目录

概述

为了实验报告整体美观，报告正文部分只显示完成实验所必须的代码部分，其余为了更好地分析实验内容而写的代码都附在附录当中。数值计算本身就是为了对实际问题得到一个较好，可以使用的结果，而不是通过非常理论得结果得到百分百精确的解。而且自己动手实验实现其中的数学方法以及更加直观地通过实验结果看到误差，可以辅助更好地去进行比较理论性的一些东西的学习。
另外，实验主要包括这些方面的内容：误差的一些定义、牛顿法、错位法、割线法、二分法、LU 分解、Gauss-Seidel 迭代、Jacobi 迭代、拉格朗日插值多项式、曲线拟合、三次样条曲线、组合梯形公式、组合辛普森公式、欧拉方法、休恩方法、龙格库塔方法、幂法、对称幂法、反幂法。

Chapter 1

算法与程序 1

题目描述：
根据习题 12(改进二次根公式)和习题 13 构造算法和 MATLAB 程序，以便精确计算所有情况下的二次方程的根，包括$|b| \approx \sqrt{b^2 - 4ac}$的情况。

实验：
对习题 12 中的公式先给出简单证明。

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a} \qquad x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a} \qquad (1)$$

对上述两式分子有理化：

$$x_1 = \frac{(-b+\sqrt{b^2-4ac})(-b-\sqrt{b^2-4ac})}{2a} \qquad x_2 = \frac{(-b-\sqrt{b^2-4ac})(-b+\sqrt{b^2-4ac})}{2a}$$

化简即得到：

$$x_1 = \frac{-2c}{b+\sqrt{b^2-4ac}} \qquad x_2 = \frac{-2c}{b-\sqrt{b^2-4ac}} \qquad (2)$$

但是由于当$|b| \approx \sqrt{b^2 - 4ac}$时，两者相减会带来精度损失，所以应当避免两者相减，当$b < 0$时用(1)计算x_1，用(2)计算x_2，当$b > 0$时用(1)计算x_2，用(2)计算x_1。对于$b = 0$的情况就可以直接包含在其中。
Code:

```
function [x1,x2] = parabola(a,b,c)
if b > 0
    x1 = -2*c/(b+sqrt(b^2-4*a*c));
    x2 = (-b-sqrt(b^2-4*a*c))/2*a;
else
    x1 = (-b+sqrt(b^2-4*a*c))/2*a;
    x2 = -2*c/(b-sqrt(b^2-4*a*c));
end
end
```

算法与程序 2

题目描述：

参照例 1.25，对下列 3 个差分方程计算出前 10 个数值近似值。在每种情况下引入一个小的初始误差。如果没有初始误差，则每个差分方程将生成序列$\{\frac{1}{2^n}\}_{n=1}^{\infty}$。构造类似表 1.4、表 1.5 以及图 1.8 至 1.10 的输出。

(a)$r_0 = 0.994, r_n = \frac{1}{2}r_{n-1}$，其中 $n = 1,2,\ldots$

(b)$p_0 = 1, p_1 = 0.497, p_n = \frac{3}{2}p_{n-1} - \frac{1}{2}p_{n-2}$,其中$n = 2,3,\ldots$

(c)$q_0 = 1, q_1 = 0.497, q_n = \frac{5}{2}q_{n-1} - q_{n-2}$,其中$n = 2,3,\ldots$

实验：

先说明递推式的合理性。

(a)式是显然的一个合理的递推式。

(b)若$p_n = \frac{1}{2^n}$,则有$p_n = \frac{1}{2^n} = \frac{(3-2)}{2^n} = \frac{3}{2^n} - \frac{2}{2^n} = \frac{3}{2}p_{n-1} - \frac{1}{2}p_{n-2}$

(c)若$q_n = \frac{1}{2^n}$,则有$q_n = \frac{1}{2^n} = \frac{(5-4)}{2^n} = \frac{5}{2^n} - \frac{1}{2^{n-2}} = \frac{5}{2}q_{n-1} - q_{n-2}$

Code:

```
clear;clc;
n=[0;1;2;3;4;5;6;7;8;9;10];
for i = 0:10
    xn(i+1,1) = 1/2^i;
end
rn(1,1) = 0.994;
pn(1,1) = 1;
pn(2,1) = 0.497;
qn(1,1) = 1;
qn(2,1) = 0.497;
```

```
for i = 2:11
    rn(i,1) = rn(i-1,1)/2;
end
for i = 3:11
    pn(i,1) = pn(i-1,1)*3/2 - pn(i-2,1)/2;
end
for i = 3:11
    qn(i,1) = qn(i-1,1)*5/2 - qn(i-2,1);
end
filename = 'table1.xlsx';
T = table(n,xn,rn,pn,qn);
writetable(T,filename,'Sheet',1);

%Second table
filename = 'table2.xlsx'
T = table(n,xn-rn,xn-pn,xn-qn);
writetable(T,filename,'Sheet',1);
%Pictures
scatter(n,xn-rn);
legend("xn-rn");
figure;
scatter(n,xn-pn);
legend("xn-pn");
figure;
scatter(n,xn-qn);
legend("xn-qn");
%plot(n,xn);hold on;plot(n,pn);hold on;plot(n,qn);hold
on;plot(n,rn);legend("xn","pn","qn","rn")
```

代码思想。直接利用递推表达式，给定有误差之后的初值之后一直迭代。

输出结果。

Table 1

n	x_n	r_n	p_n	q_n
0	1	0.994	1	1
1	0.5	0.497	0.497	0.497
2	0.25	0.2485	0.2455	0.2425
3	0.125	0.12425	0.11975	0.10925
4	0.0625	0.062125	0.056875	0.030625
5	0.03125	0.0310625	0.0254375	-0.0326875
6	0.015625	0.01553125	0.00971875	-0.11234375
7	0.0078125	0.007765625	0.001859375	-0.248171875
8	0.00390625	0.003882813	-0.002070312	-0.508085938
9	0.001953125	0.001941406	-0.004035156	-1.022042969
10	0.000976563	0.000970703	-0.005017578	-2.047021484

Table 2

n	$x_n - r_n$	$x_n - p_n$	$x_n - q_n$
0	0.006	0	0
1	0.003	0.003	0.003
2	0.0015	0.0045	0.0075
3	0.00075	0.00525	0.01575
4	0.000375	0.005625	0.031875
5	0.0001875	0.0058125	0.0639375
6	9.375E-05	0.00590625	0.12796875
7	0.000046875	0.005953125	0.255984375
8	2.34375E-05	0.005976562	0.511992188
9	1.17188E-05	0.005988281	1.023996094
10	5.85938E-06	0.005994141	2.047998047

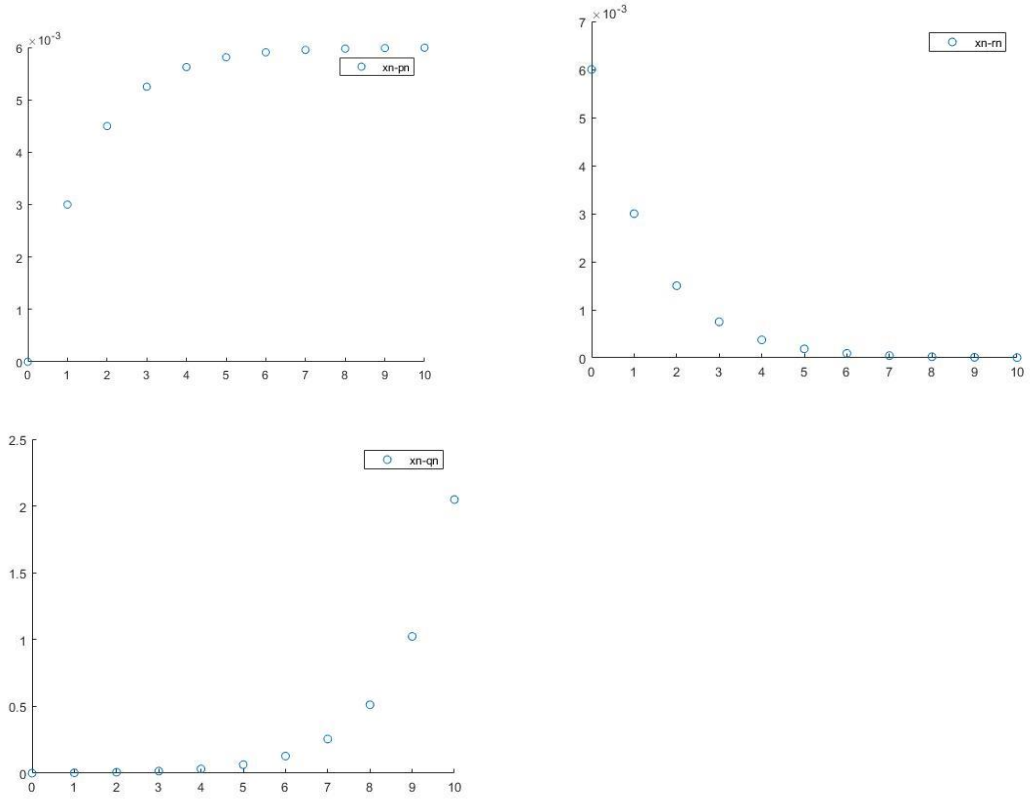


Figure 1

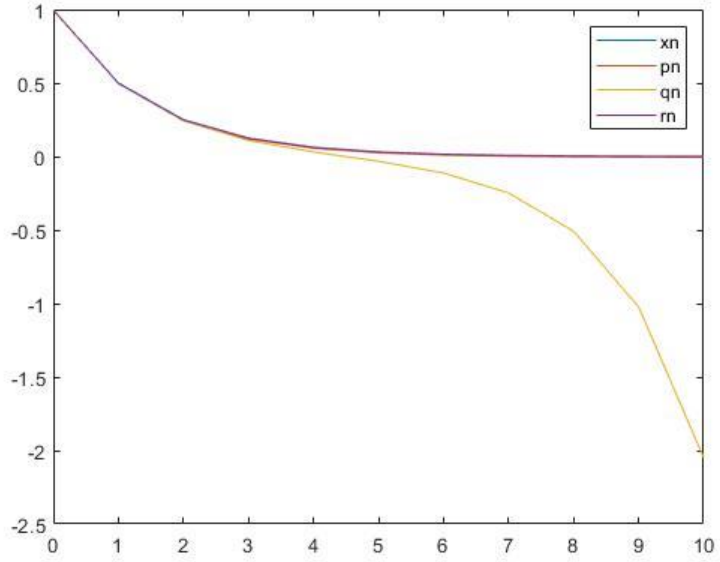


Figure 2

分析：

首先看 Table 1，发现r_n是最接近x_n的，而p_n和q_n直接看数值不是很明显可以看出来结果，Figure 2 就很好地显示出p_n和$\mathrm{r_n}$都可以较好地生成需要的序列，而q_n却离需要构造的序列真实值越来越远。

然后看 Table 2 和 Figure 1 可以发现r_n的误差从开始的一个初始误差开始，到后来越来越小，这一点可以比较容易证明：

$$x_n = \frac{1}{2^n}(x_0 - e_0)$$

e_0代表初始的误差，x_n代表想要生成的序列的真实值，序列整个的误差会在后面越来越小。

p_n的误差则迅速上升，然后基本稳定在某一个值上面，这一点也可以在 Figure 2 放大之后，发现p_n和x_n是一直有一定距离的。

q_n的误差则是越来越大，从 Figure 2 当中也可以看到，最后q_n和x_n完全分开了。

p_n和q_n的误差从迭代公式可以发现其实主要是因为误差前面的系数一个是$\frac{3}{2}$，一个是$\frac{5}{2}$，都是大于 1 的，所以可能误差会一直增大。

Chapter 2

Problem 1

题目描述：

利用牛顿法求解方程

$$\frac{1}{2}+\frac{1}{4}x^2-xsinx-\frac{1}{2}cos2x=0$$

分别取$x_0=\frac{\pi}{2},5\pi,10\pi$,使得精度不超过$10^{-5}$.比较初值对计算结果的影响。

实验：

初值可以改变的函数代码：

```
function [p1] = func2_1(p0)
f =@(x) 1/2 + x^2/4 - x*sin(x) - cos(2*x)/2;
df =@(x) x/2 - sin(x) - x*cos(x) + sin(2*x);
i = 0;
while(1)
    i = i  + 1;
    if(df(p0) == 0)
        fprintf("zero\n");
        break;
    end
     p1 = p0 - f(p0)/df(p0);
    err = abs(p1 - p0);
    if p1 == 0
        break;
    elseif err < 10^-5
        fprintf("迭代步数:%d\n",i);
        break;
    end
    p0 = p1;
end
end
```

分析：

牛顿法

假设给定p_0为初始点，如果$f\in C^2[a,b]$，在p_0点泰勒展开有：

$$f(x)=f(p_0)+f'(p_0)(x-p_0)+\frac{f''(c)(x-p_0)^2}{2!}$$

其中c位于x和p_0之间。将p带入上式，并且有$f(p)=0$有：

$$0=f(p)=f(p_0)+f'(p_0)(p-p_0)+\frac{f''(c)(p-p_0)^2}{2!}$$

因为p_0比较靠近p，忽略 2 次项：

$$0=f(p_0)+f'(p_0)(p-p_0)$$

所以有：

$$p=g(p_0)=p_0-\frac{f(p_0)}{f'(p_0)}$$

$$g(x)=x-\frac{f(x)}{f'(x)}$$

因为$f(p)=0$，所以$g(p)=p$，即将求根问题变成了求不动点问题。上式就是迭代的式子。
另外对其收敛性进行分析。对$g(x)$求导有：

$$g'(x)=1-\frac{f'(x)f'(x)-f(x)f''(x)}{\left(f'(x)\right)^2}=\frac{f(x)f''(x)}{(f'(x))^2}$$

在$x=p$附近的时候，因为$f(p)=0$，有

$$g'(p)=0$$

可以找到一个δ在区间$(p-\delta,p+\delta)$内，有$g'(p)<1$，根据不动点定理，如果$p_0\in(p-\delta,p+\delta)$,则最终会收敛到$f(x)=0$的一个根上去。
将代码函数改变初始值之后得到以下表格：

Table 3

初始值x_0	$\frac{\pi}{2}$	5π	10π
迭代次数	15	19	13052
零点	1.895488	1.895489	-1.895503
误差	10^{-5}	10^{-5}	10^{-5}

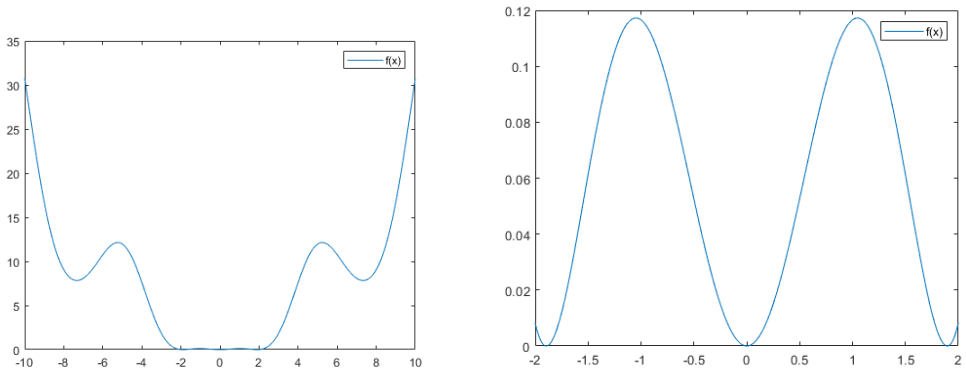


Figure 3

实现 Figure 3 的代码在附录 Code 1 中。

从图上大致看出有三个零点，从 Table 表中可以发现，$\frac{\pi}{2}$和5π都可以比较快地收敛到最大的那个零点，但是10π这个点就稍微迭代的步骤有点多，并且跳过了离10π最近的零点，收敛到了最远的一个零点，足以看出初值的选区对牛顿法的影响很大，而且函数本身从图片上可以看出来波动比较大，如果是一个波动不大的函数，例如

$$f(x)=x^2$$

这种波动不大的函数，用牛顿法对初值的依赖也只是影响迭代的步骤，而不会影响最后收敛到哪个点。
下面显示一下迭代过程在图上的点。

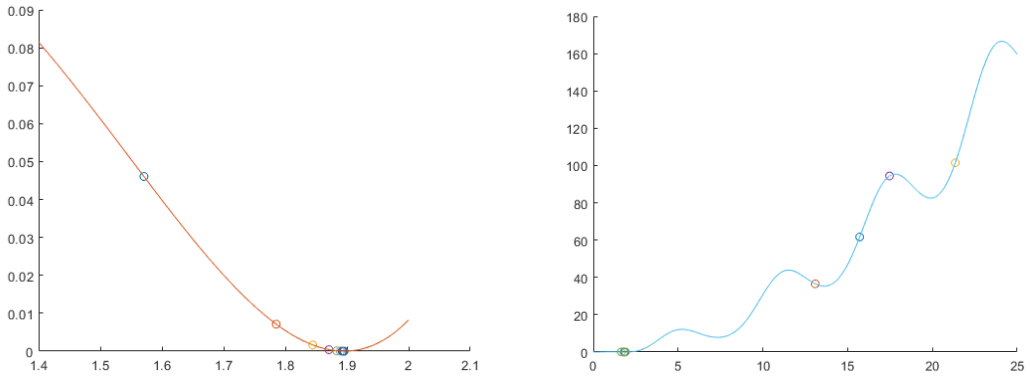


Figure 4

如果加速迭代过程，即将迭代公式变成

$$p_k = p_{k-1} - \frac{Mf(p_{k-1})}{f'(p_{k-1})}$$

其中 M 代表根的个数，在这里是 3，重新迭代得到下面结果：

初始值x_0	$\frac{\pi}{2}$	5π	10π
迭代次数	18	19	23
零点	1.8955	-2.3927e-06	1.6991e-06
误差	10^{-5}	10^{-5}	10^{-5}

