计算方法实验报告

姓名： 孟卜凡
学号： 200111314
院系：计算机科学与技术学院
专业：计算机
班级：13班

1 实验题目：拉格朗日（Lagrange）插值

1.1 问题分析

1. 利用拉格朗日插值算法，根据参考程序流程实现算法，利用多项式$P_n(x)$来求$f(x)$的近似值。
2. 通过对具体实例的分析探索插值多项式次数对结果的影响
3. 通过对具体实例的分析探索插值区间大小对结果的影响
4. 理解插值问题的内插与外推的方法，对两者可靠性进行比较

1.2 数学原理

1.2.1 插值基函数

令插值基函数 $l_j(x)\,(j=0,1,2,\ldots,n)$ 为如下的多项式：

$$l_j(x)=\begin{cases}0, & i\neq j\\ 1, & i=j\end{cases}$$

1.2.2 Lagrange插值公式

显然存在某多项式

$$y(x)=\sum_{j=0}^{n} f(x_j)l_j(x) \tag{1}$$

满足插值条件，$l_j(x)$ 有 n 个零点 $x_0,x_1,x_2,\ldots,x_{j-1},xj+1,\ldots,x_n$ ，所以应当具有形式

$$l_j(x)=A_j(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n),$$

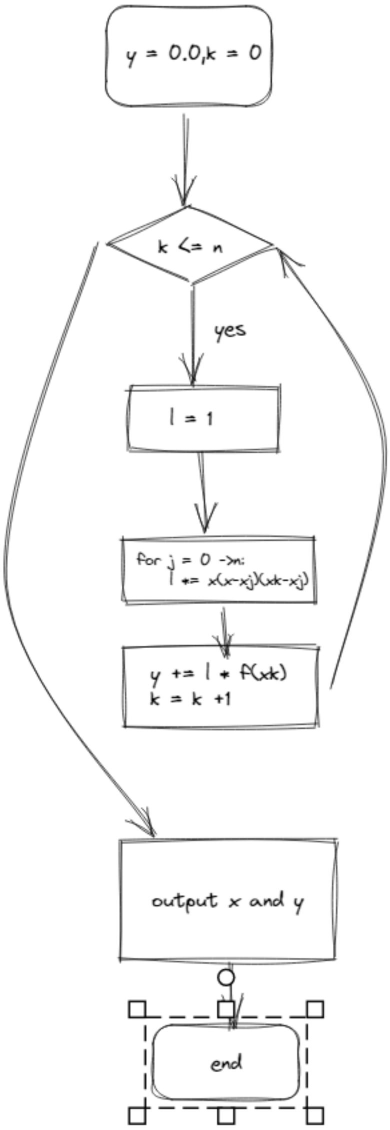
结合$l_j(x_j)=1$ 可求得 A_j，综合得

$$l_j(x)=\frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}, \quad j=0,1,2,\ldots,n \tag{2}$$

其中式（1）称为Lagrange插值公式，式（2）称为 Lagrange 插值多项式，记作 $L_n(x)$

1.3 程序设计流程

1.3.1 流程图



1.3.2 代码

1.3.2.1 函数实现

```
%------------------------file name: Lagrange_vec--------------------
function v = lagrange_vec(x_in, y_in, u)
n = length(x_in);
v = zeros(size(u));
for k = 1:n
    w = ones(size(u));
    for j = [1:k-1 k+1:n] %除去它自己不要乘
        w = (u-x_in(j))./(x_in(k)-x_in(j)).*w; %Lagrange多项式
    end
    v = v + w*y_in(k);
end
```

1.3.2.2 交互脚本

```
%----------------------file name: Lagrange_vec_interactive-----------------
-
syms x;
format long;
n = input('多项式的次数n: ');
u = input('请输入待估值点序列u的值(例如:[1,2,3,4]): ');
a = input('请输入区间下界a的值: ');
b = input('请输入区间上界b的值: ');
f = input('请输入函数f(x)的表达式(如1+x^2) : ');
x = linspace(a,b,n);
y = subs(f, symvar(f), x);
lagrange_vec(x,y,u);
ans = lagrange_vec(x,y,u);
exact_value = subs(f,symvar(f),u);
fprintf('\t\t\t结果表格\n');
fprintf('-----------------------------------------------------------------------
-\n');
fprintf('\t    u \t   f(u) \t\t   exact \t\t   error\n');
fprintf('\t %2.3f \t %2.9f \t %2.9f\t %2.9f\n ',
[u;ans;exact_value;exact_value-ans]);

```

1.4 实验结果、结论与讨论

1.4.0.3 问题1

拉格朗日插值多项式的次数n越大越好吗?

1.4.0.3.1 输出结果:

- 1.(1) $f(x) = \frac{1}{1+x^2}$

```
                    结果表格(n=5)
-----------------------------------------------------------
        u       f(u)            exact           error
     0.750   0.905441810     0.640000000     -0.265441810
     1.750   0.525799901     0.246153846     -0.279646054
     2.750   0.009553216     0.116788321     0.107235105
     3.750   -0.356826094    0.066390041     0.423216136
     4.750   -0.159544927    0.042440318     0.201985245
                     结果表格(n=10)
------------------------------------------------------------
        u       f(u)            exact           error
     0.750   0.690717622     0.640000000     -0.050717622
     1.750   0.232998135     0.246153846     0.013155711
     2.750   0.112245498     0.116788321     0.004542823
     3.750   0.108400418     0.066390041     -0.042010377
```

```
     4.750   -0.236036985    0.042440318     0.278477303

                        结果表格(n=20)
----------------------------------------------------------
        u       f(u)            exact           error
     0.750   0.641340347     0.640000000     -0.001340347
     1.750   0.249055753     0.246153846     -0.002901907
     2.750   0.128218767     0.116788321     -0.011430446
     3.750   0.190261670     0.066390041     -0.123871629
     4.750   6.415032061     0.042440318     -6.372591743
```

- 1.(2) $f(x) = e^x$

```
                        结果表格(n=5)
----------------------------------------------------------
        u       f(u)            exact           error
     -0.95   0.386293876     0.386741023     0.000447148
     -0.05   0.951334528     0.951229425     -0.000105103
     0.05    1.051164240     1.051271096     0.000106857
     0.95    2.586322530     2.585709659     -0.000612871
                        结果表格(n=10)
----------------------------------------------------------
        u       f(u)            exact           error
     -0.95   0.386741027     0.386741023     -0.000000003
     -0.05   0.951229425     0.951229425     -0.000000000
     0.05    1.051271096     1.051271096     -0.000000000
     0.95    2.585709663     2.585709659     -0.000000004
                        结果表格(n=20)
----------------------------------------------------------
        u       f(u)            exact           error
     -0.95   0.386741023     0.386741023     0.000000000
     -0.05   0.951229425     0.951229425     0.000000000
     0.05    1.051271096     1.051271096     0.000000000
     0.95    2.585709659     2.585709659     0.000000000
```

1.4.0.4 问题2

插值区间越小越好吗?

1.4.0.4.1 输出结果

- 2.(1)

```
                        结果表格(n=5)
---------------------------------------------------------
        u       f(u)            exact           error
     -0.95   0.513552500     0.525624179     0.012071679
     -0.05   0.997752500     0.997506234     -0.000246266
```

```
    0.05    0.997752500     0.997506234     -0.000246266
    0.95    0.513552500     0.525624179     0.012071679

                    结果表格(n=10)
----------------------------------------------------------
       u       f(u)            exact            error
    -0.95   0.524273975     0.525624179     0.001350204
    -0.05   0.997464702     0.997506234     0.000041533
    0.05    0.997464702     0.997506234     0.000041533
    0.95    0.524273975     0.525624179     0.001350204

                    结果表格(n=20)
----------------------------------------------------------
       u       f(u)            exact            error
    -0.95   0.525631202     0.525624179     -0.000007023
    -0.05   0.997506234     0.997506234     0.000000000
    0.05    0.997506234     0.997506234     0.000000000
    0.95    0.525631202     0.525624179     -0.000007023
```

- 2.(2)

```
                   结果表格(n=5)
-------------------------------------------------------------
       u       f(u)            exact            error
    -4.75   -1.932149264    0.008651695     1.940800959
    -0.25   1.427537021     0.778800783     -0.648736238
    0.25    0.588185464     1.284025417     0.695839953
    4.75    123.714558835   115.584284527   -8.130274308
                    结果表格(n=10)
-------------------------------------------------------------
       u       f(u)            exact            error
    -4.75   0.042515959     0.008651695     -0.033864263
    -0.25   0.779562066     0.778800783     -0.000761282
    0.25    1.284820075     1.284025417     -0.000794659
    4.75    115.663039200   115.584284527   -0.078754673
                    结果表格(n=20)
-------------------------------------------------------------
       u       f(u)            exact            error
    -4.75   0.008651704     0.008651695     -0.000000009
    -0.25   0.778800783     0.778800783     0.000000000
    0.25    1.284025417     1.284025417     0.000000000
    4.75    115.584284542   115.584284527   -0.000000014
```

1.4.0.5 问题4

考虑拉格朗日插值问题，内插比外推更可靠吗?

