作业二

敖旭扬　　**PB18071477**　　**2020 年 12 月 5 日**

第一题 解：本题均使用当前积分值与上一个计算出的积分值之差的绝对值 $|I_n - I_{n-1}| < 10^{-15}$ 作为结束计算的条件，即此时计算得到的积分值 I_n 已经是最精确的了。因为用 Cauchy 收敛准则可以证明 I_n 一定会收敛，且随着 n 增大，$|I_n - I_{n-1}|$ 也将减小，但是在双精度（double precision）的计算环境下，计算机的机器精度约为 2.22×10^{-16}。所以当 $|I_n - I_{n-1}| < 10^{-15}$ 时，由于计算机精度有限，I_n 已经不能更精确了，它已经是最精确的了。

(a) 使用自动控制误差的复化梯形公式计算积分的 MATLAB 程序显示如下：

```
clear, clc, clf
MS = 'MarkerSize'; ms = 8;
MC = 'MarkerFaceColor';

F = @(x) sin(cos(sin(cos(x)))); % 积分函数
a = -1; b = 1; % 积分区间[a,b]
tol = 1e-15; % 误差控制精度
n = 1; % 初始分点数

h = (b - a) / n; % 小区间长度
x = linspace(a, b, n + 1); % 取值点
% 新的复化梯形积分数值
Tnew = h * (F(a) / 2 + sum(F(x(2:end - 1))) + F(b) / 2);
T = zeros(100, 1); % 存储每个次迭代的复化梯形积分数值
Told = Tnew + 1; % 该Told无意义，用于进入下面的while循环
j = 1; % 迭代轮次
T(j) = Tnew; % 保存第1个轮的复化梯形积分数值

while abs(Told - Tnew) > tol
    Told = Tnew; % 保存上一轮迭代的复化梯形积分数值
    H = h * sum(F(a + (2 * (1:n) - 1) * h / 2));
    Tnew = (Told + H) / 2; % 计算当前轮的复化梯形积分数值
    T(j + 1) = Tnew;
```

```
    h = h / 2; % 小区间长度减半
    n = 2 * n; % 分点加密一倍
    j = j + 1; % 下一轮迭代
end

%% 输出计算结果
format long
I = Tnew% 使用自动控制误差的复化梯形公式计算的积分值
n% 输出最终分点数
semilogy((1:j), abs(T(1:j) - I), 'ro-', MC, 'b', MS, ms);
xlabel('迭代次数');
ylabel('误差');
```