从结果看出，只有在10π的时候结果明显好转，其他情况下并没有得到很好的好转。

Problem 2

题目描述：

已知

$$f(x) = 5x - e^x$$

在(0,1)之间有一个实根，试分别用二分法、牛顿法、割线法、错位法设计相应的计算格式，并编程求解(精确到 4 位小数)．

实验：

$f(0) = -1 < 0, f(1) = 5 - e > 0$，所以在(0,1)之间有一个实根。

Code：

二分法

```
clear;clc;
f1 = @(x) 5*x - exp(x);
df = @(x) 5 - exp(x);
x=0.2:0.01:0.6;
y=f1(x);
  %二分
```

```
    a = 0;
    b = 1;
    ya = f1(a);
    yb = f1(b);
    k = ceil(4*log(10)/log(2));
    cnt=0;
    for i = 1:k
        cnt=cnt+1;
        c = (a + b)/2;
        yc = f1(c);
        scatter(c,f1(c));
        hold on;
        if yc == 0
            result = c;
            break;
        elseif ya*yc > 0
            ya = yc;
            a = c;
        else
            yb = yc;
            b = c;
        end
    end
    result = c;
    fprintf("迭代步数:%d\n",cnt);
    plot(x,y);
```

牛顿法

```
clear;clc;
f1 = @(x) 5*x - exp(x);
df = @(x) 5 - exp(x);
%牛顿
    p0 = 0;
    i=0;
    x=0:0.01:0.3;
    y=f1(x);
    while(1)
        scatter(p0,f1(p0));
        hold on;
        i=i+1;
        p1 = p0 - f1(p0)/df(p0);
        err = abs(p1 - p0);
        if err < 10^-4
```

```
            fprintf("迭代步数:%d\n",i);
            break;
        end
        p0 = p1;
    end
    result = p0;
    plot(x,y);
```

割线法

```
clear;clc;
f1 = @(x) 5*x - exp(x);
df = @(x) 5 - exp(x);
%割线
    p0 = 0;
    p1 = 1;
    x=0.24:0.01:0.31;
    y=f1(x);
    i=0;
    while(1)
        i=i+1;
        p2 = p1 - f1(p1)*(p1-p0)/(f1(p1)-f1(p0));
        scatter(p2,f1(p2));
        hold on;
        err = abs(p2 - p1);
        if err < 10^-4
            fprintf("迭代步数:%d\n",i);
            break;
        end
        p0 = p1;
        p1 = p2;
    end
    result = p2;
    plot(x,y);
```

错位法

```
clear;clc;
f1 = @(x) 5*x - exp(x);
df = @(x) 5 - exp(x);
%错位 试值法
    p0 = 0;
```

```
    p1 = 1;
    i=0;
    while(1)
        i=i+1;
        p2 = p1 - f1(p1)*(p1-p0)/(f1(p1)-f1(p0));
        if f1(p2) == 0
            result = p2;
            break;
        elseif f1(p2)*f1(p1) > 0
            err = abs(p2 - p1);
            p1 = p2;
        elseif f1(p2)*f1(p0) > 0
            err = abs(p2 - p0);
            p0 = p2;
        end
        if err < 10^-4
            fprintf("迭代步数:%d\n",i);
            result = p2;
            break;
        end
    end
```

分析：

二分法：

如果$f \in C(a,b)$，且存在数$r \in [a,b]$，满足$f(r)=0$。如果$f(a)f(b)<0$则在区间[a,b]内有奇数个零点，若只有一个零点，可以递归用下述方法找到零点：

每次取中点z，若$f(z)=0$，则z就是零点。如果$f(z)f(a)<0$则区间变换到$[a,z]$，否则区间变为$[z,b]$。

从上述递归中容易知道，每次都将零点存在的区间缩小一倍，所以如果在区间$[a,b]$上，如果迭代n次，可以得到精度为：

$$\frac{b-a}{2^{n+1}}$$

并且如果假设零点为r，上式稍微改下一下可以得到：

$$|r-c_n| \le \frac{b-a}{2^{n+1}}$$

因此，当$n \to \infty$的时候，上式右边趋于 0，所以得到$r \to c_n$，所以只要n足够大，最后一定会收敛到目标点。

在真实函数上的迭代点的值如 Figure 4 所示。

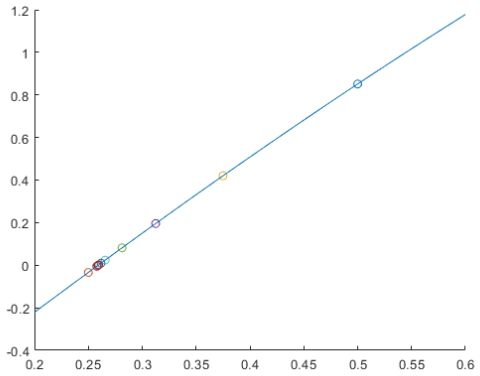


Figure 5

牛顿法。
在真实函数上的迭代点的值如 Figure 5 所示.
可以明显发现牛顿法收敛很快。从第一个点迭代了一次之后就很快就近真实零点了。

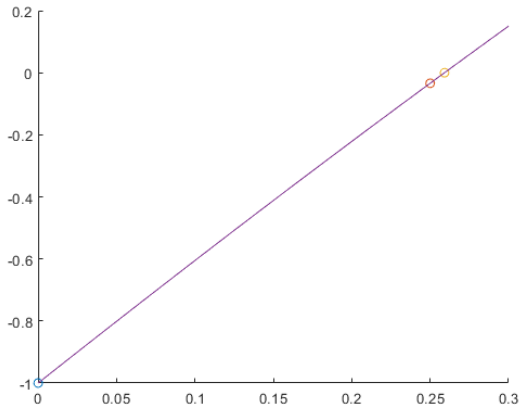


Figure 6

割线法。真实函数上的迭代点的值如 Figure 6 所示.图中在 0.26 点有两个很相近的点。图上总共有四个点，因为迭代了四步。

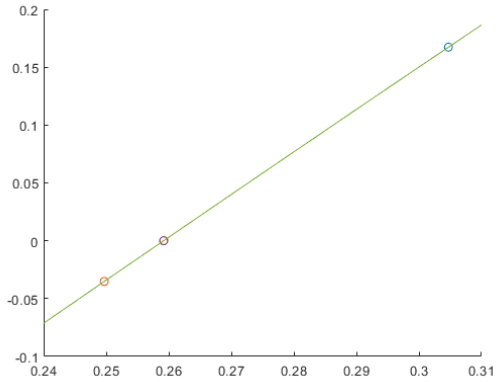


Figure 7

错位法。真实函数上的迭代点的值如 Figure 7 所示.

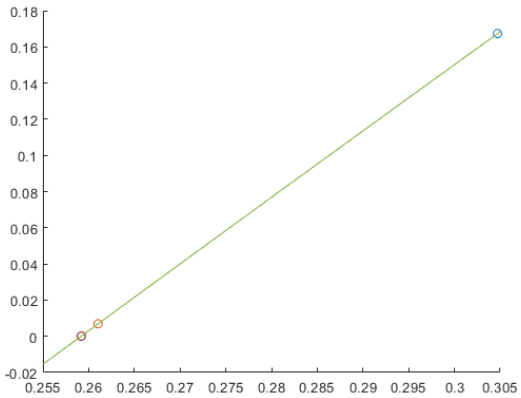


Figure 8

最后将上述几个方法进行一个总的论述。

Table 4

方法	二分法	牛顿法	割线法	错位法
迭代次数	14	3	4	4
零点	0.2592	0.2592	0.2592	0.2592
误差	10^{-4}	10^{-4}	10^{-4}	10^{-4}

从表格中可以看出，在相同精度的情况下，二分法需要的迭代次数最多，牛顿法需要的迭代次数最少，并且除二分法之外，另外三种方法基本第一次迭代之后都到了零点附近，区别就在于在零点附近之后的迭代速度。

Chapter 3

Problem 1

问题描述：

求解线性方程组

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

(1)试用 LU 分解求解此方程组.

(2)分别用 Jacobi,Gauss-Seidel 方法求解此方程组

实验：

(1)LU 分解：

给定一个无行交换的高斯消去法可求解一般线性方程组$AX = B$，则矩阵A可分解为一个下三角矩阵L和一个上三角矩阵U的乘积：

$$A = LU$$

而且L的对角线元素为 1，U的对角线元素非零。
得到LU矩阵之后，原方程变为：

$$LUX = B$$

这时可以先把UX当成新的未知数，即求解

$$LX_{new} = B$$

得到X_{new}之后，再求解

$$UX = X_{new}$$

最终得到X.
分解矩阵为 LU：
基本思想：利用已知求未知。最开始已知L的对角线全是 1，所以用L的第一行分别与U的第一列、第二列…第n列相乘，得到U的第一行。然后用L的第二行与U的第一列相乘求得L中第二行的所有元素，然后用L的第二行分别与U的所有列相乘求得U的第二行，然后一直重复这样的步骤，就可以得到分解的LU矩阵。
求解X_{new}：
用L的第一行乘以X_{new}的列，求得X_{new}的第一行，然后用L的第二行乘以X_{new}的列，求得X_{new}的第二行。重复这样的步骤就可以得到X_{new}的所有值。
求解X：
类似X_{new}的求法，但是是从U矩阵的最后一行向上分别和X的列相乘。
LU分解的代码和分解之后求解方程组的代码：

```
function [L,U] = LU(X,row)
%X 表示要进行 LU 分解的矩阵
%row 表示 X 的维数
U = zeros(row,row);
L=zeros(row,row);
for i = 1:row
    L(i,i) = 1;
end
for i = 1:row
    for j = 1:row
        if i > j    %求 L
            temp = 0;
            for k = 1:j-1
                temp = temp + L(i,k)*U(k,j);
            end
            L(i,j) = (X(i,j) - temp)/U(j,j);
        else  %求 U
            temp = 0;
            for k = 1:i-1
                temp = temp + L(i,k)*U(k,j);
            end
            U(i,j) = X(i,j) - temp;
        end
    end
end
```

```
end
```

```
function [X] = LUsolveX(L,U,B)
%L U 表示 LU 分解之后的下三角矩阵和上三角矩阵
%B 表示 Ax=B 中的 B
row = size(L,1);
X1 = zeros(row,1);
X = zeros(row,1);
for i = 1: row  %先求 X1
    temp = 0;
    for j = 1:i-1
        temp = temp + X1(j,1)*L(i,j);
    end
    X1(i,1) = (B(i,1)-temp)/L(i,i);
end
for i = row:-1:1
    temp = 0;
    for j = row:-1:i+1
        temp = temp + X(j,1)*U(i,j);
    end
    X(i,1) = (X1(i,1)-temp)/U(i,i);
end
end
```

利用这两个函数求解的脚本程序见附件 code 2.
分解之后的 LU 矩阵如下

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -0.5 & -0.0714 & 1 \end{pmatrix}$$

$$U = \begin{pmatrix} 4 & -1 & 1 \\ 0 & -7 & 0 \\ 0 & 0 & 5.5 \end{pmatrix}$$

最后解得的方程结果：

$$\begin{cases} x_1 = 2 \\ x_2 = 4 \\ x_3 = 3 \end{cases}$$

(2)
Jacobi 迭代：
例如对下面这组方程：

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = y_1$$
$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = y_2$$
$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = y_3$$

选定一组初值x_1、x_2、x_3之后，利用以下递推式求解：

$$x_1' = \frac{y_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$
$$x_2' = \frac{y_2 - a_{21}x_1 - a_{23}x_3}{a_{22}}$$
$$x_3' = \frac{(y_3 - a_{31}x_1 - a_{32}x_2)}{a_{33}}$$

Gauss-Seidel 迭代：

同样对 Jacobi 迭代中的方程进行说明用这样的递推式：

$$x_1 = \frac{y_1 - a_{12}x_2 - a_{13}x_3}{a_{11}}$$
$$x_2 = \frac{y_2 - a_{21}x_1 - a_{23}x_3}{a_{22}}$$
$$x_3 = \frac{(y_3 - a_{31}x_1 - a_{32}x_2)}{a_{33}}$$

相当于就是每次刚算出来的值立马就可以参加到对后面值的影响当中。

收敛条件：

当矩阵A具有严格对角优势的时候，Jacobi 迭代法和 Gauss-Seidel 方法都会收敛。

Jacobi 方法和 Gauss-Seidel 方法的函数调用代码如下：

```
function [x] = func3_12(x,opt)
%opt 为 1 时表示用 Gauss-Seildel 方法，为 2 时用 Jacobi 方法
err=10^(-5);
if opt == 1
    for i = 1:1000   %Gauss-Seidel
        temp=x;
        x(1,1) = (7 + x(2,1) - x(3,1))/4;
        x(2,1) = (21 + 4*x(1,1) + x(3,1))/8;
        x(3,1) = (15 + 2*x(1,1) - x(2,1))/5;
        if(abs(norm(temp-x))<err)
            fprintf("迭代次数:%d\n",i);
            fprintf("迭代结果:\n");
            x
            break;
        end
    end
elseif opt == 2  %Jacobi
    for i = 1:100
        z(1,1) = (7 + x(2,1) - x(3,1))/4;
        z(2,1) = (21 + 4*x(1,1) + x(3,1))/8;
        z(3,1) = (15 + 2*x(1,1) - x(2,1))/5;
        if(abs(norm(z-x))<err)
```

```
            fprintf("迭代次数:%d\n",i);
            fprintf("迭代结果:\n");
            z
            break;
        end
        x = z;
    end
end
end
```

利用这个函数的脚本程序见附件 Code 3.
首先感性认识一下，Jacobi 方法没有利用刚计算出来的值，而 Gauss-Seidel 方法利用了刚计算出来的值，Gauss-Seildel 方法对于相同的精度应该需要的迭代次数应该比 Jacobi 方法的要少，即收敛得更快。实际上因为有 Stein-Rosenberg 定理有：如果对每个$\mathrm{i} \neq \mathrm{j}, a_{ij} \leq 0$,且对每个$\mathrm{i} = 1,2,\ldots,\mathrm{n} \quad a_{ii} > 0$，那么一下结论有且只有一个成立：

$$0 \leq \rho(T_g) < \rho(T_j) < 1$$

$$1 < \rho(T_j) < \rho(\rho_g)$$

$$\rho(T_g) = \rho(T_j) = 0$$

$$\rho(T_g) = \rho(T_j) = 1$$

也就是说如果收敛，高斯赛德尔收敛会比雅可比收敛得更快，如果发散的话，高斯塞格尔也会比雅可比发散得更快。
运行结果如下表格(初始值:[1,2,2])：

Table 5

方法名	Jacobi	Gauss-Seidel
迭代数	12	8
$norm_1$	$< 10^{-5}$	$< 10^{-5}$
结果	$[2; 4; 3]$	$[2; 4; 3]$

结果也确实表明 Gauss-Seidel 方法对于同样的精度，所需要的迭代数比 Jacobi 方法的要少。

算法与程序 3

问题描述：
设有如下三角线性方程组，而且系数矩阵具有严格对角优势：

$$\begin{aligned} d_1x_1 + c_1x_2 &= b_1 \\ a_1x_1 + d_2x_2 + c_2x_3 &= b_2 \end{aligned}$$

$$
\begin{aligned}
a_2x_2 + d_3x_3 + c_3x_4 &= b_3 \\
&\cdots \\
&\cdots \\
&\cdots \\
a_{N-2}x_{N-2} + d_{N-1}x_{N-1} + c_{N-1}x_N &= b_{N-1} \\
a_{N-1}x_{N-1} + d_Nx_N &= b_N
\end{aligned}
$$