1.4.0.5.1 结果输出

```
                  (1) 结果表格 x_in = [1,4,9]
------------------------------------------------------------
       u      f(u)            exact           error
     5.00   2.266666667     2.236067977     -0.030598689
    50.00   -20.233333333   7.071067812     27.304401145
   115.00   -171.900000000  10.723805295    182.623805295
   185.00   -492.733333333  13.601470509    506.334803842

                  (2) 结果表格 x_in = [36,49,64]
------------------------------------------------------------
       u      f(u)            exact           error
     5.00   3.115750916     2.236067977     -0.879682938
    50.00   7.071794872     7.071067812     -0.000727060
   115.00   10.167032967    10.723805295    0.556772328
   185.00   10.038827839    13.601470509    3.562642670

                  (3) 结果表格 x_in = [100,115,185]
------------------------------------------------------------
       u      f(u)            exact           error
     5.00   4.537585505     2.236067977     -2.301517527
    50.00   7.314155736     7.071067812     -0.243087924
   115.00   10.723805295    10.723805295    0.000000000
   185.00   13.601470509    13.601470509    0.000000000


                  (4) 结果表格 x_in = [169,196,225]
------------------------------------------------------------
       u      f(u)            exact           error
     5.00   5.497172049     2.236067977     -3.261104071
    50.00   7.800127714     7.071067812     -0.729059902
   115.00   10.800492611    10.723805295    -0.076687316
   185.00   13.600620325    13.601470509    0.000850184
```

1.4.0.5.2 回答

不一定，取决于具体函数的形式，但通常来说连续函数内插的可靠程度更高。外推相当于根据已知点推测未知点的信息。而我们的已知区间内是不含有区间外的信息的。从上述结果也可以看出，当需要推测的点落在 x_in 的内部时，$error$ 的值就会非常的小，精度明显比外推情况更高。

1.4.1 思考题

1. 1.4.1.1 对实验1 存在的问题，应如何解决?

1.4.1.1.3 回答

不是，次数过高的话会出现Runge现象,使用切比雪夫节点代替等距点可以减小震荡，在这种情况下，随着多项式阶次的增加最大误差逐渐减小。这个现象表明高阶多项式通常不适合用于插值。使用分段多项式样条可以避免这个问题。如果要减小插值误差，那么可以增加构成样条的多项式的数目，而不必是增加多项式的阶次。

> 参考：维基百科：龙格现象

2. 1.4.1.2 对实验2 存在的问题的回答，试加以说明

1.4.1.2.4 回答

不一定，虽然从精度上考虑是区间变窄具有一定的合理性，但是过于密集时舍入误差被放大，而且计算量成本也大幅增加，但是却对精度的提升不大。实际计算时应当合理考虑，取合适的区间长度进行插值。

3. 1.4.1.3 如何理解插值问题中的内插和外推?

1.4.1.3.5 回答

对于内插与外推的理解：

- 内插（Interpolation）：We could use our function to predict the value of the dependent variable for an independent variable that is **in the midst of our data**. In this case, we are performing interpolation.
- 外推（Extrapolation）：We could use our function to predict the value of the dependent variable for an independent variable that is **outside the range of our data**. In this case, we are performing extrapolation.

> 参考: ThoughtCo./The Difference Between Extrapolation and Interpolation

---

提取上述的关键词，内插就是用一些已知的值去估计已知数据区间内的未知值，外推就是用一些已知的值去估计已知数据区间外的未知值。

2 实验题目：龙贝格（Romberg）积分法

2.1 问题分析

实验目的：利用龙贝格积分法计算积分 $\int_a^b f(x)\,dx$

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

在等距基点的情况下，用计算机计算积分值通常都采用把区间逐次分半的方法进行。这样，前一次分割得到的函数值在分半以后仍可被利用,且易于编程。

本实验要求给出 $a, b, N, f(x)$ 的情况下输出 Romberg 积分的数表 T

2.2 数学原理

在数值方法中经常使用一个序列 $f_1, f_2, \ldots$来逼近 f^*，并且理论上研究其收敛的误差，即余项。在余项估计式的基础上通过外推加速收敛。借助 **Richardson 外推**的思想，由梯形公式的简单组合可以得到比 h^2 更高阶的求积公式如下:

$$\begin{cases} T_0(h) = T(h) \\ T_m(h) = \dfrac{4^m T_{m-1}(\frac{h}{2}) - T_{m-1}(h)}{4^m - 1} \end{cases}$$

2.3 程序设计流程

根据数学原理易得以下交互脚本代码:

```matlab
% Romberg integration algorithm
% Interactive

 f = @(x) x^2*exp(x)
 a = input('Enter lower limit, a:  ');
 b = input('Enter upper limit, b:  ');
 n = input('Enter no. of subintervals, n:  ');

 h = b-a;
 r = zeros(2,n+1);
 r(1,1) = (f(a)+f(b))/2*h;
 fprintf('\nRomberg integration table:\n');
 fprintf('\n %11.8f\n\n', r(1,1));

 for i = 2:n
    sum = 0;
    for k = 1:2^(i-2)
       sum = sum+f(a+(k-0.5)*h);
    end
    r(2,1) = (r(1,1)+h*sum)/2;

    for j = 2:i
       l = 2^(2*(j-1));
       r(2,j) = r(2,j-1)+(r(2,j-1)-r(1,j-1))/(l-1);
    end
```

```

    for k = 1:i
        fprintf(' %11.8f',r(2,k));
    end

    fprintf('\n\n');
    h = h/2;
    for j = 1:i
        r(1,j) = r(2,j);
    end
end
```

2.4 实验结果、结论与讨论

2.4.1 实验结果

2.4.1.1 1.1 $\int_0^1 x^2e^x\,dx$

```
1.35914091

0.88566062  0.72783385

0.76059633  0.71890824  0.71831320

0.72889018  0.71832146  0.71828234  0.71828185

0.72093578  0.71828431  0.71828184  0.71828183  0.71828183

0.71894543  0.71828198  0.71828183  0.71828183  0.71828183  0.71828183
```

2.4.1.2 1.2 $\int_1^3 e^x sinx\,dx$

```
 5.12182642

 9.27976291 10.66574174

 10.52055428 10.93415141 10.95204539

 10.84204347 10.94920653 10.95021020 10.95018107

 10.92309389 10.95011070 10.95017097 10.95017035 10.95017031

 10.94339842 10.95016660 10.95017033 10.95017031 10.95017031 10.95017031
```

2.4.1.3 1.3 $\int_0^1 \frac{4}{1+x^2}\,dx$

```
3.00000000

3.10000000 3.13333333

3.13117647 3.14156863  3.14211765

3.13898849 3.14159250  3.14159409  3.14158578

3.14094161 3.14159265  3.14159266  3.14159264  3.14159267

3.14142989 3.14159265  3.14159265  3.14159265  3.14159265  3.14159265

3.14155196 3.14159265  3.14159265  3.14159265  3.14159265  3.14159265  3.14159265
```

2.4.1.4 1.4 $\int_0^1 \frac{1}{1+x}\,dx$

```
0.75000000

0.70833333  0.69444444

0.69702381  0.69325397  0.69317460

0.69412185  0.69315453  0.69314790  0.69314748

0.69339120  0.69314765  0.69314719  0.69314718  0.69314718
```

2.4.2 思考题

2.4.2.1 在实验1 中二分次数和精度的关系如何?

