计　算　方　法

实验指导与实验报告

姓名_____袁野_____

学号___1190200122__

院系__计算学部____

专业__计算机科学与技术___

哈尔滨工业大学

实验报告一

题目（摘要）

问题 1： 拉格朗日插值多项式的次数 越大越好吗？

问题 2： 插值区间越小越好吗？

问题 3： 在区间 考虑拉格朗日插值问题，为了使得插值误差较小，应如何选取插值节点？

问题 4： 考虑拉格朗日插值问题，内插比外推更可靠吗？

前　言：（目的和意义）

在数值分析中，拉格朗日插值法是以法国十八世纪数学家约瑟夫·路易斯·拉格朗日命名的一种多项式插值方法。许多实际问题中都用函数来表示某种内在联系或规律，而不少函数都只能通过实验和观测来了解。如对实践中的某个物理量进行观测，在若干个不同的地方得到相应的观测值，拉格朗日插值法可以找到一个多项式，其恰好在各个观测的点取到观测到的值。这样的多项式称为拉格朗日（插值）多项式。数学上来说，拉格朗日插值法可以给出一个恰好穿过二维平面上若干个已知点的多项式函数。

数学原理

对于给定的 $k+1$ 个点：$(x_0, y_0) \cdots (x_n, y_n)$，拉格朗日插值法的思路是找到一个在一点 x_j 取值为1，而在其他点取值都是 0 的多项式 $\ell_j(x)$。这样，多项式 $y_j \ell_j(x)$ 在点 x_j 取值为 y_j，而在其他点取值都是 0。而多项式 $L(x) = \sum_{j=0}^{k} y_j \ell_j(x)$ 就可以满足

$$L(x_j) = \sum_{i=0}^{k} y_i \ell_i(x_j) = 0 + 0 + \cdots + y_j + \cdots + 0 = y_j$$

在其它点取值为 0 的多项式容易找到，例如：

$$(x - x_0) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_k)$$

它在点 x_j 取值为：

$$(x_j - x_0) \cdots (x_j - x_{j-1})(x_j - x_{j+1}) \cdots (x_j - x_k)$$

由于已经假定 x_i 两两互不相同，因此上面的取值不等于 0。于是，将多项式除以这个取值，就得到一个满足“在 x_j 取值为 1，而在其他点取值都是 0 的多项式”：

$$\ell_j(x) = \prod_{i=0, i \neq j}^{k} \frac{x - x_i}{x_j - x_i} = \frac{x - x_0}{x_j - x_0} \cdots \frac{x - x_{j-1}}{x_j - x_{j-1}} \frac{x - x_{j+1}}{x_j - x_{j+1}} \cdots \frac{x - x_k}{x_j - x_k}$$

这就是拉格朗日基本多项式。

次数不超过 k 的拉格朗日多项式至多只有一个，因为对任意两个次数不超过 k 的拉格朗日多项式：P_1 和 P_2，它们的差 $P_1 - P_2$ 在所有 $k+1$ 个点上取值都是 0，因此必然是多项式 $(x - x_0)(x - x_1) \cdots (x - x_k)$ 的倍数。因此，如果这个差 $P_1 - P_2$ 不等于 0，次数就一定不小于 $k+1$。但是 $P_1 - P_2$ 是两个次数不超过 k 的多项式之差，它的次数也不超过 k。所以 $P_1 - P_2 = 0$，也就是说 $P_1 = P_2$。这样就证明了唯一性。

程序设计流程

```
double Lagrange(double *x, double *y, int n, double X) {
    /* *x *y 为数据点， n 为数据点个数-1， x 为插值点*/
    double Y = 0;
    for (int k=0; k<=n; ++k) {
        double l = 1;
        for (int j=0; j<=n; ++j) {
            if (k == j) continue;
            l = l*(X-x[j])/(x[k]-x[j]);
        }
        Y += l*y[k];
    }
    return Y;
}
```

实验结果、结论与讨论

问题一：

```
PS D:\github\School_Pro\计算方法> .\Lagrange 1
 ------------------------
This is f(x) = 1/(x^2+1)
When n is equal to 5
x = 0.750000  f(x) = 0.528974 err = -0.111026
x = 1.750000  f(x) = 0.373325 err = 0.127171
x = 2.750000  f(x) = 0.153733 err = 0.036945
x = 3.750000  f(x) = -0.025954 err = -0.092344
x = 4.750000  f(x) = -0.015738 err = -0.058178
When n is equal to 10
x = 0.750000  f(x) = 0.678990 err = 0.038990
x = 1.750000  f(x) = 0.190580 err = -0.055573
x = 2.750000  f(x) = 0.215592 err = 0.098804
x = 3.750000  f(x) = -0.231462 err = -0.297852
x = 4.750000  f(x) = 1.923631 err = 1.881191
When n is equal to 20
x = 0.750000  f(x) = 0.636755 err = -0.003245
x = 1.750000  f(x) = 0.238446 err = -0.007708
x = 2.750000  f(x) = 0.080660 err = -0.036128
x = 3.750000  f(x) = -0.447052 err = -0.513442
x = 4.750000  f(x) = -39.952449 err = -39.994889
 ------------------------
This is f(x) = e^x
When n is equal to 5
x = -0.950000  f(x) = 0.386798 err = 0.000057
x = -0.050000  f(x) = 0.951248 err = 0.000019
x = 0.050000  f(x) = 1.051290 err = 0.000019
x = 0.950000  f(x) = 2.585785 err = 0.000075
When n is equal to 10
x = -0.950000  f(x) = 0.386741 err = -0.000000
x = -0.050000  f(x) = 0.951229 err = -0.000000
x = 0.050000  f(x) = 1.051271 err = 0.000000
x = 0.950000  f(x) = 2.585710 err = 0.000000
When n is equal to 20
x = -0.950000  f(x) = 0.386741 err = -0.000000
x = -0.050000  f(x) = 0.951229 err = 0.000000
x = 0.050000  f(x) = 1.051271 err = -0.000000
x = 0.950000  f(x) = 2.585710 err = 0.000000
 ------------------------
```

err 为插值结果和实际值的误差。
f(x) = 1/(1+x^2): 随着插值多项式次数增大，中间值的误差越来越小，而边界处的误差迅速增大，发生龙格现象。
f(x) = exp(x) 随着次数增大误差越来越小。
一般情况下，插值多项式的次数越大，预测也越准确。
而有的时候，会发生龙格现象：插值多项式序列在只在一定区间内收敛，而在区间外发散，导致随着次数增大，误差越来越大。