(1)根据方程组(9)、式(10)和式(11)，设计一个算法来求解上述方程组。算法必须有效地利用系数矩阵的稀疏性。

(2)根据(1)中设计的算法构造一个 MATLAB 程序，并求解下列三角线性方程组。

$$
\begin{aligned}
4m_1 + m_2 &= 3 \\
m_1 + 4m_2 + m_3 &= 3 \\
m_2 + 4m_3 + m_4 &= 3 \\
m_3 + 4m_4 + m_5 &= 3 \\
&\cdots \\
&\cdots \\
&\cdots \\
m_{48} + 4m_{49} + m_{50} &= 3 \\
m_{49} + 4m_{50} &= 3
\end{aligned}
$$

第二组：

$$
\begin{aligned}
4m_1 + m_2 &= 1 \\
m_1 + 4m_2 + m_3 &= 2 \\
m_2 + 4m_3 + m_4 &= 1 \\
m_3 + 4m_4 + m_5 &= 2 \\
&\cdots \\
&\cdots \\
&\cdots \\
m_{48} + 4m_{49} + m_{50} &= 1 \\
m_{49} + 4m_{50} &= 2
\end{aligned}
$$

实验：

(1)题目中说利用的两个式子就是指分别利用 Gauss-Seidel 方法和 Jacobi 方法。

将题目中的方程转化成系数矩阵会发现这个矩阵是一个，而且整个矩阵很有规律，并且具有严格对角优势，所以可以直接用三个向量来存储三个对角线的信息。

Code:

```
function [X] = func3_2i(A,D,C,B,opt)
%A、D、C 分别表示对角线信息
%B 表示结果矩阵
%opt 为 1 表示用 Jacobi 迭代
row = size(B,1);
X = zeros(row,1);
err=10^(-5);
```

```
if opt == 1 %Jacobi
    for i = 0:1000
        X2 = X;
        X(1,1) = (B(1,1) - C(1,1)*X2(2,1))/D(1,1);
        for j = 2:row-1
            X(j,1) = (B(j,1) - A(j-1,1)*X2(j-1,1) - C(j,1)*X2(j+1,1))/D(j,1);
        end
        X(row,1) = (B(row,1) - A(row-1,1)*X2(row-1,1))/D(row,1);
        if(abs(norm(X-X2))<err)
            fprintf("迭代次数:%d\n",i);
            fprintf("迭代结果:");
            X
            break;
        end
    end
elseif opt == 2 %Gauss-Seidel
    temp=X;
    for i = 0:1000
        X(1,1) = (B(1,1) - C(1,1)*X(2,1))/D(1,1);
        for j = 2:row-1
            X(j,1) = (B(j,1) - A(j-1,1)*X(j-1,1) - C(j,1)*X(j+1,1))/D(j,1);
        end
        X(row,1) = (B(row,1) - A(row-1,1)*X(row-1,1))/D(row,1);
        if(abs(norm(temp-X))<err)
            fprintf("迭代次数:%d\n",i);
            fprintf("迭代结果:");
            X
            break;
        end
    end
end
end
```

(2)利用第一题的函数，具体将这道题的系数输入之后得到结果.

Code:

```
clear;clc;
A = zeros(49,1);
C = A;
D = zeros(50,1);
B = D;
B2 = B;
for i = 1:49
```

```
    A(i,1) = 1;
    C(i,1) = 1;
end
for i = 1:50
    D(i,1) = 4;
    B(i,1) = 3;
end
X = func3_2i(A,D,C,B,1);
for i = 1:2:50
    B2(i,1) = 1;
    B2(i+1,1) = 2;
end
X2 = func3_2i(A,D,C,B2,1);
```

得到结果：

$$(a): x = [0.6340,0.4641,0.5096,0.4974,0.5007,0.4998,0.5000,0.5000, \ldots, 0.4998,0.5007,0.4974,0.5096,0.4641,0.6340]$$

$(b): x$

$= [0.1340,0.4641,0.00960.4974,0.0007,0.4998,0.0001,0.5000,0, \ldots,0.5000,0,0.5000]$

其中 Gauss-Seidel 迭代 18 次，Jacobi 方法迭代 19 次。

算法与程序 4

问题描述：

利用 Gause-Seidel 迭代法求解下列带状方程。

$$\begin{aligned}
12x_1 - 2x_2 + x_3 &= 5 \\
-2x_1 + 12x_2 - 2x_3 + x_4 &= 5 \\
x_1 - 2x_2 + 12x_3 - 2x_4 + x_5 &= 5 \\
x_2 - 2x_3 + 12x_4 - 2x_5 + x_6 &= 5 \\
\cdots & \\
x_{46} - 2m_{47} + 12x_{48} - 2x_{49} + x_{50} &= 5 \\
x_{47} - 2x_{48} + 12x_{49} - 2x_{50} &= 5 \\
x_{48} - 2x_{49} + 12x_{50} &= 5
\end{aligned}$$

实验：

综合前面一个实验可以知道这个系数也具有稀疏性 。

Code：

```
clear;clc;
X = zeros(50,1);
err=10^(-5);
cnt=0;
```

```
for i =0:100
    cnt=cnt+1;
    temp=X;
    X(1,1) = (5-X(3,1)+2*X(2,1))/12;
    X(2,1) = (5-X(4,1)+2*X(3,1)+2*X(1,1))/12;
    for j=3:48
        X(j,1)=(5-X(j-2,1)-X(j+2,1)+2*X(j+1,1)+2*X(j-1,1))/12;
    end
    X(49,1)=(5+2*X(50,1)+2*X(48,1)-X(47,1))/12;
    X(50,1)=(5+2*X(49,1)-X(48,1))/12;
    if norm(X-temp)<err
        fprintf("迭代次数:%d\n",cnt);
        fprintf("最后结果:\n");
        X
        break;
    end
end
```

最后求解得到的结果为:

$$x = [0.4638,0.5373,0.5090,0.4982,0.4989,0.5,0.5001,0.5 \ldots,0.5,0.5001,$$
$$0.5000,0.4989,0.4982,0.5090,0.5373,0.4683]$$

迭代次数:8 次。

Chapter 4

Problem 1

问题描述:

在区间$[-5,5]$上，生成 11 个等距插值节点，$x_i, i = 0,1,\ldots,10$.在相应插值节点上计算函数

$$y(x) = 1/(1+x^2)$$

的函数值作为观测值，$y(x_i), i = 0,1,2,\ldots,10.$

(1)利用这 11 个数据点，生成一个 10 次拉格朗日插值多项式$P_{10}(x)$，并做出插值函数与原函数的对比结果图。

(2)利用此多项式近似计算

$$\int_{-5}^{5} \frac{1}{1+x^2} dx \approx \int_{-5}^{5} P_{10}(x) dx$$

与解析解比较,分析误差的产生原因。

(3)利用$\{(x_i, y(x_i))\}_{i=0}^{10}$,构造分段线性插值多项式$P(x)$，并利用此分段插值多项式近似计算积分

$$\int_{-5}^{5}\frac{1}{1+x^2}dx \approx \int_{-5}^{5}P(x)dx$$

与解析解比较，分析误差产生原因。

(4)若希望提高积分的计算精度,试提出你自己的建议，并进行实验测试验证。

实验：

拉格朗日逼近：

先可以从线性的插值开始，也就是直接将两个点用线段连接起来。对点(x_0,y_0)和(x_1,y_1)，用点斜式可以将过该两点的直线表达出来：

$$y = y_0 + \frac{(y_1-y_0)(x-x_0)}{(x_1-x_0)}$$

将上式稍微变化一下可以得到：

$$y = y_0\frac{x-x_1}{x_0-x_1} + y_1\frac{x-x_0}{x_1-x_0}$$

如果将其再稍微改变一下：

$$L_{1,0}(x) = \frac{(x-x_1)}{x_0-x_1}, \qquad L_{1,1} = \frac{x-x_0}{x_1-x_0}$$

$$y = y_0L_{1,0} + y_1L(1,1)$$

写成这种形式是为了更好地推广这种形式，上式x的指数都还只是 1，如果推广到多项，就有：

$$P_N(x) = \sum_{k=0}^{N} y_kL_{N,k}(x)$$

其中

$$L_{N,k} = \frac{(x-x_0)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N)}{(x_k-x_0)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_N)}$$

很容易看出右式的规律，即分子没有$(x-x_k)$和分母没有(x_k-x_k)，所以规律性的总结一下有：

$$L_{N,k}(x) = \prod_{j=0,j\neq k}^{N}\frac{x-x_j}{x_k-x_j}$$

拉格朗日的误差逼近项$E_N(x)$形式：

$$E_N(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_N)f^{N+1}(c)}{(N+1)!}$$

首先求出 10 次拉格朗日多项式，然后在图上和原函数一起画出来。下面的代码是可以求任意次的多项式：

Code:

```
function s = lagran(jie)
%jie 是需要求几阶多项式
syms m
f = 1./(1+m.^2);
step=10/jie;
x = -5:step:5;
x = x';
```

```
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:jie+1
    P = 1;
    for j = 1:jie+1
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
end
```

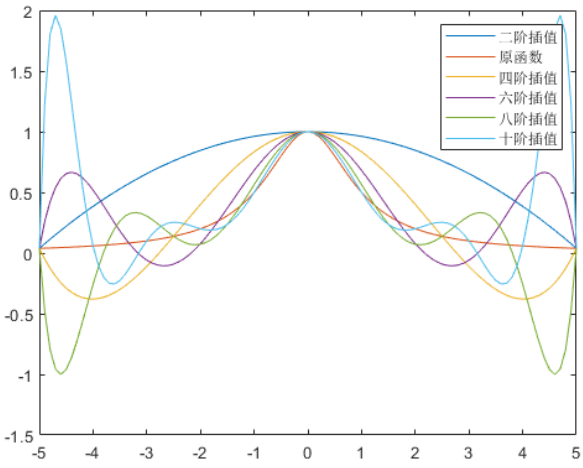


Figure 9

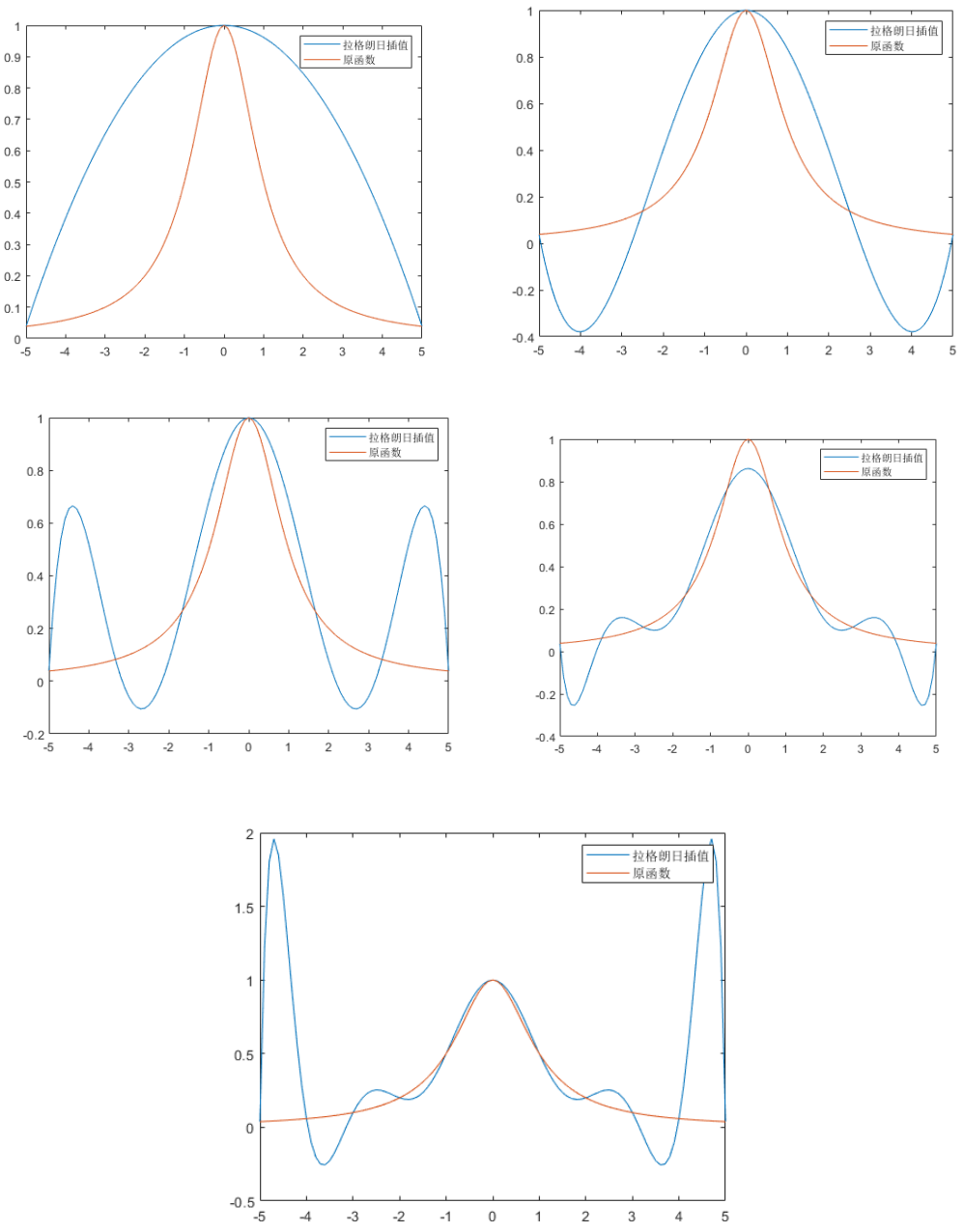


Figure 10

画图所用的代码在附件 Code 4 中 Figure 8 显示了多个阶的情况。Figure 9 将这些单独拿出来了，然后在这些插值多项式逼近当中，会发现阶数越大极大极值点的个数越多。

误差公式为：

$$E_N(x) = \frac{(x-x_0)(x-x_1)\dots(x-x_N)f^{N+1}(c)}{(N+1)!}$$

针对 10 次拉格朗日插值进行分析。

直接将准确值和估计值进行相减可以比较直观看出误差：

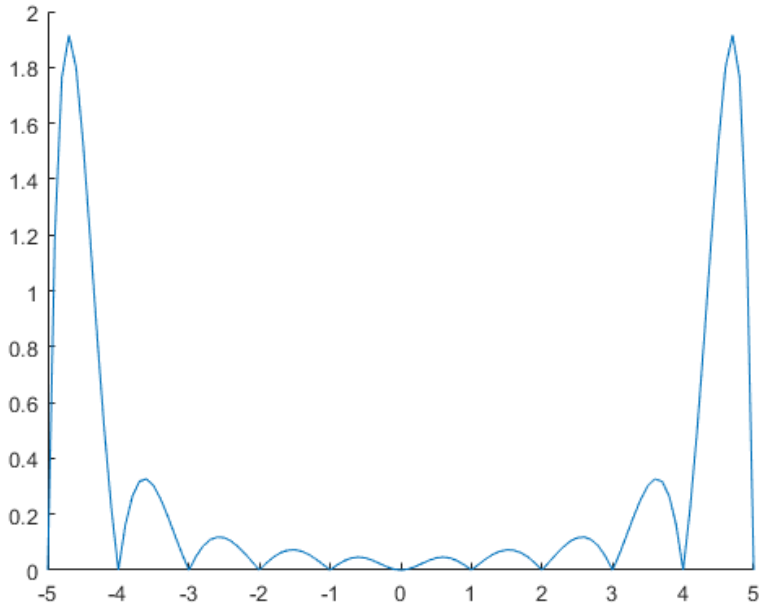


Figure 11

可以看出离$x=0$这个点越远，误差越大。

$$\max\left(f^{N+1}(c)\right)=3.6289\times10^{7}$$

$$\max\left((x-x_0)(x-x_1)\dots(x-x_N)\right)=4.1655\times10^{5}$$

所以

$$E_N(x)\le 3.6289\times4.1644\times10^{12}=3.7869\times10^{5}$$

直接用余项作为误差项的出来的误差很大。误差估计代码见附录 Code 8.下面为多项式$(x-x_0)(x-x_1)\dots(x-x_N)$画出来的图。

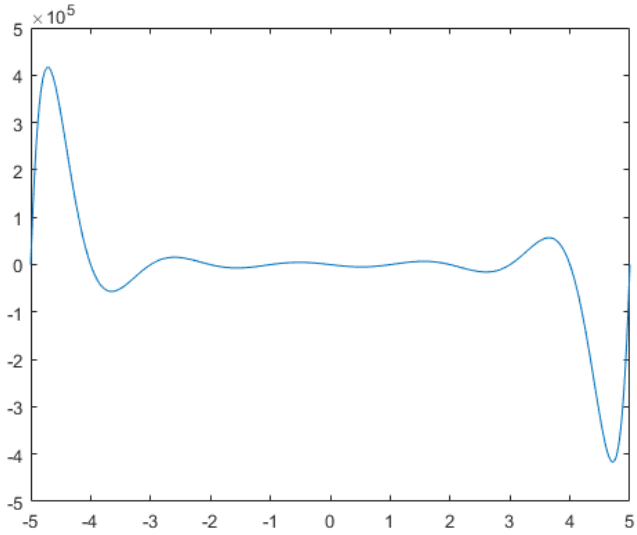


Figure 12

积分：

10 次拉格朗日插值的积分：4.6733，准确积分：2.7468。误差很大，观察之前插值曲线的逼近情况，发现在 0 附近逼近还比较可以，但是在两端的时候插值函数波动很大，这是龙格现象，当等距节点数目增加的时候，震荡会变得很剧烈。