上述程序在命令行输出的结果为：

$$I = 1.339880713117284$$

$$n = 33554432$$

即在误差控制精度为 10^{-15} 的情况下，计算得最精确的结果为

$$I = 1.339880713117284$$

上述程序输出的绝对误差随着迭代次数变化的 semilogy 图为：

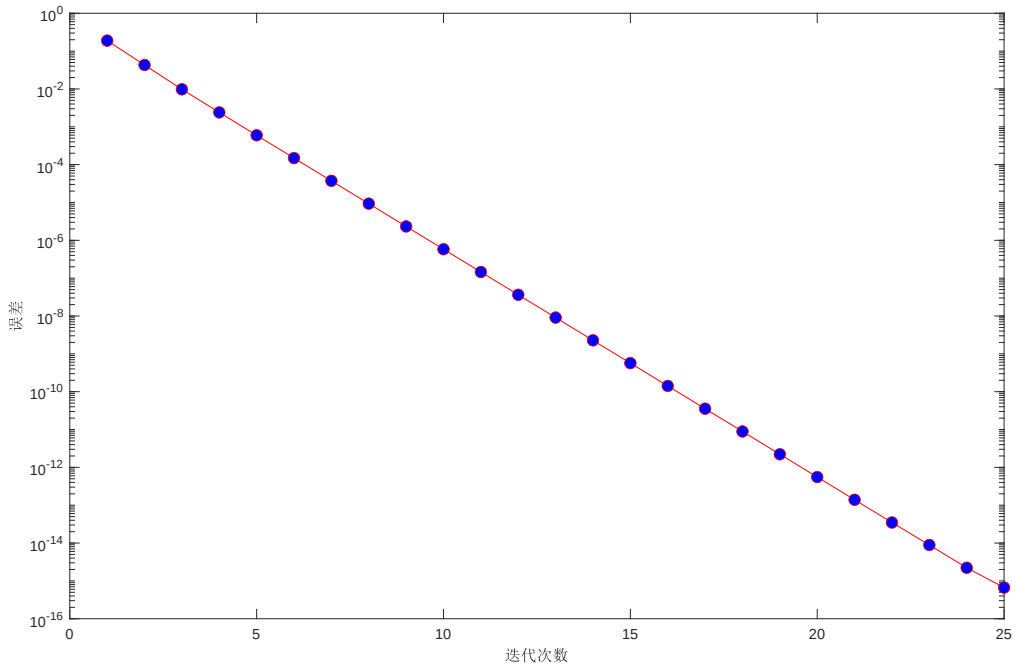

图 1: 使用自动控制误差的复化梯形公式的效果

(b) 使用 Richardson 外推方法计算积分的 MATLAB 程序显示如下：

```
clear,clc, clf
MS = 'MarkerSize'; ms = 8;
MC = 'MarkerFaceColor';
%% 第1步
F = @(x) sin(cos(sin(cos(x)))); % 积分函数
a = -1; b = 1; % 积分区间[a,b]
e=1e-15; % 精度控制值
M=30; %最大循环次数
n=1;
h = b-a;
I = zeros(M); %存储每一轮迭代计算得到的积分值
%% 第2步
R= zeros(M,M);
R(1,1) = (F(a)+F(b))*h/2;
I(1) = R(1,1);
%% 第3步
for k=2 :M
```

```
18      hk = h/2^(k-1);
19      Fsum=sum(F(a + (2 * (1:2^(k-2)) - 1) * hk));
20      R(k,1) = (R(k-1,1)+(2*hk)* Fsum)/2; % 2 * h_{k}=h_{k
            -1}
21      for j=2:k
22          R(k,j) = R(k,j-1)+(R(k,j-1)-R(k-1,j -1))/(4^(j-1)
                -1);
23      end
24      I(k) = R(k,k);
25      n=n+1;
26      if( abs(R(k,k )-R(k-1,k-1)) < e)
27          Irichardson = R(k,k) % 第4步,输出最终的积分值
28          break
29      end
30  end
31  %% 输出计算结果
32  format long
33  n %迭代次数
34  semilogy((1:n), abs(I(1:n)-Irichardson), 'ro-', MC, 'b',MS
        ,ms);
35  xlabel('迭代次数');
36  ylabel('误差');
```

上述程序在命令行输出的结果为：

$$Irichardson = 1.339880713117284$$

$$n = 9$$

即在精度控制值为 10^{-15} 的情况下，计算得最精确的结果为

$$I_{richardson} = 1.339880713117284$$

上述程序输出的绝对误差随着迭代次数变化的 semilogy 图为：

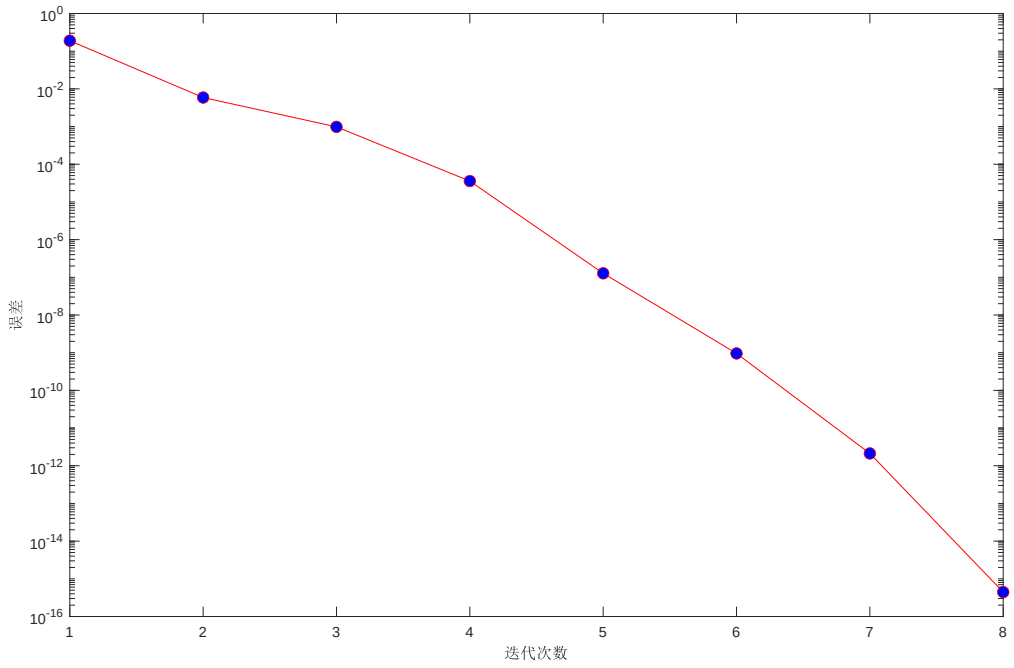

图 2: 使用 Richardson 外推方法的效果

(c) 使用 Gauß 积分计算积分的 MATLAB 程序显示如下:

```
clear, clc, clf
MS = 'MarkerSize'; ms = 8;
MC = 'MarkerFaceColor';

F = @(x) sin(cos(sin(cos(x)))); % 积分函数
a = -1; b = 1; % 积分区间[a,b]
e = 1e-15; % 精度控制值
M = 30; %最大循环次数
I = zeros(M);
[x, w] = gauss(1);
I(1) = w * F(x);

for n = 2:M
    [x, w] = gauss(n);
    I(n) = w * F(x);

    if (abs(I(n) - I(n - 1)) < e)
```

```
        Igauss = I(n)% 输出最终的积分值
        break
    end

end


n%迭代次数
semilogy((1:n), abs(I(1:n) - Igauss), 'ro-', MC, 'b', MS,
   ms);
xlabel('迭代次数');
ylabel('误差');

% GAUSS  nodes x (Legendre points) and weights w
%        for Gauss quadrature
function [x, w] = gauss(N)
beta = .5 ./ sqrt(1 - (2 * (1:N - 1)).^(-2));
T = diag(beta, 1) + diag(beta, -1);
[V, D] = eig(T);
x = diag(D); [x, i] = sort(x);
w = 2 * V(1, i).^2;
end
```