问题二：

```
PS D:\github\School_Pro\计算方法> .\Lagrange 2
 ------------------------
This is f(x) = 1/(x^2+1)
When n is equal to 5
x = -0.950000  f(x) = 0.517147 err = -0.008477
x = -0.050000  f(x) = 0.992791 err = -0.004716
x = 0.050000  f(x) = 0.992791 err = -0.004716
x = 0.950000  f(x) = 0.517147 err = -0.008477
When n is equal to 10
x = -0.950000  f(x) = 0.526408 err = 0.000784
x = -0.050000  f(x) = 0.997507 err = 0.000001
x = 0.050000  f(x) = 0.997507 err = 0.000001
x = 0.950000  f(x) = 0.526408 err = 0.000784
When n is equal to 20
x = -0.950000  f(x) = 0.525620 err = -0.000004
x = -0.050000  f(x) = 0.997506 err = 0.000000
x = 0.050000  f(x) = 0.997506 err = 0.000000
x = 0.950000  f(x) = 0.525620 err = -0.000004
 ------------------------
This is f(x) = e^x
When n is equal to 5
x = -4.750000  f(x) = 1.147035 err = 1.138383
x = -0.250000  f(x) = 1.302152 err = 0.523352
x = 0.250000  f(x) = 1.841210 err = 0.557185
x = 4.750000  f(x) = 119.621007 err = 4.036723
When n is equal to 10
x = -4.750000  f(x) = -0.001957 err = -0.010608
x = -0.250000  f(x) = 0.778686 err = -0.000114
x = 0.250000  f(x) = 1.284144 err = 0.000119
x = 4.750000  f(x) = 115.607360 err = 0.023076
When n is equal to 20
x = -4.750000  f(x) = 0.008652 err = -0.000000
x = -0.250000  f(x) = 0.778801 err = 0.000000
x = 0.250000  f(x) = 1.284025 err = -0.000000
x = 4.750000  f(x) = 115.584285 err = 0.000000
 ------------------------
```

由上面结果与问题一种的结果对比可以看出：
f(x) = 1/(1+x^2):随着插值区间的减小，误差也越来越小，同时龙格现象也消失了。
f(x) = exp(x):随着插值区间的减小，误差也越来越小。
通常情况下，减小插值区间会使误差减小，一是因为插值点和取值点更接近，使得误差天然减小，二是因为在更小的区间内，龙格现象会消失，即插值多项式会收敛，这也极大地减小了误差。

问题三

```
PS D:\github\School_Pro\计算方法> .\Lagrange 3
 ------------------------
This is f(x) = 1/(x^2+1)
When n is equal to 5
x = -0.950000  f(x) = 0.523881 err = -0.001743
x = -0.050000  f(x) = 0.987881 err = -0.009625
x = 0.050000  f(x) = 0.987881 err = -0.009625
x = 0.950000  f(x) = 0.523881 err = -0.001743
When n is equal to 10
x = -0.950000  f(x) = 0.525682 err = 0.000058
x = -0.050000  f(x) = 0.997509 err = 0.000003
x = 0.050000  f(x) = 0.997509 err = 0.000003
x = 0.950000  f(x) = 0.525682 err = 0.000058
When n is equal to 20
x = -0.950000  f(x) = 0.525624 err = 0.000000
x = -0.050000  f(x) = 0.997506 err = 0.000000
x = 0.050000  f(x) = 0.997506 err = 0.000000
x = 0.950000  f(x) = 0.525624 err = 0.000000
 ------------------------
This is f(x) = e^x
When n is equal to 5
x = -0.950000  f(x) = 0.386754 err = 0.000013
x = -0.050000  f(x) = 0.951272 err = 0.000042
x = 0.050000  f(x) = 1.051314 err = 0.000043
x = 0.950000  f(x) = 2.585727 err = 0.000017
When n is equal to 10
x = -0.950000  f(x) = 0.386741 err = -0.000000
x = -0.050000  f(x) = 0.951229 err = -0.000000
x = 0.050000  f(x) = 1.051271 err = 0.000000
x = 0.950000  f(x) = 2.585710 err = 0.000000
When n is equal to 20
x = -0.950000  f(x) = 0.386741 err = -0.000000
x = -0.050000  f(x) = 0.951229 err = 0.000000
x = 0.050000  f(x) = 1.051271 err = 0.000000
x = 0.950000  f(x) = 2.585710 err = 0.000000
 ------------------------
```

这是采用切比雪夫插值结点的结果，与之前的实验结果相对比可以发现，对于 Cauchy 函数：使用切比雪夫节点进行插值的效果优于等距结点。exp(x)函数中用切比雪夫节点，和等距结点插值的效果基本相同。

所以一般情况下（如柯西函数），使用切比雪夫节点插值，可得到更好的效果。少数情况下（如指数函数），利用切比雪夫节点进行插值与等距插值效果均很好，差距不大。

问题四

```
PS D:\github\School_Pro\计算方法> .\Lagrange 4
 ------------------------
When use x0=1, x1=4, x2=9
x = 5.000000  f(x) = 2.266667  err = 0.030599
x = 50.000000  f(x) = -20.233333  err = -27.304401
x = 115.000000  f(x) = -171.900000  err = -182.623805
x = 185.000000  f(x) = -492.733333  err = -506.334804
 ------------------------
When use x0=36, x1=49, x2=64
x = 5.000000  f(x) = 3.115751  err = 0.879683
x = 50.000000  f(x) = 7.071795  err = 0.000727
x = 115.000000  f(x) = 10.167033  err = -0.556772
x = 185.000000  f(x) = 10.038828  err = -3.562643
 ------------------------
When use x0=100, x1=121, x2=144
x = 5.000000  f(x) = 4.439112  err = 2.203044
x = 50.000000  f(x) = 7.284961  err = 0.213894
x = 115.000000  f(x) = 10.722756  err = -0.001050
x = 185.000000  f(x) = 13.535667  err = -0.065803
 ------------------------
When use x0=169, x1=196, x2=225
x = 5.000000  f(x) = 5.497172  err = 3.261104
x = 50.000000  f(x) = 7.800128  err = 0.729060
x = 115.000000  f(x) = 10.800493  err = 0.076687
x = 185.000000  f(x) = 13.600620  err = -0.000850
 ------------------------
```

我们可以发现，凡是在数据点中内插的数据的误差都不超过 0.1，而外推常常误差极大，有时误差也较小。绝大多数情况下，内插比外推更可靠，因为在已知区间内预测内部数据更加可靠，而且也可以避免龙格现象。

实验报告二

题目（摘要）

利用复化梯形求积公式、复化辛普生求积公式、复化柯特斯求积公式的误差估计式计算积分$\int_a^b f(x)dx$。记$h=\frac{b-a}{n}$，$x_k=a+k\cdot h$，$k=0,1,\cdots,n$，

前　言：（目的和意义）

利用龙贝格(Romberg)积分法计算积分$\int_a^b f(x)dx$

数学原理

利用复化梯形求积公式、复化辛普生求积公式、复化柯特斯求积公式的误差估计式计算积分$\int_a^b f(x)dx$。记$h=\frac{b-a}{n}$，$x_k=a+k\cdot h$，$k=0,1,\cdots,n$，其计算公式：

$$T_n=\frac{1}{2}h\sum_{k=1}^{n}[f(x_{k-1})+f(x_k)]$$

$$T_{2n}=\frac{1}{2}T_n+\frac{1}{2}h\sum_{k=1}^{n}f(x_k-\frac{1}{2}h)$$

$$S_n=\frac{1}{3}(4T_{2n}-T_n)$$

$$C_n=\frac{1}{15}(16S_{2n}-S_n)$$

$$R_n=\frac{1}{63}(64C_{2n}-C_n)$$

一般地，利用龙贝格算法计算积分，要输出所谓的$T-$数表

$$\begin{array}{ccccc} T_1 & & & & \\ T_2 & S_1 & & & \\ T_4 & S_2 & C_1 & & \\ T_8 & S_4 & C_2 & R_1 & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