答: 二分次数越多，精度就越高，所以一般设定足够的二分次数以达到精度，进行大型运算时要提前确定好精度要求，避免二分次数过多带来不必要的时间开销。

3 实验题目：四阶龙格—库塔(Runge—Kutta)方法

3.1 问题分析

龙格-库塔(Runge-Kutta)方法是一种在工程上应用广泛的高精度单步算法，其中包括著名的欧拉法，用于数值求解微分方程。由于此算法精度高，采取措施对误差进行抑制，所以其实现原理也较复杂。在各种龙格- 库塔法当中有一个方法十分常用，以至于经常被称为“RK4”或者就是“龙格- 库塔法”。该方法主要是在已知方程导数和初值信息，利用计算机仿真时应用，省去求解微分方程的复杂过程。

本实验要求对于如下形式微分方程和初值

$$
\begin{cases}
\dfrac{\mathrm{d}y}{\mathrm{d}x} = f(x,y), a \le x \le b \\
y(a) = \alpha \qquad h = \dfrac{b-a}{N}
\end{cases}
$$

对于给定的输入a, b, α, N，输出给定

3.2 数学原理

建立微分方程，

$$y' = f(x,y), \quad y(x_0) = y_0, \quad x_0 \le x \le x_n$$

可将其表示为，

$$y_{i+1} = y_i + h\varphi(x_i, y_i)$$

再令$\varphi(x_i, y_i) = a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4$，可得到

$$y_{i+1} = y_i + h(a_1 k_1 + a_2 k_2 + a_3 k_3 + a_4 k_4)$$

其中，

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + b_1 h, y_i + c_{11} k_1 h) \\
k_3 &= f(x_i + b_2 h, y_i + c_{21} k_1 h + c_{22} k_2 h) \\
k_4 &= f(x_i + b_3 h, y_i + c_{31} k_1 h + c_{32} k_2 h + c_{33} k_3 h)
\end{aligned}
$$

令，

$$y_{i+1} = y_i + \frac{1}{6} h (k_1 + 2k_2 + 2k_3 + k_4)$$

其中，

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f\left(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_1 h\right) \\
k_3 &= f\left(x_i + \tfrac{1}{2}h, y_i + \tfrac{1}{2}k_2 h\right) \\
k_4 &= f(x_i + h, y_i + k_3 h)
\end{aligned}
$$

可利用此方法求解微分方程，且对于一般的ODE来说精度已经足够

3.3 程序设计流程

3.3.1 函数主体

```
function [x,y] = RungeKutta4(fun,a, b, y0, n)
h = (b-a)/n;
x = zeros(n+1,1);
y = zeros(n+1,1);
x(1) = a;
y(1) = y0;
for i = 1:n
```

```
    x(i+1) = x(i)+h;
    K1 = fun(x(i), y(i));
    K2 = fun(x(i)+h/2, y(i)+K1*h/2);
    K3 = fun(x(i)+h/2, y(i)+K2*h/2);
    K4 = fun(x(i)+h, y(i)+K3*h);
    y(i+1) = y(i)+h/6*(K1+2*K2+2*K3+K4);
end
end
```

3.3.2 测试脚本

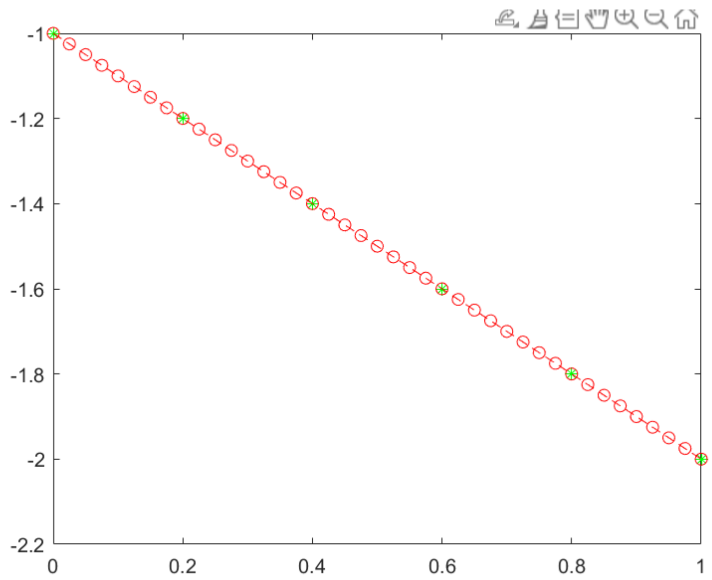
```
clc,clear
set(0,'defaultfigurecolor','w')

f = @(x,y) x +y

n = 5

a = 0; %x初值
b = 1; %x终值
alpha = -1; %y初值

[x,y] = RungeKutta4(f, a, b, alpha, n);
[x2,y2] = ode45(f, [a,b], alpha);    %matlab内部函数,用作参考
plot(x,y,'g--*',x2,y2,'r--o') % 自己实现的是绿色的, 标准库的是红色的