上述程序在命令行输出的结果为：

$$Igauss = 1.339880713117285$$

$$n = 16$$

即在误差控制精度为 10^{-15} 的情况下，计算的最精确的结果为

$$I_{gauss} = 1.339880713117285$$

上述程序输出的绝对误差随着迭代次数变化的 semilogy 图为：

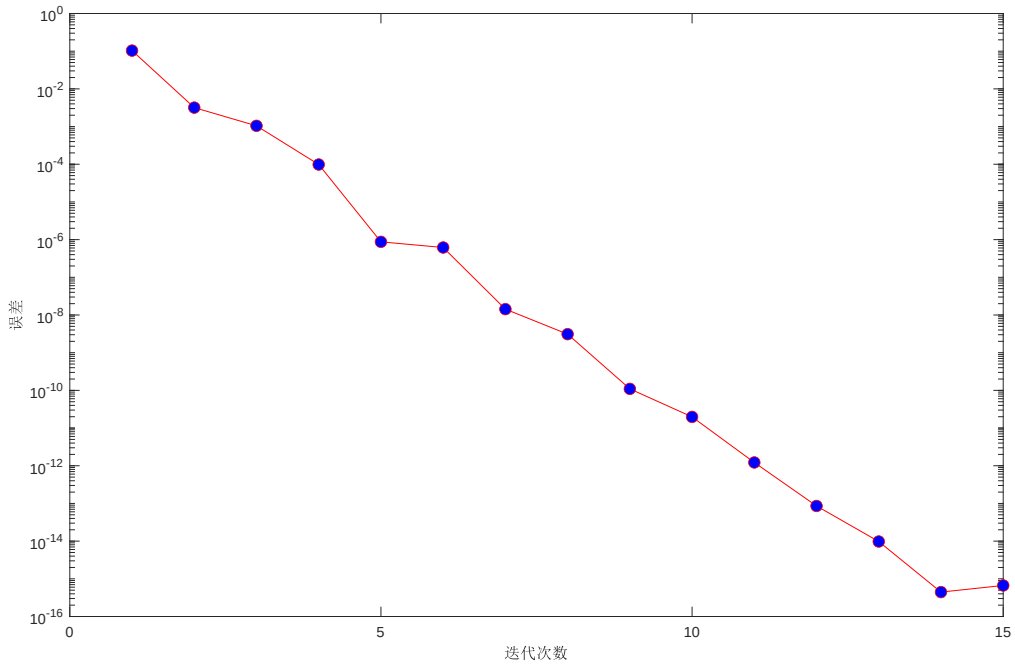

图 3: 使用 Gauß 积分的效果

(d) 三种方法均使用自动误差控制逐步迭代来达到一个满意的最精确的结果：

(a) 中 MATLAB 程序输出分点数为 $n = 33554432$，则使用自动控制误差的复化梯形公式的方法的采样点总数约为

$$N_a = n = 33554432$$

(b) 中 MATLAB 程序输出迭代次数为 $n = 9$，则使用 Richardson 外推方法的方法的采样点总数约为

$$N_b = \sum_{k=2}^{n} 2^{k-2} = \sum_{i=0}^{n-2} 2^i = 2^{n-1} - 1 = 2^{9-1} - 1 = 255$$

(c) 中 MATLAB 程序输出迭代次数为 $n = 16$，则使用 Gauß 积分的方法的采样点总数约为

$$N_c = \sum_{k=1}^{n} k = \frac{n(n+1)}{2} = \frac{16(16+1)}{2} = 136$$

综上所述，$N_c < N_b \ll N_a$，所以从取样总数考虑，三种方法的效率：(c) 的效率高于 (b) 的效率，(b)、(c) 的效率远高于 (a) 的效率。

第二题 解：

(a) 证明：

由于

$$\ell_j(x) = \frac{1}{\pi_j} \prod_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k) = \frac{e_j(x)}{\pi_j} \tag{1}$$

其中

$$e_j(x) = \prod_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k) \tag{2}$$

而

$$e_j'(x) = \sum_{\substack{t=0 \\ t\neq j}}^{n} \prod_{\substack{k=0 \\ k\neq t,j}}^{n} (x - x_k) = \sum_{\substack{k=0 \\ k\neq j}}^{n} \frac{e_j(x)}{(x - x_k)} = e_j(x) \sum_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k)^{-1} \tag{3}$$

所以

$$\ell_j'(x) = \left(\frac{e_j(x)}{\pi_j}\right)' = \frac{e_j'(x)}{\pi_j} = \frac{e_j(x)}{\pi_j} \sum_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k)^{-1} = \ell_j(x) \sum_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k)^{-1} \tag{4}$$

则

$$p'(x) = \sum_{j=0}^{n} f_j \ell_j'(x) = \sum_{j=0}^{n} \left(f_j \ell_j(x) \sum_{\substack{k=0 \\ k\neq j}}^{n} (x - x_k)^{-1} \right) \tag{5}$$

