计算方法实验报告

姓名：王良希

学号：210110205

院系：计算机科学与技术学院

专业：计算机科学与技术

班级：2 班

实验报告一

第一部分：问题分析 *（描述并总结出实验题目）*

编写程序, 利用拉格朗日插值多项式 $P_n(x)$ 求 $f(x)$ 的近似值。

同时，还要回答以下三个问题：

问题 1 拉格朗日插值多项式的次数 n 越大越好吗？

问题 2 插值区间越小越好吗？

问题 4 考虑拉格朗日插值问题，内插比外推更可靠吗？

第二部分：数学原理

给定平面上 n+1 个不同的数据点 $\left(x_k, f\left(x_k\right)\right),\quad k=0,1,\cdots,n,\quad x_i \neq x_j,\quad i \neq j$；则满足条件

$$P_n\left(x_k\right)=f\left(x_k\right),\quad k=0,1,\cdots,n$$

的 n 次拉格朗日插值多项式

$$P_n(x)=\sum_{k=0}^{n} f\left(x_k\right) l_k(x)$$

是存在唯一的。若 $x_k \in[a,b], k=0,1,\cdots,n$，且函数 $f(x)$ 充分光滑, 则当 $x \in[a,b]$ 时, 有误差估计式

$$f(x)-P_n(x)=\frac{f^{(n+1)}(\xi)}{(n+1)!}\left(x-x_0\right)\left(x-x_1\right)\cdots\left(x-x_n\right),\quad \xi \in[a,b]$$

第三部分：程序设计流程

根据给出的程序流程可设计出：

```
n = 5;
x = linspace(-5, 5, n+1);
f = 1./(1+x.^2);
Pn1 = my_lagrange_interp(x, f, 0.75);
fprintf('x=0.75, Pn(x)= %.8f\n',Pn1)
Pn2 = my_lagrange_interp(x, f, 1.75);
fprintf('x=1.75, Pn(x)= %.8f\n',Pn2)
Pn3 = my_lagrange_interp(x, f, 2.75);
fprintf('x=2.75, Pn(x)= %.8f\n',Pn3)
Pn4 = my_lagrange_interp(x, f, 3.75);
fprintf('x=3.75, Pn(x)= %.8f\n',Pn4)
Pn5 = my_lagrange_interp(x, f, 4.75);
fprintf('x=4.75, Pn(x)= %.8f\n',Pn5)


function Pn = my_lagrange_interp(x, f, xi)
n = length(x) - 1; Pn = 0;
for k = 1:n+1
    l = 1;
    for j = [1:k-1, k+1:n+1]
        l = l*(xi - x(j))/(x(k) - x(j));
    end
    Pn = Pn + f(k)*l;
end
end
```

% 注：以上代码对应第(1)问的情形，其他情况需要修改代码.

第四部分：实验结果、结论与讨论

问题一(n 是否越大越好)

第(1)小问

n=5

```
>> lagrange_interp
x=0.75, Pn(x)= 0.52897386
x=1.75, Pn(x)= 0.37332482
x=2.75, Pn(x)= 0.15373347
x=3.75, Pn(x)= -0.02595403
x=4.75, Pn(x)= -0.01573768

n=10
>> lagrange_interp
x=0.75, Pn(x)= 0.67898958
x=1.75, Pn(x)= 0.19058047
x=2.75, Pn(x)= 0.21559188
x=3.75, Pn(x)= -0.23146175
x=4.75, Pn(x)= 1.92363115

n=20
>> lagrange_interp
x=0.75, Pn(x)= 0.63675534
x=1.75, Pn(x)= 0.23844593
x=2.75, Pn(x)= 0.08065999
x=3.75, Pn(x)= -0.44705196
x=4.75, Pn(x)= -39.95244903
```

实际值:

```
Pn(0.75)=0.64
Pn(1.75)=0.24615384
Pn(2.75)=0.11678832
Pn(3.75)=0.06639004
Pn(4.75)=0.04244031
```