分段线性插值：

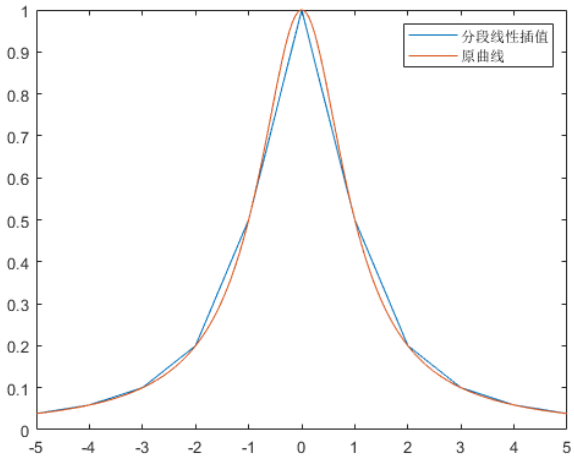
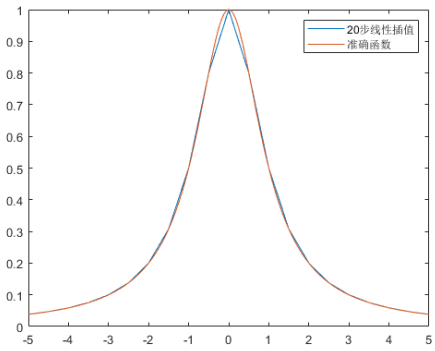
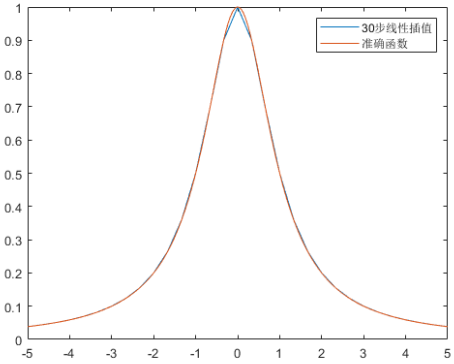


Figure 13

分段线性插值积分： 2.7561，准确积分：2.7468。误差主要产生在于线段和曲线之间不重合，有一点空隙，当分段越来越细，误差会越来越小。这就是*Matlab*画曲线的方法，取得步长越来越小，看上去的图形就是曲线。

分段数	估计积分	准确积分	误差（准确－估计）
10	2.756109	2.746802	-0.0093
20	2.746208	2.746802	0.000594
30	2.746528	2.746802	0.000274
40	2.746648	2.746802	0.000154
50	2.746703	2.746802	0.000099
60	2.746733	2.746802	0.000069





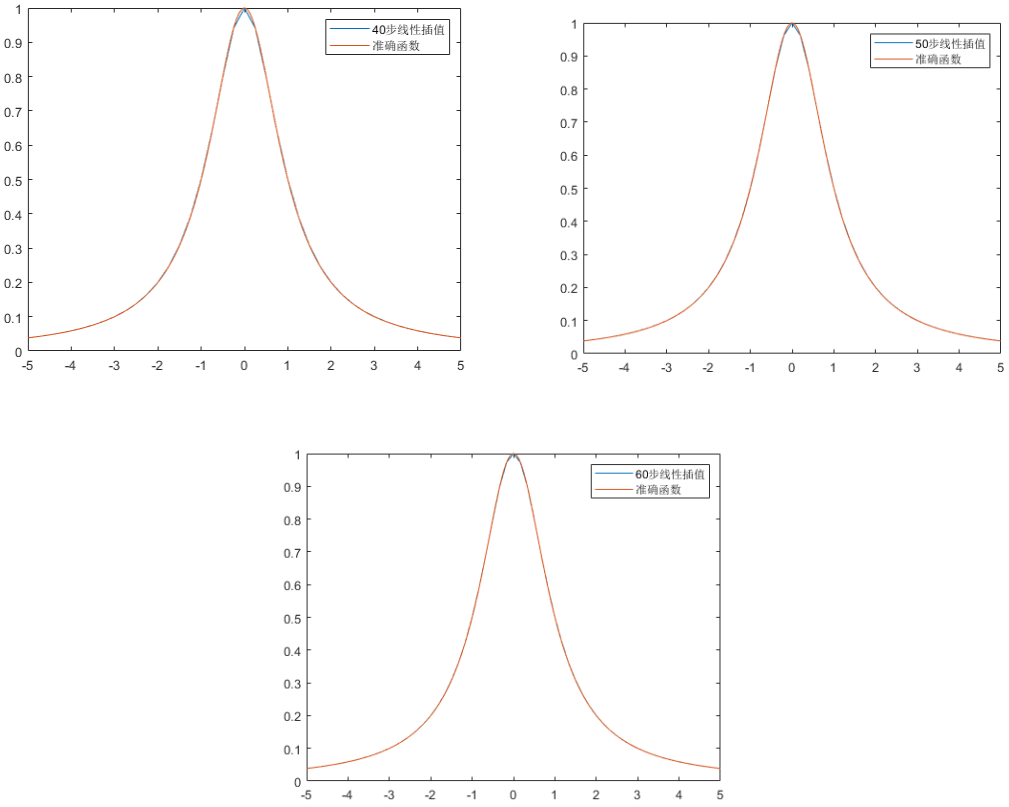


Figure 14

从表格和图像都可以看出积分随着步数的增加越来越精确，误差越来越小，但是到后来误差改善不明显，在图像上看出来主要是在$x = 0$这点附近的积分不是很准确，所以最好还是在$x = 0$这一点附近多取几个插值点，而不是所有点都采用等距的插值点。

Chapter 5

算法与程序 2

问题描述：

根据下列数据，使用下面的幂曲线拟合，写一个程序求解重力常量 g。

$$d = \frac{1}{2}gt^2$$

(a)

时间t_k	距离d_k
0.200	0.1960
0.400	0.7835
0.600	1.7630
0.800	3.1345
1.000	4.8975

(b)

时间t_k	距离d_k
0.200	0.1965
0.400	0.7855
0.600	1.7675
0.800	3.1420
1.000	4.9095

实验：

最小二乘拟合的幂函数拟合形式：

$$y = Ax^M，\ M已知$$

使用最小二乘计数，需要求函数$E(A)$的最小值：

$$E(A) = \sum_{k=1}^{N}(Ax_k^M - y_k)^2$$

高中的办法是用了一些数学技巧(而且只是针对线性函数)，学了微积分之后就可以直接把A当作未知数，其他参数当作已知数，然后对A求偏导数：

$$E'(A) = 2\sum_{k=1}^{N}(Ax_k^M - y_k)(x_k^M) = 2\sum_{k=1}^{N}(Ax_k^{2M} - x_k^M y_k)$$

求解上式可以得到：

$$A = \frac{\sum_{k=1}^{N} x_y^M y_k}{\sum_{k=1}^{N} x_k^{2M}}$$

至此原表达式已没有未知数。

Code：

```
clear;clc;
tk1=[0.200;0.400;0.600;0.800;1.000];
dk1=[0.1960;0.7835;1.7630;3.1345;4.8975];
tk2=[0.200;0.400;0.600;0.800;1.000];
dk2=[0.1965;0.7855;1.7675;3.1420;4.9095];
dktk21=zeros(5,1);
tk41=zeros(5,1);
dktk22=zeros(5,1);
tk42=zeros(5,1);
for i = 1:5
    dktk21(i,1)=dk1(i,1)*tk1(i,1)*tk1(i,1);
    dktk22(i,1)=dk2(i,1)*tk2(i,1)*tk2(i,1);
    tk41(i,1)=tk1(i,1)^4;
    tk42(i,1)=tk2(i,1)^4;
end
g1= 2*sum(dktk21)/sum(tk41);
```

```
g2=2*sum(dktk22)/sum(tk42);
fprintf("g1:%d\n",g1);
fprintf("g2:%d\n",g2);
```

求到的结果：

$$g_1 = 9.795020$$
$$g_2 = 9.818973$$

Table 6

	g_1	g_2
最大误差	1.0368e-4	1.2053e-4
平均误差	8.1716e-5	6.1593e-5
均方根误差	8.9260e-5	7.3924e-5

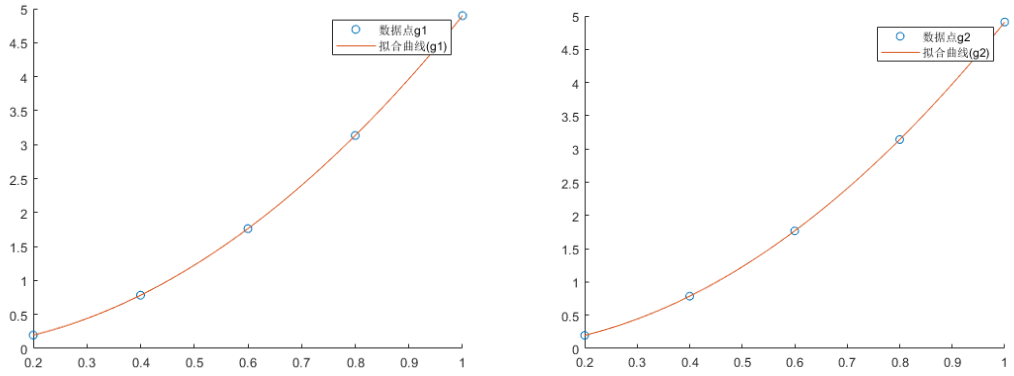


Figure 15

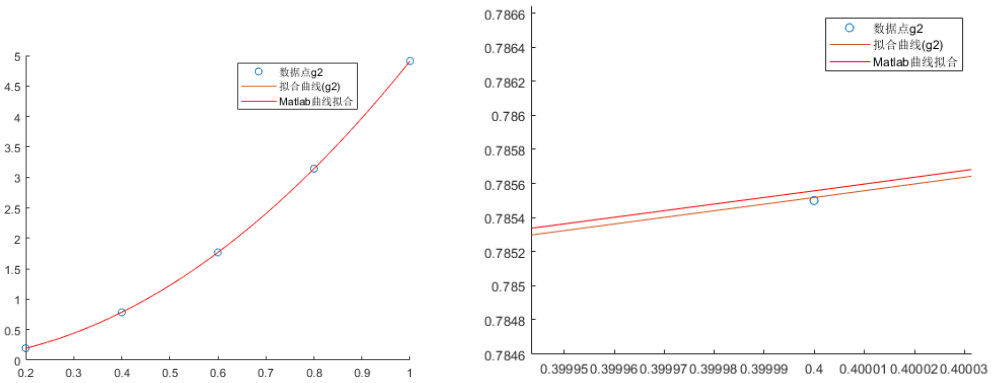


Figure 16

左图是*Matlab*的多项式拟合曲线和自己写的最小二乘幂函数拟合，结果非常相近，右图是局部放大图。

算法与程序 3

问题描述：

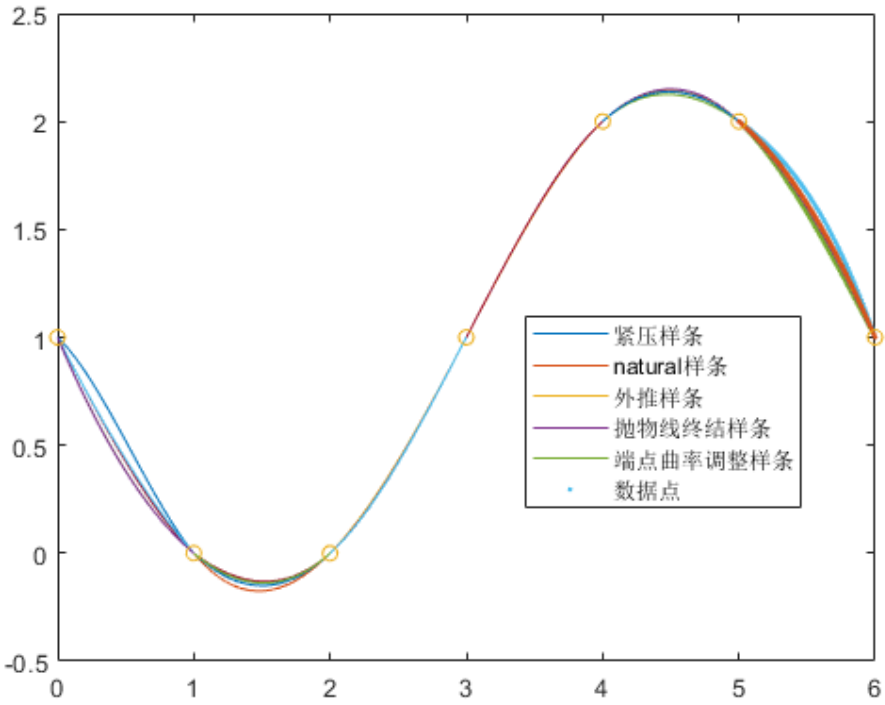
使用上题中的程序，根据点(0,1),(1,0),(2,0)(3,1)(4,2)(5,2)和(6,1)，求 5 种不同的三次样条插值，其中$s'(0)=-0.6, s'(6)=-0.8$，$s''(0)=1$和$s''(6)=-1$。在同一坐标系中，画出这 5 个三次样条插值和这些数据点。

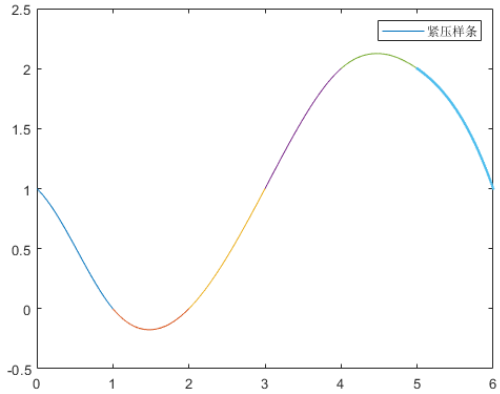
实验：

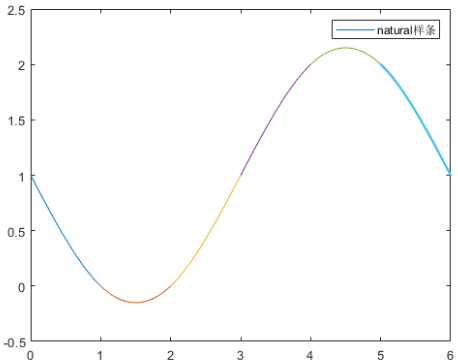
这五种三次样条曲线的绘制的代码就是在解方程处有一点不同，所以在正文当中显示一下 natural 样条的代码，其他的代码详细见附录 Code 5.