(b) 用等距插值计算 $f(x) = \sin(x)$ 在 $[-1, 1]$ 上的导数的 MATLAB 程序显示如下：

```
clear, clc, clf
LW = 'linewidth'; lw = 2;

n = 15;
x = linspace(-1, 1, n + 1)';
m = 1001;
xx = linspace(-1, 1, m)';
F = @sin;
f = F(x);
P = @cos;
p = P(xx);
pp = zeros(m, 1);
```

```
13
14 for k = 1:n
15     l = ones(m, 1);
16     sum = zeros(m, 1); % 公式(1)中大括号内的求和部分
17
18     for j = 1:k - 1
19         l = l .* (xx - x(j)) / (x(k) - x(j));
20         sum = sum +1 ./ (xx - x(j));
21     end
22
23     for j = k + 1:n
24         l = l .* (xx - x(j)) / (x(k) - x(j));
25         sum = sum +1 ./ (xx - x(j));
26     end
27
28     pp = pp + f(k) * l .* sum;
29 end
30
31 R = abs(pp - p); %与真实解的误差绝对值
32 figure (1)
33 plot(xx, P(xx), 'k', LW, lw), hold on
34 plot(xx, pp, 'b', LW, lw);
35 xlabel('x');
36 ylabel('p(x)或cos(x)');
37 legend('exact', 'interpolant', 'location', 'nw');
38 figure (2)
39 semilogy(xx, R, 'k', LW, lw);
40 xlabel('x');
41 ylabel('error');
```

