计算方法实验报告

姓名：	朱海峰
学号：	190110716
院系：	计算机科学与技术
专业：	计算机类
班级：	7 班

实验报告一 拉格朗日插值

题目（摘要）

给定若干个插值点，利用拉格朗日插值多项式

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

求$f(x)$的近似值。

前言：（目的和意义）

目的：

1. 学习和掌握拉格朗日插值多项式；
2. 运用拉格朗日插值多项式进行计算。

意义：

在实际问题中，某些函数存在且连续，但是很难找到解析表达式，只能通过有限点上的函数表，来构造简单函数$P(x)$作为$f(x)$的近似值。插值法是解决此类问题的一种比较古老的、但却很常用的方法。它不仅直接广泛地应用于生产实际和科学研究中，而且也是进一步学习数值计算方法的基础。

数学原理

给定平面上$n+1$个不同的数据点$\big(x_k, f(x_k)\big), k = 1,2, \dots, n\ x_i \neq x_j$, 则满足条件

$$P_n(x_k) = f(x_k), k = 0,1, \dots, n$$

的 n 次拉格朗日插值多项式

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

是唯一存在的。若$x_k \in [a,b], k = 0,1, \dots, n$，且函数$f(x)$充分光滑，则当$x \in [a,b]$时，有误差估计式

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)(x - x_1) \dots (x - x_n)$$

程序设计流程

输入：$n+1$个数据点$\big(x_k, f(x_k)\big), k = 1,2, \dots, n\ x_i \neq x_j$；插值点$x$

输出：$f(x_k)$在插值点x的近似值$P_n(x)$

具体流程：

1　置$y = 0; k = 0$;

2　当$k \leq n$时，循环：

2.1　置$l = 0$;

2.2　对$j = 1,2, \dots, j-1, j+1, \dots, n$, 置$l = l * \frac{x - x_j}{x_k - x_j}$

2.3　置$y = y + l * f(x_k)$

2.4　置$k = k + 1$

3　输出x, y

4　停机

实验结果、结论与讨论

代码发布报告、具体结果在下页。

思考题：

问题 1：

拉格朗日插值多项式的次数并不是越大越好，根据定义，插值式可以在节点处与实际函数匹配，但不能保证在节点之间很好的逼近实际函数。这个现象就是多项式摆动——Runge 现象

有时多项式摆动可以通过谨慎选择基础函数的取样点来减小；或者通过分段插值来减小。

问题 2：

一般来说，插值区间越大，误差越大，而函数在比较小的区间上的函数值变化较缓和，因此即使出现摆动也不会偏离函数太大。

问题 4：

内插是指插值点在给定的数据点范围内，外推指插值点给定的数据点范围外面。仅从本次问题 4 的实验结果来看，内插的精确度比外推更好，内插比外推更可靠。

```
function px = Lagrange(X,Y,x)
if(length(X) ~=length(Y))
    error('check the length of X and Y')
end
y = 0;
k = 0;
n = length(X) - 1;
while(k<=n)
    l = 1;
    for j = 0:n
        if(j == k)
            continue
        end
        l = l * (x - X(j+1)) / (X(k+1) - X(j+1));

    end
    y = y + l*Y(k+1);
    k = k + 1;
end
px = y;
```