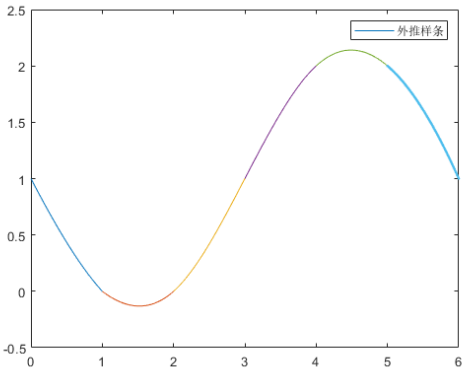
Code:

```
function S = natural(X,Y)
N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);
for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);
for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=0;
M(N+1)=0;
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
end
```


紧压样条
natural样条
外推样条
抛物线终结样条
端点曲率调整样条
数据点


紧压样条


natural样条


外推样条

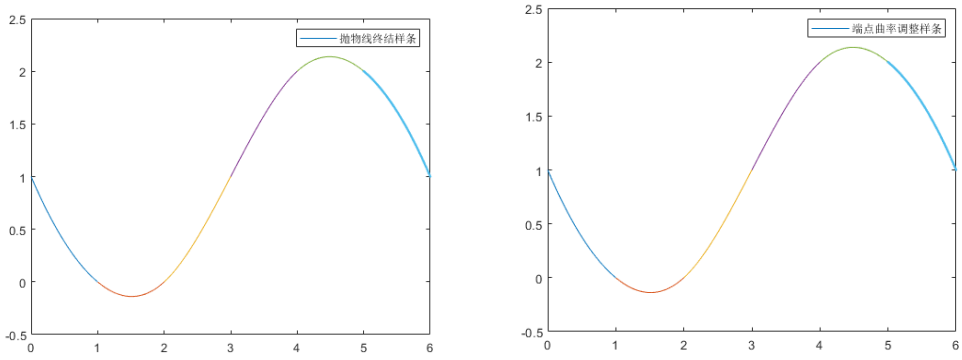


Figure 17

Chapter 6

算法与程序 1(b)

问题描述：
用程序 6.1 求解下列函数在x处的导数近似值，精度为小数点后 13 位。注：有必要改写程序中的$max1$的值和h的初始值。

$$f(x)=\tan\left(\cos\left(\frac{\sqrt{5}+\sin(x)}{1+x^2}\right)\right);x=\frac{1+\sqrt{5}}{3}$$

实验：
Code：

```
function [L,n] = difflim(f,x,toler)
%x 是要求导数的点
%toler 是容差值
%H 是步长序列
%D 是近似导数
%E 是误差边界
max1=15;
h=1;
H(1)=h;
D(1)=((f(x+h)-f(x-h))/2*h);
E(1)=0;
R(1)=0;
for n=1:2
    h=h/10;
    H(n+1)=h;
    D(n+1)=((f(x+h)-f(x-h))/2*h);
    E(n+1)=abs(D(n+1)-D(n));
```

```
    R(n+1)=2*E(n+1)-(abs(D(n+1))+abs(D(n))+eps);
end
n=2;
while (E(n)>E(n+1))&(R(n)>toler)&n<max1
    h=h/10;
    H(n+2)=h;
    D(n+2)=(f(x+h)-f(x-h))/2*h;
    E(n+2)=abs(D(n+2)-D(n+1));
    R(n+2)=2*E(n+2)-(abs(D(n+2))+abs(D(n+1))+eps);
    n=n+1;
end
n=length(D)-1;
L=[H' D' E'];
end
```

使用极限的微分求解。在研究求解函数$f(x)$的导数的近似值的过程，$f(x)$的导数可以表示为：

$$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

非常理想情况下就是直接选取一个序列$\{h_k\}$，并且序列极限为 0，然后计算序列的极限：

$$D_k=\frac{f(x+h_k)-f(x)}{h_k},\qquad k=1,2,\dots,n,\dots$$

但是在计算机求解的过程中，序列$\{h_k\}$在趋近于 0 的时候，$f(x+h)-f(x)$同样也会趋近于 0，而且$f(x+h)-f(x)$的差值会和之前求抛物线的根一样的情况，会损失精度，使得求出的导数值不是想要的精度，所以并不是迭代的步骤越多越好，需要及时终止。

具体求解过程：

Table 7

h	导数值	E
1	0.944508969133127	0
0.1	0.0122912035588668	0.932217765574261
0.01	0.000122860294102802	0.0121683432647640
0.001	1.22859747858110e-06	0.000121631696624220
0.0001	1.22859742392997e-08	1.21631150434180e-06
1e-05	1.22859742340137e-10	1.21631144969596e-08
1e-06	1.22859742318626e-12	1.21631144916951e-10
1e-07	1.22859742365811e-14	1.21631144894968e-12
1e-08	1.22859741269465e-16	1.21631144953116e-14
1e-09	1.22859759865701e-18	1.21631143670808e-16

利用$Matlab$函数求出精确的导数：

$$f'(x) = -sin(\frac{sin(x)+\sqrt{5}}{x^2+1}) * (tan\left(cos\left(\frac{sin(x)+\sqrt{5}}{x^2+1}\right)\right)^2 + 1)$$

$$* \left(\frac{cos(x)}{x^2+1} - \frac{2x\left(sin(x)+\sqrt{5}\right)}{(x^2+1)^2}\right)$$

求得的精确值为 1.2285974233。

Chapter 7

算法与程序 2

问题描述：
用程序 7.2 求习题 2 中的定积分，精确到小数点后 11 位。
实验：
辛普森公式：
用拉格朗日逼近多项式来求积分。

$$P_2(x) = \frac{f_0(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + \frac{f_1(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + \frac{f_2(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

用$P_2(x)$来逼近求积分，就是对$p_2(x)$求积分：

$$\int_{x_0}^{x_2} f(x)dx \approx f_0 \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}dx + f_1 \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}dx$$

$$+ f_2 \int_{x_0}^{x_2} \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}dx$$

变量替换：$x = x_0 + ht$，于是有$dx = hdt$，上式变为：

$$f_0 \int_0^2 \frac{h(t-1)h(t-2)h}{(-h)(-2h)}dt + f_1 \int_0^2 \frac{h(t-0)h(t-2)h}{h(-h)}dt$$

$$+ f_2 \int_0^2 \frac{h(t-0)h(t-1)h}{(2h)h}dt$$

$$= \frac{h}{3}(f_0 + 4f_1 + f_2)$$

梯形公式：
梯形公式可以看作是梯形求面积的公式，上底加下底乘高除以二。
梯形公式的误差$E_T(f,h)$为：

$$E_T(f,h) = -\frac{(b-a)f^{(2)}(c)h^2}{12}$$

辛普森公式误差$E_S(f,h)$为：

$$E_S(f,h) = -\frac{(b-a)f^{(4)}(c)h^4}{180}$$

组合辛普森公式 Code：

```
function s = simprl(f,a,b,M)
%a、b 表示边界
h=(b-a)/(2*M);
s1=0;
s2=0;
for  k=1:M
    x=a+h*(2*k-1);
    s1=s1+f(x);
end
for k=1:(M-1)
    x=a+h*2*k;
    s2=s2+f(x);
end
s=h*(f(a)+f(b)+4*s1+2*s2)/3;
end
```

组合梯形公式 Code：

```
function s = traprl(f,a,b,M)
%a、b 表示函数边界
h=(b-a)/M;
s=0;
for i=1:(M-1)
    x=a+h*i;
    s=s+f(x);
end
s=h*(f(a)+f(b))/2+h*s;
end
```

习题 2 中曲线长的公式为：

$$\text{长度} = \int_a^b \sqrt{1+f'(x)^2}dx$$

$$f_1(x) = x^3, 0 \le x \le 1$$

$$f_2(x) = \sin(x), 0 \le x \le \frac{\pi}{4}$$

$$f_3(x) = e^{-x}, 0 \le x \le 1$$

将上述几个式子分别积分转化成长度之后：

$$f_1(x) = \sqrt{1+9x^4}$$

$$f_2(x) = \sqrt{1+\cos^2(x)}$$

$$f_3(x) = \sqrt{1+e^{-2x}}$$

调用自己写的函数以及利用$Matlab$标准的int函数算出积分之后得到下表.

Table 8

方法	$f_1(x)$	$f_2(x)$	$f_3(x)$
组合辛普森	1.54786419399	1.05809581807	1.19270185509
组合梯形	1.56684521697	1.05725480798	1.19463323503
$Matlab$ int函数	1.54786565468	1.05809550139	1.19270140197

可以看出组合辛普森得到的结果和标准结果更接近。

算法与程序 3

问题描述:
修改组合梯形公式，使之可以求只有若干点函数值已知的函数积分。将程序 7.1 修改为求区间$[a,b]$上过M个给定点的函数$f(x)$的积分逼近。注意节点不需要等距。

利用该程序求过点$\{\sqrt{k^2+1}, k^{\frac{1}{3}}\}_{k=0}^{13}$的函数的积分逼近。

实验:
修改组合梯形公式只需要不用固定步长去求每个区间的面积，而是每次用给定的端点带入梯形公式去求。即每一步的迭代变成:

$$s = s + \frac{(y_{i+1}+y_i)(x_{i+1}-x_i)}{2}$$

将序列修改一些可以得到对应的函数:

$$y = (x^2-1)^{\frac{1}{6}}, x \geq 1$$

```
function s = traprl2(x,y)
s=0;
for i=1:size(x,2)-1
    s=s+(y(i+1)+y(i))*(x(i+1)-x(i))/2;
end
fprintf("改进组合梯形公式求得面积:%f\n",s);
end
```

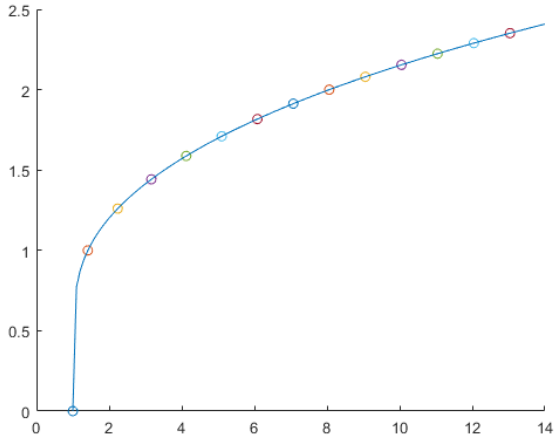


Figure 18

图上的点是不定长的端点。

修改之后的 **Code:**

不定步长和定步长之间的区别可以类比一下自适应算法，只是这里的不定步长没有什么依据，自适应算法的不定步长是含有误差判断在其中的。误差主要出现在区间[1,2]中。

Table 9

方法	积分
精确值	22.0094
修改组合梯形公式	21.8411

Chapter 9

Problem 1

问题描述:

流行病模型。流行病的数学模型描述如下：设有L个成员的构成的群落，其中有P个感染个体，Q为未感染个体。令$y(t)$表示时刻t感染个体的数量。对于温和的疾病，如普通感冒，每个个体保持存活，流行病从感染者传播到未感染者。由于两组间有PQ种可能的接触，$y(t)$的变化率正比于PQ。故该问题可以描述为初值问题:

$$y' = ky(L - y) \qquad y(0) = y_0$$

(a)用$L = 25000, k = 0.00003, h = 0.2$和初值条件$y(0) = 250$，并用程序 9.1 计算[0,60]上的欧拉近似解。

(b)画出(a)中的近似解。

(c)通过求(a)中欧拉方法的纵坐标平均值来估计平均感染个体的数目。

(d)通过曲线拟合(a)中的数据，并用定理 1.10(积分均值定理)，估计平均感染个体的数目。

实验：

欧拉方法：

欧拉方法就是泰勒级数法的一种特殊情况。即可以有下述泰勒展开式：

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(\xi)}{2}(x - x_0)^2$$

其中$x_0 \le \xi \le x$，用步长改写上面方程式，并且忽略掉误差项有：

$$f(x) = f(x_0) + hf'(x_0)$$

其中$f'(x_0)$已经在方程最开始给过了，所以只需要将式子带进去就行，而这样的式子在几何上的意义其实就是用切线逼近曲线。

欧拉方法在[0,60]上的欧拉近似解：

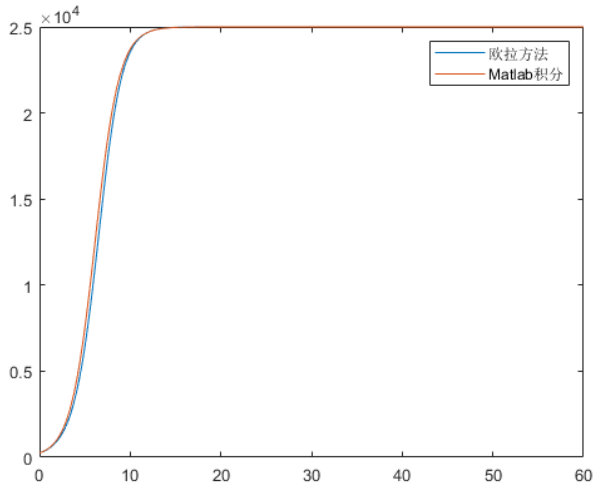


Figure 19

Figure 11 中蓝色的曲线是用欧拉方法求出来的曲线，红色曲线使用$Matlab$方法求出来的解：

$$y = \frac{25000}{e^{\ln(99) - \frac{3t}{4}} + 1}$$

Code：

```
1.  y=dsolve('Dy=0.00003*y*(25000-y)','y(0)=250')
```

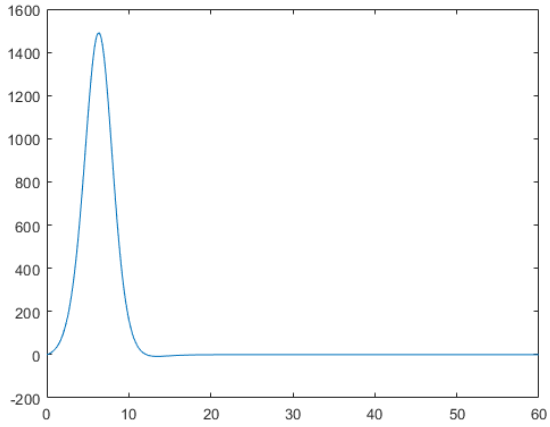


Figure 20 **误差(真实值-估计值)**

从上图可以看出误差主要出现在曲线比较陡峭的地方，而当曲线本身比较平缓的时候，欧拉方法的估计误差就比较小。

如果单纯只用欧拉方法的纵坐标的平均值来估计平均感染个体的数目，得到的结果是 22301.559616.

Table 10

积分方法	积分S	$\frac{S}{b-a}$
精确值	1346494.32	22441.57
梯形求积分	1340028.88	22333.81

如果用积分的方法，估计到的平均感染个体数目是 223333.81。
曲线在外形上有点像人口增长曲线，也就是曲线：

$$y = \frac{L}{1 + C * e^{At}}$$

这样的形式，但是在$Matlab$上面用这样的参数进行非线性拟合效果并不好。

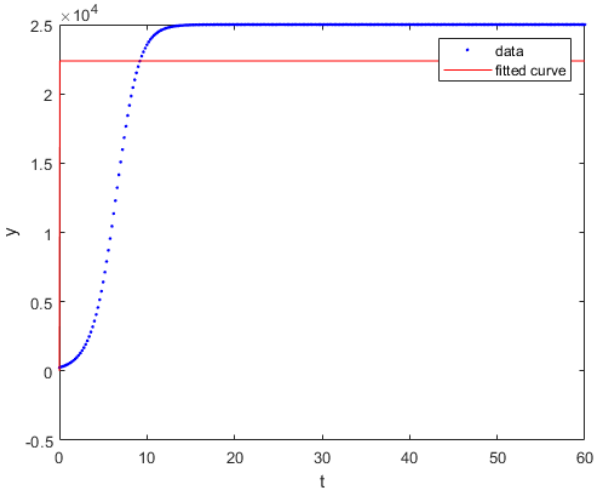


Figure 21

然后用$Matlab\ \ cfitool$工具箱里面的$Smooth\ \ Spline$拟合之后画出曲线和欧拉方法比较：

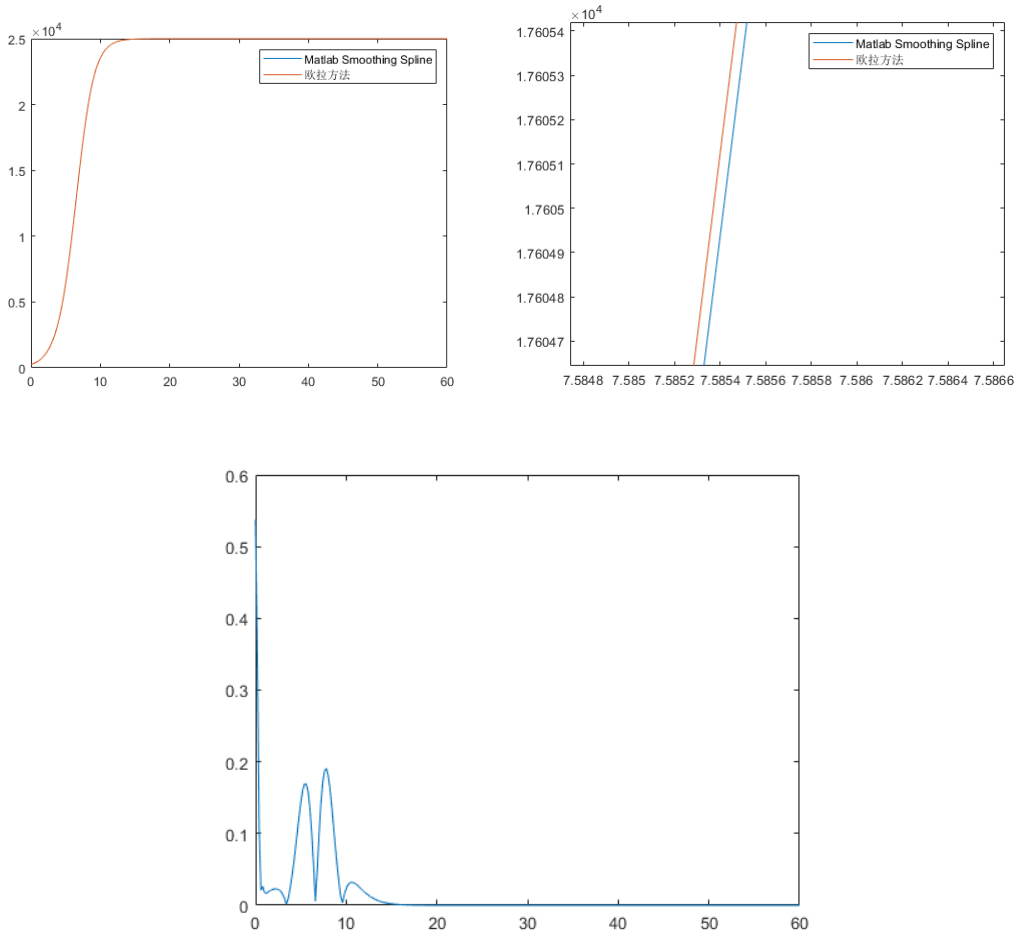


Figure 22

右上角图片是左上角图片的局部放大。
最下面一张图是欧拉方法和$Matlab$工具箱拟合曲线的函数值相减的绝对值，可以看出欧拉方法的效果和工具箱的效果还是比较接近。