```

3.4 实验结果、结论与讨论

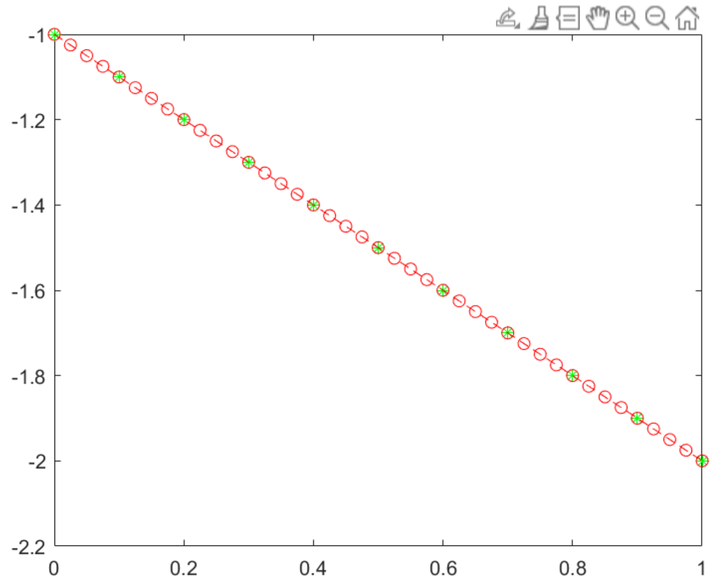
3.4.1 题目 1

3.4.1.1 1.1 $\frac{dy}{dx} = x + y$

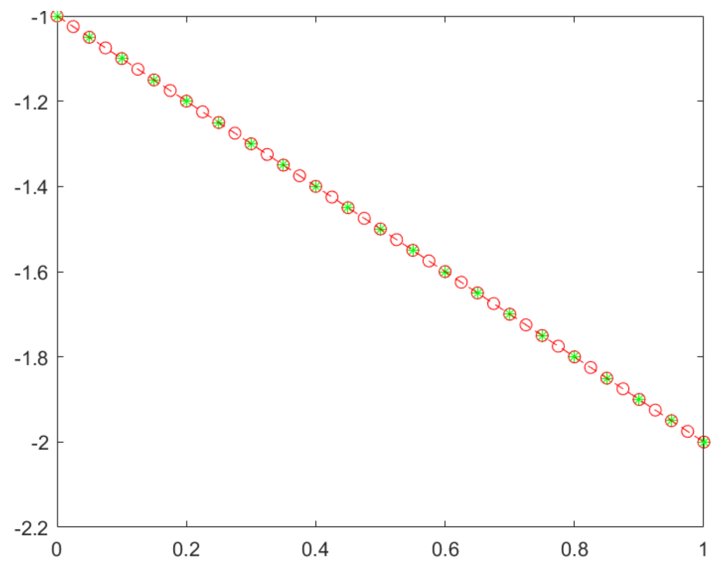
- n = 5


-1
-1.2
-1.4
-1.6
-1.8
-2
-2.2
0
0.2
0.4
0.6
0.8
1

- n = 10

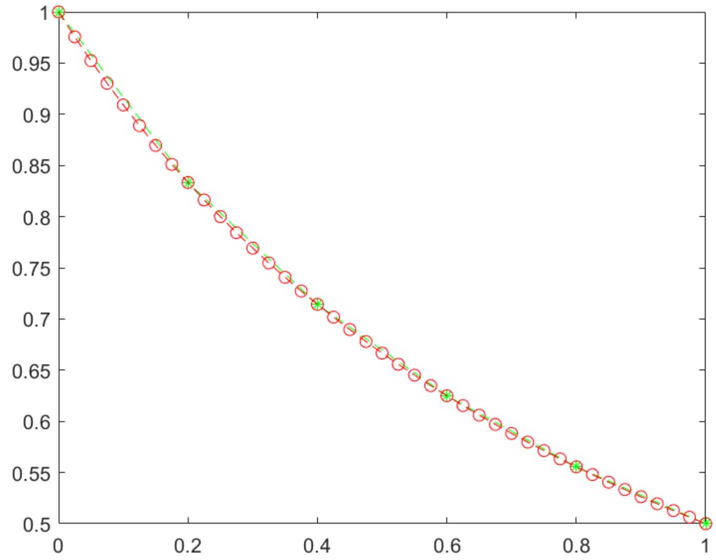

-1
-1.2
-1.4
-1.6
-1.8
-2
-2.2
0
0.2
0.4
0.6
0.8
1

- n = 20

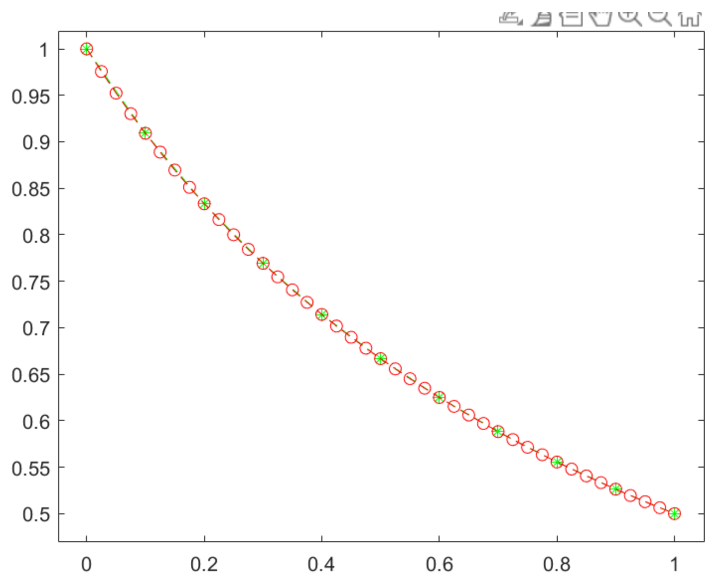

-1
-1.2
-1.4
-1.6
-1.8
-2
-2.2
0
0.2
0.4
0.6
0.8
1

3.4.1.2 $1.2\frac{\mathrm{d}y}{\mathrm{d}x} = -y^2$

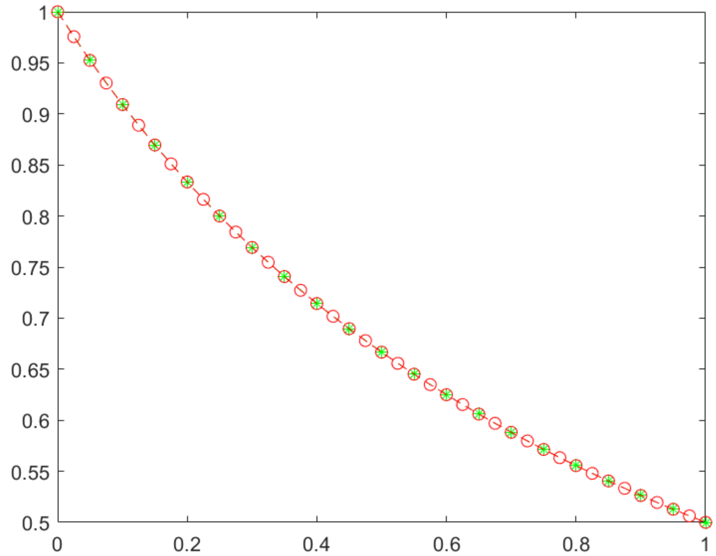
- n = 5



- n = 10



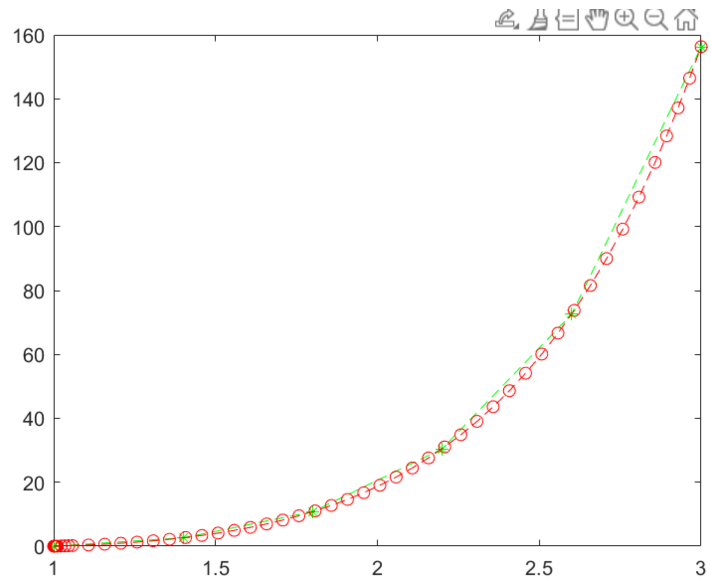
- n = 20



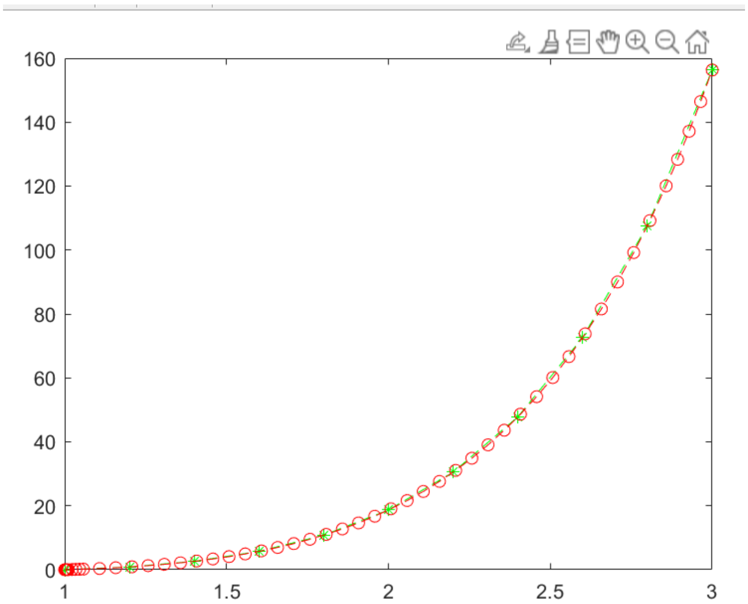
3.4.2 题目2

3.4.2.1 2.1 $\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{2y}{x} + x^2e^x$

