{k=1}^{n-1} f(x_k) + f(b) \right]$$

### 2.5 高阶 Newton - Cotes 公式

对于  $n$  阶的 Newton - Cotes 公式的一般形式。

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

称  $C_k$  为 Cotes 系数。

因此：

$$\int_a^b f(x)dx \cong (b-a) \sum_{k=0}^n C_k f(x_k)$$

将被积区间划分成许多小区间，每个小区间对应的原函数近似为一个高阶多项式，对此高阶多项式进行积分，结果准确性高，但因为实际计算需要很大的计算储存空间以及计算量太大，导致应用范围有限。

### 2.6 Romberg 积分

Romberg 积分的具体步骤为：

1. 在区间  $[a, b]$  上， $h = (b-a)$ ，计算分点的函数值  $f(a)$  和  $f(b)$  后，用梯形求积公式计算  $T_1$ ：

$$T_1 = \frac{b-a}{2} [f(a) + f(b)]$$

2. 将区间  $[a, b]$  二分,  $h = (b - a)/2$ , 计算新增分点的函数值  $f\left(\frac{a+b}{2}\right)$  后, 用  $T_{2n} = \frac{1}{2} T_n + \frac{1}{2} H \sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)$  计算  $T_2$  :

$$T_2 = \frac{1}{2} T_1 + h \sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)$$

用  $S_n = \frac{4}{3} T_{2n} - \frac{1}{3} T_n$  计算  $S_1$  :

$$S_1 = \frac{4}{3} T_2 - \frac{1}{3} T_1$$

3. 将区间  $[a, b]$  再二分,  $h = (b - a)/4$ , 计算新增分点  $f\left(a + \frac{a+b}{4}\right)$  和  $f\left(a + \frac{3}{4}(a + b)\right)$  后, 用  $T_{2n} = \frac{1}{2} T_n + \frac{1}{2} H \sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)$  式计算  $T_4$ , 用  $S_n = \frac{4}{3} T_{2n} - \frac{1}{3} T_n$  计算  $S_2$ , 用  $C_n = \frac{16}{15} S_{2n} - \frac{1}{15} S_n$  计算  $C_1$  :

$$C_1 = \frac{16}{15} S_2 - \frac{1}{15} S_1$$

4. 将区间  $[a, b]$  再二分,  $h = (b - a)/8$ , 计算新增分点的函数值后, 用  $T_{2n} = \frac{1}{2} T_n + \frac{1}{2} H \sum_{k=0}^{n-1} f\left(x_{k+\frac{1}{2}}\right)$  计算  $T_8$ , 用  $S_n = \frac{4}{3} T_{2n} - \frac{1}{3} T_n$  计算  $S_4$ , 用  $C_n = \frac{16}{15} S_{2n} - \frac{1}{15} S_n$  计算  $C_2$ , 用  $R_n = \frac{64}{63} C_{2n} - \frac{1}{63} C_n$  计算  $R_1$  :

$$R_1 = \frac{64}{63} C_2 - \frac{1}{63} C_1$$

5. 将区间  $[a, b]$  再二分,  $h = (b - a)/16$ , 计算新增分点的函数值后, 计算  $T_{16}, S_8, C_4, R_2$ 。

6. 将区间  $[a, b]$  继续二分下去, 可得 *Romberg* 序列  $R_1, R_2, R_4, \dots, R_n, R_{2n}$  直至满足精度要求  $|R_{2n} - R_n| \leq \epsilon$  为止。

由于加速过程中不需要再计算函数值, 所以加速过程的计算量非常小, 可以忽略不计, 而加速效果则是十分显著的。

总而言之, 该方法通过若干个积分近似值来推算更精确的新的近似值, 从而提高计算结果准确度, 结果准确性高, 但因为要求已知  $f(x)$  表达式, 所以应用范围受限, 不适用于列表型数据。

