计算方法实验报告

姓名：谢宇航

学号：200110505

院系：计算机学院

专业：计算机科学与技术

班级：5 班

实验报告一：拉格朗日(Lagrange)插值法

第一部分：问题分析 *（描述并总结出实验题目）*

本次实验主要要求实现*Lagrange*插值方法的程序编写以及分析比较不同情况下如插值点的个数、插值区间大小、 内插外插情况下程序的可靠性等一系列问题。在此次实验中，通过给定平面上的$n+1$个不同数据点$\left(x_k,f\left(x_k\right)\right)$，构建出*Lagrange*插值函数，并根据此插值函数计算一些定点的拟合函数值。通过不断分析与比较，进一步得出*Lagrange*插值函数的可靠性、拟合程度与不同插值情况的关系，据此探讨了在程序求解中出现的各种问题，并回答思考题相关题目，加深对于*Lagrange*插值方法在不同维度的理解。此次实验，一方面锻炼了同学的代码编写和程序设计能力，另一方面通过编写代码和分析结果，使同学们更进一步理解了*Lagrange*插值相关理论，提高了问题分析思考能力。

第二部分：数学原理

给定平面上$n+1$个不同数据点$(x_k,f(x_k))$ $k=0,1,\ldots,n$， $x_i\neq x_j,i\neq j$，则满足条件

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

的n次拉格朗日多项式

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

是存在唯一的，其中

$$l_k(x)=\prod_{\substack{j=0\\ j\neq k}}^{n}\frac{x-x_j}{x_k-x_j}$$

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，有误差估式

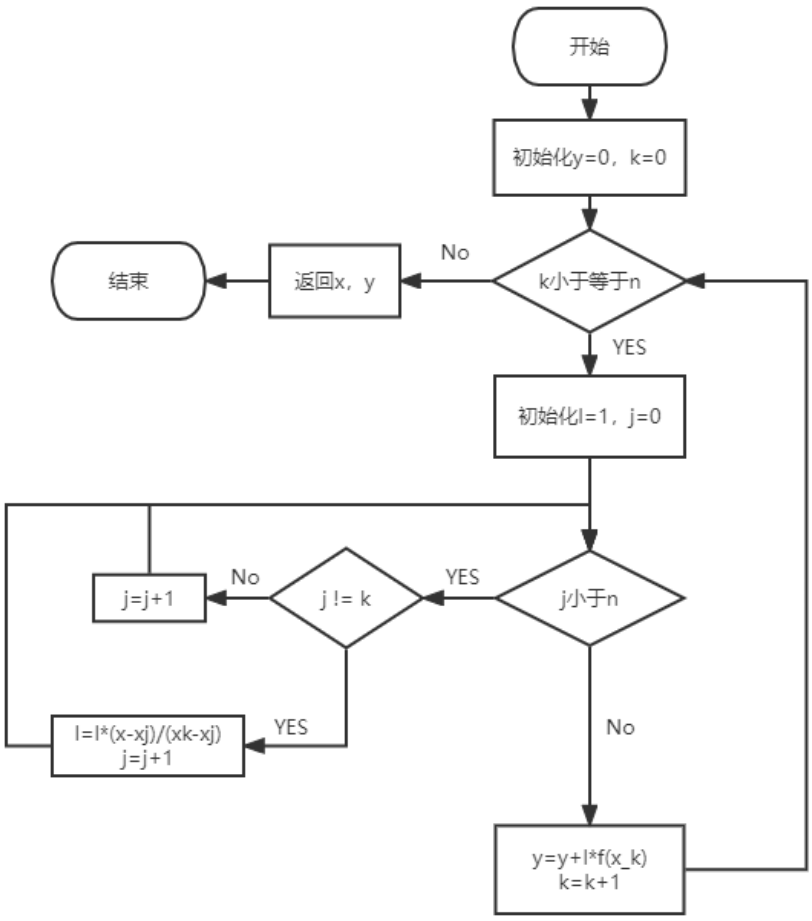
$$E(x) = f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x - x_i), \xi \in [a, b]$$