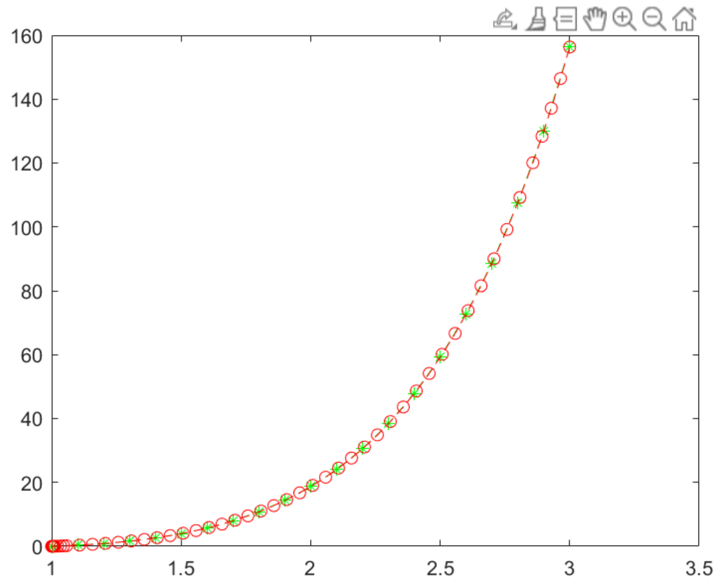
- n = 5



- n = 10

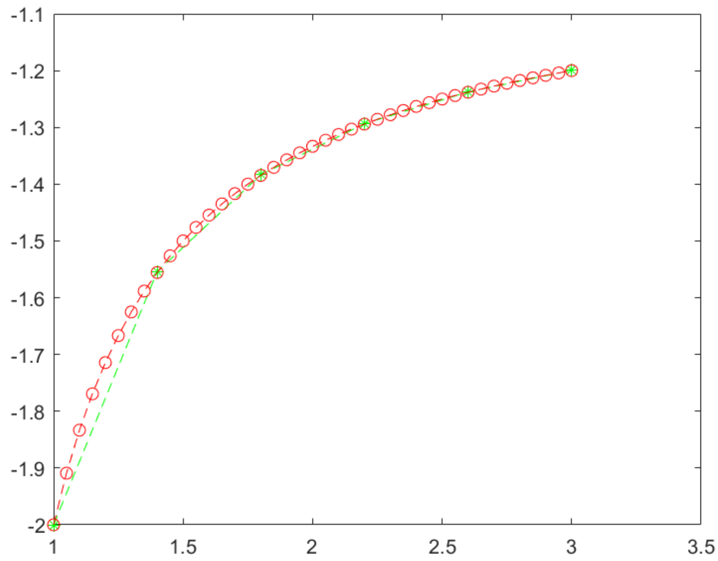


- n = 20

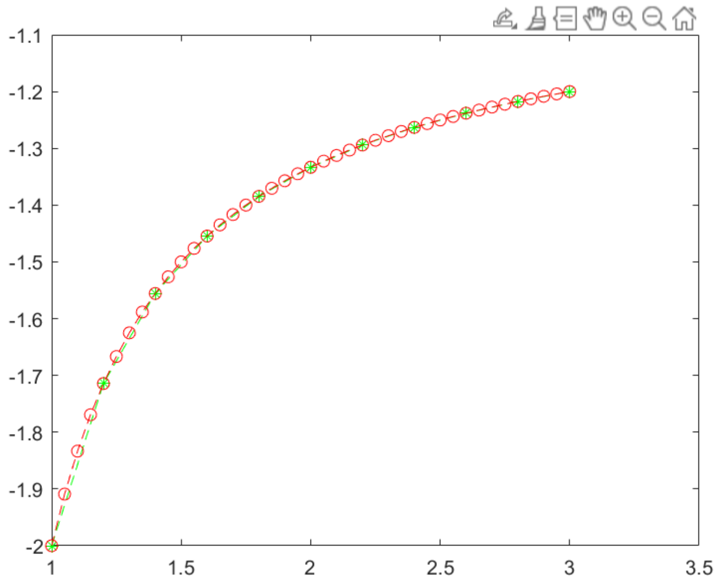


3.4.2.2 2.2 $\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{y^2+y}{x}$

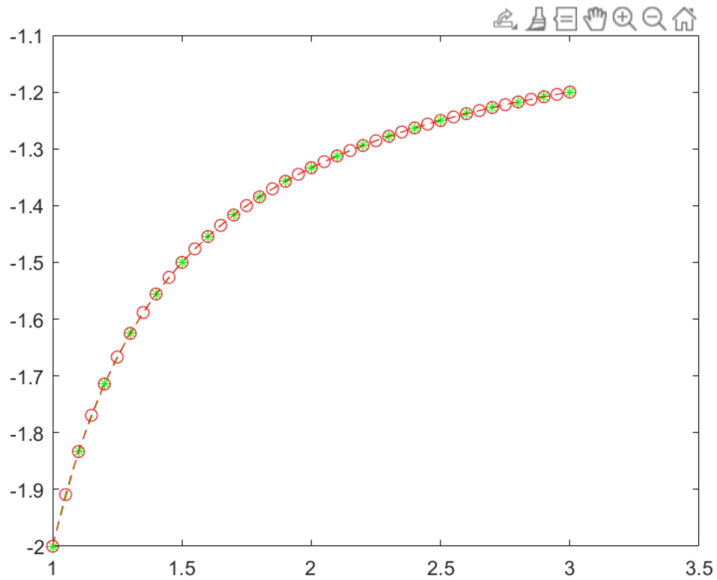
- n = 5


-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
-1.8
-1.9
-2
1
1.5
2
2.5
3
3.5

- n = 10


-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
-1.8
-1.9
-2
1
1.5
2
2.5
3
3.5

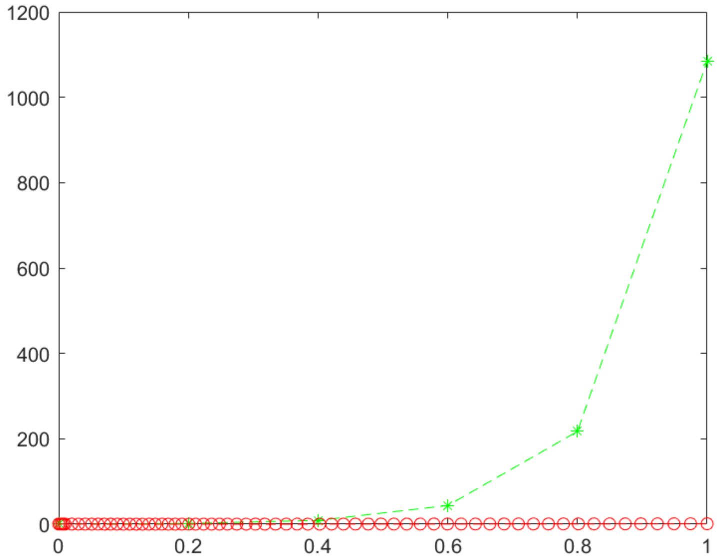
- n = 20


-1.1
-1.2
-1.3
-1.4
-1.5
-1.6
-1.7
-1.8
-1.9
-2
1
1.5
2
2.5
3
3.5

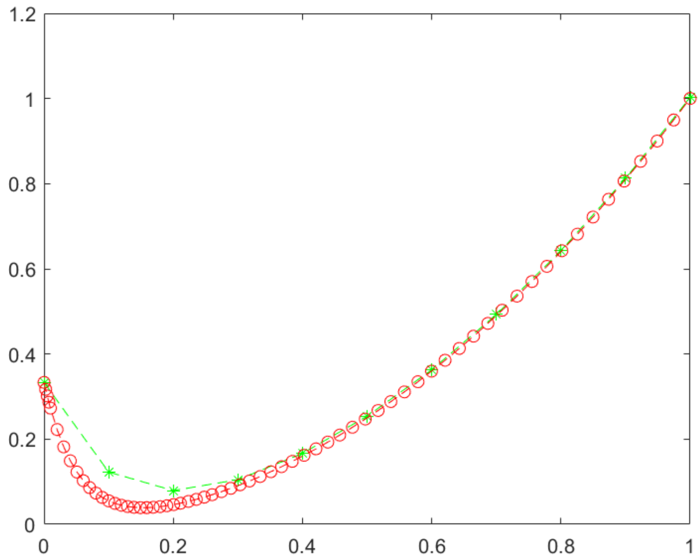
3.4.3 题目3

3.4.3.1 3.1 $\frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - x^2) + 2x$