## 2.7 Gauss 求积公式

*Gauss* 求积公式是指通过选取合适的节点和系数, 构造具有高精度的数值积分公式通过较少的节点得到较高的数值积分精度。因为要求已知  $f(x)$  表达式, 所以应用范围受限, 不适用于列表型数据。

## 3 运行结果与分析

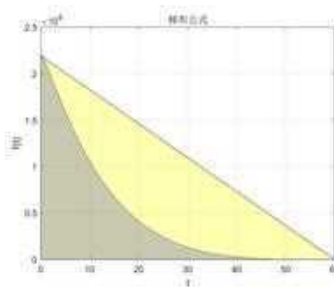
### 3.1 梯形公式

运行计算得到结果并绘制图像如下:

```

命令窗口
disp(' 梯形公式计算结果相对真实值相对误差')
esps = double(esps);
disp(esps);
解析解为 26361215.183948
梯形公式计算结果65645611.410492
梯形公式计算结果相对真实值相对误差为
1.490234647849823
fx >>

```



在图像中，深黄色表示的是原函数的积分值，浅黄色表示的是相对误差，由此可见对于梯形方法来说误差还是相当大的。

原因是：

$$E_1(f) = \int_a^b \frac{f''(\xi)}{2!} (x-a)(x-b) dx = \frac{f''(\xi)}{2!} \int_a^b (x-a)(x-b) dx = \frac{-(b-a)^3}{12} f''(\xi)$$

估计误差大约为原函数二阶导数的积分的平均值。

### 3.2 复合梯形公式

运行计算结果并绘制图像如下：

```

命令窗口
解析解为 26361215.183948
区间个数为 5 时
复合梯形公式计算结果 27713687.680354
复合梯形公式计算结果相对真实值相对误差为
0.051305392675137

区间个数为 10 时
复合梯形公式计算结果 26613929.660010
复合梯形公式计算结果相对真实值相对误差为
0.009588601918707

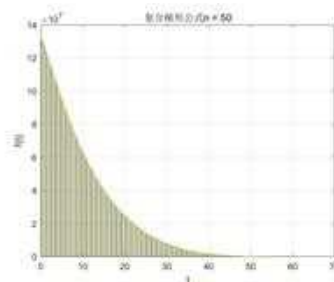
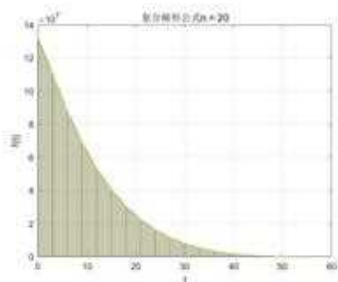
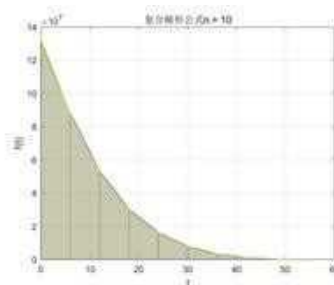
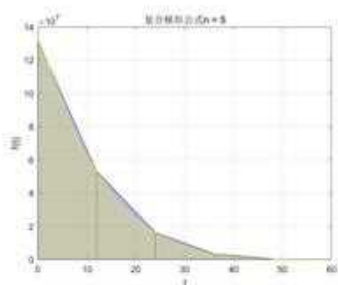
```

```

命令窗口
区间个数为 20 时
复合梯形公式计算结果 26398042.902264
复合梯形公式计算结果相对真实值相对误差为
0.001397041754685

区间个数为 50 时
复合梯形公式计算结果 26359374.357321
复合梯形公式计算结果相对真实值相对误差为
6.983087139672641e-05
fx