Published with MATLAB® R2020a

```
% problem 1

X_p = .75:1:4.75;
answer = zeros(3,5);
for id = 1:length(X_p)
    X = -5:2:5;
    answer(1,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
    X = -5:1:5;
    answer(2,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
    X = -5:.5:5;
    answer(3,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
end
answer1_1 = answer

X_p = [-.95 -.05 .05 .95];
answer = zeros(3,4);
for id = 1:length(X_p)
    X = -1:.4:1;
    answer(1,id) = Lagrange(X,  exp(X),  X_p(id));
    X = -1:.2:1;
    answer(2,id) = Lagrange(X,  exp(X),  X_p(id));
    X = -1:.1:1;
    answer(3,id) = Lagrange(X,  exp(X),  X_p(id));
end
answer1_2 = answer

% problem 2
X_p = [-.95 -.05 .05 .95];
answer = zeros(3,4);
for id = 1:length(X_p)
    X = -1:.4:1;
    answer(1,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
    X = -1:.2:1;
    answer(2,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
    X = -1:.1:1;
    answer(3,id) = Lagrange(X,  1./(X.^2 + 1),  X_p(id));
end
answer2_1 = answer

X_p = [-4.75 -.25 .25 4.75];
answer = zeros(3,4);
for id = 1:length(X_p)
    X = -5:2:5;
    answer(1,id) = Lagrange(X,  exp(X),  X_p(id));
    X = -5:1:5;
    answer(2,id) = Lagrange(X,  exp(X),  X_p(id));
    X = -5:.5:5;
    answer(3,id) = Lagrange(X,  exp(X),  X_p(id));
end
answer2_2 = answer

% problem 5
```

```
X_p = [5 50 115 185];
answer = zeros(4,4);
for id = 1:length(X_p)
    X = [1 4 9];
    answer(1,id) = Lagrange(X,  sqrt(X),  X_p(id));
    X = [36 49 64];
    answer(2,id) = Lagrange(X,  sqrt(X),  X_p(id));
    X = [100 121 144];
    answer(3,id) = Lagrange(X,  sqrt(X),  X_p(id));
    X = [169 196 225];
    answer(4,id) = Lagrange(X,  sqrt(X),  X_p(id));
end
answer4 = answer
```

```
answer1_1 =

    0.5290    0.3733    0.1537   -0.0260   -0.0157
    0.6790    0.1906    0.2156   -0.2315    1.9236
    0.6368    0.2384    0.0807   -0.4471  -39.9524


answer1_2 =

    0.3868    0.9512    1.0513    2.5858
    0.3867    0.9512    1.0513    2.5857
    0.3867    0.9512    1.0513    2.5857


answer2_1 =

    0.5171    0.9928    0.9928    0.5171
    0.5264    0.9975    0.9975    0.5264
    0.5256    0.9975    0.9975    0.5256


answer2_2 =

    1.1470    1.3022    1.8412  119.6210
   -0.0020    0.7787    1.2841  115.6074
    0.0087    0.7788    1.2840  115.5843


answer4 =

    2.2667  -20.2333 -171.9000 -492.7333
    3.1158    7.0718   10.1670   10.0388
    4.4391    7.2850   10.7228   13.5357
    5.4972    7.8001   10.8005   13.6006
```

Published with MATLAB® R2020a

实验报告二 龙贝格积分法

题目（摘要）

利用龙贝格积分法计算积分$\int_a^b f(x)\,dx$。给定积分上下限$a,b,f(x)$,和误差限ε,得到龙贝格T −数表。

其中,$\int_a^b f(x)\,dx$:

$$\int_0^1 x^2 e^x dx,\ \varepsilon = 10^{-6}$$

$$\int_1^3 e^x \sin x dx,\ \varepsilon = 10^{-6}$$

$$\int_0^1 \frac{4}{1+x^2} dx,\ \varepsilon = 10^{-6}$$

$$\int_0^1 \frac{1}{x+1} dx,\ \varepsilon = 10^{-6}$$

前言：（目的和意义）

目的：

1. 进一步理解和掌握龙贝格积分的原理；
2. 运用龙贝格积分法数值计算定积分。

意义：

龙贝格积分法实在梯形公式、辛普森公式和柯特斯公式之间的关系基础上，构造出的一种加速计算积分的方法，作为一种外推算法，在不增加计算量的前提下提高了误差的精度。

数学原理

利用复化梯形求积公式、复化辛普森求积公式、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$h=\frac{b-a}{n}, x_k=a+k*h, k=0,1,\dots,n$, 其计算公式:

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

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

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

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

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

得到$T-$数表:

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

程序设计流程

输入：a, b, N, ε

输出：龙贝格T −数表

具体流程：

1　置$h = b - a, i = 1$；$T_{1,1} = \frac{1}{2}h[f(a) + f(b)], err = \varepsilon + 1$

2　当$err > \varepsilon$, 循环：

2.1　$i = i + 1$;

2.2　置$sum = \sum_{k=1}^{2^{i-2}} f\left(a + kh - \frac{h}{2}\right)$

2.3　置$T_{i,1} = \frac{T_{i-1,1}}{2} + sum * \frac{h}{2}$

2.4　置$end = \max(4, i)$

2.5　对$j = 2,3, \ldots, end$, 循环:

2.5.1$T_{i,j} = T_{i,j-1} + \frac{T_{i,j-1} - T_{i-1,j-1}}{4^{j-1} - 1}$

2.5.2如果$j = 4$, 且$abs\left(T_{i,1} - T_{i,1}\right) < \varepsilon$, 则停机

2.6　$h = \frac{h}{2}$

实验结果、结论与讨论

代码发布报告、具体结果在下页。结论与预期相符。

思考题：

显然, 随着二分次数的增加, 精度也在提升。

```
function [ R ] = rombergIntegration( f, a, b, epsilon)
% epsilon is the tolerance for each iteration
h = (b-a);
i = 1;
err = epsilon + 1;
R = zeros(5,4);
R(1,1) = h * (f(a)+f(b))/2;
while(err > epsilon)
    i = i + 1;
    sum = 0;
    for k=1:2^(i-2)
        sum = sum + f(a + (k-.5)*h);
    end
    R(i,1) = R(i-1,1)/2 + sum * h / 2;

    if(i>=4)
        j_end = 4;
    else
        j_end = i;
    end

    for j=2:j_end
        R(i,j) = R(i,j-1) + (R(i,j-1)-R(i-1,j-1))/(4^(j-1)-1);
        if(i>=5 && j==4)
            err = abs(R(i,j) - R(i-1,j));
            if err < epsilon
                return
            end
        end
    end
    h = h/2;
end
```

Published with MATLAB® R2020a

```
opts = {};
tmp = {@(x)x^2*exp(x),  0,1,1e-6};opts = [opts;tmp];
tmp = {@(x)exp(x)*sin(x),1,3,1e-6};opts = [opts;tmp];
tmp = {@(x)4/(1+x^2),   0,1,1e-6};opts = [opts;tmp];
tmp = {@(x)1/(1+x),     0,1,1e-6};opts = [opts;tmp];

for i = 1:length(opts)
    res = rombergIntegration(opts{i,:})
end
```

```
res =

  # 1 # 3

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

  # 4

                   0
                   0
                   0
   0.718281850112090
   0.718281828546943
```

```
res =

  # 1 # 3

   5.121826419665847                   0                   0
   9.279762907261173  10.665741736459614                   0
  10.520554283818644  10.934151409337801  10.952045387529679
  10.842043467557430  10.949206528803691  10.950210203434750
  10.923093889613778  10.950110696965893  10.950170974843374
  10.943398421186796  10.950166598377802  10.950170325138595

  # 4

                   0
                   0
                   0
  10.950181073528482
  10.950170352167321
  10.950170314825820
```

```
res =
```

```
  # 1 # 3

   3.000000000000000                   0                   0
   3.100000000000000   3.133333333333333                   0
   3.131176470588235   3.141568627450980   3.142117647058823
   3.138988494491089   3.141592502458707   3.141594094125888
   3.140941612041389   3.141592651224822   3.141592661142563
   3.141429893174974   3.141592653552836   3.141592653708037

  # 4

                    0
                    0
                    0
   3.141585783761874
   3.141592638396796
   3.141592653590029


res =

  # 1 # 3

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

  # 4

                    0
                    0
                    0
   0.693147477644832
   0.693147183071933
```

Published with MATLAB® R2020a

实验报告四 牛顿迭代法

题目（摘要）

使用牛顿迭代法计算公式：

$$x_0 = \alpha$$

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

$$n = 0,1,\ldots$$

求非线性方程$f(x) = 0$的根x^*。

其中：

问题	$f(x)$	ε_1	ε_2	N	x_0
1	$cosx - x$	10^{-6}	10^{-4}	10	$\pi/4$
2	$e^{-x} - sinx$				0.6
3	$x - e^{-x}$				0.5
4	$x^2 - 2xe^{-x} + e^{-2x}$				0.5

前言：（目的和意义）

目的：

1. 用牛顿迭代法求解方程的根
2. 了解迭代法的原理
3. 代码实现牛顿迭代法
4. 考虑特殊情况

意义：

1. 学习使用 matlab 语言
2. 提升对牛顿迭代法的认识
3. 增加对上机作业的经验

数学原理

一般地，牛顿迭代法具有局部收敛性，为了保证迭代收敛，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$x^*$,且收敛速度为 2 阶的；