程序设计流程

```
double Romberg(double a, double b, int N, double eps, double (*f)(double x)) {
    double h = (b-a)/2, T1, T2, R1, R2, S1, S2, C1, C2, ret[50][50];
    T1 = h*(f(a)+f(b));
    ret[0][0] = T1;
    for (int i=1,ii=1; i<=N; ++i,ii<<=1) {
        double now = 0;
        for (int j=1; j<=ii; ++j) now += f(a+(j*2-1)*h);
        T2 = T1/2 + h*now;
        ret[0][i] = T2;
        S2 = (4*T2-T1)/3;
        ret[1][i] = S2;
        if (i > 1) C2 = (16*S2-S1)/15, ret[2][i] = C2;
        if (i > 2) R2 = (64*C2-C1)/63, ret[3][i] = R2;
        if (i > 3) {
            if (fabs(R2 - R1) < eps) return R2;
        }
        R1 = R2, C1 = C2, S1 = S2, T1 = T2, h /= 2;
    }
    return R2;
}
```

实验结果、结论与讨论

问题 1

```
When N is 3
(1) the ans is 0.7182818501
(2) the ans is 10.9501810735
(3) the ans is 3.1415857838
(4) the ans is 0.6931474776
When N is 10
(1) the ans is 0.7182818285
(2) the ans is 10.9501703148
(3) the ans is 3.1415926536
(4) the ans is 0.6931471831
When N is 20
(1) the ans is 0.7182818285
(2) the ans is 10.9501703148
(3) the ans is 3.1415926536
(4) the ans is 0.6931471831
```

问题 2

```
(1) the ans is 0.946082
(2) the ans is 1.4996
(3) the ans is 1.809048
(4) the ans is 1.38246
```

（1）可以近似认为当 x 接近 0 时，f(x)=1。

（2）做完变换后，原积分式可转化为 $2\int_0^1 \cos(1-t^2)dt$，此时可直接对该式积分。

（3）利用等式 $\int_0^1 \frac{\cos x}{\sqrt{x}}dx = \int_0^1 \frac{1}{\sqrt{x}}dx + \int_0^1 \frac{\cos x - 1}{\sqrt{x}}dx$，第一个积分值等于 2，第二个积分，利用 $f(0) = \lim_{x\to 0} \frac{\cos x - 1}{\sqrt{x}} = 0$，可认为当 x 接近于 0 时，f(x)=0。

（4）利用 Gauss-Chebyshev 求积公式可把原积分式 $\int_{-1}^1 \frac{x\sin x}{\sqrt{1-x^2}}dx$ 转化为求 $\frac{\pi}{n+1}\sum_{k=0}^{n} f(x_k)$ 的值（其中 $f(x) = x\sin x$），又 $x_k = \cos\frac{(2k+1)\pi}{2n+2}$，可取 $n=100$，算得原积分值为 1.38246。

问题 3

```
(1) the ans is 1.77245
(2) the ans is 1.5708
(3) the ans is 1.08195
(4) the ans is 0.388195
```

（1）可利用 $\int_{-10}^{+10} e^{-x^2}dx$ 作近似求解。

（2）利用变换 $t = \sqrt{x^{-1}}$，原积分式可转化为 $2\int_0^1 \frac{1}{1+t^2}dt$

（3）根据 Gauss-Hermite 求积公式，$\int_{-\infty}^{+\infty} e^{-x^2}\cos^3 x dx \approx \sum_{j=0}^{n} H_j f(x_j)$，

$H_j = \dfrac{2^{n+2}(n+1)!\sqrt{n}}{H_{n+2}(x_j)H'_{n+1}(x_j)}$，通过查阅 Gauss-Hermite 求积节点和求积系数表，取 $n+1=8$，算得 $\sum_{j=0}^{n} H_j f(x_j) = 1.08195$。

（4）根据 Gauss-Lagurre 求积公式，$\int_0^{+\infty} \mathrm{e}^{-x}\sin^2 x dx \approx \sum_{j=0}^{n} H_j f(x_j)$，

$H_j = \dfrac{[(n+1)!]^2}{L_{n+2}(x_j)L'_{n+1}(x_j)}$，通过查阅 Gauss-Lagurre 求积节点和求积系数表，取 $n+1=5$，算得 $\sum_{j=0}^{n} H_j f(x_j) = 0.388195$。

思考题：

1. N 的值越大，对有限区间的划分就会越细，分段就会越小，使用复化梯形公式的误差就越小，计算精度更高。

2. 实验 1 中二分次数越多，计算精度越高。

3. 实验 2 中给出的提示具有一定程度上的普遍性，当被积函数在某个区间上无界时，可以对积分变量做一些变换，或者套用 Gauss 积分公式进行处理。对于某个函数的积分来说，处理的方法往往不止一种。例如问题 2 中的第 4 题，除了使用 Gauss-Chebyshev 求积公式外，还可以做 $x=\sin t$ 的变换，变换之后原积分式变为 $2\int_0^{\frac{\pi}{2}} \sin t \cdot \sin(\sin t)dt$，从而可以使用常规方法处理。

4. 实验 3 中给出的提示也具有一定程度上的普遍性，当积分区间无界时，一方面可以对积分变量做一些变换，让积分区间变成有界的；另一方面，可以套用一些 Gauss 积分公式直接进行计算。对于某个函数的积分来说，处理的方法往往不止一种。例如问题 3 中的第 1 题，处除了使用直接近似处理的办法，还可以使用 Gauss-Hermite 求积公式进行处理。

实验报告三

题目（摘要）

利用四阶龙格—库塔(Runge—Kutta)方法求解微分方程初值问题

前　言：（目的和意义）

目的：利用四阶龙格—库塔(Runge—Kutta)方法求解微分方程初值问题

输　　入：a,b,α,N

输　　出：初值问题的数值解x_n, y_n, $n=0,1,2,\cdots,N$

数学原理

给定常微分方程初值问题

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

记$x_n=a+n\cdot h$， $n=0,1,\cdots,N$，利用四阶龙格—库塔方法

$$\begin{aligned}K_1&=f(x_n,y_n)\\ K_2&=f(x_n+\frac{h}{2},y_n+\frac{hK_1}{2})\\ K_3&=f(x_n+\frac{h}{2},y_n+\frac{hK_2}{2})\\ K_4&=f(x_n+h,y_n+hK_3)\\ y_{n+1}&=y_n+\frac{h}{6}(K_1+2K_2+2K_3+K_4)\end{aligned}$$