上述程序输出的计算出的导函数与真实解 $f'(x) = \cos(x)$ 的图像为：

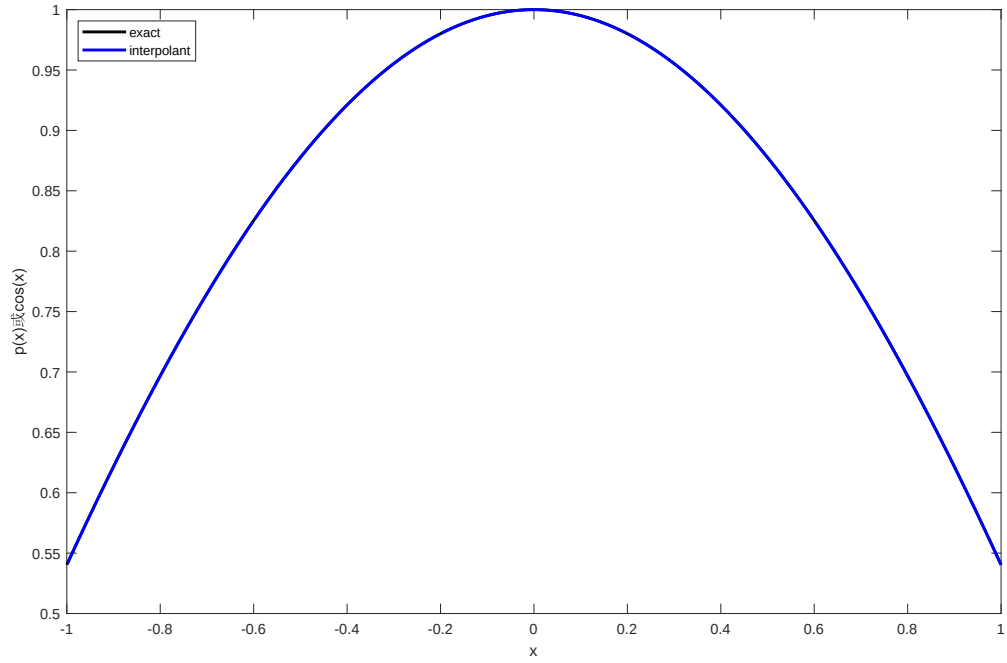

图 4: 导函数图像

逐点误差绝对值的 semilogy 图为：

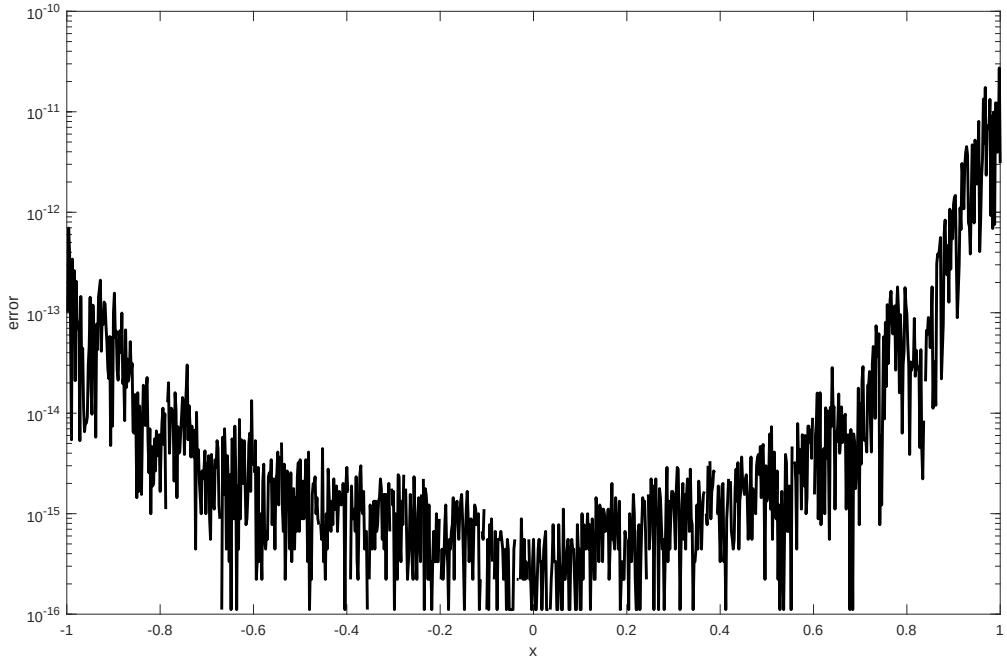

图 5: 逐点误差的绝对值

(c) 证明：

由

$$\boldsymbol{p}' = \boldsymbol{D}\boldsymbol{f} \tag{6}$$

得

$$p'(x_i) = \sum_{j=0}^{n} f_j D_{ij} \tag{7}$$

又由 (5) 式，有

$$D_{ij} = \ell_j(x_i) \sum_{\substack{k=0 \\ k\neq j}}^{n} (x_i - x_k)^{-1} \tag{8}$$

又

$$\ell_j(x_k) = \begin{cases} 1 & k = j \\ 0 & k \neq j \end{cases} \tag{9}$$

所以 $i = j$ 时

$$D_{ij} = \ell_j(x_i) \sum_{\substack{k=0 \\ k\neq j}}^{n} (x_i - x_k)^{-1} = \sum_{\substack{k=0 \\ k\neq j}}^{n} (x_i - x_k)^{-1} \tag{10}$$

而 $i \neq j$ 时，根据 (5) 式前一个等式部分

$$D_{ij} = \ell_j'(x_i) = \ell_j'(x)\Big|_{x=x_i} = \frac{1}{\pi_j} e_j'(x)\Big|_{x=x_i} \tag{11}$$

由 (3) 式得

$$e_j'(x)\Big|_{x=x_i} = \sum_{\substack{t=0 \\ t\neq j}}^{n} \prod_{\substack{k=0 \\ k\neq t,j}}^{n} (x - x_k)\Big|_{x=x_i} = \sum_{\substack{t=0 \\ t\neq j}}^{n} \prod_{\substack{k=0 \\ k\neq t,j}}^{n} (x_i - x_k) = \prod_{\substack{k=0 \\ k\neq i,j}}^{n} (x_i - x_k) \tag{12}$$

所以 $i \neq j$ 时

$$D_{ij} = \frac{1}{\pi_j} e_j'(x)\Big|_{x=x_i} = \frac{1}{\pi_j} \prod_{\substack{k=0 \\ k\neq i,j}}^{n} (x_i - x_k) = \frac{\pi_i}{\pi_j (x_i - x_j)} \tag{13}$$

综上，有

$$D_{ij} = \begin{cases} \dfrac{1}{\pi_j} \displaystyle\prod_{\substack{k=0 \\ k\neq i,j}}^{n} (x_i - x_k) = \dfrac{\pi_i}{\pi_j (x_i - x_j)} & i \neq j \\ \displaystyle\sum_{\substack{k=0 \\ k\neq j}}^{n} (x_i - x_k)^{-1} & i = j \end{cases} \tag{14}$$

(d) MATLAB 程序显示如下：

```
clear, clc, clf
MC = 'MarkerFaceColor';

F = @(x) sin(3 * x.^2);
dF = @(x) 6 * x .* cos(3 * x.^2);
maxe = zeros(30, 1)';
maxc = zeros(30, 1)';
nvec = (1:2:59); % 依次令 n=1,3,5,...,57,59
j = 1;

for n = nvec
    %% 等距点
    xe = linspace(-1, 1, n + 1)';
    dfe = dF(xe); %精确导数值
    fe = F(xe); % 等距点上的函数值
    dpe = diffM(xe) * fe; %用微分矩阵计算的导数值
    maxe(j) = max(abs(dpe - dfe)); %误差绝对值最大值
    %% Chebyshev点
    xc = cos((0:n) * pi ./ n)'; % xc(j)中是x_{j-1}
    dfc = dF(xc);
    fc = F(xc);
    dpc = diffM(xc) * fc;
    maxc(j) = max(abs(dpc - dfc));
    j = j + 1;
end
%% 画图
figure(1)
plote = semilogy(nvec, maxe, 'b^-', MC, 'b'); hold on
plotc = semilogy(nvec, maxc, 'k*-', MC, 'k'); hold on
legend([plote, plotc], '等距点误差', 'Chebyshev点误差');
%% 计算微分矩阵D的函数
function D = diffM(x)
n = length(x);
D = zeros(n, n);
```

```
35 PI = zeros(n);
36 for k = 1:n
37     PI(k) = prod(x(k) - x((1:n) ~= k));
38 end
39 for i = 1:n
40     for j = 1:n
41         if (i ~= j)
42             D(i, j) = PI(i) / (PI(j) * (x(i) - x(j)));
43         else
44             D(i, j) = sum(1 ./ (x(j) - x((1:n) ~= j)));
45         end
46     end
47 end
48 end
```