第三部分：程序设计流程

主要代码：

```
def Lagrange(function, sample_points, data_points):
    n = sample_points.size
    function_values = []
    y = 0
    for i in range(len(data_points)):
        for k in range(n):
            l = 1
            for j in range(n):
                if j != k:
                    l *= ((data_points[i] - sample_points[j]) / (sample_points[k] - sample_points[j]))
            y += l * function(sample_points[k])
        function_values.append((data_points[i], y))
        y = 0
    return function_values
```

设计流程图：



第四部分：实验结果、结论与讨论

1.实验结果：

Question1_1

When n = 5, the result is

(0.75, 0.528974) (1.75, 0.373325) (2.75, 0.153733) (3.75, -0.025954) (4.75, -0.015738)

When n = 10, the result is

(0.75, 0.678990) (1.75, 0.190580) (2.75, 0.215592) (3.75, -0.231462) (4.75, 1.923631)

When n = 20, the result is

(0.75, 0.636755) (1.75, 0.238446) (2.75, 0.080660) (3.75, -0.447052) (4.75, -39.952449)

Question1_2

When n = 5, the result is

(-0.95, 0.386798) (-0.05, 0.951248) (0.05, 1.051290) (0.95, 2.585785)

When n = 10, the result is

(-0.95, 0.386741) (-0.05, 0.951229) (0.05, 1.051271) (0.95, 2.585710)

When n = 20, the result is

(-0.95, 0.386741) (-0.05, 0.951229) (0.05, 1.051271) (0.95, 2.585710)

Question2_1

When n = 5, the result is

(-0.95, 0.517147) (-0.05, 0.992791) (0.05, 0.992791) (0.95, 0.517147)

When n = 10, the result is

(-0.95, 0.526408) (-0.05, 0.997507) (0.05, 0.997507) (0.95, 0.526408)

When n = 20, the result is

(-0.95, 0.525620) (-0.05, 0.997506) (0.05, 0.997506) (0.95, 0.525620)

Question2_2

When n = 5, the result is

(-4.75, 1.147035) (-0.25, 1.302152) (0.25, 1.841210) (4.75,119.621007)

When n = 10, the result is

(-4.75, -0.001957) (-0.25, 0.778686) (0.25, 1.284144)

(4.75, 115.607360)

When n = 20, the result is

(-4.75, 0.008652) (-0.25, 0.778801) (0.25, 1.284025)

(4.75, 115.584285)

Question4_1

The result is

(5.00, 2.266667) (50.00, -20.233333) (115.00, -171.900000)

(185.00, -492.733333)

Question4_2

The result is

(5.00, 3.115751) (50.00, 7.071795) (115.00, 10.167033)

(185.00, 10.038828)

Question4_3

The result is

(5.00, 4.439112) (50.00, 7.284961) (115.00, 10.722756)

(185.00, 13.535667)

Question4_4

The result is

(5.00, 5.497172) (50.00, 7.800128) (115.00, 10.800493)

(185.00, 13.600620)

2.结论：

*Lagrange*插值方法，可以利用给定平面上的$n+1$个不同数据点$\left(x_k, f\left(x_k\right)\right)$，构建出*Lagrange*插值函数，并根据此插值函数计算一些定点的拟合函数值。通过与函数在这些定点的准确值y^*比较，发现*Lagrange*插值函数拟合效果较好。但会随着插值点数的增多出现龙格现象，从而在某些点处产生较大误差，且插值区间的选定对于插值的效果也有一定影响，可以通过不断修改区间，得到一个较好的拟合

效果。

3.讨论：

思考题 1

插值点增多，会导致插值多项式的次数变高，当次数过高时，在插值区间的边缘，插值函数会出现非常明显的波动，变得非常不确定，这种现象称为龙格现象。将构造拉格朗日插值函数时的等距节点转化为切比雪夫零点，可以有效解决该问题

思考题 2

插值区间不是越小越好，虽然通过对比 1.（1）和 2.（1）以及 1.（2）和 2.（2）可以得到较小的区间的相对误差都较小，但是当插值区间过小时，虽然可以在局部得到比较好的拟合效果，但是泛化能力减弱，在区间外的点的拟合效果不一定提升，故插值区间并不是越小越好