```



在图像中，深黄色表示的是采用复合梯形方法得到的结果，紫色表示的是相对误差，由此可见采用复合梯形方法可以使相对误差大大减少，且区间个数越多，误差越小。

### 3.3 Simpson 公式

运行结果如下：

```

命令窗口
解析解为 26361215.183948
区间个数为 5 时
复合Simpson法则计算结果I=26247343.6532285437
复合Simpson法则计算结果相对真实值相对误差为esps=0.0043196617
区间个数为 10 时
复合Simpson法则计算结果I=26326080.6496824361
复合Simpson法则计算结果相对真实值相对误差为esps=0.0013328116
区间个数为 20 时
复合Simpson法则计算结果I=26349650.5882485062
复合Simpson法则计算结果相对真实值相对误差为esps=0.0004386974
区间个数为 50 时
复合Simpson法则计算结果I=26358402.8991579786
复合Simpson法则计算结果相对真实值相对误差为esps=0.0001066827
fx >>

```

### 3.4 复合 Simpson 公式

运行结果如下：

```

命令窗口
解析解为 26361215.183948
Simpson1/3法则计算结果 I1=27132668.123073
Simpson1/3法则相对误差为 esps1=0.029265
Simpson3/8法则计算结果 I2=26327244.689562
Simpson3/8法则相对真实值相对误差为 esps2=0.001289
fx >> |

```

### 3.5 高阶 Newton - Cotes 公式

运行结果如下：

```

命令窗口
end
解析解为 26361215.183948
区间个数为 5 时
Newton-Cotes公式计算结果I=26331329.782779369503
Newton-Cotes公式计算结果相对真实值相对误差为esps=0.001133688298
区间个数为 10 时
Newton-Cotes公式计算结果I=26351221.917486254126
Newton-Cotes公式计算结果相对真实值相对误差为esps=0.000379088750
区间个数为 20 时
Newton-Cotes公式计算结果I=26357767.346697103232
Newton-Cotes公式计算结果相对真实值相对误差为esps=0.000130792045
区间个数为 50 时
Newton-Cotes公式计算结果I=26360354.944818153977
Newton-Cotes公式计算结果相对真实值相对误差为esps=0.000032632757
fx >> |

```

### 3.6 Romberg 积分

运行结果如下：

```
命令行窗口
end
解析解为 26361215.183948
区间个数为 5 时
Romberg积分计算结果I=26361215.1834025606513023
Romberg积分计算结果相对真实值相对误差为esps=0.0000000000206955
区间个数为 10 时
Romberg积分计算结果I=26361215.1834025606513023
Romberg积分计算结果相对真实值相对误差为esps=0.0000000000206955
区间个数为 20 时
Romberg积分计算结果I=26361215.1834025606513023
Romberg积分计算结果相对真实值相对误差为esps=0.0000000000206955
区间个数为 50 时
Romberg积分计算结果I=26361215.1839354000985622
Romberg积分计算结果相对真实值相对误差为esps=0.000000000004825
fx >>
```

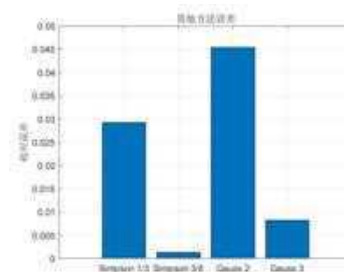
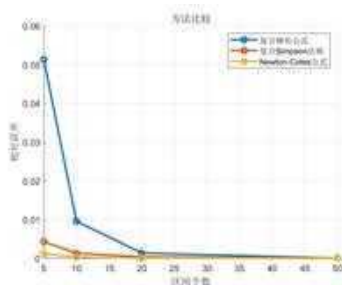
### 3.7 Gauss 求积公式

运行结果如下：

```
命令行窗口
解析解为 26361215.183948
两点高斯公式求解: 25165113.147622
两点高斯公式求解相对误差: 0.045374
三点高斯公式求解: 26576866.770124
三点高斯公式求解相对误差: 0.008181
fx >>
```