第(2)小问

```
n=5
>> lagrange_interp
x=-0.95, Pn(x)= 0.38679816
x=-0.05, Pn(x)= 0.95124833
x=0.05, Pn(x)= 1.05129028
x=0.95, Pn(x)= 2.58578455
```

```
n=10
>> lagrange_interp
x=-0.95, Pn(x)= 0.38674102
x=-0.05, Pn(x)= 0.95122942
x=0.05, Pn(x)= 1.05127110
x=0.95, Pn(x)= 2.58570966

n=20
>> lagrange_interp
x=-0.95, Pn(x)= 0.38674102
x=-0.05, Pn(x)= 0.95122942
x=0.05, Pn(x)= 1.05127110
x=0.95, Pn(x)= 2.58570966
```

实际值为:

```
Pn(-0.95)=0.38674102
Pn(-0.05)=0.95122942
Pn(0.05)=1.05127110
Pn(0.95)=2.58570966
```

由实验结果可得:并不是插值次数越多越好，比如在函数 f(x)=1/(1+x^2)中, n 取 20 会发生严重的振荡现象,导致误差极大. 而在 f(x)=e^x 中，插值次数越多，精确度越高.

问题二(插值区间越小越好吗?)

(1)

```
n=5
>> lagrange_interp
x=-0.95, Pn(x)= 0.51714729
x=-0.05, Pn(x)= 0.99279067
x=0.05, Pn(x)= 0.99279067
x=0.95, Pn(x)= 0.51714729

n=10
```

```
>> lagrange_interp
x=-0.95, Pn(x)= 0.52640798
x=-0.05, Pn(x)= 0.99750686
x=0.05, Pn(x)= 0.99750686
x=0.95, Pn(x)= 0.52640798

n=20
>> lagrange_interp
x=-0.95, Pn(x)= 0.52562037
x=-0.05, Pn(x)= 0.99750623
x=0.05, Pn(x)= 0.99750623
x=0.95, Pn(x)= 0.52562037

实际值为:
Pn(-0.95)=0.52562418
Pn(-0.05)=0.99750623
Pn(0.05)=0.99750623
Pn(0.95)=0.52562418
```

(2)

```
n=5
>> lagrange_interp
x=-4.75, Pn(x)= 1.14703473
x=-0.25, Pn(x)= 1.30215246
x=0.25, Pn(x)= 1.84121041
x=4.75, Pn(x)= 119.62100706

n=10
>> lagrange_interp
x=-4.75, Pn(x)= -0.00195655
x=-0.25, Pn(x)= 0.77868634
x=0.25, Pn(x)= 1.28414449
x=4.75, Pn(x)= 115.60736006

n=20
>> lagrange_interp
x=-4.75, Pn(x)= 0.00865169
x=-0.25, Pn(x)= 0.77880078
```

```
x=0.25, Pn(x)= 1.28402542
x=4.75, Pn(x)= 115.58428453
```

实际值为:

```
Pn(-4.75)=0.00865169
Pn(-0.25)=0.77880078
Pn(0.25)=1.28402542
Pn(4.75)=115.58428453
```

由实验结果和问题一中的数据对比可知：在合适的插值区间内,插值区间越小，插值多项式精度越高.但是，如果插值区间过小，可能会导致插值多项式存在 Runge 现象，在插值点附近的误差会比较大,因此，插值区间大小的选择应该根据被插值函数的特点进行选择。对于变化剧烈的函数，可以采用分段插值的方式来提高插值的精度；如果被插值函数的变化比较平缓，则可以适当扩大插值区间。

问题 4 内插比外推更可靠吗？

（1）

得到的答案为：

```
>> lagrange_interp
x=5, Pn(x)= 2.26666667
x=50, Pn(x)= -20.23333333
x=115, Pn(x)= -171.90000000
x=185, Pn(x)= -492.73333333
```

（2）

得到的答案为：

```
>> lagrange_interp
x=5, Pn(x)= 3.11575092
```

```
x=50, Pn(x)= 7.07179487
x=115, Pn(x)= 10.16703297
x=185, Pn(x)= 10.03882784
```

（3）

得到的答案为：

```
>> lagrange_interp
x=5, Pn(x)= 4.43911161
x=50, Pn(x)= 7.28496142
x=115, Pn(x)= 10.72275551
x=185, Pn(x)= 13.53566723
```