思考题 4

通过对比 4.（1）（2）（3）（4）中各个样本点的相对误差，可以发现位于插值点的区间内的定点，其相对误差均相对较小，因此可以从一定程度上得出对于拉格朗日插值问题而言，内插法会比外推法更加可靠

实验报告二 龙贝格（Romberg）积分法

第一部分：问题分析 *（描述并总结出实验题目）*

本次实验主要要求实现龙贝格(*Romberg*)积分法的程序编写以及分析迭代过程中求解的数值逐渐收敛等一系列问题。在此次实验中，首先利用编程实现龙贝格积分算法，一般而言我们可以利用复化梯形求积公式、复化辛普森求积公式、复化科特斯求积公式的误差估计式来计算积分$\int_a^b f(x)dx$，然而龙贝格算法可以进一步通过不断二分和外延提高算法的阶，以获取精度更高的数值解。通常而言，龙贝格算法会输出一个$T-$数表来记录迭代过程，并根据所给精度范围，返回符合条件的数值解。此次实验，一方面锻炼了同学的代码编写和程序设计能力，另一方面通过编写代码和分析结果，使同学们更进一步理解了龙贝格(*Romberg*)积分算法相关理论和计算步骤，提高了问题分析思考能力。

第二部分：数学原理

利用龙贝格(*Romberg*)积分算法求解$\int_a^b f(x)dx$的数值解，首先定义$h=\frac{b-a}{n}$，$x_k=a+k\cdot h, k=0,1,\dots,n$，其中龙贝格算法如下

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

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

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

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

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

利用龙贝格积分算法得到的$T-$数表记录迭代过程，在此次实验中

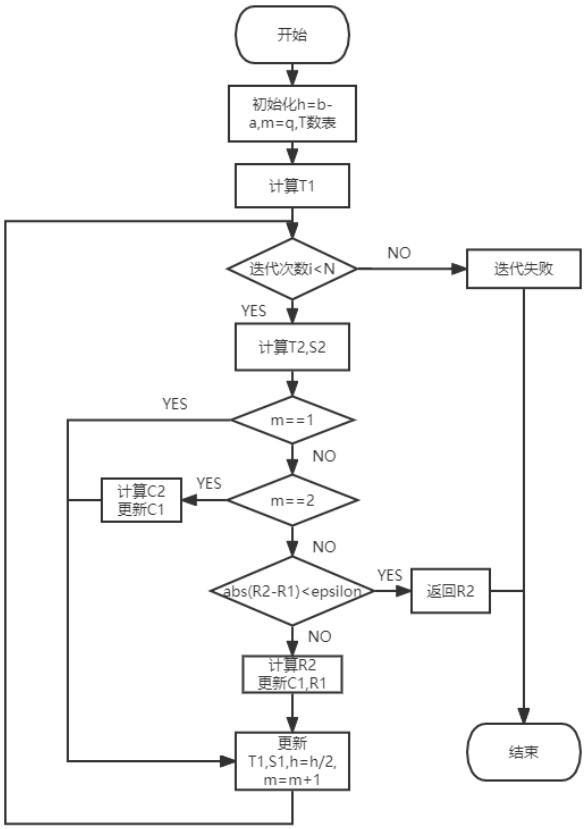
当满足小于最大迭代次数的条件下，有$\left|T_{i+1,0}-T_{i,0}\right|<\varepsilon$即满足条件，返回$T_{i+1,0}$作为积分算法的近似数值解

第三部分：程序设计流程

主要代码：

```
def Romberg(a, b, N, epsilon, f):
    h = b - a
    T = np.zeros((N, N))
    T[0, 0] = 1/2 * h * (f(a) + f(b))
    for i in range(1, N):
        T[0, i] = 1/2*T[0, i-1] + h/2*np.sum(f(a + np.arange(1, 2**i+1, 2)*h/2))
        for j in range(1, i + 1):
            T[j, i - j] = (4**j*T[j-1, i-j+1] - T[j-1, i-j])/(4**j - 1)
        if abs(T[i, 0] - T[i - 1, 0]) < epsilon:
            return T
        h /= 2
    return False
```