上述程序输出的图像为：

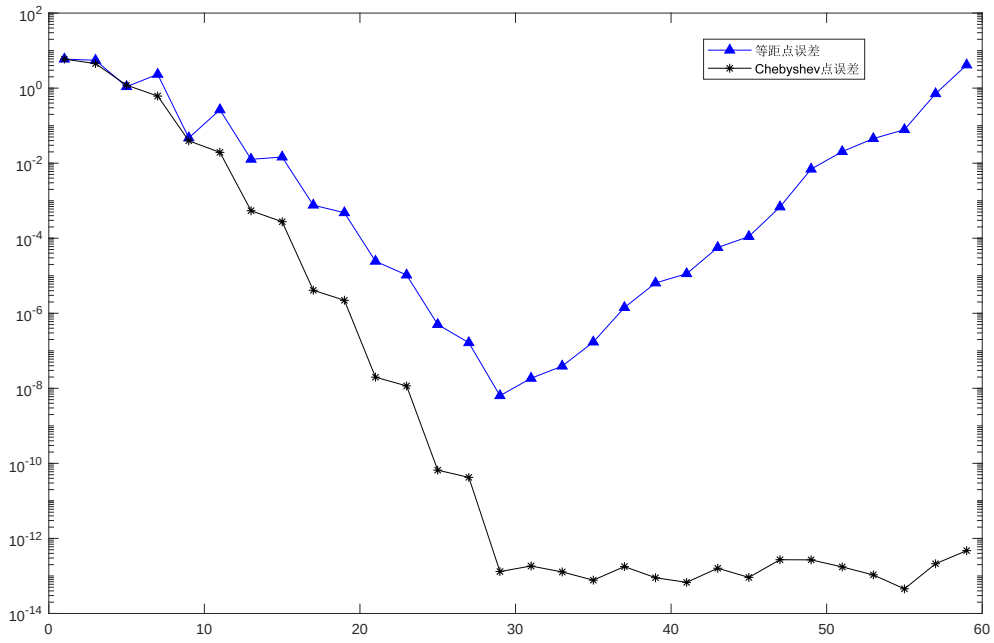

图 6: 等距点和 Chebyshev 点上的误差绝对值

观察发现，使用 Chebyshev 点时，在最大误差下降到 10^{-13} 之后就基本保持（同一数量级）不变了；而使用等距点时，$n = 29$ 时最大误差下降到最小值，之后却

一直增大，甚至到 $n = 59$ 时，最大误差高达 4.139913901026180。这是因为出现了 Runge（龙格）现象，高次多项式的插值效果不一定优于低次多项式的插值的效果，同时等距插值不能保证有较好的插值效果。

第三题

(a) 该多步法公式中的系数为

$$\alpha = \int_{x_{n-1}}^{x_{n+1}} \frac{(x-x_n)(x-x_{n-1})(x-x_{n-2})}{(x_{n+1}-x_n)(x_{n+1}-x_{n-1})(x_{n+1}-x_{n-2})}\mathrm{d}x = \frac{h}{3}$$
$$\beta = \int_{x_{n-1}}^{x_{n+1}} \frac{(x-x_{n+1})(x-x_{n-1})(x-x_{n-2})}{(x_n-x_{n+1})(x_n-x_{n-1})(x_n-x_{n-2})}\mathrm{d}x = \frac{4h}{3}$$
$$\gamma = \int_{x_{n-1}}^{x_{n+1}} \frac{(x-x_{n+1})(x-x_n)(x-x_{n-2})}{(x_{n-1}-x_{n+1})(x_{n-1}-x_n)(x_{n-1}-x_{n-2})}\mathrm{d}x = \frac{h}{3}$$
$$\mu = \int_{x_{n-1}}^{x_{n+1}} \frac{(x-x_{n+1})(x-x_n)(x-x_{n-1})}{(x_{n-2}-x_{n+1})(x_{n-2}-x_n)(x_{n-2}-x_{n-1})}\mathrm{d}x = 0$$

(b) 截断为

$$\begin{aligned} T_{n+1} &= \int_{x_{n-1}}^{x_{n+1}} R(x)\mathrm{d}x = \int_{x_{n-1}}^{x_{n+1}} \frac{y^{(5)}(\xi)}{5!}(x-x_{n+1})(x-x_n)(x-x_{n-1})(x-x_{n-2})\mathrm{d}x \\ &= \frac{y^{(5)}(\xi)}{5!}\int_{x_{n-1}}^{x_{n+1}} (x-x_{n+1})(x-x_n)(x-x_{n-1})(x-x_{n-2})\mathrm{d}x \\ &= \frac{y^{(5)}(\xi)}{5!}\cdot\frac{-4h^5}{15} = -\frac{1}{450}h^5 y^{(5)}(\xi) = O(h^5) = O(h^{4+1}) \end{aligned}$$

所以此格式是 4 阶的。

(c) 由 (a) 得到计算格式

$$y_{n+1} = y_{n-1} + \frac{h}{3}(f_{n-1} + 4f_n + f_{n+1})$$

代入

$$f_n = f(x_n, y_n) = x_n e^{-5x_n} - 5y_n$$

得

$$y_{n+1} = y_{n-1} + \frac{h}{3}[x_{n-1}e^{-5x_{n-1}} - 5y_{n-1} + 4(x_n e^{-5x_n} - 5y_n) + x_{n+1}e^{-5x_{n+1}} - 5y_{n+1}]$$