（4）

得到的答案为：

```
>> lagrange_interp
x=5, Pn(x)= 5.49717205
x=50, Pn(x)= 7.80012771
x=115, Pn(x)= 10.80049261
x=185, Pn(x)= 13.60062032
```

由实验结果可知：通常情况下，内推（将插值点配置在有限范围内）的精度比外推的精度更高

思考题

1. 对实验 1 存在的问题，如何解决？

插值次数与插值精度之间存在一种权衡关系，插值次数越高，插值精度越高，但在过高的插值次数下，插值多项式会发生病态现象，导致插值结果不准确。因此，有以下几种解决方案：

a.采用其他高效准确的方法进行插值，如样条插值方法。样

条插值法能够通过一定的参数选择达到很高的插值精度。

b.在函数变化剧烈的区域内，增加插值节点的密度。

c.将多项式表达式进行分段拟合， 使得每段的斜率变化范围较小。

2. 对实验 2 存在的问题的回答，试加以说明

插值区间的大小并不是越小越好，而是要根据实际问题的需要进行灵活选择。当插值区间变小时，插值多项式的阶数会变高，从而导致插值多项式的病态程度更严重，进而导致插值误差的增大。但如果插值区间过大，则无法准确地捕捉函数的局部变化规律，也会导致插值误差的增大。

3. 如何理解插值问题中的内插和外推？

内插：当目标点在插值点之间时，我们可以通过插值公式得到目标点的值。这种情况称为内插，插值点范围包含目标点。

外推：当目标点超出插值点的范围时，我们可以通过插值公式对目标点进行预测。这种情况称为外推，插值点范围不包括目标点。

在内插和外推中，内插通常比外推更加可靠，因为在内插中目标点与插值点之间的距离更近，可以更好地反映原函数的变化趋势。而在外推中，目标点可能超出了原始数据范围，从而导致插值误差的增加。

实验报告二

第一部分：问题分析　*（描述并总结出实验题目）*

方法概要：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

$h=\dfrac{b-a}{n},\quad x_k=a+k\cdot h,\quad k=0,1,\cdots,n$，其计算公式:

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

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

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

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

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

一般地，利用龙贝格算法计算积分，要输出所谓的$T-$数表:

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

第二部分：数学原理

假设要计算区间[a, b]上的某个函数 f(x)的定积分，我们将区间[a, b]进行等分，将区间步长 h 逐步减半，并用梯形公式求得每个步长下积分的估计值，即：

$$T_{0,0}=\frac{1}{2}h(f(a)+f(b))$$

$$T_{1,0}=\frac{1}{2}T_{0,0}+h\sum_{i=1}^{n}f(a+i\frac{b-a}{n})$$

$$T_{2,0}=\frac{1}{2}T_{1,0}+\frac{h}{2}\sum_{i=1}^{n}f(a+i\frac{b-a}{n})$$

$$\cdots$$

$$T_{m,0}=\frac{1}{2}T_{m-1,0}+\frac{h}{2^{m}}\sum_{i=1}^{n}f(a+i\frac{b-a}{n})$$