设计流程图：



第四部分：实验结果、结论与讨论

1.实验结果：

Question1_1

the result is 0.71828183

Question1_2

the result is 10.95017031

Question1_3

the result is 3.14159265

Question1_4

the result is 0.69314718

2.结论：

此次实验使用龙贝格(*Romberg*)积分法来计算积分$\int_a^b f(x)dx$的数值解，通过与解析解进行比较，发现二者近似程度较高，并且数值解精度会随迭代次数以及二分次数的增加会得到进一步提高，但算法的时间代价和空间代价也会增加，因此在利用龙贝格(*Romberg*)积分法计算时，需要根据可接受误差限度、时间空间复杂度可控程度范围来确定迭代次数

3.讨论：

思考题2

通过分析实验1的计算结果，可以得出当二分次数增加时，其数值解往往会更加贴近解析解，原因是当二分次数增加时，龙贝格积分算法可以利用到更多函数上的点，所以拟合程度也会因此增加，故其数值解的精度也会因此增高，但相应地时间代价和空间代价也会因此增加

实验报告三 四阶龙格—库塔(Runge—Kutta)方法

第一部分：问题分析 *（描述并总结出实验题目）*

本次实验主要要求实现龙格-库塔(*Runge − Kutta*)方法的程序编写以及分析比较不同情况下如不同区间和步长，对于数值解和解析解精确程度的影响等一系列问题。在此次实验中，首先通过编程实现常微分方程四阶龙格-库塔方法，通过求解给定常微分方程的数值解，分析比较不同步长对于数值解与解析解近似程度的影响，加深对于龙格-库塔方法求解过程的进一步理解，并探究和思考龙格-库塔方法求解常微分方程数值解过程中出现的各种问题和解决方案。此次实验，一方面锻炼了同学的代码编写和程序设计能力，另一方面通过编写代码和分析结果，使同学们更进一步理解了龙格-库塔方法相关理论和计算步骤，提高了问题分析思考能力。

第二部分：数学原理

给定常微分方程初值问题，求解其在等距节点的数值解

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

记做$x_n=a+n\cdot h, n=0,1,\cdots,N$，利用四阶龙格-库塔(*Runge − Kutta*)方法

$$\begin{aligned}K_1&=hf(x_n,y_n)\\ K_2&=hf\left(x_n+\frac{h}{2},y_n+\frac{K_1}{2}\right)\\ K_3&=hf\left(x_n+\frac{h}{2},y_n+\frac{K_2}{2}\right)\\ K_4&=hf(x_n+h,y_n+K_3)\\ y_{n+1}&=y_n+\frac{1}{6}(K_1+2K_2+2K_3+K_4)\end{aligned}$$

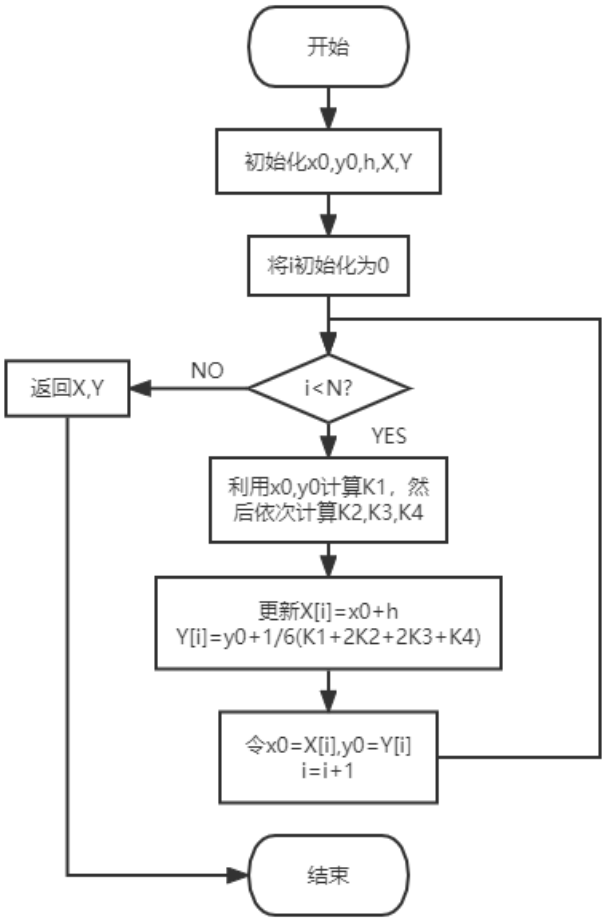
其中$n = 0,1,\cdots,N-1$，可逐次求出微分方程初值问题在等距节点的数值解$y_n(n = 1,2,\cdots,N)$