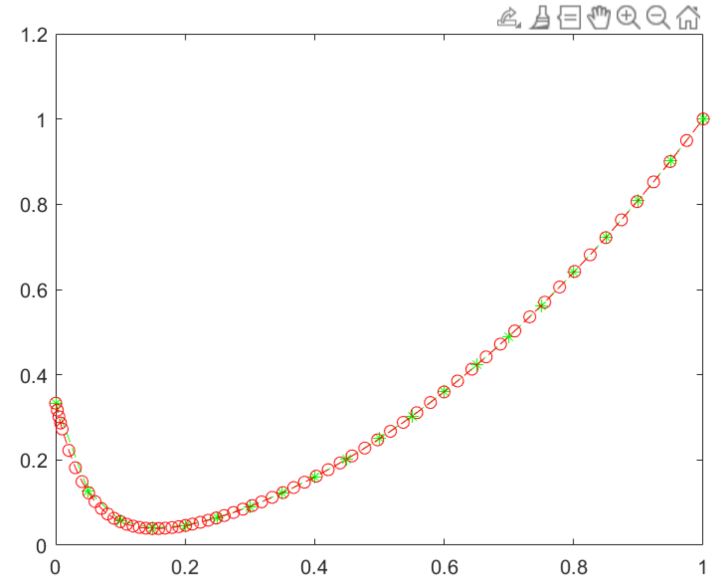
- n = 5



- n = 10

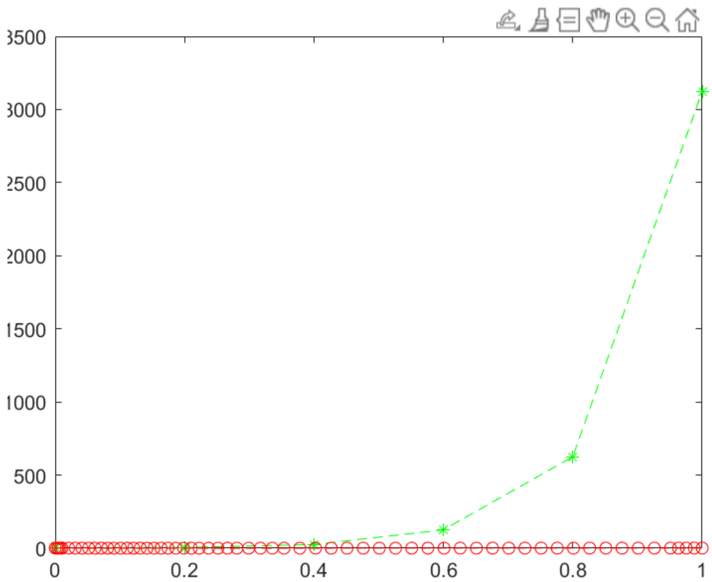


- n = 20

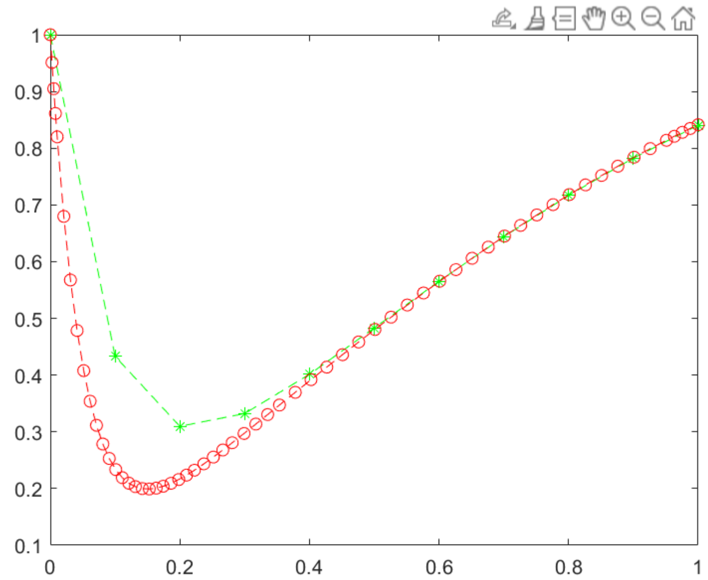


3.4.3.2 3.2 $\frac{\mathrm{d}y}{\mathrm{d}x} = -20y + 20\sin x + \cos x$

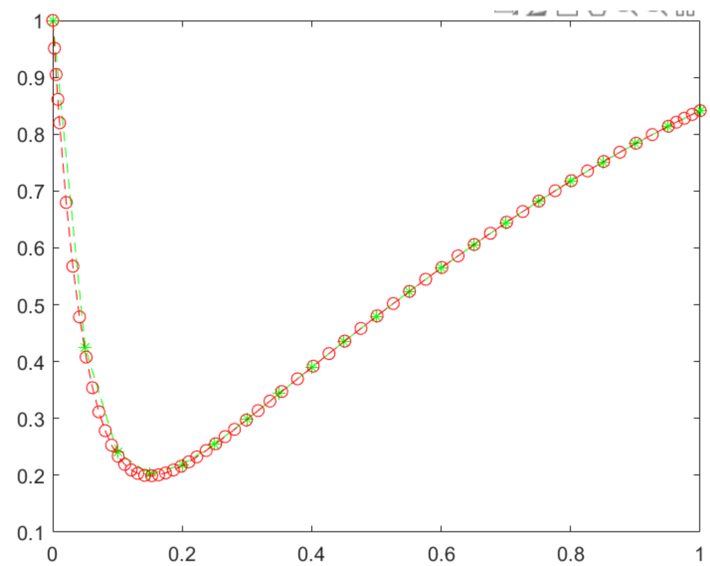
- n = 5



- n = 10

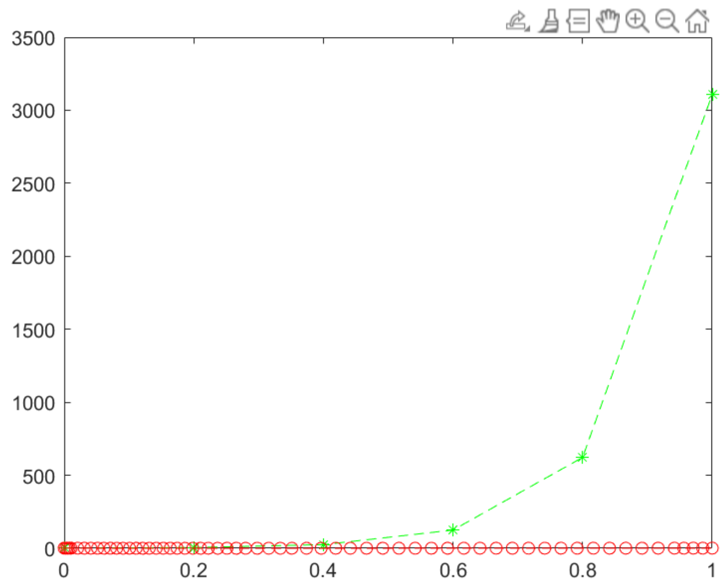


- n = 20

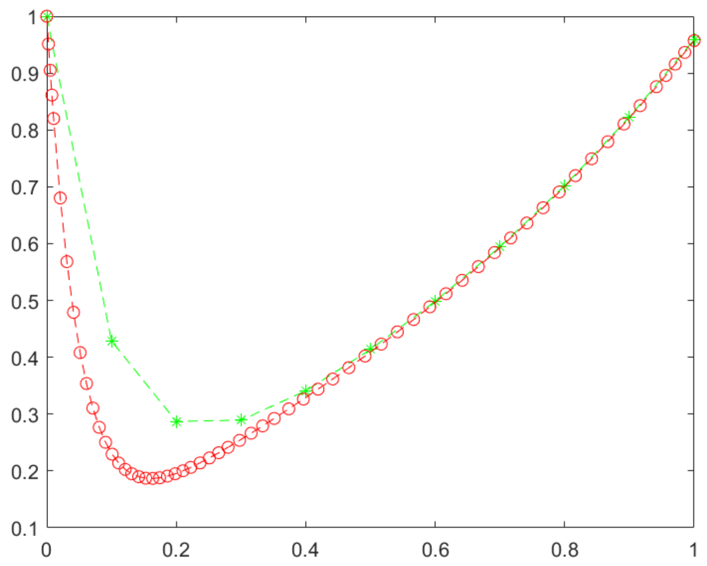


3.4.3.3 3.3 $\frac{\mathrm{d}y}{\mathrm{d}x} = -20(y - x^x \sin x) + e^x(\sin x + \cos x)$