其中，m 表示递归次数，n 表示子区间数，h 表示每个子区间的长度，$T_{m,0}$ 表示递归 m 次后的积分估计值。

接下来，我们使用龙贝格公式计算更精确的估计值，公式如下：

$$T_{m,n}=\frac{4^{n}T_{m,n-1}-T_{m-1,n-1}}{4^{n}-1}$$

其中，$T_{m,n}$ 表示递归 m 次、精度为 n 的积分估计值。

通过递归计算，可以不断提高积分的精度，直到达到预设的误差要求。

第三部分：程序设计流程

根据给出的程序流程可得:

```
f = @(x) x.^2 .* exp(x);
%f = @(x) exp(x).*sin(x);
%f = @(x) 4./(1+x.^2);
%f = @(x) 1./(x+1);
a= 0;
b = 1;
eps = 1e-6;
R = my_romberg(a, b, f, eps);
format long;
display(R)
fprintf("最终近似结果:%.15f\n", R(end, end))
fprintf("实际精确值是:%.15f\n", integral(f,a,b))


function R = my_romberg(a, b, f, eps)
h = b - a;
R = zeros(20);
R(1, 1) = h / 2 * (f(a) + f(b));
for j = 2 : 10
    h = h / 2;
    sum = 0;
    for k = 1 : 2^(j-2)
        sum = sum + f(a + (2*k-1)*h);
    end
    R(j, 1) = 1 / 2 * R(j-1, 1) + h * sum;
    for i = 2 : j
        R(j, i) = (4^(i-1) * R(j, i-1) - R(j-1, i-1)) / (4^(i-1) - 1);
    end
    if abs(R(j, j) - R(j-1, j-1)) < eps
        break;
    end
end
R = R(1:j, 1:j);
end
```

第四部分：实验结果、结论与讨论

（1）：$\int_0^1 x^2 \mathrm{e}^x dx,\quad \varepsilon = 10^{-6}$：

R =

1.359140914229523	0	0	0	0
0.885660615952277	0.727833849859862	0	0	0
0.760596332448042	0.718908237946630	0.718313197152415	0	0
0.728890177014693	0.718321458536910	0.718282339909595	0.718281850112090	0
0.720935778937658	0.718284312911980	0.718281836536984	0.718281828546943	0.718281828462374

最终近似结果:0.718281828462374
实际精确值是:0.718281828459045

（2）：$\int_1^3 \mathrm{e}^x \sin x dx,\quad \varepsilon = 10^{-6}$

R =

5.121826419665847	0	0	0	0	0
9.279762907261173	10.665741736459614	0	0	0	0
10.520554283818644	10.934151409337801	10.952045387529681	0	0	0
10.842043467557430	10.949206528803691	10.950210203434750	10.950181073528482	0	0
10.923093889613778	10.950110696965893	10.950170974843372	10.950170352167319	10.950170310122767	0
10.943398421186796	10.950166598377800	10.950170325138595	10.950170314825822	10.950170314679385	10.950170314683838

最终近似结果:10.950170314683838
实际精确值是:10.950170314685517

（3）：$\int_0^1 \frac{4}{1+x^2} dx,\quad \varepsilon = 10^{-6}$

R =

3.000000000000000	0	0	0	0	0
3.100000000000000	3.133333333333333	0	0	0	0
3.131176470588235	3.141568627450980	3.142117647058823	0	0	0
3.138988494491089	3.141592502458707	3.141594094125888	3.141585783761874	0	0
3.140941612041389	3.141592651224822	3.141592661142563	3.141592638396796	3.141592665277717	0
3.141429893174974	3.141592653552836	3.141592653708038	3.141592653590029	3.141592653649611	3.141592653638244

最终近似结果:3.141592653638244

实际精确值是:3.141592653589793

(4) : $\int_0^1 \frac{1}{x+1}dx, \quad \varepsilon = 10^{-6}$

R =

0.750000000000000	0	0	0	0
0.708333333333333	0.694444444444444	0	0	0
0.697023809523809	0.693253968253968	0.693174603174603	0	0
0.694121850371850	0.693154530654531	0.693147901481235	0.693147477644832	0
0.693391202207527	0.693147652819419	0.693147194297078	0.693147183071933	0.693147181916745

最终近似结果:0.693147181916745
实际精确值是:0.693147180559946

思考题:

龙贝格积分法中二分次数和精度有直接关系，二分次数越多，精度越高。具体地，龙贝格积分法的精度可以通过迭代公式中的误差估计来衡量，每迭代一次就会把区间分成两半，即增加一次二分次数。因此，二分次数和精度之间是近似成线性关系的，即当二分次数翻倍时，精度也翻倍。