令 $g_n = x_n e^{-5x_n}$，整理得到下面的递归计算式

$$y_{n+1} = \frac{y_{n-1} + \frac{h}{3}\left[g_{n-1} - 5y_{n-1} + 4g_n - 20y_n + g_{n+1}\right]}{1 + \frac{5h}{3}} \tag{15}$$

所以可以用三阶 Runge-Kutta 方法起步计算出前两项，再用上式计算出其后所有项，求解的 MATLAB 程序显示如下：

```
clear, clc, clf
LW = 'linewidth'; lw = 2;
%% 问题输入
a = 0; b = 2; % 求解区间[0, 2]
n = 101; % 取样点数
h = (b - a) / (n - 1); %步长
x = linspace(a, b, n)'; % 取样点
g = x .* exp(-5 .* x);
yn = zeros(n, 1); % 存储计算出的函数在取样点处的数值解
yn(1) = 0; %初值条件
f = @(x, y) x * exp(-5 * x) - 5 * y;
%% 使用三阶Runge-Kutta方法起步, 计算前两项即可
k1 = zeros(2, 1);
k2 = zeros(2, 1);
k3 = zeros(2, 1);
for i = 1:2
    k1(i) = f(x(i), yn(i));
    k2(i) = f(x(i) + h / 2, yn(i) + (h / 2) * k1(i));
    k3(i) = f(x(i) + h, yn(i) - h * k1(i) + 2 * h * k2(i))
        ;
    yn(i + 1) = yn(i) + (h / 6) * (k1(i) + 4 * k2(i) + k3(
        i));
end

%% 使用4阶的线性多步法从第3项开始计算
for i = 3:n - 1
    yn(i + 1) = (yn(i - 1) + (h / 3) * (g(i - 1) - 5 * yn(
        i - 1) + 4 * g(i) - 20 * yn(i) + g(i + 1))) / (1 +
        5 * h / 3);
end

%% 符号运算求解析解
syms y(t)
Y = dsolve(diff(y, 1) == t * exp(-5 * t) - 5 * y, y(0) ==
    0, t);
```

```
y = matlabFunction(Y); %初值问题的解析解
ya = y(x); % 解析解在取样点处的取值
figure(1)
pn = plot(x, yn, 'k', LW, lw); hold on
pa = plot(x, ya, 'r', LW, lw);
legend([pn, pa], '数值解', '解析解');
```

上述程序输出的图像为：

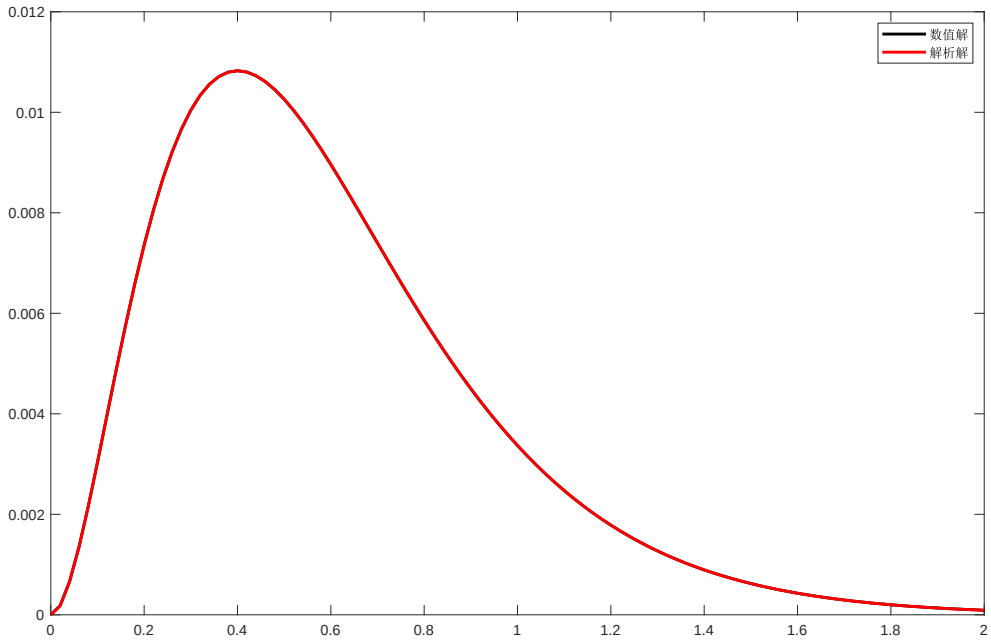

图 7: Runge-Kutta 方法起步和线性多步法求解初值问题的结果

可以看到数值解与解析解完全重合，三阶的 Runge-Kutta 方法作为三阶格式之所以不会影响使用该多步法时的精度，是因为三阶的 Runge-Kutta 方法局部截断误差为 $O(h^4)$，而多步法的局部截断误差为 $O(h^5)$，满足“起步计算的格式的精度至多只能比该格式的精度低一阶”的条件，所以不会影响求解精度。