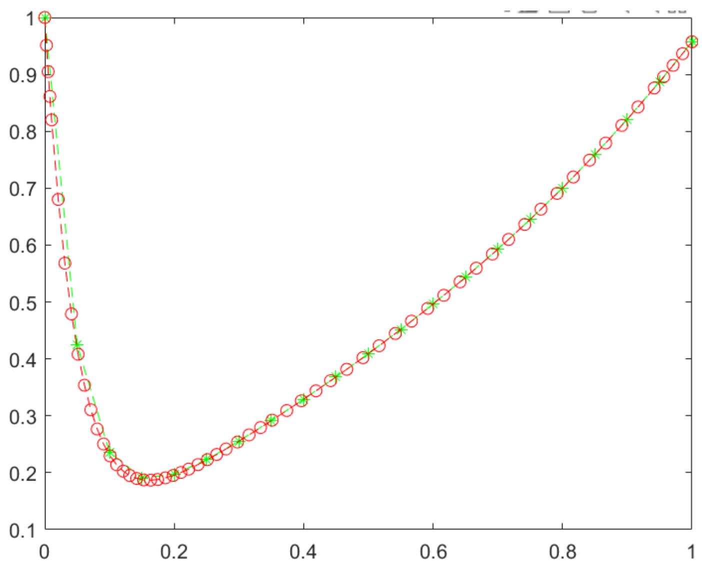
- n = 5



- n =10



- n = 20



3.4.4 思考题

1. 对实验 1，数值解和解析解相同吗？为什么？试加以说明。

答：

- 对于1.1：数值解和解析解是相同的，因为本题的解是线性的，能够通过所得数值解的两个点确定所求直线。
- 对于1.2：虽然数值解与解析解之间的差值已经极小，但是仍然是不相等的，RK-4方法本就有误差，不可能保证数值解和解析解完全相同

2. 对实验 2， N 越大越精确吗？试加以说明。

答：

- 是的，随着 N 的增大精度确实在提高，但是精度增加不大的同时计算量大幅提升，所以实际运用中还是需要按需进行精度的设置。

2. 对于实验 3， N 较小时会出现什么现象，试加以说明。

答:

- 在 N 较小时数值解与解析解出现了较大的差一，说明根据函数性质的不同，区间的不同，还是需要保证有一定量的 N 来保证结果的精度，不然就会出现较大的失真，至于这个 N 该如何选取，需要根据实际进行判断。

4 实验题目：牛顿（Newton）迭代法

4.1 问题分析

实验的目的为使用牛顿迭代法，在给定初值的条件下数值求解非线性方程的根。

在给定输入初值精度以及最大迭代次数的情况下输出方程 $f(x)=0$ 的根 x^* 的近似值或者计算失败的标志

牛顿迭代法：多数方程不存在求根公式，因此求精确根非常困难，甚至不可解，从而寻找方程的近似根就显得特别重要。方法使用函数 $f(x)$ 的泰勒级数的前面几项来寻找方程 $f(x)=0$ 的根。牛顿迭代法是求方程根的重要方法之一，其最大优点是在方程 $f(x)=0$ 的单根附近具有平方收敛，而且该法还可以用来求方程的重根、复根，此时线性收敛，但是可通过一些方法变成超线性收敛。另外该方法广泛用于计算机编程中。

4.2 数学原理

用牛顿迭代法解非线性方程，是把非线性方程 $f(x)=0$ 线性化的一种近似方法。把 $f(x)$ 在点 x_0的某邻域内展开成泰勒级数

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)(x-x_0)^2}{2!}+\cdots+\frac{f^{(n)}(x_0)(x-x_0)^n}{n!}+R_n(x)$$

取其线性部分（即泰勒展开的前两项），并令其等于0，即

$$f(x_0)+f'(x_0)(x-x_0)=0$$

以此作为非线性方程 $f(x)=0$ 的近似方程，若 $f\prime(x)\neq 0$ 则其解为

$$x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

这样，得到牛顿迭代法的一个迭代关系式：

$$x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}$$

4.3 程序设计流程

4.3.1 函数实现

```
function Newton(alpha,eps1,eps2,N)

syms x;
f(x) = cos(x)-x;
```

```matlab

display('The equation to be solved is: f(x) = cos(x) - x ')

n = 1;
x0 = alpha;
while(n<=N)
    F = double(subs(f(x),x,x0));
    Diif_F = double(subs(diff(f(x)),x,x0));
    if(abs(F)<eps1)
        fprintf('The root is: %f\n',x0)
        return;
    end
    if(abs(Diif_F)<eps2) % 寻找失败了，停机输出
        disp("Not found");
        return;
    end
    x1 = double(x0 - F/Diif_F);
    Tol = double(abs(x1-x0));
    if(Tol<eps1)
        fprintf('The root is: %f\n',x0)
        return;
    end
    n = n+1;
    x0 = x1;
end
disp("Not found");
end
```

4.3.2 测试脚本

```matlab
clc
clear
format long

alpha = input('请输入初值α:');
eps1 = input('请输入精度ε1:');
eps2 = input('请输入精度ε2:');
n = input('请输入最大迭代次数N:');
Newton(alpha,eps1,eps2,n);
```

4.4 实验结果、结论与讨论

4.4.1 问题1

4.4.1.1 （1） $\cos x - x = 0 \quad \varepsilon_1 = 10^{-6} \quad \varepsilon_2 = 10^{-4} \quad x_0 = \frac{\pi}{4} \quad N = 10$

```
>> Newton(pi/4,1e-6,1e-4,10)
The equation to be solved is: f(x) = cos(x) - x
The root is: 0.739085
```

4.4.1.2 （2） $e^{-x} - \sin x \quad \varepsilon_1 = 10^{-6} \quad \varepsilon_2 = 10^{-4} \quad x_0 = 0.6 \quad N = 10$

```
>> Newton(0.6,1e-6,1e-4,10)
The equation to be solved is: f(x) = exp(-x) - sin(x)
The root is: 0.588533
```

4.4.2 问题2

4.4.2.1 (1) $x - e^{-x} = 0 \quad \varepsilon_1 = 10^{-6} \quad \varepsilon_2 = 10^{-4} \quad x_0 = 0.5 \quad N = 10$

```
>> Newton(0.5,1e-6,1e-4,10)
The equation to be solved is: f(x) = x - exp(-x)
The root is: 0.567143
```

4.4.2.2 (2) $x^2 - 2xe^{-x} + e^{-2x} = 0 \quad \varepsilon_1 = 10^{-6} \quad \varepsilon_2 = 10^{-4} \quad x_0 = 0.5 \quad N = 20$

```
>> Newton(0.5,1e-6,1e-4,20)
The equation to be solved is: f(x) = x - exp(-x)
The root is: 0.566606
```

4.4.3 思考题:

1. #### 4.4.3.1 对实验1 确定初值的原则是什么？实际计算中应如何解决?