可逐次求出微分方程初值问题的数值解y_n， $n=1,2,\cdots,N$。

程序设计流程

```
void RungeKutta(db a, db b, db alpha, int N, db(*f)(db x, db y), vector <pair<db, db> >& ans) {
	db x0 = a, y0 = alpha;
	db h = (b-a)/N;
	for (int i=1; i<=N; ++i) {
		db k1 = f(x0, y0);
		db k2 = f(x0+h/2, y0+h*k1/2);
		db k3 = f(x0+h/2, y0+h*k2/2);
		db k4 = f(x0+h, y0+h*k3);
		db x1 = x0+h, y1 = y0+h*(k1+2*k2+2*k3+k4)/6;
		ans.push_back(make_pair(x1, y1));
		x0 = x1, y0 = y1;
	}
	return;
}
```

实验结果、结论与讨论

问题一：

```
PS D:\github\School_Pro\计算方法> .\RungeKutta
f(x,y)=x+y
X = 0.050000 Y = -1.0500000000 g(x) = -1.0500000000
X = 0.100000 Y = -1.1000000000 g(x) = -1.1000000000
X = 0.150000 Y = -1.1500000000 g(x) = -1.1500000000
X = 0.200000 Y = -1.2000000000 g(x) = -1.2000000000
X = 0.250000 Y = -1.2500000000 g(x) = -1.2500000000
X = 0.300000 Y = -1.3000000000 g(x) = -1.3000000000
X = 0.350000 Y = -1.3500000000 g(x) = -1.3500000000
X = 0.400000 Y = -1.4000000000 g(x) = -1.4000000000
X = 0.450000 Y = -1.4500000000 g(x) = -1.4500000000
X = 0.500000 Y = -1.5000000000 g(x) = -1.5000000000
X = 0.550000 Y = -1.5500000000 g(x) = -1.5500000000
X = 0.600000 Y = -1.6000000000 g(x) = -1.6000000000
X = 0.650000 Y = -1.6500000000 g(x) = -1.6500000000
X = 0.700000 Y = -1.7000000000 g(x) = -1.7000000000
X = 0.750000 Y = -1.7500000000 g(x) = -1.7500000000
X = 0.800000 Y = -1.8000000000 g(x) = -1.8000000000
X = 0.850000 Y = -1.8500000000 g(x) = -1.8500000000
X = 0.900000 Y = -1.9000000000 g(x) = -1.9000000000
X = 0.950000 Y = -1.9500000000 g(x) = -1.9500000000
X = 1.000000 Y = -2.0000000000 g(x) = -2.0000000000
f(x,y)=-y*y
X = 0.050000 Y = 0.9523809630 g(x) = 0.9523809524
X = 0.100000 Y = 0.9090909268 g(x) = 0.9090909091
X = 0.150000 Y = 0.8695652397 g(x) = 0.8695652174
X = 0.200000 Y = 0.8333333586 g(x) = 0.8333333333
X = 0.250000 Y = 0.8000000269 g(x) = 0.8000000000
X = 0.300000 Y = 0.7692307971 g(x) = 0.7692307692
X = 0.350000 Y = 0.7407407689 g(x) = 0.7407407407
X = 0.400000 Y = 0.7142857423 g(x) = 0.7142857143
X = 0.450000 Y = 0.6896552000 g(x) = 0.6896551724
X = 0.500000 Y = 0.6666666937 g(x) = 0.6666666667
X = 0.550000 Y = 0.6451613166 g(x) = 0.6451612903
X = 0.600000 Y = 0.6250000255 g(x) = 0.6250000000
X = 0.650000 Y = 0.6060606307 g(x) = 0.6060606061
X = 0.700000 Y = 0.5882353179 g(x) = 0.5882352941
X = 0.750000 Y = 0.5714285944 g(x) = 0.5714285714
X = 0.800000 Y = 0.5555555776 g(x) = 0.5555555556
X = 0.850000 Y = 0.5405405618 g(x) = 0.5405405405
X = 0.900000 Y = 0.5263158099 g(x) = 0.5263157895
X = 0.950000 Y = 0.5128205325 g(x) = 0.5128205128
X = 1.000000 Y = 0.5000000189 g(x)_= 0.5000000000
```