Problem 2

问题描述：
考虑一阶积分-常微分方程

$$y' = 1.3y - 0.25y^2 - 0.0001y\int_0^t y(\tau)d\tau$$

(a)在区间[0,20]上，用欧拉方法和$h = 0.2, y(0) = 250$以及梯形公式求方程的近似解。
提示：欧拉方法的一般步长可以修改为

$$y_{k+1} = y_k + h(1.3y_k - 0.25y_k^2 - 0.0001y_k\int_0^{t_k} y(\tau)d\tau)$$

如果梯形公式用于逼近积分，则该表达式为

$$y_{k+1} = y_k + h(1.3y_k - 0.25y_k^2 - 0.0001y_k T_k(h))$$

其中$T_0(h) = 0$且

$$T_k(h) = T_{k-1}(h) + \frac{h}{2}(\mathrm{y_{k-1}} + \mathrm{y_k}) \quad \text{其中 } k = 0,1,\dots,99$$

(b)用初值$y(0) = 200$和$y(0) = 300$重复(a)的计算。

(c)在同一坐标系中画出(a)和(b)的近似解。

实验：

欧拉方法：

欧拉方法其实是泰勒级数法的特殊情况：

$$y(t) = y(t_0) + y'(t_0)(t - t_0) + \frac{y''(c)}{2}(t - t_0)^2$$

将要求的式子：$y' = f(t,y)$带入，并且$t - t_0 = h$，忽略掉误差项有：

$$y(t) = y(t_0) + y'(t_0)h$$

误差：

误差也就是泰勒展开式中忽略的误差项的和（因为每一步迭代都是在原来基础上进行的，即误差应该不断相加，假设进行 M 次迭代）：

$$\frac{\sum_{k=1}^{M} y^{(2)}(\mathrm{c_k})\mathrm{h}^2}{2} \approx \frac{\mathrm{M}y^{(2)}(c)h^2}{2} = \frac{(\mathrm{b-a})\mathrm{y}^{(2)}(c)h^2}{2\mathrm{h}} = \frac{(\mathrm{b-a})\mathrm{y}^{(2)}h}{2} = O(h)$$

所以如果步长变为原来的$\frac{1}{2}$，误差应该变成原来的$\frac{1}{2}$

利用欧拉方法和梯形公式去逼近积分，可以得到题目提示中的表达式，利用该表达式可以写出代码完成方程的解。

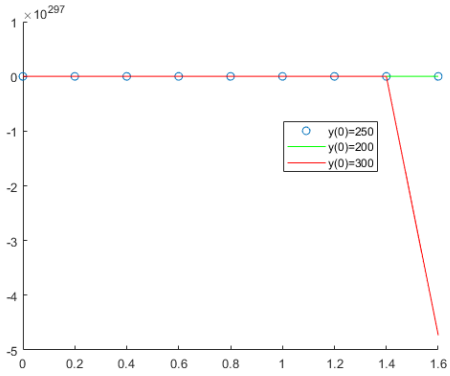
Code：

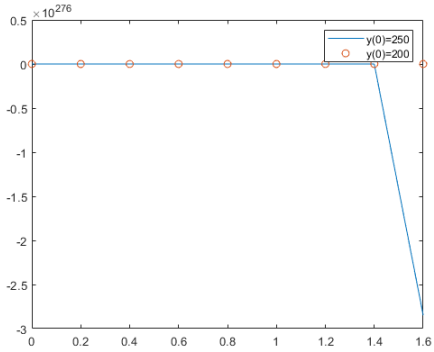
```
function E = euler2(a,b,ya,h)
%ya 是边界值
%a b 是边界线
M = (b-a)/h;
X=zeros(1,M+1);
T=zeros(1,M+1);
Y=zeros(1,M+1);
X=a:h:b;
Y(1)=ya;
T(1)=0;
for i=1:M
    if i>1
        T(i)=T(i-1)+h*(Y(i-1)+Y(i))/2;
    end
    Y(i+1)=Y(i)+h*(1.3*Y(i)-0.25*Y(i)*Y(i)-0.0001*Y(i)*T(i));
end
E=[X',Y'];
end
```

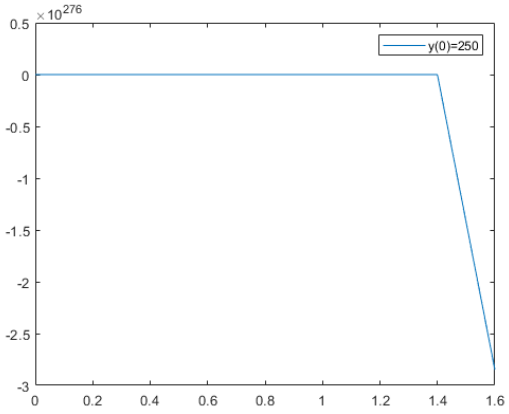
得到的近似解：

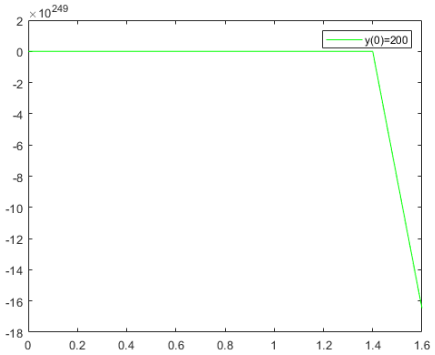
Table 11

k	$y(0)=200$	$y(0)=250$	$y(0)=300$
0	200	250	300
0.2	-1748	-2810	-4122
0.4	-154983.10	-398359.99	-854769.43
0.6	-1201232277.18	-7935357563.43	-36534090652.33
0.8	-7.2151e+16	-3.1486e+18	-6.6740e+19
1	-2.6030e+32	-4.9571e+35	-2.2272e+38
1.2	-3.3879e+63	-1.2287e+70	-2.4803e+75
1.4	-5.7391e+125	-7.5487e+138	-3.0760e+149
1.6	-1.6469e+250	-2.8493e+276	-4.7310e+297









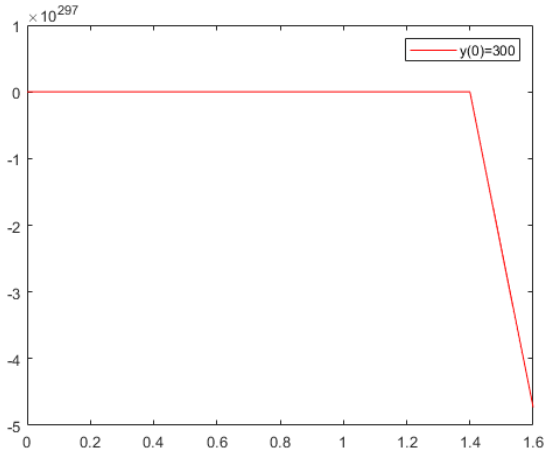


Figure 23

表中的数据在 1.6 之后都到了 INF，所以在图片上上面显示不出来，将三个函数画在一张图上的时候，因为值相差比较大，所以有的曲线就直接变成了直线。

Problem 3

问题描述：

对于下面的微分方程，用 heun 方法求解微分方程。

(a)令$h = 0.1$，程序 9.2 执行 20 步，然后令$h = 0.05$，程序 9.2 执行 40 步。

(b)比较(a)中的两个近似解与精确解$y(2)$。

(c)当h减半时，(a)中的最终全局误差是否和预期相符?

(d)将两个近似解和精确解画在同一坐标系中。

微分方程：

$$y' = 3y + 3t, \quad y(0) = 1, \quad y(t) = \frac{4}{3}e^{3t} - t - \frac{1}{3}$$

实验：

Heun 方法：

对于方程：$y'(t) = f\big(t, y(t)\big)$,还可以用另一种方法来得到解，就是利用积分：

$$\int_{t_0}^{t_1} f\big(t, y(t)\big)dt = \int_{t_0}^{t_1} y'(t)dt = y(t_1) - y(t_0)$$

从而得到：

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y(t))dt$$

然后利用梯形公式可以逼近其中的积分：

$$y(t_1) \approx y(t_0) + \frac{h}{2}\big(f\big(t_0, y(t_0)\big) + f\big(t_1, y(t_1)\big)\big)$$

但是因为式子右边还有未知的$y(t_1)$，所以可以用之前的欧拉公式将这里的$y(t_1)$替换掉，也就是用欧拉公式来预报，然后用积分来校正。

$$y(t_1) = y(t_0) + \frac{h}{2}(f(t_0, y(t_0)) + f(t_1, y_0 + hf(t_0, y_0)))$$

误差：

积分公式中有误差：

$$-\frac{y^{(2)}(c_k)h^3}{12}$$

所以将每步的合起来有：

$$\frac{b-a}{12}y^{(2)}(c)h^2 = O(h^2)$$

使用欧拉方法也有误差$O(h)$，和外面的$\frac{2}{h}$合在一起就是$O(h^2)$，所以总的误差就是$O(h^2)$。

所以如果步长变为原来的$\frac{1}{2}$误差应该变为原来的$\frac{1}{4}$

Heun 方法 **Code：**

```
function H = heun(f,a,b,ya,h)
M = (b-a)/h;
T = zeros(1,M+1);
Y=zeros(1,M+1);
T=a:h:b;
Y(1)=ya;
for i=1:M
    k1=f(T(i),Y(i));
    k2=f(T(i+1),Y(i)+h*k1);
    Y(i+1)=Y(i)+(h/2)*(k1+k2);
end
H=[T' Y'];
end
```

利用 Heun 方法画图的代码放在附录 Code 6 中。

Table 12

t_k	y_k	$y(t_k)$	误差$y(t_k) - y_k$
0.0	1.0000	1.0000	0
0.1	1.3600	1.3665	0.0065
0.2	1.8787	1.8962	0.0175
0.3	2.6109	2.6461	0.0353
0.4	3.6301	3.6935	0.0634
0.5	5.0355	5.1423	0.1068
0.6	6.9602	7.1329	0.1726
0.7	9.5835	9.8549	0.2714
0.8	13.1463	13.5642	0.4179
0.9	17.9728	18.6063	0.6335
1.0	24.4989	25.4474	0.9485

1.1	33.3110	34.7169	1.4059
1.2	45.1978	47.2643	2.0665
1.3	61.2200	64.2366	3.0166
1.4	82.8044	87.1818	4.3774
1.5	111.8699	118.1895	6.3196
1.6	150.9976	160.0806	9.0830
1.7	203.6587	216.6625	13.0038
1.8	274.5225	293.0752	18.5527
1.9	369.8688	396.2565	26.3878
2.0	498.1440	535.5717	37.4277

40 步的相应表格放在附录当中。

20 步的最终误差是 37.4277，40 步的最终误差是 10.7143，比值为 3.4933.可以看出基本还是满足预期的。

Table 13

步数	误差
20	37.4277
40	10.7143
比值	3.4933

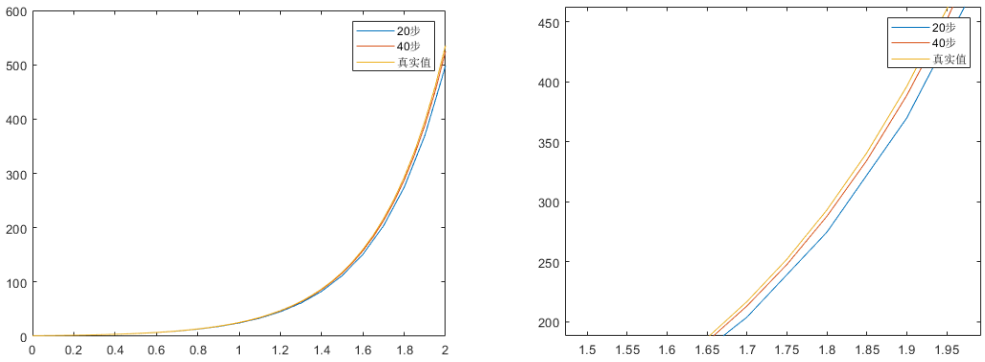


Figure 24 **左图为整体图样，右图为局部放大图**

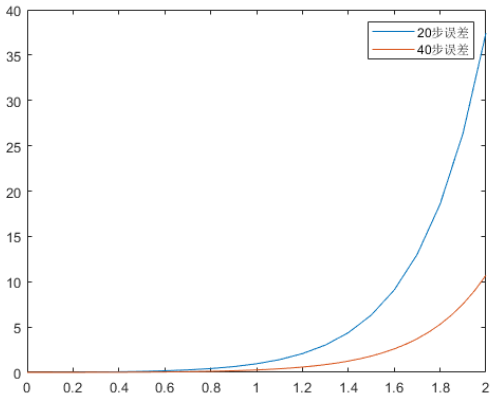


Figure 25 **误差曲线图**

将图片放大之后还是可以比较明显地看出，20 步和真实值的偏离程度是大于

40 步的，而且从误差曲线也可以看出，20 步的误差到后来比 40 步的误差越来越大。

Problem 4

问题描述：

对于下面的微分方程，用$N = 4$的龙格-库塔方法求解微分方程。

(a)令$h = 0.1$，程序 9.4 执行 20 步，然后令$h = 0.05$，程序 9.4 执行 40 步。

(b)比较(a)中的两个近似解与精确解$y(2)$。

(c)当h减半时，(a)中的最终全局误差是否和预期相符?

(d)在同一坐标系中画出两个近似解和精确解。

微分方程：

$$y' = 3y + 3t, \quad y(0) = 1, \quad \mathrm{y(t)} = \frac{4}{3}e^{3t} - t - \frac{1}{3}$$

实验：

4 阶龙格-库塔方法 Code：

```
function R = rk4(f,a,b,ya,h)
M = (b-a)/h;
T=zeros(1,M+1);
Y=zeros(1,M+1);
T=a:h:b;
Y(1)=ya;
for i = 1:M
    k1 = h*f(T(i),Y(i));
    k2= h*f(T(i)+h/2,Y(i)+k1/2);
    k3=h*f(T(i)+h/2,Y(i)+k2/2);
    k4=h*f(T(i)+h,Y(i)+k3);
    Y(i+1)=Y(i)+(k1+2*k2+2*k3+k4)/6;
end
R=[T' Y'];
end
```

利用 4 阶龙格-库塔方法绘图的代码放在附录 Code 7 中。

Table 14 20 步 RK4 计算

t_k	$RK4$近似值y_k	真解$y(t_k)$	误差$y(t_k) - y_k$
0.0	1	1	0
0.1	1.3664	1.3664	2.8410e-05
0.2	1.8960	1.8961	7.6698e-05
0.3	2.6459	2.6461	0.0001
0.4	3.6932	3.6934	0.0002
0.5	5.1417	5.1422	0.0004
0.6	7.1320	7.1328	0.0007

0.7	9.8536	9.8548	0.0012
0.8	13.5623	13.5642	0.0018
0.9	18.6034	18.6063	0.0028
1.0	25.4431	25.4473	0.0042
1.1	34.7105	34.7168	0.0062
1.2	47.2550	47.2643	0.0092
1.3	64.2230	64.2365	0.0135
1.4	87.1621	87.1817	0.0196
1.5	118.1610	118.1895	0.0284
1.6	160.0396	160.0805	0.0409
1.7	216.6038	216.6625	0.0586
1.8	292.9913	293.0752	0.0838
1.9	396.1370	396.2565	0.1194
2.0	535.4019	535.5717	0.1697

40 步的相应表格放在附录当中。
20 步的最终误差为 0.1697，40 步的最终误差为 0.012，比值为 14.1276.