4.4.3.1.1 回答

原则应该是:

1. 区间存在根，可以通过二分法来找到一个含有根的区间
2. 尽可能与根接近，实际计算中可以先绘图，通过图像估计初值，通常来说会获得更加准确的初值

2. #### 4.4.3.2 对实验2 如何解释在计算中出现的现象？试加以说明

4.4.3.2.2 回答

实验2中（2）的收敛速度明显较慢，原因是方程存在重根，当存在重根时牛顿迭代法的收敛速度为线性收敛，所以收敛速度变慢了。

5 实验题目：高斯（Gauss）列主元消去法

5.1 问题分析

本实验为高斯列主元消去法，本实验主要学习高斯消元法的代码实现，对于 n 阶线性方程组 $Ax=b$，首先进行列主元消元过程，最后进行回代，得到方程的解或确定该方程为奇异的。

高斯消去法从第k步到第k+1步的消元过程，必须满足条件$a_{kk}^{(k-1)} \neq 0$。而这个元素即被称为第k步的主元(素)。显然，高斯消去法是按方程排列的自然顺序产生主元的，这样，一旦出现

$a_{kk}^{(k-1)}=0$ 计算就归于失败，而且即使$a_{kk}^{(k-1)} \neq 0$，但若其绝对值很小，也将会因用它作除数，引起其他元素的数量级及舍人误差急剧增大，导致最终计算结果不可靠。为了避免在高斯消去法应用中可能出现的这类问题，就发展形成了列主元、全主元等多种消去法。这些方法的基本点在于对高斯消去法的过程作某些技术性修改，全面或局部地选取绝对值最大的元素为主元素，从而构成了相应的主元(素)消去法。列主元(素)消去法以处理简单、相对计算量小的特点，在各类主元消去法中得到最为广泛的应用。

5.2 数学原理

高斯消去法中第k-1步消元得到的结果可由分块矩阵记为

$$\left[\boldsymbol{A}^{(k-1)}, \boldsymbol{b}^{(k-1)}\right]=\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \vdots & \boldsymbol{b}_1 \\ \mathbf{0} & \boldsymbol{A}_{kk} & \vdots & \boldsymbol{b}_2 \end{bmatrix}. \qquad (1)$$

式中 b_1, b_2 对应于右端常数列的两个子块，而

$$\boldsymbol{A}_{11}=\begin{bmatrix} a_{11}^{(0)} & a_{12}^{(0)} & \cdots & a_{1,k-1}^{(0)} \\ & a_{22}^{(1)} & \cdots & a_{2,k-1}^{(1)} \\ & & & \vdots \\ & & & a_{k-1,k-1}^{(k-2)} \end{bmatrix},$$

$$\boldsymbol{A}_{kk}=\begin{bmatrix} a_{kk}^{(k-1)} & a_{k,k+1}^{(k-1)} & \cdots & a_{k,n}^{(k-1)} \\ a_{k+1,k}^{(k-1)} & a_{k+1,k+1}^{(k-1)} & \cdots & a_{k+1,n}^{(k-1)} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n,k}^{(k-1)} & a_{n,k+1}^{(k-1)} & \cdots & a_{n,n}^{(k-1)} \end{bmatrix}.$$

与高斯消去法不同的是，列主元消去法在第k步消元之前，先在它的第k列主对角元$a_{kk}^{(k-1)}$及其下方的所有元素中(亦即 A_{kk} 的第一列元素中)选出绝对值最大的元素 $a_{ik,k}^{(k-1)}$ 作为这一列的主元，即

$$\left|a_{i_k,k}^{(k-1)}\right|=\max_{k\le i\le n}\left|a_{ik}^{(k-1)}\right| \qquad (k=1,2,\ldots,n-1),$$

对式(1)，作初等行变换

$$[i_k] \leftrightarrow [k], \qquad (i_k \ge k),$$

这样，就把$a_{ik,k}^{(k-1)}$换到主对角元位置上。经过选列主元与行交换之后，再如高斯消去法一样作行的消元变换。上述过程从第一步消元开始执行，即k=1，2，⋯，n—1，这就构成了**列主元消去法**

5.3 程序设计流程

5.3.1 函数主体

```
function Result = Gauss(n, A, b)
    for k = 1:n-1
        max = abs(A(k, k));
        p = k;
        for j = k+1:n
            if(abs(A(j, k)) > max)
                max = abs(A(j, k));
                p = j;
            end
        end
        if(A(p, k) == 0)
            Result = 'Singular matrix!';
            return;
        end
        if(p ~= k)
           A([k p], :) = A([p k], :);
           b([k p], :) = b([p k], :);
        end
        for i = k+1:n
            Mik = A(i, k)/A(k, k);
            for j = k:n
                A(i, j) = A(i, j) - A(k, j)*Mik;
            end
            b(i) = b(i) - b(k)*Mik;
        end
    end
    if(A(n, n) == 0)
        Result = 'Singular matrix!';
        return;
    end
    Result = zeros(n, 1);
    Result(n, 1) = b(n)/A(n, n);
    for k = n-1:-1:1
        Sum = 0;
        for j = k+1:n
            Sum = Sum + A(k, j)*Result(j, 1);
        end
        Result(k, 1) = (b(k) - Sum)/A(k, k);
    end
end
```

5.4 实验结果、结论与讨论

5.4.1 问题1

$$（1）\begin{bmatrix} 0.4096 & 0.1234 & 0.3678 & 0.2943 \\ 0.2246 & 0.3872 & 0.4015 & 0.1129 \\ 0.3645 & 0.1920 & 0.3781 & 0.0643 \\ 0.1784 & 0.4002 & 0.2786 & 0.3927 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1.1951 \\ 1.1262 \\ 0.9989 \\ 1.2499 \end{bmatrix}$$

```
ans =

   1.000000000000003
   1.000000000000002
   0.999999999999997
   0.999999999999999
```

$$（2）\begin{bmatrix} 136.01 & 90.860 & 0 & 0 \\ 90.860 & 98.810 & -67.590 & 0 \\ 0 & -67.590 & 132.01 & 46.260 \\ 0 & 0 & 46.260 & 177.17 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 226.87 \\ 122.08 \\ 110.68 \\ 223.43 \end{bmatrix}$$

```
ans =

   1.000000000000118
   0.999999999999824
   0.999999999999901
   1.000000000000026
```

$$（3）\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 \\ 1/2 & 1/3 & 1/4 & 1/5 \\ 1/3 & 1/4 & 1/5 & 1/6 \\ 1/4 & 1/5 & 1/6 & 1/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 25/12 \\ 77/60 \\ 57/60 \\ 319/420 \end{bmatrix}$$

```
ans =

   0.999999999999993
   1.000000000000069
   0.999999999999855
   1.000000000000085
```

$$（4）\begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}$$

```
Gauss 1.(4)

ans =

     1
     1
     1
     1
```

实验题目的准确结果：

（1） $\boldsymbol{x}=(x_1,x_2,x_3,x_4)^T=(1,1,1,1)^T$；

（2） $\boldsymbol{x}=(x_1,x_2,x_3,x_4)^T=(1,1,1,1)^T$；

（3） $\boldsymbol{x}=(x_1,x_2,x_3,x_4)^T=(1,1,1,1)^T$；

（4） $\boldsymbol{x}=(x_1,x_2,x_3,x_4)^T=(1,1,1,1)^T$。

5.4.2　　问题2

（1） $$\begin{bmatrix} 197 & 305 & -206 & -804 \\ 46.8 & 71.3 & -47.4 & 52.0 \\ 88.6 & 76.4 & -10.8 & 802 \\ 1.45 & 5.90 & 6.13 & 36.5 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}=\begin{bmatrix} 136 \\ 11.7 \\ 25.1 \\ 6.60 \end{bmatrix}$$

```
ans =

   0.953679106901772
   0.320956845521104
   1.078708075793238
  -0.090108509539579
```

（2） $$\begin{bmatrix} 0.5398 & 0.7161 & -0.5554 & -0.2982 \\ 0.5257 & 0.6924 & 0.3565 & -0.6255 \\ 0.6465 & -0.8187 & -0.1872 & 0.1291 \\ 0.5814 & 0.9400 & -0.7779 & -0.4042 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}=\begin{bmatrix} 0.2058 \\ -0.0503 \\ 0.1070 \\ 0.1859 \end{bmatrix}$$

```
ans =

   0.516177297958542
   0.415219472830135
   0.109966102867889
   1.036539223336201
```

$$（3）\begin{bmatrix} 10 & 1 & 2 \\ 1 & 10 & 2 \\ 1 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 13 \\ 13 \\ 7 \end{bmatrix}$$

```
ans =

   1.000000000000000
   1.000000000000000
   1.000000000000000
```

$$（4）\begin{bmatrix} 4 & -2 & -4 \\ -2 & 17 & 10 \\ -4 & 10 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 25 \\ 15 \end{bmatrix}$$

```
ans =

     1
     1
     1
```