### 3.8 小结

将得到的运行结果绘制成图像如下：



其中由于梯形公式的误差非常大，因此不画入图像中。对于三种可以选择区间段数的方法，在区间段数相同情况下，可以发现精度是 *Newton - Cotes* 公式大于复合 *Simpson* 公式大于复合梯形公式；而对于每种方法，都是随着区间个数增加，计算精度增加。

总而言之，高阶 *Newton - Cotes* 公式和 *Romberg* 积分计算结果较为理想。

对于要求自己设置精度的 *Romberg* 积分，对于 *Romberg* 序列计算差值应  $\div 225$ ，但是原程序电脑并不能正常运行，因此修改简便后通过只调节误差精度保证程序的运行。个人认为在对于误差的理解上还需要加强。

## 4 实验心得

本次实验我熟悉了数值积分的多种计算方法，从难度上，我认为本次实验的难度较高，要求对于积分本质有着更清晰的理解。同时由于计算过程可能较为复杂，有时候精度选取不当经常会导致程序难以完成运行。

同时，一开始由于误用了函数 *int* 而不是现在选取的函数 *integral*，而后来了解 *int* 是要求原函数，因此会导致程序运行时间较长。

且本次实验由于积分理解的难度，本打算全部绘制图像来展示运算的过程，但后面由于对于 *matlab* 内置函数的不熟练，只绘制了梯形公式和复合梯形公式的可视化结果。

希望后面通过大作业可以更加熟悉这些方法。

## 5 源代码

### 5.1 绘制函数图像

```
1 % 定义函数
2 f = @(t) (1/60) * ((60 - t).^2 + (60 - t) .* sin(sqrt(t))) .* (10 *
   ((60 - t).^2 + (60 - t) .* sin(sqrt(t))) + 2 * ((60 - t).^2 + (60
   - t) .* sin(sqrt(t))).^(2/3));
3
4 % 生成0到60之间的一系列点
5 t = linspace(0, 60, 1000);
6
7 % 计算函数值
8 y = f(t);
9
10 % 绘制图像
11 plot(t, y);
12 title('Function Plot');
13 xlabel('t');
14 ylabel('f(t)');
15 grid on;
```

### 5.2 求解准确值

```
1 % 定义被积函数
```



```

2 fun = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-
   t).*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
3
4 % 求解积分
5 V_avg = 1/60 * integral(fun, 0, 60);
6
7 % 将结果格式化为6位小数
8 result_str = sprintf('%0.6f', V_avg);
9
10 disp(['平均电压准确值是: ', result_str]);

```

### 5.3 梯形公式

```

1 syms t;
2
3 % 定义被积函数
4 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
   .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
5
6 % 解析解
7 y = integral(f,0,60);
8
9 fprintf("解析解为 %0.6f\n", y/60);
10
11 a = 0; % 积分区间下界
12 b = 60; % 积分区间上界
13 t = b - a; % 积分区间长度
14
15 % 梯形公式计算结果
16 I = t * (f(b) + f(a)) / 2;
17
18 % 计算相对误差
19 esps = abs((I - y) / y);
20
21 fprintf("梯形公式计算结果 %0.6f\n", I/60);
22
23 disp('梯形公式计算结果相对真实值相对误差为');
24 esps = double(esps);
25 disp(esps);
26
27 % 定义函数

```

```

28 f = @(t) (1/60) * ((60 - t).^2 + (60 - t) .* sin(sqrt(t))) .* (10 *
    ((60 - t).^2 + (60 - t) .* sin(sqrt(t))) + 2 * ((60 - t).^2 + (60
    - t) .* sin(sqrt(t))).^(2/3));
29
30 % 生成0到60之间的一系列点
31 t = linspace(0, 60, 1000);
32
33 % 计算函数值
34 y = f(t);
35
36 % 绘制图像
37 figure;
38 plot(t, y, 'b');
39 hold on;
40
41 % 绘制0到60部分图像与x轴围成的阴影
42 fill([t, fliplr(t)], [y, zeros(size(y))], 'b', 'FaceAlpha', 0.3);
43
44 %绘制0处的函数值与60处的函数值的直线并标注梯形阴影
45 y0 = f(0);
46 y60 = f(60);
47 plot([0, 60], [y0, y60], 'r');
48 area([0, 60], [y0, y60], 'FaceColor', 'y', 'FaceAlpha', 0.3);
49
50 title('梯形公式');
51 xlabel('t');
52 ylabel('f(t)');
53 grid on;
54 hold off;