由结果可知第一个问题的准确解为$y=-x-1$，第二个问题的准确解为$y=\frac{1}{x+1}$

问题二

```
f(x,y)=2*y/x+x*x*exp(x)
N is 5
X = 1.400000 Y = 2.6139427925 g(x) = 2.6203595512 err = -0.006417
X = 1.800000 Y = 10.7763131664 g(x) = 10.7936246605 err = -0.017311
X = 2.200000 Y = 30.4916542038 g(x) = 30.5245812875 err = -0.032927
X = 2.600000 Y = 72.5855986060 g(x) = 72.6392839562 err = -0.053685
X = 3.000000 Y = 156.2251982758 g(x) = 156.3052958526 err = -0.080098
N is 10
X = 1.200000 Y = 0.8663791120 g(x) = 0.8666425358 err = -0.000263
X = 1.400000 Y = 2.6197405205 g(x) = 2.6203595512 err = -0.000619
X = 1.600000 Y = 5.7198952795 g(x) = 5.7209615256 err = -0.001066
X = 1.800000 Y = 10.7920175975 g(x) = 10.7936246605 err = -0.001607
X = 2.000000 Y = 18.6808523645 g(x) = 18.6830970819 err = -0.002245
X = 2.200000 Y = 30.5215981354 g(x) = 30.5245812875 err = -0.002983
X = 2.400000 Y = 47.8323658327 g(x) = 47.8361926206 err = -0.003827
X = 2.600000 Y = 72.6345035377 g(x) = 72.6392839562 err = -0.004780
X = 2.800000 Y = 107.6088519912 g(x) = 107.6147011503 err = -0.005849
X = 3.000000 Y = 156.2982574429 g(x) = 156.3052958526 err = -0.007038
N is 20
X = 1.100000 Y = 0.3459102873 g(x) = 0.3459198765 err = -0.000010
X = 1.200000 Y = 0.8666216927 g(x) = 0.8666425358 err = -0.000021
X = 1.300000 Y = 1.6071813477 g(x) = 1.6072150782 err = -0.000034
X = 1.400000 Y = 2.6203113059 g(x) = 2.6203595512 err = -0.000048
X = 1.500000 Y = 3.9676018980 g(x) = 3.9676662942 err = -0.000064
X = 1.600000 Y = 5.7208793242 g(x) = 5.7209615256 err = -0.000082
X = 1.700000 Y = 7.9637717926 g(x) = 7.9638734778 err = -0.000102
X = 1.800000 Y = 10.7935017836 g(x) = 10.7936246605 err = -0.000123
X = 1.900000 Y = 14.3229357276 g(x) = 14.3230815359 err = -0.000146
X = 2.000000 Y = 18.6829265677 g(x) = 18.6830970819 err = -0.000171
X = 2.100000 Y = 24.0249894197 g(x) = 24.0251864509 err = -0.000197
X = 2.200000 Y = 30.5243558898 g(x) = 30.5245812875 err = -0.000225
X = 2.300000 Y = 38.3834586600 g(x) = 38.3837143134 err = -0.000256
X = 2.400000 Y = 47.8359047809 g(x) = 47.8361926206 err = -0.000288
X = 2.500000 Y = 59.1510038275 g(x) = 59.1513258265 err = -0.000322
X = 2.600000 Y = 72.6389257808 g(x) = 72.6392839562 err = -0.000358
X = 2.700000 Y = 88.6565733309 g(x) = 88.6569697449 err = -0.000396
X = 2.800000 Y = 107.6142643893 g(x) = 107.6147011503 err = -0.000437
X = 2.900000 Y = 129.9833331157 g(x) = 129.9838123797 err = -0.000479
X = 3.000000 Y = 156.3047718808 g(x) = 156.3052958526 err = -0.000524
f(x,y)=(y*y+y)/x
N is 5
X = 1.400000 Y = -1.5539889981 g(x) = -1.5555555556 err = 0.001567
X = 1.800000 Y = -1.3836172899 g(x) = -1.3846153846 err = 0.000998
X = 2.200000 Y = -1.2934015269 g(x) = -1.2941176471 err = 0.000716
X = 2.600000 Y = -1.2375401579 g(x) = -1.2380952381 err = 0.000555
X = 3.000000 Y = -1.1995479585 g(x) = -1.2000000000 err = 0.000452
N is 10
X = 1.200000 Y = -1.7142451805 g(x) = -1.7142857143 err = 0.000041
X = 1.400000 Y = -1.5555228848 g(x) = -1.5555555556 err = 0.000033
X = 1.600000 Y = -1.4545197492 g(x) = -1.4545454545 err = 0.000026
X = 1.800000 Y = -1.3845945063 g(x) = -1.3846153846 err = 0.000021
X = 2.000000 Y = -1.3333158561 g(x) = -1.3333333333 err = 0.000017
X = 2.200000 Y = -1.2941026606 g(x) = -1.2941176471 err = 0.000015
X = 2.400000 Y = -1.2631447989 g(x) = -1.2631578947 err = 0.000013
X = 2.600000 Y = -1.2380836212 g(x) = -1.2380952381 err = 0.000012
X = 2.800000 Y = -1.2173808733 g(x) = -1.2173913043 err = 0.000010
X = 3.000000 Y = -1.1999905397 g(x) = -1.2000000000 err = 0.000009
N is 20
X = 1.100000 Y = -1.8333328294 g(x) = -1.8333333333 err = 0.000001
X = 1.200000 Y = -1.7142851698 g(x) = -1.7142857143 err = 0.000001
X = 1.300000 Y = -1.6249995002 g(x) = -1.6250000000 err = 0.000000
X = 1.400000 Y = -1.5555551111 g(x) = -1.5555555556 err = 0.000000
X = 1.500000 Y = -1.4999996057 g(x) = -1.5000000000 err = 0.000000
X = 1.600000 Y = -1.4545451028 g(x) = -1.4545454545 err = 0.000000
X = 1.700000 Y = -1.4166663505 g(x) = -1.4166666667 err = 0.000000
X = 1.800000 Y = -1.3846150982 g(x) = -1.3846153846 err = 0.000000
X = 1.900000 Y = -1.3571425958 g(x) = -1.3571428571 err = 0.000000
X = 2.000000 Y = -1.3333330933 g(x) = -1.3333333333 err = 0.000000
X = 2.100000 Y = -1.3124997783 g(x) = -1.3125000000 err = 0.000000
X = 2.200000 Y = -1.2941174411 g(x) = -1.2941176471 err = 0.000000
X = 2.300000 Y = -1.2777775857 g(x) = -1.2777777778 err = 0.000000
X = 2.400000 Y = -1.2631577147 g(x) = -1.2631578947 err = 0.000000
X = 2.500000 Y = -1.2499998307 g(x) = -1.2500000000 err = 0.000000
X = 2.600000 Y = -1.2380950784 g(x) = -1.2380952381 err = 0.000000
X = 2.700000 Y = -1.2272725761 g(x) = -1.2272727273 err = 0.000000
X = 2.800000 Y = -1.2173911609 g(x) = -1.2173913043 err = 0.000000
X = 2.900000 Y = -1.2083331969 g(x) = -1.2083333333 err = 0.000000
X = 3.000000 Y = -1.1999998699 g(x) = -1.2000000000 err = 0.000000
```