Table 15

步数	误差
20	0.1697
40	0.0120
比值	14.1276

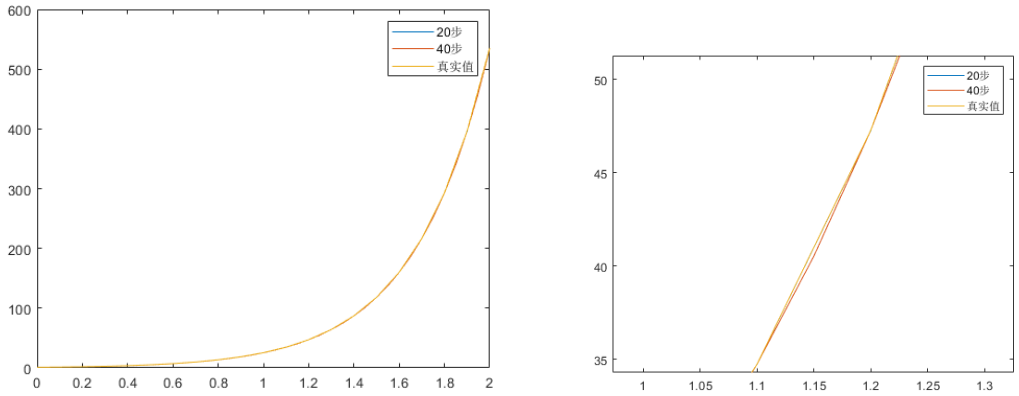


Figure 26 左图为整体图样，右图为局部放大图

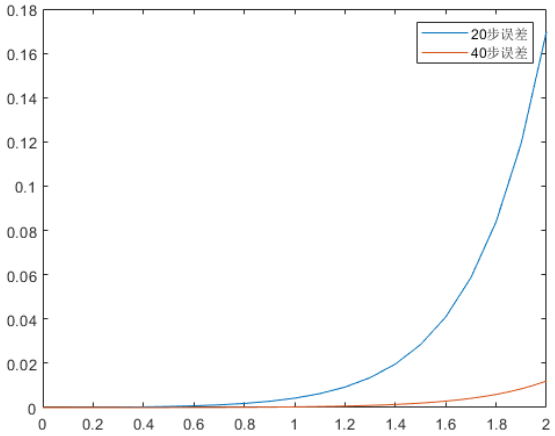


Figure 27 **误差曲线图**

上图中 40 步基本和真实值重合了，20 步稍微放大之后还是可以看见和真实值不同情况。上图中的误差曲线图可以看出，20 步的误差相比 40 步的误差到后来越来越大。

Chapter 11

Problem 1

问题描述：

已知矩阵

$$A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$$

是一个对称矩阵，且其特征值为$\lambda_1 = 6, \lambda_2 = 3, \lambda_3 = 1.$

分别利用幂法、对称幂法、反幂法求其最大特征值和特征向量。

注意：可取初始向量$x^{(0)} = (111)^T$.

实验：

幂法

由于A有n个特征值，所以有对应的特征向量$V_j, j = 1,2, \dots, n$。而且是线性无关的，所以初始向量X_0可表示为这样的线性组合：

$$X_0 = b_1V_1 + B_2V_2 + \cdots + b_nV_n$$

两边同时乘以A可以得到：

$$\begin{aligned} Y_0 = AX_0 &= A(b_1V_1 + b_2V_2 + \cdots + b_nV_n) \\ &= b_1AV_1 + b_2AV_2 + \cdots + b_nAV_n \\ &= b_1\lambda_1V_1 + b_2\lambda_2V_2 + \cdots + b_n\lambda_nV_n \\ &= \lambda_1\left(b_1V_1 + b_2\left(\frac{\lambda_2}{\lambda_1}\right)V_2 + \cdots + b_n\left(\frac{\lambda_n}{\lambda_1}\right)V_n\right) \end{aligned}$$

那么有：

$$X_1 = \frac{\lambda_1}{c_1}\left(b_1V_1 + b_2\left(\frac{\lambda_2}{\lambda_1}\right)V_2 + \cdots + b_n\left(\frac{\lambda_n}{\lambda_1}\right)V_n\right)$$

经过k个迭代后可以得到：

$$Y_{k-1} = AX_{k-1}$$

$$= A\frac{\lambda_1^{k-1}}{c_1c_2\cdots c_{k-1}}(\mathrm{b_1V_1} + \mathrm{b_2}\left(\frac{\lambda_2}{\lambda_1}\right)^{\mathrm{k-1}}\mathrm{V_2} + \cdots + \mathrm{b_n}\left(\frac{\lambda_n}{\lambda_1}\right)^{\mathrm{k-1}}\mathrm{V_n})$$

$$= \frac{\lambda_1^{k-1}}{c_1c_2\cdots c_{k-1}}(b_1AV_1 + b_2\left(\frac{\lambda_2}{\lambda_1}\right)^{k-1}AV_2 + \cdots + b_n\left(\frac{\lambda_n}{\lambda_1}\right)^{k-1}AV_n)$$

$$= \frac{\lambda_1^{k-1}}{c_1c_2\cdots c_{k-1}}(b_1\lambda_1V_1 + b_2\left(\frac{\lambda_2}{\lambda_1}\right)^{k-1}\lambda_2V_2 + \cdots + b_n\left(\frac{\lambda_n}{\lambda_1}\right)^{k-1}\lambda_nV_n)$$

$$= \frac{\lambda_1^{k}}{c_1c_2\cdots c_{k-1}}(b_1V_1 + b_2\left(\frac{\lambda_2}{\lambda_1}\right)^{k}V_2 + \cdots + b_n\left(\frac{\lambda_n}{\lambda_1}\right)^{k}V_n)$$

由于$\frac{|\lambda_j|}{|\lambda_1|} < 1, j = 2,3,\ldots,n$

所以有

$$\lim_{k\to\infty}\left(\frac{\lambda_j}{\lambda_1}\right)^k V_j = 0, j = 2,3,\ldots,n$$

因此进一步得到：

$$\lim_{k\to\infty} X_k = \lim_{k\to\infty}\frac{b_1\lambda_1^k}{c_1c_2\ldots c_k}V_1$$

然后有

$$\lim_{k\to\infty}\frac{b_1\lambda_1^k}{c_1c_2\ldots c_k} = 1$$

利用k项和第$k-1$项相比可以得到：

$$\lim_{k\to\infty}\frac{\lambda_1}{c_k} = \lim_{k\to\infty}\frac{\frac{b_1\lambda_1^k}{c_1c_2\ldots c_k}}{\frac{b_1\lambda_1}{c_1c_2\ldots c_{k-1}}} = 1$$

所以有

$$\lim_{\mathrm{k}\to\infty} c_k = \lambda_1$$

Code：

```
function [lambda,V] = power2(A,X,step,epsilon)
%epsilon 是误差
%step 是迭代步数
lambda = 0;
```

```
cnt = 0;
err = 1;
while(err>epsilon&cnt<step)
    cnt = cnt + 1;
    Y=A*X;
    C(cnt) = max(abs(Y));
    dc = abs(lambda - C(cnt));
    Y = (1/C(cnt))*Y;
    dv = norm(X-Y);
    err=max(dc,dv);
    X=Y;
    lambda=C(cnt);
end
V=X;
C
end
```

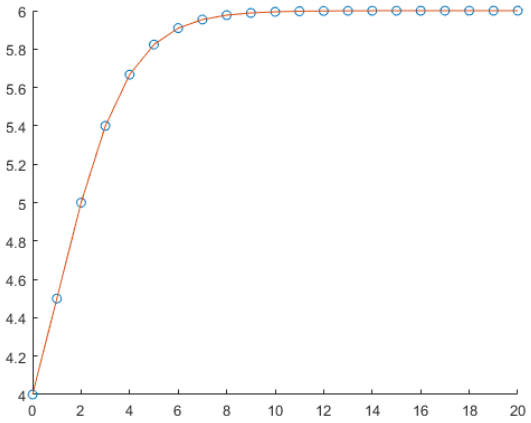


Figure 28

最大特征值:6.0000,特征向量：$[1;-1;1]$.精度10^{-5}迭代次数 21

对称幂法

对称幂法的特征就是有一个矩阵A是对称矩阵。因此A就有n个特征值：

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$$

对用有正交的特征向量：

$$V_1, V_2, \cdots, V_n$$

和幂法一样：

多以对于迭代$x_k = A^k x_0$，和上式联系起来可以有：

$$x_k = A^k x = \beta_1 \lambda^k V_1 + \beta_2 \lambda_k V_2 + \cdots + \beta_n \lambda_n V_n$$

因为$V_1, V_2, \cdots, V_n$都是正交的，所以有：

$$x_k^T x_k = \sum_{j=1}^{n} \beta_j^2 \lambda_j^{2k} = \beta_1^2 \lambda_1^{2k} \left\{ 1 + \sum_{j=2}^{n} \left(\frac{\beta_j}{\beta_1} \right)^2 \left(\frac{\lambda_j}{\lambda_1} \right)^{2k} \right\}$$

并且有

$$x_k^T A x_k = \sum_{j=1}^{n} \beta_j^2 \lambda_j^{2k+1}$$

$$= \beta_1^2 \lambda_1^{2k+1} \{1 + \sum_{j=2}^{n} \left(\frac{\beta_j}{\beta_1}\right)^2 \left(\frac{\lambda_j}{\lambda_1}\right)^{2k+1}\}$$

所以最终有：

$$\lim_{k\to\infty} \frac{x_k^T A x_k}{x_k^T x_k} = \lambda_1$$

$$\lim_{k\to\infty} \frac{x_k}{||x_k||_2} = \frac{V_1}{||V_1||_2}$$

Code：

```
function [lambda,V] = sympower(A,X,epsilon,step)
%step 表示迭代次数
%epsilon 表示误差
err = 1;
cnt = 0;
X = X/norm(X,2);
while(cnt<step&err>epsilon)
    Y = A*X;
    u = X'*Y;
    dc = norm(Y,2);
    err = norm(X-Y/norm(Y,2),2);
    X = Y/norm(Y,2);
    cnt = cnt + 1;
end
V = X;
lambda = u;
end
```

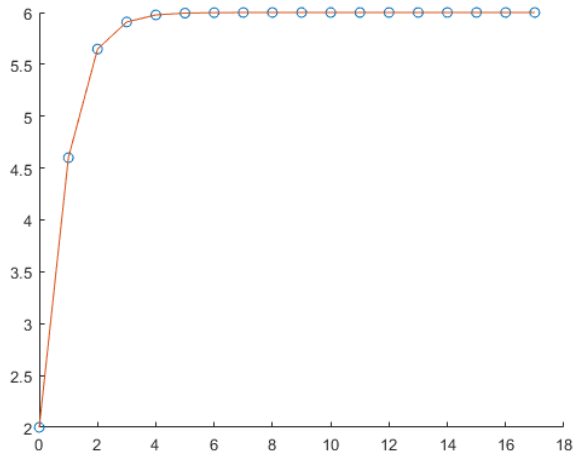


Figure 29

最大特征值:6.0000,特征向量:[0.5774; −0.5773; 0.5773]，迭代次数 18。

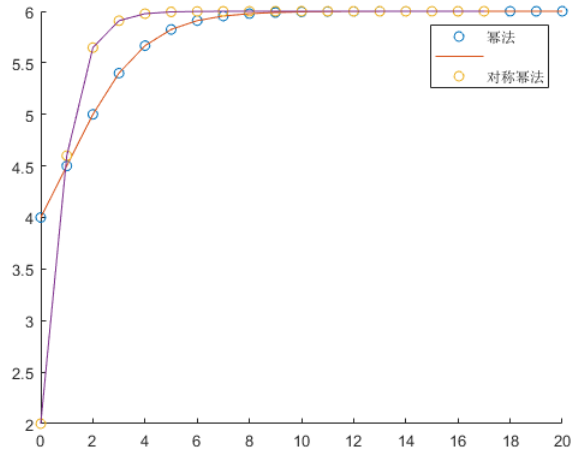


Figure 30

上图是将幂法和反幂法画在一张图上。

反幂法

首先有两个基本定理：

定理 1：设λ, V为A的特征对。如果α是任意常量，则$\lambda - \alpha$，V是矩阵$A - \alpha I$的特征对。

证明：

$$AV = \lambda V$$

所以

$$AV - \alpha V = \lambda V - \alpha V$$

所以有

$$(A - \alpha I)V = (\lambda - \alpha)V$$

得证。

定理 2：设λ, V是A的特征对。如果$\lambda \neq 0$，则$\frac{1}{\lambda}$,V是矩阵A^{-1}的特征对。

证明：

$$AV = \lambda V$$

所以

$$A^{-1}AV = A^{-1}\lambda V$$

所以

$$\frac{1}{\lambda}V = A^{-1}V$$

证毕。

将上述两个定理合在一起有：

设λ, V是A的特征对。如果$\alpha \neq \lambda$，则$\frac{1}{\lambda-\alpha}, V$是矩阵$(A-\alpha I)^{-1}$的特征对。

所以反幂法的迭代过程就是将幂法中的A看成是反幂法中的$(A-\alpha I)^{-1}$，将幂法中的λ看成是反幂法中的$\frac{1}{\lambda-\alpha}$，最后证明大致一样，但是要求α是对λ的一个好的近似。

Code：

```
function [lambda,V] = invpower(A,X,alpha,epsilon,step)
%alpha 就是位移
%step 代表迭代步骤
n = size(A);
A = A- alpha*eye(n);
lambda = 0;
cnt = 0;
err = 1;
while(err>epsilon&cnt<step)
    cnt = cnt + 1;
    Y = A\X;
    C(cnt) = max(Y);
    dc = abs(lambda-C(cnt));
    Y = (1/C(cnt))*Y;
    dv = norm(X-Y);
    err = max(dc,dv);
    X = Y;
    lambda = C(cnt);
end
lambda = alpha+1/C(cnt);
V = X;
C
end
```

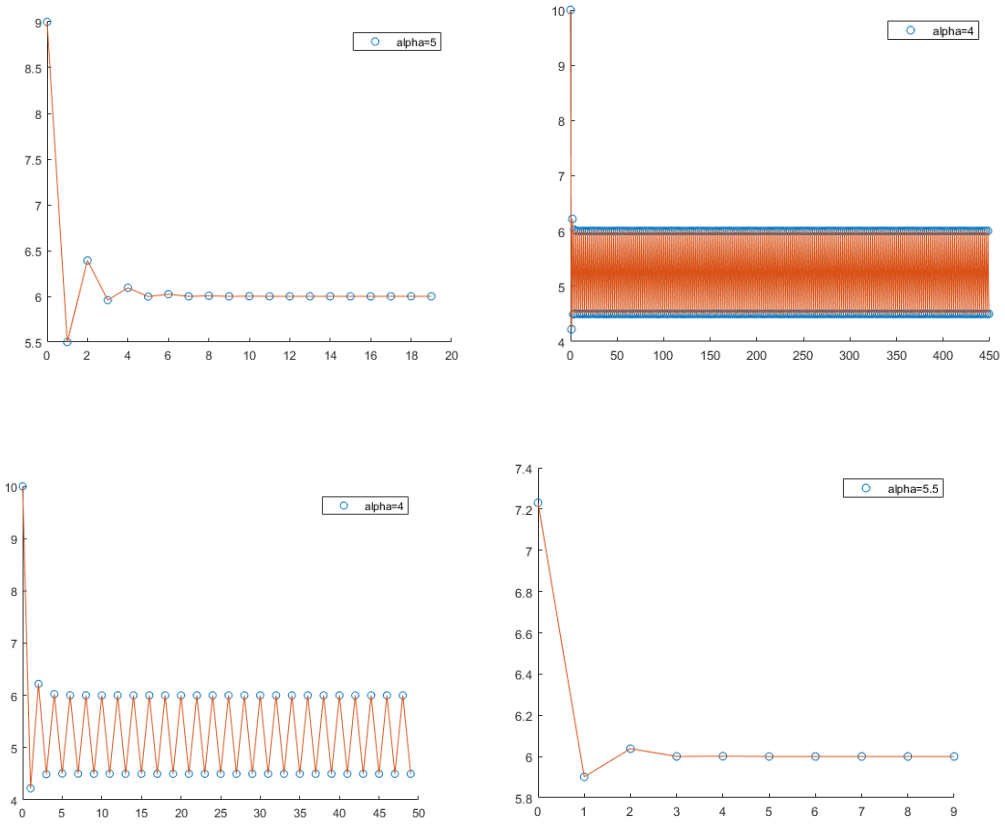


Figure 31

从左到右，从上到下，第一张是 alpha 为 5 的情况，第二张是α为 4 的情况，第三张是第二张局部放大，第四张是α是 5.5 的情况。

最大特征值为 6.0000，特征向量为[1,−1,1].当α是 5 的时候，迭代了 20 次，当 α是 5.5 的时候迭代了 10 次。

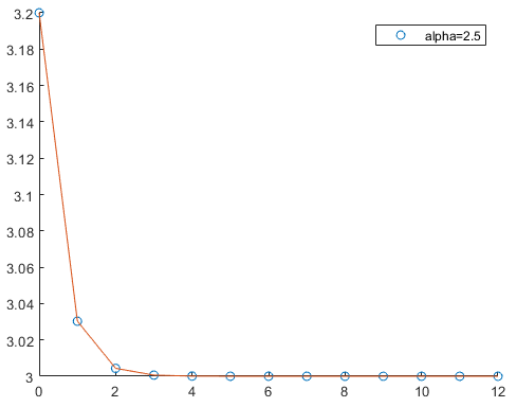


Figure 32

求得的特征值为 3.0000，特征向量为[1,0.5,−0.5]，迭代了 13 次。

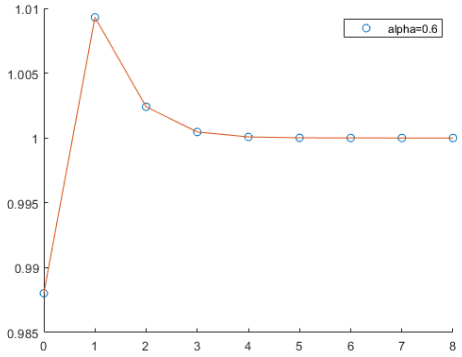


Figure 33

求得的特征值为 **1.0000**，特征向量为$[6.6175e-08,1,1.0000]$，迭代了 9 次。

Conclusion

本次实验总体感觉确实学到了很多，在书本的基础上做实验可以更加直观地看到书本上的理论知识在实际中的用处，而且误差这个东西通过公式和通过图像比较显示出来，给人的印象是完全不一样的，误差通过自己实验的出来的结论可以直接非常直观看到大小，而不是公式抽象的分析。