```

## 5.4 复合梯形公式

```

1 syms t;
2
3 % 定义被积函数
4 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
    .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
5
6 % 解析解
7 y = integral(f,0,60);
8

```

```

9 fprintf("解析解为 %.6f\n", y/60);
10
11 a = 0; % 积分区间下界
12 b = 60; % 积分区间上界
13 t = b - a; % 积分区间长度
14
15 % 待测试的区间个数
16 n_values = [5, 10, 20, 50];
17
18 for k = 1:length(n_values)
19     n = n_values(k);
20     h = t/n; % 步长
21     x(1) = a;
22     for i = 2:n+1
23         x(i) = x(i-1) + h; % 各个节点的值
24     end
25     sum = f(a) + f(b);
26     for i = 2:n
27         sum = sum + 2*f(x(i));
28     end
29     I = t*sum/(2*n); % 计算结果
30     esps = abs((I-y)/y); % 计算结果相对真实值相对误差
31     fprintf("区间个数为 %d 时\n", n);
32     fprintf("复合梯形公式计算结果 %.6f\n", I/60);
33     disp('复合梯形公式计算结果相对真实值相对误差为');
34     disp(double(esps));
35
36 % 绘制图像
37 figure;
38 t_values = linspace(a, b, 1000);
39 plot(t_values, f(t_values), 'b'); % 绘制原始函数
40 hold on;
41 for i = 1:n
42     fill([x(i), x(i+1), x(i+1), x(i)], [0, 0, f(x(i+1)), f(x(i))],
43         'b', 'FaceAlpha', 0.3); % 绘制每个小区间的积分结果
44 end
45
46 % 添加原函数与阴影
47 t_values = linspace(a, b, 1000);
48 y_values = f(t_values);
49 area(t_values, y_values, 'FaceColor', 'y', 'FaceAlpha', 0.3); % 添

```

```

    加阴影
49     plot(t_values, y_values, 'y'); % 添加原函数曲线
50
51     title(sprintf('复合梯形公式n = %d', n));
52     xlabel('t');
53     ylabel('f(t)');
54     grid on;
55     hold off;
56 end

```

## 5.5 Simpson 公式

```

1  syms t;
2
3  % 定义被积函数
4  f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
    .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
5
6  % 解析解
7  y = integral(f,0,60);
8
9  fprintf('解析解为 %.6f\n', y/60);
10
11 % Simpson1/3法则
12 a = 0; % 积分区间下界
13 b = 60; % 积分区间上界
14 t = b - a; % 积分区间长度
15 I1 = t/(3*2)*(f(a)+4*f((a+b)/2)+f(b)); % 计算结果
16 esps1 = abs((I1 - y) / y); % 相对误差
17 fprintf('Simpson1/3法则计算结果 I1=%.6f\n', I1/60);
18 fprintf('Simpson1/3法则相对误差为 esps1=%.6f\n', double(esps1));
19
20 % Simpson3/8法则
21 n = 3;
22 h = t / n; % 步长
23 x = a:h:b; % 各个节点的值
24 I2 = t/8 * (f(x(1)) + 3*f(x(2)) + 3*f(x(3)) + f(x(4))); % 计算结果
25 esps2 = abs((I2 - y) / y); % 相对误差
26 fprintf('Simpson3/8法则计算结果 I2=%.6f\n', I2/60);
27 fprintf('Simpson3/8法则相对真实值相对误差为 esps2=%.6f\n', double(
    esps2));