如果

$$\begin{cases} f(x) \in C^m[a,b] \\ f(x^*) = f'(x^*) = \cdots = f^{(m-1)}(x^*) = 0 \\ f^{(m)}(x^*) \neq 0 \end{cases}$$

那么，对于充分小的$\delta > 0$，当$\alpha \in O(x^*, \delta)$时，由牛顿迭代法计算出的$\{x_n\}$收敛于$x^*$,且收敛速度为 1 阶的。

程序设计流程

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^*的近似值或者计算失败的标志

流程：

1　置$n = 1$

2　当$n \leq N$时：

　　2.1　置$F = f(x_{n-1}), DF = f'(x_{n-1})$

　　如果$|F| < \varepsilon_1$,输出x_0；停机

　　如果$|DF| < \varepsilon_2$,输出失败标志；停机

　　2.2　置$x_n = x_{n-1} - \frac{F}{DF}$

　　2.3　置$Tol = |x_n - x_{n-1}|$

　　如果$Tol < \varepsilon_1$,输出x_n；停机

　　2.4　置$n = n + 1$

3　输出失败标志

4　停机

实验结果、结论与讨论

问题	$f(x)$	ε_1	ε_2	N	x_0	x_n
1	$cosx - x$	10^{-6}	10^{-4}	10	$\pi/4$	0.739085178106010
2	$e^{-x} - sinx$				0.6	0.588532742847979
3	$x - e^{-x}$				0.5	0.567143165034862
4	$x^2 - 2xe^{-x} + e^{-2x}$				0.5	0.566605704128158

显然，四个问题都能成功完成迭代，没有失败输出。

而且在 N 足够大的情况下，根的近似值能足够精确。

思考题：

1. 选择一个x_0，使得$f(x)$接近于精确解，实际运算中可以考虑二分法大致确定一个不太精确的数开始迭代；
2. 数学上来看显然两个解是完全相等的，但是在程序顺利完成的情况下得出的结果有所不同。

   是因为前一问是单根，二阶收敛；后一问是重根，是局部线性收敛，的迭代次数就会更多，要求更准确的解才能满足题目给出的误差条件。所以两者的结果有所不同。

```
function [x_end,X,status]=newton(fun,alpha,epsilon1,epsilon2,N)%######
#####
    syms x%####
    df = diff(fun(x));%###

    status = 'succeed';
    X = [];
    if abs(fun(alpha)) < epsilon1
        x_end = alpha;
    end

    if abs(subs(df,alpha)) < epsilon2
        status = 'error';
        return
    end

    X(1) = alpha - fun(alpha)/subs(df,alpha);

    for i = 2:N
        X(i) = X(i-1) -  fun(X(i-1))/subs(df,X(i-1));
%         disp(double(X(i)))

        if abs(fun(X(i))) < epsilon1
            x_end = X(i);
%             disp('1-----------')
%             disp(X(i))
            return
        end

        if abs(subs(df,X(i))) < epsilon2
            status = 'error';
            return
        end
    end


    status = 'error';
    x_end = X(length(X));

%     disp('2-----------')
%     disp(x_end)
end
```

Published with MATLAB® R2020a

```
fun1 = @(x)cos(x)-x;
[res1,a,b] = newton(fun1,pi/4,1e-6,1e-4,10);

fun2 = @(x)exp(-x) - sin(x);
[res2,a,b] = newton(fun2,.6,1e-6,1e-4,10);

fun3 = @(x)x-exp(-x);
[res3,a,b] = newton(fun3,.5,1e-6,1e-4,10);


fun4 = @(x)x^2-2*x*exp(-x)+exp(-2*x);
[res4,a,b] = newton(fun4,.5,1e-6,1e-4,10);
disp('res1')
disp(res1)
disp('res2')
disp(res2)
disp('res3')
disp(res3)
disp('res4')
disp(res4)
```

```
res1
   0.739085178106010

res2
   0.588532742847979

res3
   0.567143165034862

res4
   0.566605704128158
```

Published with MATLAB® R2020a

实验报告五

题目（摘要）

利用 Gauss 列主元消去法求解线性方程组：

$$Ax = b$$

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

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

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

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

前言：（目的和意义）

1. 熟悉运用已学的数值运算方法求解线性方程——Gauss 列主消去法；
2. 加深对计算方法技巧的认识，正确使用计算方法来求解方程；
3. 培养用计算机来实现科学计算和解决问题的能力。