Appendix

Code 1

```
clear;clc;
f =@(x) 1/2 + x.^2/4 - x.*sin(x) - cos(2*x)/2;
x=-10:0.01:10;
y=f(x);
plot(x,y);
legend("f(x)");
```

Code 2

```
X=[4,-1,1;4,-8,1;-2,1,5];
B=[7;-21;15];
[L,U]=LU(X,3);
fprintf("LU 分解结果:\n");
L
U
res=LUsolveX(L,U,B);
```

```
fprintf("求解结果:\n");
res
```

Code 3

```
clear;clc;
X=[1;2;2];
fprintf("Jacobi 方法:\n");
func3_12(X,2);
fprintf("Gauss-Seidel 方法:\n");
func3_12(X,1);
```

Code 4

把多阶拉格朗日插值多项式都画在一起：

```
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:5:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:3
    P = 1;
    for j = 1:3
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold on;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
hold on;
%四阶
x = -5:2.5:5;
```

```
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:5
    P = 1;
    for j = 1:5
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold on;
%六阶
x = -5:10/6:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:7
    P = 1;
    for j = 1:7
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold on;
%8 阶
x = -5:1.25:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:9
    P = 1;
```

```
    for j = 1:9
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold on;
%10阶
x = -5:1:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:11
    P = 1;
    for j = 1:11
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold on;
legend("二阶插值","原函数","四阶插值","六阶插值","八阶插值","十阶插值");
```

单独将拉格朗日插值多项式和原函数画出来进行比较：

```
%二次
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:5:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:3
    P = 1;
```

```
    for j = 1:3
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
legend('拉格朗日插值','原函数');
eval(int(f,m,-5,5))
eval(int(L,t,-5,5))
```

```
%四次
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:2.5:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:5
    P = 1;
    for j = 1:5
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
```

```
legend('拉格朗日插值','原函数');
eval(int(f,m,-5,5))
eval(int(L,t,-5,5))
```

```
%六次
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:10/6:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:7
    P = 1;
    for j = 1:7
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
legend('拉格朗日插值','原函数');
eval(int(f,m,-5,5))
eval(int(L,t,-5,5))
```

```
%八次
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:(10/9):5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
```

```
syms t;
for i = 1:9
    P = 1;
    for j = 1:9
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
legend('拉格朗日插值','原函数');
eval(int(f,m,-5,5))
eval(int(L,t,-5,5))
```

```
%十次
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:5;
x = x';
yreal = subs(f,x);%插值点
L = 0;
syms t;
for i = 1:11
    P = 1;
    for j = 1:11
        if i~= j
            P = P*(t - x(j,1))/(x(i,1)-x(j,1));
        end
    end
    L = L + P*yreal(i,1);
end
x2 = -5:0.1:5;
y2 = subs(L,x2);
plot(x2,y2)
hold;
```

```
x = -5:0.1:5;
yreal = subs(f,x2);
plot(x,yreal);
legend('拉格朗日插值','原函数');
eval(int(f,m,-5,5))
eval(int(L,t,-5,5))
```

Code 5

Natural 样条：

```
function S = natural(X,Y)
N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);
for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);
for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=0;
M(N+1)=0;
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
end
```

紧压样条：

```
function S = csfit(X,Y,dx0,dxn)
N=length(X)-1;
```

```
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);

B(1)=B(1)-H(1)/2;
U(1)=U(1)-3*(D(1)-dx0);
B(N-1)=B(N-1)-H(N)/2;
U(N-1)=U(N-1)-3*(dxn-D(N));
for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);
for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=3*(D(1)-dx0)/H(1)-M(2)/2;
M(N+1)=3*(dxn-D(N))/H(N)-M(N)/2;
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
end
```

外推样条：

```
function S = extra(X,Y)
N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);
B(1)=B(1)+H(1)+H(1)^2/H(2);
B(N-1)=B(N-1)+H(N)^2/H(N-1)+H(N);
A(N-2)=A(N-2)-H(N)^2/H(N-1);
for k=2:N-1
    temp=A(k-1)/B(k-1);
```

```
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);
for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=M(2)-(H(1)*(M(3)-M(2)))/H(2);
M(N+1)=M(N)+(H(N)*(M(N)-M(N-1)))/H(N-1);
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
end
```

抛物线终结样条：

```
function S = parabolically(X,Y)
N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);
B(1)=B(1)+H(1);
B(N-1)=B(N-1)+H(N);
for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);
for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=M(2);
M(N+1)=M(N);
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
```

```
end
end
```

端点曲率调整样条：

```
function S = end_point(X,Y,ddx0,ddxn)
N=length(X)-1;
H=diff(X);
D=diff(Y)./H;
A=H(2:N-1);
B=2*(H(1:N-1)+H(2:N));
C=H(2:N);
U=6*diff(D);

U(1)=U(1)-H(1)*ddx0;
U(N-1)=U(N-1)-H(N)*ddxn;
for k=2:N-1
    temp=A(k-1)/B(k-1);
    B(k)=B(k)-temp*C(k-1);
    U(k)=U(k)-temp*U(k-1);
end
M(N)=U(N-1)/B(N-1);

for k=N-2:-1:1
    M(k+1)=(U(k)-C(k)*M(k+2))/B(k);
end
M(1)=ddx0;
M(N+1)=ddxn;
for k=0:N-1
    S(k+1,1)=(M(k+2)-M(k+1))/(6*H(k+1));
    S(k+1,2)=M(k+1)/2;
    S(k+1,3)=D(k+1)-H(k+1)*(2*M(k+1)+M(k+2))/6;
    S(k+1,4)=Y(k+1);
end
end
```

将所有曲线画到一张图上：

```
clear;clc;
X=[0 1 2 3 4 5 6];Y=[1 0 0 1 2 2 1];dx0=-0.6;dxn=-1.8;ddx0=1;ddxn=-1;
%紧压样条
S=csfit(X,Y,dx0,dxn);
x1=0:.01:1;y1=polyval(S(1,:),x1-X(1));
x2=1:.01:2;y2=polyval(S(2,:),x2-X(2));
x3=2:.01:3;y3=polyval(S(3,:),x3-X(3));
```

```
 x4=3:.01:4;y4=polyval(S(4,:),x4-X(4));
 x5=4:.01:5;y5=polyval(S(5,:),x5-X(5));
x6=5:.01:6;y6=polyval(S(6,:),x6-X(6));
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,'.');
hold on;
%natural 样条
S=natural(X,Y);
x1=0:.01:1;y1=polyval(S(1,:),x1-X(1));
x2=1:.01:2;y2=polyval(S(2,:),x2-X(2));
x3=2:.01:3;y3=polyval(S(3,:),x3-X(3));
x4=3:.01:4;y4=polyval(S(4,:),x4-X(4));
x5=4:.01:5;y5=polyval(S(5,:),x5-X(5));
x6=5:.01:6;y6=polyval(S(6,:),x6-X(6));
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,'.');
hold on;
%外推样条
S=extra(X,Y);
x1=0:.01:1;y1=polyval(S(1,:),x1-X(1));
x2=1:.01:2;y2=polyval(S(2,:),x2-X(2));
x3=2:.01:3;y3=polyval(S(3,:),x3-X(3));
x4=3:.01:4;y4=polyval(S(4,:),x4-X(4));
x5=4:.01:5;y5=polyval(S(5,:),x5-X(5));
x6=5:.01:6;y6=polyval(S(6,:),x6-X(6));
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,'.');
hold on;
%抛物线终结样条
S=parabolically(X,Y);
x1=0:.01:1;y1=polyval(S(1,:),x1-X(1));
x2=1:.01:2;y2=polyval(S(2,:),x2-X(2));
x3=2:.01:3;y3=polyval(S(3,:),x3-X(3));
x4=3:.01:4;y4=polyval(S(4,:),x4-X(4));
x5=4:.01:5;y5=polyval(S(5,:),x5-X(5));
x6=5:.01:6;y6=polyval(S(6,:),x6-X(6));
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,'.');
hold on;
%端点曲率调整样条
S=end_point(X,Y,ddx0,ddxn);
x1=0:.01:1;y1=polyval(S(1,:),x1-X(1));
x2=1:.01:2;y2=polyval(S(2,:),x2-X(2));
x3=2:.01:3;y3=polyval(S(3,:),x3-X(3));
x4=3:.01:4;y4=polyval(S(4,:),x4-X(4));
x5=4:.01:5;y5=polyval(S(5,:),x5-X(5));
x6=5:.01:6;y6=polyval(S(6,:),x6-X(6));
plot(x1,y1,x2,y2,x3,y3,x4,y4,x5,y5,x6,y6,'.');
```

```
52. hold on;
```

单独画就只需要将其中的内容拆开就行，不在这里罗列。

Code 6

调用 Heun 方法绘图 Code：

```
1.  clear;clc;
2.  a=0;b=2;
3.  freal=@(t) 4*exp(3*t)/3-t-1/3;
4.  f=@(t,y) 3*y+3*t;
5.  ya=1;
6.  h=0.1;
7.  H=heun(f,a,b,ya,h);
8.  plot(H(:,1),H(:,2));
9.  hold on;
10. h=0.05;
11. H=heun(f,a,b,ya,h);
12. plot(H(:,1),H(:,2));
13. hold on;
14. x=0:h:2;
15. y=freal(x);
16. plot(x,y);
17. legend("20 步","40 步","真实值");
```

Code 7

调用龙格-库塔方法绘图：

```
1.  clear;clc;
2.  freal=@(t) 4*exp(3*t)/3-t-1/3;
3.  f=@(y,t) 3*y + 3*t;
4.  a=0;b=2;
5.  h=0.1;ya=1;
6.  R=rk4(f,a,b,ya,h);
7.  plot(R(:,1),R(:,2));
8.  hold on;
9.  h=0.05;
10. R=rk4(f,a,b,ya,h);
11. plot(R(:,1),R(:,2));
12. hold on;
13. x=0:h:2;
```

```
y=freal(x);
plot(x,y);
legend("20 步","40 步","真实值");
```

Code 8

```
clear;clc;
syms m
f = 1./(1+m.^2);
x = -5:5;
fnew = @(t) (t-x(1)).*(t-x(2)).*(t-x(3)).*(t-x(4)).*(t-x(5)).*(t-x(6)).*(t-x(7)).*(t-x(8)).*(t-x(9)).*(t-x(10)).*(t-x(11));
t=-5:0.01:5;
res=max(abs(fnew(t)));
df=@(m)(479001600*m)/(m^2 + 1)^7 - (11176704000*m^3)/(m^2 + 1)^8 + (71530905600*m^5)/(m^2 + 1)^9 - (183936614400*m^7)/(m^2 + 1)^10 + (204374016000*m^9)/(m^2 + 1)^11 - (81749606400*m^11)/(m^2 + 1)^12;
res2=0;
for i=0:0.01:5
    res2=max(res2,abs(df(i)));
end
res
res2
```

Heun 方法 40 步

t_k	估计值y_k	精确值$y(t_k)$	误差$\mathrm{y(t_k)-y_k}$
0	1.0000	1.0000	0
0.05	1.1650	1.1658	0.0008
0.10	1.3647	1.3665	0.0018
0.15	1.6046	1.6077	0.0032
0.20	1.8913	1.8962	0.0049
0.25	2.2322	2.2393	0.0071
0.30	2.6363	2.6461	0.0099
0.35	3.1135	3.1269	0.0134
0.40	3.6757	3.6935	0.0178
0.45	4.3367	4.3599	0.0232
0.50	5.1123	5.1423	0.0300
0.55	6.0210	6.0593	0.0383
0.60	7.0843	7.1329	0.0485
0.65	8.3272	8.3883	0.0611

0.70	9.7785	9.8549	0.0764
0.75	11.4719	11.5670	0.0951
0.80	13.4464	13.5642	0.1178
0.85	15.7474	15.8928	0.1454
0.90	18.4275	18.6063	0.1788
0.95	21.5478	21.7670	0.2192
1	25.1793	25.4474	0.2681
1.05	29.4045	29.7314	0.3269
1.10	34.3190	34.7169	0.3978
1.15	40.0341	40.5172	0.4831
1.20	46.6788	47.2643	0.5855
1.25	54.4030	55.1114	0.7085
1.30	63.3808	64.2366	0.8558
1.35	73.8143	74.8466	1.0323
1.40	85.9383	87.1818	1.2435
1.45	100.0253	101.5213	1.4959
1.50	116.3920	118.1895	1.7975
1.55	135.4058	137.5633	2.1575
1.60	157.4937	160.0806	2.5868
1.65	183.1513	186.2500	3.0986
1.70	212.9543	216.6625	3.7083
1.75	247.5710	252.0050	4.4340
1.80	287.7778	293.0752	5.2974
1.85	334.4760	340.8001	6.3241
1.90	388.7123	396.2565	7.5442
1.95	451.7023	460.6958	8.9936
2	524.8575	535.5717	10.7143

4 阶龙格–库塔方法 40 步

t_k	RK4 近似值y_k	真解$y(t_k)$	误差$y(t_k) - y_k$
0	1	1	0
0.05	1.1657	1.1657	8.6530e-07
0.10	1.3664	1.3664	2.0106e-06
0.15	1.6077	1.6077	3.5041e-06
0.20	1.8961	1.8961	5.4282e-06
0.25	2.2393	2.2393	7.8834e-06
0.30	2.6461	2.6461	1.0991e-05
0.35	3.1268	3.1268	1.4898e-05
0.40	3.6934	3.6934	1.9781e-05
0.45	4.3598	4.3599	2.5856e-05
0.50	5.1422	5.1422	3.3378e-05

0.55	6.0592	6.0593	4.2658e-05
0.60	7.1328	7.1328	5.4067e-05
0.65	8.3881	8.3882	6.8051e-05
0.70	9.8548	9.8548	8.5147e-05
0.75	11.5668	11.5669	0.0001
0.80	13.5641	13.5642	0.0001
0.85	15.8926	15.8928	0.0001
0.90	18.6061	18.6063	0.0001
0.95	21.7667	21.7670	0.0002
1	25.4470	25.4473	0.0002
1.05	29.7310	29.7314	0.0003
1.10	34.7164	34.7168	0.0004
1.15	40.5166	40.5171	0.0005
1.20	47.2636	47.2643	0.0006
1.25	55.1106	55.1114	0.0007
1.30	64.2356	64.2365	0.0009
1.35	74.8454	74.8466	0.0011
1.40	87.1803	87.1817	0.0013
1.45	101.5196	101.5212	0.0016
1.50	118.1874	118.1895	0.0020
1.55	137.5608	137.5633	0.0024
1.60	160.0776	160.0805	0.0028
1.65	186.2464	186.2499	0.0034
1.70	216.6583	216.6625	0.0041
1.75	252.0000	252.0050	0.0049
1.80	293.0692	293.0752	0.0059
1.85	340.7929	340.8000	0.0070
1.90	396.2480	396.2565	0.0084
1.95	460.6857	460.6958	0.0100
2	535.5597	535.5717	0.0120