第一个准确解为$y = x^2(e^x - e)$第二个准确解为$y = \frac{2x}{1-2x}$

问题三

```
f(x,y)=-20(y-x^2)+2x
N is 5
X = 0.200000 Y = 1.7600000000 g(x) = 0.0461052130 err = 1.713895
X = 0.400000 Y = 8.8133333333 g(x) = 0.1601118209 err = 8.653222
X = 0.600000 Y = 43.6800000000 g(x) = 0.3600020481 err = 43.319998
X = 0.800000 Y = 217.2933333333 g(x) = 0.6400000375 err = 216.653333
X = 1.000000 Y = 1084.3200000000 g(x) = 1.0000000007 err = 1083.320000
N is 10
X = 0.100000 Y = 0.1227777778 g(x) = 0.0551117611
X = 0.200000 Y = 0.0792592593 g(x) = 0.0461052130
X = 0.300000 Y = 0.1047530864 g(x) = 0.0908262507
X = 0.400000 Y = 0.1665843621 g(x) = 0.1601118209
X = 0.500000 Y = 0.2538614540 g(x) = 0.2500151333
X = 0.600000 Y = 0.3629538180 g(x) = 0.3600020481
X = 0.700000 Y = 0.4926512727 g(x) = 0.4900002772
X = 0.800000 Y = 0.6425504242 g(x) = 0.6400000375
X = 0.900000 Y = 0.8125168081 g(x) = 0.8100000051
X = 1.000000 Y = 1.0025056027 g(x) = 1.0000000007
N is 20
X = 0.050000 Y = 0.1275520833 g(x) = 0.1251264804
X = 0.100000 Y = 0.0569466146 g(x) = 0.0551117611
X = 0.150000 Y = 0.0401570638 g(x) = 0.0390956895
X = 0.200000 Y = 0.0466734823 g(x) = 0.0461052130
X = 0.250000 Y = 0.0650546392 g(x) = 0.0647459823
X = 0.300000 Y = 0.0910100730 g(x) = 0.0908262507
X = 0.350000 Y = 0.1229308607 g(x) = 0.1228039607
X = 0.400000 Y = 0.1602136561 g(x) = 0.1601118209
X = 0.450000 Y = 0.2026322044 g(x) = 0.2025411366
X = 0.500000 Y = 0.2501016600 g(x) = 0.2500151333
X = 0.550000 Y = 0.3025902058 g(x) = 0.3025055672
X = 0.600000 Y = 0.3600859105 g(x) = 0.3600020481
X = 0.650000 Y = 0.4225842998 g(x) = 0.4225007534
X = 0.700000 Y = 0.4900836957 g(x) = 0.4900002772
X = 0.750000 Y = 0.5625834692 g(x) = 0.5625001020
X = 0.800000 Y = 0.6400833843 g(x) = 0.6400000375
X = 0.850000 Y = 0.7225833524 g(x) = 0.7225000138
X = 0.900000 Y = 0.8100833405 g(x) = 0.8100000051
X = 0.950000 Y = 0.9025833360 g(x) = 0.9025000019
X = 1.000000 Y = 1.0000833343 g(x) = 1.0000000007
f(x,y)=-20y+20sin(x)+cos(x)
N is 5
X = 0.200000 Y = 5.1973381062 g(x) = 0.2169849697
X = 0.400000 Y = 25.3761707044 g(x) = 0.3897538049
X = 0.600000 Y = 125.4868152611 g(x) = 0.5646486176
X = 0.800000 Y = 625.3120955171 g(x) = 0.7173562034
X = 1.000000 Y = 3123.7951509472 g(x) = 0.8414709869
N is 10
X = 0.100000 Y = 0.4331389965 g(x) = 0.2351686999
X = 0.200000 Y = 0.3096604680 g(x) = 0.2169849697
X = 0.300000 Y = 0.3323246667 g(x) = 0.2979989588
X = 0.400000 Y = 0.4014139713 g(x) = 0.3897538049
X = 0.500000 Y = 0.4830743415 g(x) = 0.4794709385
X = 0.600000 Y = 0.5654352797 g(x) = 0.5646486176
X = 0.700000 Y = 0.6439890045 g(x) = 0.6442185188
X = 0.800000 Y = 0.7167223471 g(x) = 0.7173562034
X = 0.900000 Y = 0.7824991512 g(x) = 0.7833269249
X = 1.000000 Y = 0.8405257206 g(x) = 0.8414709869
N is 20
X = 0.050000 Y = 0.4249785186 g(x) = 0.4178586104
X = 0.100000 Y = 0.2404562221 g(x) = 0.2351686999
X = 0.150000 Y = 0.2021684390 g(x) = 0.1992252008
X = 0.200000 Y = 0.2184386634 g(x) = 0.2169849697
X = 0.250000 Y = 0.2548116511 g(x) = 0.2541419063
X = 0.300000 Y = 0.2982910222 g(x) = 0.2979989588
X = 0.350000 Y = 0.3439285511 g(x) = 0.3438096894
X = 0.400000 Y = 0.3897953364 g(x) = 0.3897538049
X = 0.450000 Y = 0.4350961733 g(x) = 0.4350889439
X = 0.500000 Y = 0.4794626229 g(x) = 0.4794709385
X = 0.550000 Y = 0.5226880879 g(x) = 0.5227039306
X = 0.600000 Y = 0.5646286384 g(x) = 0.5646486176
X = 0.650000 Y = 0.6051659863 g(x) = 0.6051886661
X = 0.700000 Y = 0.6441937626 g(x) = 0.6442185188
X = 0.750000 Y = 0.6816125255 g(x) = 0.6816390659
X = 0.800000 Y = 0.7173280379 g(x) = 0.7173562034
X = 0.850000 Y = 0.7512507634 g(x) = 0.7512804465
X = 0.900000 Y = 0.7832958132 g(x) = 0.7833269249
X = 0.950000 Y = 0.8133830538 g(x) = 0.8134155104
X = 1.000000 Y = 0.8414372689 g(x) = 0.8414709869
```

```
f(x,y)=-20(y-e^xsin(x))+e^x(sin(x)+cos(x))
N is 5
X = 0.200000 Y = 0.2986462128 g(x) = 0.2426552686
X = 0.400000 Y = 0.9272198700 g(x) = 0.5809439008
X = 0.600000 Y = 2.8354773389 g(x) = 1.0288456663
X = 0.800000 Y = 10.7108853309 g(x) = 1.5965053406
X = 1.000000 Y = 47.9414463816 g(x) = 2.2873552872
N is 10
X = 0.100000 Y = 0.1120551091 g(x) = 0.1103329887
X = 0.200000 Y = 0.2451165144 g(x) = 0.2426552686
X = 0.300000 Y = 0.4017780967 g(x) = 0.3989105538
X = 0.400000 Y = 0.5840969566 g(x) = 0.5809439008
X = 0.500000 Y = 0.7938220530 g(x) = 0.7904390832
X = 0.600000 Y = 1.0324183053 g(x) = 1.0288456663
X = 0.700000 Y = 1.3010149884 g(x) = 1.2972951119
X = 0.800000 Y = 1.6003210120 g(x) = 1.5965053406
X = 0.900000 Y = 1.9305210338 g(x) = 1.9266733040
X = 1.000000 Y = 2.2911569231 g(x) = 2.2873552872
N is 20
X = 0.050000 Y = 0.0525950400 g(x) = 0.0525416561
X = 0.100000 Y = 0.1104089863 g(x) = 0.1103329887
X = 0.150000 Y = 0.1737093905 g(x) = 0.1736223395
X = 0.200000 Y = 0.2427490009 g(x) = 0.2426552686
X = 0.250000 Y = 0.3177716916 g(x) = 0.3176729719
X = 0.300000 Y = 0.3990135525 g(x) = 0.3989105538
X = 0.350000 Y = 0.4867020696 g(x) = 0.4865951510
X = 0.400000 Y = 0.5810544891 g(x) = 0.5809439008
X = 0.450000 Y = 0.6822757725 g(x) = 0.6821617474
X = 0.500000 Y = 0.7905562930 g(x) = 0.7904390832
X = 0.550000 Y = 0.9060693250 g(x) = 0.9059492169
X = 0.600000 Y = 1.0289683441 g(x) = 1.0288456663
X = 0.650000 Y = 1.1593841417 g(x) = 1.1592592694
X = 0.700000 Y = 1.2974217528 g(x) = 1.2972951119
X = 0.750000 Y = 1.4431571960 g(x) = 1.4430292663
X = 0.800000 Y = 1.5966340222 g(x) = 1.5965053406
X = 0.850000 Y = 1.7578596710 g(x) = 1.7577308348
X = 0.900000 Y = 1.9268016338 g(x) = 1.9266733040
X = 0.950000 Y = 2.1033834233 g(x) = 2.1032563278
X = 1.000000 Y = 2.2874803507 g(x) = 2.2873552872
```

第一个准确解为$y = x^2 + \frac{1}{3}e^{-20x}$，第二个准确解为$y = e^{-20x} + sin\,x$，第三个准确解为$y = e^x\,sin\,x$

1. 对实验 1，数值解和解析解相同吗？为什么？试加以说明。
   实验一中(1)数值解和解析解相同，(2)数值解和解析解稍有不同，因为四阶 Runge-Kutta 方法是以小段的线性算法来近似获得微分方程的数值解，(1)的准确解是 1 阶的，(2)的准确解是无限阶的，因此对于(1)数值解和解析解相同。
2. 对实验 2，N 越大越精确吗？试加以说明。
   N 越大越精确。
3. 对实验 3，N 较小会出现什么现象？试加以说明
   N 太小时会导致误差巨大