(d) 一阶常微分方程有如下求解公式

$$y' + P(x)y = Q(x) \Rightarrow y = e^{-\int P\mathrm{d}x}\left[\int e^{-\int P\mathrm{d}x}Q(x)\mathrm{d}x + C\right]$$

所以对于所求的方程，即 $P(x) = 5, \quad Q(x) = xe^{-5x}$，代入上面的求解公式得精确

解为

$$y = \frac{1}{2}x^2 e^{-5x} \tag{16}$$

取一系列 n 值或 h 值，计算出相应的数值解和精确解误差的最大值，再将这一系列误差最大值和 h 值画成 loglog 图，求解的 MATLAB 程序显示如下：

```
clear, clc, clf
LW = 'linewidth'; lw = 2;
a = 0; b = 2; % 求解区间[0, 2]
y = @(x) (x.^2) .* exp(-5 .* x) / 2; % 推导出的精确解
nvec = 101:10001; % 取样点数从101到1001
hvec = (b - a) ./ (nvec - 1);
maxe = zeros(length(nvec), 1)';
iter = 1; % 计数
for n = nvec
    h = (b - a) / (n - 1); %步长
    x = linspace(a, b, n)'; % 取样点
    g = x .* exp(-5 .* x);
    yn = zeros(n, 1); % 存储计算出的函数在取样点处的数值解
    yn(1) = 0; %初值条件
    f = @(x, y) x * exp(-5 * x) - 5 * y;
    %% 使用三阶Runge-Kutta方法起步，计算前两项即可
    k1 = zeros(2, 1);
    k2 = zeros(2, 1);
    k3 = zeros(2, 1);
    for i = 1:2
        k1(i) = f(x(i), yn(i));
        k2(i) = f(x(i) + h / 2, yn(i) + (h / 2) * k1(i));
        k3(i) = f(x(i) + h, yn(i) - h * k1(i) + 2 * h * k2
            (i));
        yn(i + 1) = yn(i) + (h / 6) * (k1(i) + 4 * k2(i) +
            k3(i));
    end
    %% 使用4阶的线性多步法从第3项开始计算
    for i = 3:n - 1
        yn(i + 1) = (yn(i - 1) + (h / 3) * (g(i - 1) - 5 *
            yn(i - 1) + 4 * g(i) - 20 * yn(i) + g(i + 1)))
```

```
            / (1 + 5 * h / 3);
    end
    ya = y(x); % 精确解在取样点处的取值
    maxe(iter) = max(abs(ya - yn));
    iter = iter + 1;
end

figure(1)
pn = loglog(hvec, maxe, 'k', LW, lw); hold on
xlabel('h')
ylabel('max error')
set(gca, 'yaxislocation', 'right');
```

上述程序输出的图像为：

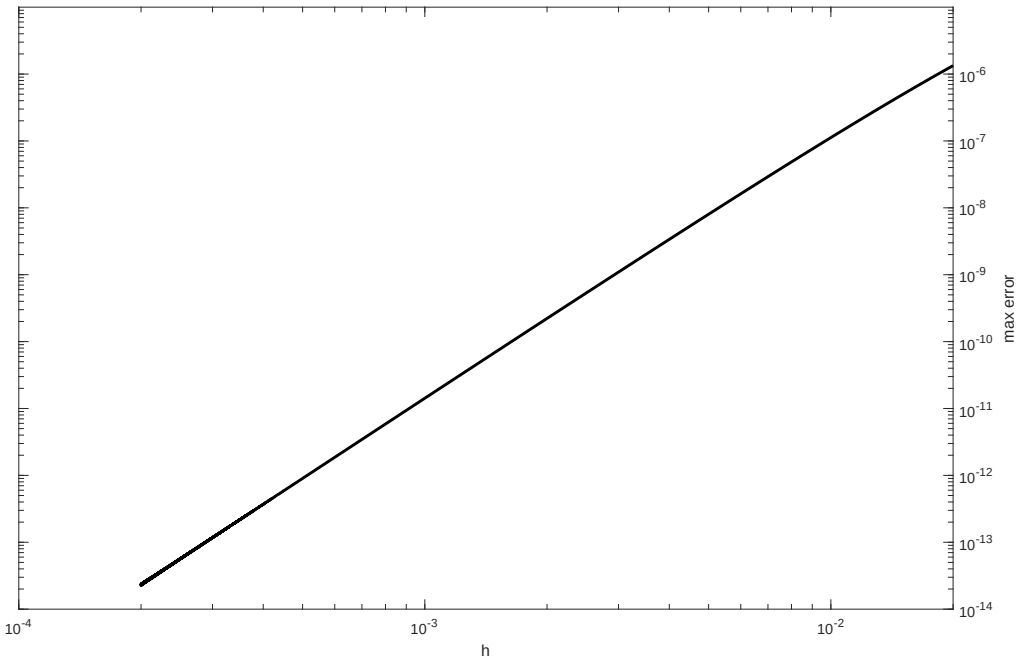

图 8: 线性多步法求解初值问题的误差和步长的 loglog 图

可以看到误差和步长 h 的对数斜率约为 4，这就证实了 (b) 中推断的阶数 4 是正确的，此格式的整体截断误差为 $O(h^4)$。