```

## 5.6 复合 Simpson 公式

```
1 syms t;
2
3 % 定义被积函数
4 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
    .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
5
6 % 解析解
7 y = integral(f,0,60);
8
9 fprintf('解析解为 %.6f\n', y/60);
10
11 % Simpson1/3 法则
12 a = 0; % 积分区间下界
13 b = 60; % 积分区间上界
14 t = b - a; % 积分区间长度
15 I1 = t/(3*2)*(f(a)+4*f((a+b)/2)+f(b)); % 计算结果
16 esps1 = abs((I1 - y) / y); % 相对误差
17 fprintf('Simpson1/3 法则计算结果 I1=%.6f\n', I1/60);
18 fprintf('Simpson1/3 法则相对误差为 esps1=%.6f\n', double(esps1));
19
20 % Simpson3/8 法则
21 n = 3;
22 h = t / n; % 步长
23 x = a:h:b; % 各个节点的值
24 I2 = t/8 * (f(x(1)) + 3*f(x(2)) + 3*f(x(3)) + f(x(4))); % 计算结果
25 esps2 = abs((I2 - y) / y); % 相对误差
26 fprintf('Simpson3/8 法则计算结果 I2=%.6f\n', I2/60);
27 fprintf('Simpson3/8 法则相对真实值相对误差为 esps2=%.6f\n', double(
    esps2));
```

## 5.7 高阶 Newton - Cotes 公式

```
1 syms t;
2
3 % 定义被积函数
4 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
    .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
5
6 % 解析解
```

```

7 y = integral(f,0,60);
8
9 fprintf("解析解为 %.6f\n", y/60);
10
11 % Simpson1/3法则
12 a = 0; % 积分区间下界
13 b = 60; % 积分区间上界
14 t = b - a; % 积分区间长度
15 I1 = t/(3*2)*(f(a)+4*f((a+b)/2)+f(b)); % 计算结果
16 esps1 = abs((I1 - y) / y); % 相对误差
17 fprintf('Simpson1/3法则计算结果 I1=%.6f\n', I1/60);
18 fprintf('Simpson1/3法则相对误差为 esps1=%.6f\n', double(esps1));
19
20 % Simpson3/8法则
21 n = 3;
22 h = t / n; % 步长
23 x = a:h:b; % 各个节点的值
24 I2 = t/8 * (f(x(1)) + 3*f(x(2)) + 3*f(x(3)) + f(x(4))); % 计算结果
25 esps2 = abs((I2 - y) / y); % 相对误差
26 fprintf('Simpson3/8法则计算结果 I2=%.6f\n', I2/60);
27 fprintf('Simpson3/8法则相对真实值相对误差为 esps2=%.6f\n', double(
    esps2));

```

## 5.8 复化梯形公式求积分函数

```

1 function y=Trape(f, a, b, n)
2 % TrapezoidInteg 用复化梯形公式求积分
3 t=b-a;
4 h=t/n;
5 x(1)=a;
6 for i=2:n
7     x(i)=x(i-1)+h; %各个节点的值
8 end
9 sum=f(a)+f(b);
10 for i=2:n
11     sum=sum+2*f(x(i));
12 end
13 y=t*sum/(2*n); %计算结果
14 end