但是，也有可能先达到了设定的精度 eps，这种情况次数和精度的关系有限，只要二分次数大于在达到设定的精度 eps 所需的二分次数。

实验报告三

第一部分：问题分析　*（描述并总结出实验题目）*

利用牛顿迭代法，在给定精度、最大迭代次数、初值的条件下求近似零点。需要根据给定的程序设计流程，设计 matlab 代码。

第二部分：数学原理

求非线性方程 $f(x)=0$ 的根 x^*，有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 $\delta>0$，$\alpha\in O\left(x^*,\delta\right)$。如果 $f(x)\in C^2[a,b]$，$f\left(x^*\right)=0$，$f'\left(x^*\right)\neq 0$，那么，对充分小的 $\delta>0$，当 $\alpha\in O\left(x^*,\delta\right)$ 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 $f(x)\in C^m[a,b]$，$f\left(x^*\right)=f'\left(x^*\right)=\cdots=f^{(m-1)}\left(x^*\right)=0$，$f^{(m)}\left(x^*\right)\neq 0(m>1)$，那么，对充分小的 $\delta>0$，当 $\alpha\in O\left(x^*,\delta\right)$ 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。

第三部分：程序设计流程

根据给出的程序设计流程可得：

```
syms x;

f = cos(x)-x; %这些参数(包括下面的eps等等)根据题目修改

fh = matlabFunction(f);

df = matlabFunction(diff(f));

eps1 = 1e-6;

eps2 = 1e-4;

N = 10;

x0 = pi/4;

[x_star, n] = newton(fh, df, eps1, eps2, x0, N);

display(x_star)


function [x_star, n] = newton(f, df, eps1, eps2, x0, N)

    n=1;

    while n<= N

        %2.1

        F = f(x0);

        DF = df(x0);

        if abs(F)<eps1

            x_star = x0;

            return
```

```
        end
        if abs(DF)<eps2
            display("Failed")
            return
        end
        %2.2
        x1=x0-F/DF;
        %2.3
        Tol = abs(x1-x0);
        if abs(Tol)<eps1
            x_star=x1;
            return
        end
        %2.4
        n=n+1;
        x0=x1;
    end
    display("Failed")
    return
end
```

第四部分：实验结果、结论与讨论

问题 1:

(1)： $\cos x - x = 0$， $\varepsilon_1 = 10^{-6}$， $\varepsilon_2 = 10^{-4}$， $N = 10$， $x_0 = \frac{\pi}{4} \approx 0.785398163$

```
>> Newton

x_star =

  0.739085178106010
```

(2)： $e^{-x} - \sin x = 0$， $\varepsilon_1 = 10^{-6}$， $\varepsilon_2 = 10^{-4}$， $N = 10$， $x_0 = 0.6$

```
>> Newton

x_star =

  0.588532742847979
```

问题 2:

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

```
>> Newton

x_star =

  0.567143165034862
```

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

```
>> Newton

x_star =

   0.566605704128158
```

思考题:

1. 确定初值的原则是什么? 实际计算中如何解决?

可以遵循以下原则:

a. 理解问题并利用图形辅助选择初始点。将函数绘制成图像，大致确定根的位置，并选择与其尽可能接近的初始点。

b. 选择已知根的领域内的点。如果已知根的大致范围，则可以选择该范围内的点作为初始点。

c. 利用其他数值方法得到初始点。例如，使用二分法或割线法等方法，得到一个粗略的近似解，然后使用牛顿迭代法进一步提高精度。

在实际计算中，通常需要结合以上原则进行选择。对于复杂的问题，可以采用试错的方法，针对不同的初始点进行计算，并检查结果是否满足要求。此外，还可以使用自适应算法，根据迭代过程中的误差调整初始点，以提高收敛速度和精度。

2. 对实验二出现的现象怎么解释?

对于求出的零点不精确的情况，有以下解释:

1. 函数可能存在多个零点，零点间的距离较远，而初值 x0 将决定最终收敛的情况，如果 x0 离所需的零点较远，则可能收敛到其他的点，使得结果偏离真实值.