数学原理

一般地，牛顿迭代高斯（Gauss）列主元消去法：对给定的 n 阶线性方程组 $Ax = b$，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$Ax = b$，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$Ax = b$，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

输入：n，$A_{n\times n}, b_n$

输出：线性方程组$Ax = b$根x^*的近似值或者计算失败的标志

流程：

1 对 k= $1,2,\dots,n-1$:

1.1 寻找最小的正整数 p，$k \le p \le n$和$|a_{pk}| = \max\limits_{k\le j\le n} |a_{jk}|$。如果$|a_{jk}| = 0$，输出奇异标志，停机；

1.2 如果$p \neq k$,那么交换 p,k 两行；

1.3 对$i = k+1,\dots,n$，记$m_{ik} = a_{ik}/a_kk$，计算

$$\begin{cases} a_{ij} = a_{ij} - a_{ij}m_{ik} \\ i = k+1,\dots,n \\ j = k+1,\dots,n \\ b_i = b_i - b_k m_{ik} \\ i = k+1,\dots,n \end{cases}$$

2 当$a_{nn} = 0$，输出奇异标志，停机；

3 置$x_n = b_n/a_{nn}$，回代过程

4 对$k = n-1,\dots,2,1$，置$x_k = \frac{b_k - \sum_{j=k+1}^{n} a_{kj}x_j}{a_{kk}}$

实验结果、结论与讨论

实验具体结果见下一页。

结论：

1. 实验结果与理论一致；
2. 使用列主元消元法的误差较小；
3. 运用程序能更好地实现理论与科学计算的统一和协调。

```
function [x ] = gauss_jordon(A,b)

%####A####
   [m1,m2] = size(A);
   if m1 ~= m2
       error('####A####')
       return
   end
   siz = m1;
   [n1,n2] = size(b);

   %####b####
   if n2 == siz && n1==1
       b = b';
   elseif n1==1 && n2 == siz
       b = b;
   else
       error('b####A####')
   end

   if rank(A)~=rank([A,b])
       error('A#########################');
       return;
   end
   %#A##
   A = [A,b];
   x = zeros(siz,1);

   for i = 1:siz-1
       a_ii = abs(A(i,i));
       [max_v,max_id_relative] = max(abs(A(i:siz)));
       max_id = i + max_id_relative - 1;
       if max_id ~=i
           A([i,max_id],:) = A([max_id,i],:);
       end
       %##
       for j = i+1:siz
           A(j,:) = A(j,:) - (A(j,i)/A(i,i)) * A(i,:);
       end
   end

   x(siz) = A(siz,siz+1)/A(siz,siz);
   for i = siz-1:-1:1
       x(i) = (A(i,siz+1) - A(i,i+1:siz)*x(i+1:siz)) / A(i,i);
   end

return
```

Published with MATLAB® R2020a

```
A1 = [
    .4096 .1234 .3678 .2943;
    .2246 .3872 .4015 .1129;
    .3645 .1920 .3781 .0643;
    .1784 .4002 .2786 .3927];
b1 = [1.1951 1.1262 .9989 1.2499];
res1 = gauss_jordon(A1,b1)

A2 = [136.01,90.860,0 0;
    90.860 98.810 -67.59 0 ;
    0 -67.59 132.01 46.26;
    0 0 46.26 177.17];
b2 = [226.87 122.08 110.68 223.43];
res2 = gauss_jordon(A2,b2)

A3 = [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];
b3 = [25/12 77/60 57/60 319/420];
res3 = gauss_jordon(A3,b3)

A4 = [10 7 8 7;7 5 6 5;8 6 10 9;7 5 9 10];
b4 = [32 23 33 31];

res4 = gauss_jordon(A4,b4)


res1 =

    1.0000
    1.0000
    1.0000
    1.0000


res2 =

    1.0000
    1.0000
    1.0000
    1.0000


res3 =

    1.0000
    1.0000
    1.0000
    1.0000
```

```
res4 =

    1.0000
    1.0000
    1.0000
    1.0000
```

Published with MATLAB® R2020a