```

## 5.9 Romberg 积分

```
1 %Romberg 积分
2 syms t;
3
4 % 定义被积函数
5 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
    .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
6
7 % 解析解
8 y = integral(f,0,60);
9
10 fprintf("解析解为 %.6f\n", y/60);
11 a=0; %积分区间下界
12 b=60; %积分区间上界
13 t=b-a; %积分区间长度
14 % 待测试的区间个数
15 n_values = [5, 10, 20, 50];
16
17 for k = 1:length(n_values)
18     n = n_values(k);
19     h=t/n;
20     esp=0.1; %计算误差上限
21     err=1; %初始化误差
22
23     T(1,1)=Trape(f,a,b,n);
24     l=2;
25
26     while err>=esp
27         T(l,1)=Trape(f,a,b,n*2^l);
28         T(l,1)=0;
29         for i=2:l
30             T(l,i)=(4^(i-1)*T(l,i-1)-T(l-1,i-1))/(4^(i-1)-1);
31         end
32         err=abs(T(l,1)-T(l-1,1-1));
33         l=l+1;
34     end
35
36     I=T(l-1,1-1);
37
38     esps=abs((I-y)/y); %计算结果相对真实值相对误差
39     esps=double(esps);
```

```

40     fprintf("区间个数为 %d 时\n", n);
41     fprintf('Romberg 积分计算结果 I=%.16f\n', I/60);
42     fprintf('Romberg 积分计算结果相对真实值相对误差为 esps=%.16f\n', esps
43         );
44 end

```

## 5.10 Gauss 求积公式

```

1 %高斯积分公式
2 syms t;
3
4 % 定义被积函数
5 f = @(t) ((60-t).^2 + (60-t).*sin(sqrt(t))) .* (10*((60-t).^2 + (60-t)
6     .*sin(sqrt(t))) + 2*((60-t).^2 + (60-t).*sin(sqrt(t))).^(2/3));
7
8 % 解析解
9 y = integral(f,0,60);
10
11 fprintf("解析解为 %.6f\n", y/60);
12 a=0; %积分上限
13 b=60; %积分下限
14 T1=(b-a)/2*(f((b+a)/2-(b-a)/6*(3^0.5))+(f((b+a)/2+(b-a)/6*(3^0.5))));
15 %两点 Gauss—Legendre 公式
16 T2=(b-a)/2*(5/9*f((b+a)/2-(b-a)/2*(-15^0.5/5))+5/9*(f((b+a)/2+(b-a)
17     /2*(-15^0.5/5)))+8/9*f((b+a)/2)); %三点 Gauss—Legendre 公式
18
19 %误差分析
20 er1=abs(T1-y)/y;
21 er2=abs(T2-y)/y;
22
23 fprintf("两点高斯公式求解: %.6f\n", T1/60);
24 fprintf("两点高斯公式求解相对误差: %.6f\n", er1);
25 fprintf("三点高斯公式求解: %.6f\n", T2/60);
26 fprintf("三点高斯公式求解相对误差: %.6f\n", er2);

```

## 5.11 运行结果比较

```

1 % 数据
2 esps1 = 0.029265;
3 esps2 = 0.001289;
4 gauss2 = 0.045374;

```



```

5 gauss3 = 0.008181;
6 trap_errors = [0.051305392675138, 0.009586601918707,
    0.001397041754685, 6.983087139665088e-05];
7 simpson_errors = [0.0043196617, 0.0013328116, 0.0004386974,
    0.0001066827];
8 newton_errors = [0.001133688298, 0.000379089750, 0.000130792045,
    0.000032632757];
9
10 figure;
11 bar([esps1, esps2, gauss2, gauss3]);
12 title('其他方法误差');
13 xticklabels({'Simpson 1/3', 'Simpson 3/8', 'Gauss 2', 'Gauss 3'});
14 ylabel('相对误差');
15 grid on;
16
17 figure;
18 hold on;
19 plot([5, 10, 20, 50], trap_errors, '-o', 'LineWidth', 2);
20 plot([5, 10, 20, 50], simpson_errors, '-o', 'LineWidth', 2);
21 plot([5, 10, 20, 50], newton_errors, '-o', 'LineWidth', 2);
22 hold off;
23 title('方法比较');
24 xlabel('区间个数');
25 ylabel('相对误差');
26 legend('复合梯形公式', '复合Simpson法则', 'Newton-Cotes公式');
27 grid on;

```