2. 因此，在使用牛顿迭代法时，要根据实际情况选择合适的初值，并且对迭代结果进行检查。同时，对于存在多个零点的函数，可以考虑使用其他的数值求解方法，例如二分法、割线法，以免收敛到错误的解。

实验报告四

第一部分：问题分析 *（描述并总结出实验题目）*

用 matlab 解决以下问题：给定 n 阶线性方程 $Ax=b$，首先进行列主元消去过程，然后进行回代，最后得到这个方程的解或者确定该方程组是奇异的。

输入：n；a_{ij}，b_i，$i,j=1,2,\cdots,n$

输出：线性方程组 $Ax=b$ 的近似解 x_i，$i=1,2,\cdots,n$

第二部分：数学原理

假设有一个线性方程组：$A\mathbf{x}=\mathbf{b}$，其中 A 是系数矩阵，$\mathbf{x}$ 是未知量向量，$\mathbf{b}$ 是常数向量。为了求解 $\mathbf{x}$，可以使用高斯列主元消去法。

让我们首先考虑如何将系数矩阵 A 变换为一个上三角矩阵。这可以通过矩阵的行变换和列变换来实现。具体来说，可以通过以下步骤来将 A 变换为上三角矩阵：

1. 找到第 j 列中绝对值最大的元素 $a_{k,j}$，即满足 $|a_{k,j}|=\max_{i=j,\ldots,n}|a_{i,j}|$ 的 k。将第 k 行交换到第 j 行，即 $a_{j,j}\leftrightarrow a_{k,j}$。

2. 将第 j 行除以 $a_{j,j}$，使得 $a_{j,j}=1$。

3. 对于下面的所有行 $i=j+1,\ldots,n$，将 $a_{i,j}$ 这个元素所在的列进行列变换，使得 $a_{i,j}=0$。

重复以上步骤，直到 A 变换为一个上三角矩阵 U。

高斯列主元消去法的核心就是在每一步中选择主元，即选取第 j 列中绝对值最大的元素作为主元。这样可以保证主元的绝对值最大，从而减小了舍入误差的影响。

现在，将 A 变换为上三角矩阵 U 后，我们可以通过回代求解 $\mathbf{x}$。回代的过程与高斯消去法类似，但顺序是相反的。具体来说，我们从最后一行开始，往前逐行求解

$\mathbf{x}_i$，并将其代入下一行的方程中，直到求出 $\mathbf{x}_1$。回代的过程如下：

1. 对于最后一行 n，有 $x_n = b_n / a_{n,n}$。

2. 对于 $i = n-1, n-2, \ldots, 1$，计算 $\mathbf{x} * i$，有 $x_i = (b_i - \sum * j = i+1^n a_{i,j} x_j) / a_{i,i}$。

这样，就可以用高斯列主元消去法求解线性方程组了。

第三部分：程序设计流程

```
x1 = gauss(A1,b1)
x2 = gauss(A2,b2)
x3 = gauss(A3,b3)
x4 = gauss(A4,b4)
x5 = gauss(A5,b5)
x6 = gauss(A6,b6)
x7 = gauss(A7,b7)
x8 = gauss(A8,b8)

function [x] = gauss(A,b)
n = size(A,1);

for k = 1:n-1
    [~,p] = max(abs(A(k:n,k)));
    p = p + k - 1;
    if A(p,k) == 0
        display('奇异')
        return
    end
    if p ~= k
        A([k,p],:) = A([p,k],:);
        b([k,p]) = b([p,k]);
    end
    for i = k+1:n
        m = A(i,k) / A(k,k);
        A(i,k+1:n) = A(i,k+1:n) - m * A(k,k+1:n);
```

```
            b(i) = b(i) - m * b(k);
      end
end

if A(n,n) == 0
      display('奇异')
      return
end
x = zeros(n,1);
x(n) = b(n) / A(n,n);
for k = n-1:-1:1
      x(k) = (b(k) - A(k,k+1:n) * x(k+1:n)) / A(k,k);
end

end
```

第四部分：实验结果、结论与讨论

```
>> Gauss

x1 =

    1.0000
    1.0000
    1.0000
    1.0000


x2 =

    1.0000
    1.0000
    1.0000
    1.0000


x3 =

    1.0000
    1.0000
    1.0000
    1.0000


x4 =

     1
     1
     1
     1
```

```
x5 =

    0.9537
    0.3210
    1.0787
   -0.0901


x6 =

    0.5162
    0.4152
    0.1100
    1.0365


x7 =

    1.0000
    1.0000
    1.0000


x8 =

     1
     1
     1
```