实验报告四

题目（摘要）

求非线性方程 $f(x)=0$ 的根 x^*，牛顿迭代法计算公式

$$x_0=\alpha$$

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

$$n=0,1,\cdots$$

前　言：（目的和意义）

实验目的：利用牛顿迭代法求 $f(x)=0$ 的根

输　　入：初值 α，精度 $\varepsilon_1,\varepsilon_2$，最大迭代次数 N

输　　出：方程 $f(x)=0$ 根 x^* 的近似值或计算失败标志

数学原理

求非线性方程 $f(x)=0$ 的根 x^*，牛顿迭代法计算公式

$$x_0=\alpha$$

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

$$n=0,1,\cdots$$

一般地，牛顿迭代法具有局部收敛性，为保证迭代收敛，要求，对充分小的 $\delta>0$，$\alpha\in O(x^*,\delta)$。如果 $f(x)\in C^2[a,b]$，$f(x^*)=0$，$f'(x^*)\neq 0$，那么，对充分小的$\delta>0$，当$\alpha\in O(x^*,\delta)$时，由牛顿迭代法计算出的$\{x_n\}$收敛于x^*，且收敛速度是 2 阶的；如果 $f(x)\in C^m[a,b]$，$f(x^*)=f'(x^*)=\cdots=f^{(m-1)}(x^*)=0$，$f^{(m)}(x^*)\neq 0(m>1)$，那么，对充分小的$\delta>0$，当$\alpha\in O(x^*,\delta)$时，由牛顿迭代法计算出的$\{x_n\}$收敛于$x^*$，且收敛速度是 1 阶的；

程序设计流程

```
# define pr pair<bool,double>
# define mp make_pair
//适用于问题一二
pr Newton(double x_0, double eps1, double eps2, int N, double (*f)(double x), double (*df)(double x)) {
        for (int n=1; n<=N; ++n) {
            double F = f(x_0), DF = df(x_0);
            if (fabs(F) < eps1) return mp(true, x_0);
            if (fabs(DF) < eps2) return mp(false, 0);
            double x_1 = x_0 - F/DF;
            double TOL = fabs(x_0 - x_1);
            if (TOL < eps1) return mp(true, x_1);
            x_0 = x_1;
        }
        return mp(false, 0);
    }

//适用于问题一三
pr Newton(double x_0, double eps1, double eps2, int N, int n, double (*f)(double x, int n), double (*df)(double x, int n)) {
        for (int t=1; t<=N; ++t) {
            double F = f(x_0, n), DF = df(x_0, n);
            if (fabs(F) < eps1) return mp(true, x_0);
            if (fabs(DF) < eps2) return mp(false, 0);
            double x_1 = x_0 - F/DF;
            double TOL = fabs(x_0 - x_1);
            if (TOL < eps1) return mp(true, x_1);
            x_0 = x_1;
        }
        return mp(false, 0);
    }
```

实验结果、结论与讨论

问题一

```
------------------------
f(x) = cosx-x
0.739085
------------------------
f(x) = e^(-x)-sinx
0.588533
------------------------
```

问题二

```
------------------------
f(x) = x-e^(-x)
0.567143
------------------------
f(x) = x^2-2xe^(-x)+e^(-2x)
0.566606
------------------------
```

问题三

```
The zero point of the function Legendre is:
-0.932470 -0.661209 -0.238619 0.238619 0.661209 0.932470
The zero point of the function Laguerre is:
0.263560 1.413403 3.596426 7.085810 12.640801
The zero point of the function Hermite is:
-2.350605 -1.335849 -0.436077 0.436077 1.335849 2.350605
```

思考题 1：实验 1 中，牛顿迭代法选取初值的原则？

牛顿迭代法是局部收敛的方法，它是否收敛与初值的选取有关。

选取初值的基本原则是要充分接近根，在实际中我们通常不能预先知道根，所以可以先通过二分法预先确定根的大致范围。

选取初值还要保证序列收敛，根据牛顿迭代的收敛定理，我们选取初值 x0 应保证 f 在 x0 处的函数值与二阶导数值的乘积为正。

思考题 2：实验 2 中的现象及分析

不难发现，问题 2 中函数（2）正好是函数（1）的平方，两者应该具有相同的值，但是使用牛顿迭代法计算得到根却有较大偏差。将根的数值解分别回代到两个函数中，发现对第一个函数的牛顿迭代求解更为精确。本实验中使用了函数值的绝对值作为牛顿迭代的终止条件，这两个函数虽然理论上具有相同的零点，但后者为前者的平方，在取值点接近根时的函数值会更快收敛于 0，于是导致牛顿迭代程序提前终止，降低了求解精度。

思考题 3：

实验三中发现的问题：在实验三中，发现对于系数较大的多项式的牛顿迭代求根较精确，能够到达小数点后十位的精度，而对于系数较小的多项式则只有小数点后六七位的精度，原因和实验二中的现象是一样的：系数较小的矩阵在取值点接近根时会更快收敛，导致牛顿迭代提前结束，降低精度。

实验报告五

题目（摘要）

高斯(Gauss)列主元消去法
方法概要：对给定的 n 阶线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，首先进行列主元消元过程，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。

前　言：（目的和意义）

实验目的：利用 Gauss 列主元消去法、显式相对 Gauss 列主元消去法、隐式相对 Gauss 列主元消去法求解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$。

输　　入：n；a_{ij}，　b_i，$i,j=1,2,\cdots,n$

输　　出：线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的近似解 x_i，$i=1,2,\cdots,n$

数学原理

高斯（Gauss）列主元消去法：对给定的 n 阶线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，首先进行列主元消元过程，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。

如果系数矩阵的元素按绝对值在数量级方面相差很大，那么，在进行列主元消元过程前，先把系数矩阵的元素进行行平衡：系数矩阵的每行元素和相应的右端向量元素同除以该行元素绝对值最大的元素。这就是所谓的平衡技术。然后再进行列主元消元过程。

如果真正进行运算去确定相对主元，则称为显式相对 Gauss 列主元消去法；如果不进行运算，也能确定相对主元，则称为隐式相对 Gauss 列主元消去法。

显式相对 Gauss 列主元消去法：对给定的 n 阶线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，首先进行列主元消元过程，在消元过程中利用显式平衡技术，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。

隐式相对 Gauss 列主元消去法：对给定的 n 阶线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$，首先进行列主元消元过程，在消元过程中利用隐式平衡技术，然后进行回代过程，最后得到解或确定该线性方程组是奇异的。

程序设计流程

Gauss 列主元消去法

```
double* Gauss(int n, double ** a, double * b, bool& flag) {
	double *x = (double *) malloc((n + 1) * sizeof(double));
	for (int k=1; k<n; ++k) {
		int p = 0;
		for (int j=k; j<=n; ++j)
			if (!p || sgn(fabs(a[j][k]) - fabs(a[p][k])) > 0) p = j;
		if (sgn(a[p][k]) == 0) {
			flag = false;
			return NULL;
		}
		if (p != k) {
			for (int j=1; j<=n; ++j) swap(a[p][j], a[k][j]);
			swap(b[p], b[k]);
		}
		for (int i=k+1; i<=n; ++i) {
			double m = a[i][k]/a[k][k];
			a[i][k] = 0;
			for (int j=k+1; j<=n; ++j) a[i][j] -= a[k][j]*m;
			b[i] = b[i]-b[k]*m;
		}
	}
	if (sgn(a[n][n]) == 0) {
		flag = false;
		return NULL;
	}
	x[n] = b[n] / a[n][n];
	for (int k=n-1; k; --k) {
		double now = 0;
		for (int j=k+1; j<=n; ++j) {
			now += a[k][j]*x[j];
		}
		x[k] = (b[k] - now) / a[k][k];
	}
	flag = true;
	return x;
}
```

显式相对 Gauss 列主元消去法

```
double* Gauss(int n, double ** a, double * b, bool& flag) {
	double *x = (double *) malloc((n + 1) * sizeof(double));
	// print(a ,b ,4);
	for (int k=1; k<n; ++k) {
		for (int i=k; i<=n; ++i) {
			double mx = 0;
```

```
                for (int j=k; j<=n; ++j)
                        mx = max(mx, fabs(a[i][j]));
                if (sgn(mx) == 0) {
                        flag = false;
                        return NULL;
                }
                for (int j=k; j<=n; ++j)
                        a[i][j] = a[i][j]/mx;
                b[i] /= mx;
        }
        // print(a, b, 4);
        int p = 0;
        for (int j=k; j<=n; ++j)
                if (p == 0 || sgn(fabs(a[j][k]) - fabs(a[p][k])) > 0) p = j;
        if (sgn(a[p][k]) == 0) {
                flag = 0;
                return NULL;
        }
        if (p != k) {
                for (int j=1; j<=n; ++j) swap(a[p][j], a[k][j]);
                swap(b[p], b[k]);
        }
        for (int i=k+1; i<=n; ++i) {
                double m = a[i][k]/a[k][k];
                a[i][k] = 0;
                for (int j=k+1; j<=n; ++j) a[i][j] -= a[k][j]*m;
                b[i] = b[i]-b[k]*m;
        }
        // print(a, b, n);
    }
    if (sgn(a[n][n]) == 0) {
        flag = false;
        return NULL;
    }
    x[n] = b[n] / a[n][n];
    for (int k=n-1; k; --k) {
        double now = 0;
        for (int j=k+1; j<=n; ++j) {
            now += a[k][j]*x[j];
        }
        x[k] = (b[k] - now) / a[k][k];
    }
    flag = true;
    return x;
}
```

隐式相对 Gauss 列主元消去法

```
double* Gauss(int n, double ** a, double * b, bool& flag) {
    double *x = (double *) malloc((n + 1) * sizeof(double));
    // print(a ,b ,4);
    double s[n+1] = {0};
    for (int k=1; k<n; ++k) {
        for (int i=k; i<=n; ++i) {
            for (int j=k; j<=n; ++j)
                s[i] = max(s[i], fabs(a[i][j]));
            if (sgn(s[i]) == 0) {
                flag = false;
                return NULL;
            }
        }
        // print(a, b, 4);
        int p = 0;
        for (int j=k; j<=n; ++j)
            if (p == 0 || sgn(fabs(a[j][k]/s[j]) - fabs(a[p][k]/s[p])) > 0) p = j;
        if (sgn(a[p][k]) == 0) {
            flag = 0;
            return NULL;
        }
        if (p != k) {
            for (int j=1; j<=n; ++j) swap(a[p][j], a[k][j]);
            swap(b[p], b[k]);
        }
        for (int i=k+1; i<=n; ++i) {
            double m = a[i][k]/a[k][k];
            a[i][k] = 0;
            for (int j=k+1; j<=n; ++j) a[i][j] -= a[k][j]*m;
            b[i] = b[i]-b[k]*m;
        }
        // print(a, b, n);
    }
    if (sgn(a[n][n]) == 0) {
        flag = false;
        return NULL;
    }
    x[n] = b[n] / a[n][n];
    for (int k=n-1; k; --k) {
        double now = 0;
        for (int j=k+1; j<=n; ++j) {
            now += a[k][j]*x[j];
        }
        x[k] = (b[k] - now) / a[k][k];
    }
    flag = true;
    return x;
}
```

实验结果、结论与讨论

问题一

```
Gauss列主元消去法
1.0000000000000000266454 1.0000000000000000199840 0.9999999999999999711342 0.9999999999999999922284
显式相对Gauss列主元消去法
1.0000000000000000177636 1.0000000000000000133227 0.9999999999999999789058 0.9999999999999999977796
隐式相对Gauss列主元消去法
1.0000000000000000266454 1.0000000000000000199840 0.9999999999999999711342 0.9999999999999999922284

Gauss列主元消去法
1.0000000000000011790569 0.9999999999982380761 0.9999999999999990096811 1.0000000000000002597922
显式相对Gauss列主元消去法
0.9999999999993219868O 1.0000000000000101496589 1.0000000000000057220895 0.9999999999985067500
隐式相对Gauss列主元消去法
1.0000000000000011790569 0.9999999999982380761 0.9999999999999990096811 1.0000000000000002597922

Gauss列主元消去法
0.9999999999999940048O 1.0000000000000005861978 0.9999999999987587707 1.0000000000000007283063
显式相对Gauss列主元消去法
0.9999999999999913402O 1.0000000000000008837375 0.9999999999998040456O 1.0000000000000011990409
隐式相对Gauss列主元消去法
1.0000000000000000399680 0.9999999999994815258 1.0000000000000013722357 0.9999999999999529798

Gauss列主元消去法
1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000
显式相对Gauss列主元消去法
1.0000000000000001598721 0.9999999999997646327 0.9999999999999988898 1.0000000000000000111022
隐式相对Gauss列主元消去法
1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000
```

问题二

```
Gauss列主元消去法
0.95367910690177182254 0.32095684552110359533 1.07870807579323835235 -0.09010850953957894038
显式相对Gauss列主元消去法
0.95367910690177204458 0.32095684552110353982 1.07870807579323835235 -0.09010850953957895426
隐式相对Gauss列主元消去法
0.95367910690177104538 0.32095684552110387289 1.07870807579323813030 -0.09010850953957892651

Gauss列主元消去法
0.51617729795854161434 0.41521947283013527219 0.10996610286788915944 1.03653922333620052143
显式相对Gauss列主元消去法
0.51617729795854228048 0.41521947283013604935 0.10996610286788954802 1.03653922333620229779
隐式相对Gauss列主元消去法
0.51617729795854183639 0.41521947283013549423 0.10996610286788929822 1.03653922333620096552

Gauss列主元消去法
1.0000000000000000000000 0.9999999999999977796 1.0000000000000022204
显式相对Gauss列主元消去法
1.0000000000000000000000 0.9999999999999977796 1.0000000000000000000000
隐式相对Gauss列主元消去法
1.0000000000000000000000 0.9999999999999977796 1.0000000000000022204

Gauss列主元消去法
1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000
显式相对Gauss列主元消去法
0.9999999999999866773 1.0000000000000111022 0.9999999999999822364
隐式相对Gauss列主元消去法
1.0000000000000000000000 1.0000000000000000000000 1.0000000000000000000000
```

问题一：

系数矩阵的元素按绝对值在数量级方面相差很小的情况，， 三种方法表现基本是一致的，而对于值的数量级相差较大的矩阵，如值集中在对角线附近的条带状矩阵，三种方法表现区别较大。