第三部分：程序设计流程

主要代码：

```
def RungeKutta(a, b, alpha, N, function):
    x0 = a
    y0 = alpha
    h = (b - a) / N
    X = np.zeros(N)
    Y = np.zeros(N)
    for i in range(N):
        K1 = h * function.subs({x: x0, y: y0})
        K2 = h * function.subs({x: x0+h/2, y: y0+1/2*K1})
        K3 = h * function.subs({x: x0+h/2, y: y0+1/2*K2})
        K4 = h * function.subs({x: x0+h, y: y0+K3})
        X[i] = x0 + h
        Y[i] = y0 + 1/6*(K1 + 2*K2 + 2*K3 + K4)
        x0 = X[i]
        y0 = Y[i]
    return X, Y
```

设计流程图：



第四部分：实验结果、结论与讨论

1.实验结果：

Question1_1

When N = 5, the result is
(0.20, -1.2000000000) (0.40, -1.4000000000) (0.60, -1.6000000000)
(0.80, -1.8000000000) (1.00, -2.0000000000)
When N = 10, the result is
(0.10, -1.1000000000) (0.20, -1.2000000000) (0.30, -1.3000000000)
(0.40, -1.4000000000) (0.50, -1.5000000000) (0.60, -1.6000000000)
(0.70, -1.7000000000) (0.80, -1.8000000000) (0.90, -1.9000000000)
(1.00, -2.0000000000)
When N = 20, the result is
(0.05, -1.0500000000) (0.10, -1.1000000000) (0.15, -1.1500000000)
(0.20, -1.2000000000) (0.25, -1.2500000000) (0.30, -1.3000000000)
(0.35, -1.3500000000) (0.40, -1.4000000000) (0.45, -1.4500000000)
(0.50, -1.5000000000) (0.55, -1.5500000000) (0.60, -1.6000000000)
(0.65, -1.6500000000) (0.70, -1.7000000000) (0.75, -1.7500000000)
(0.80, -1.8000000000) (0.85, -1.8500000000) (0.90, -1.9000000000)
(0.95, -1.9500000000) (1.00, -2.0000000000)

Question1_2

When N = 5, the result is
(0.20, 0.8333390356) (0.40, 0.7142921305) (0.60, 0.6250058936)
(0.80, 0.5555606879) (1.00, 0.5000044062)
When N = 10, the result is
(0.10, 0.9090911863) (0.20, 0.8333337288) (0.30, 0.7692312058)
(0.40, 0.7142861539) (0.50, 0.6666670911) (0.60, 0.6250004009)
(0.70, 0.5882356686) (0.80, 0.5555559032) (0.90, 0.5263161113)
(1.00, 0.5000002976)
When N = 20, the result is
(0.05, 0.9523809630) (0.10, 0.9090909268) (0.15, 0.8695652397)
(0.20, 0.8333333586) (0.25, 0.8000000269) (0.30, 0.7692307971)
(0.35, 0.7407407689) (0.40, 0.7142857423) (0.45, 0.6896552000)
(0.50, 0.6666666937) (0.55, 0.6451613166) (0.60, 0.6250000255)
(0.65, 0.6060606307) (0.70, 0.5882353179) (0.75, 0.5714285944)
(0.80, 0.5555555776) (0.85, 0.5405405618) (0.90, 0.5263158099)
(0.95, 0.5128205325) (1.00, 0.5000000189)

Question2_1

When N = 5, the result is
(1.40, 2.6139427925) (1.80, 10.7763131664) (2.20, 30.4916542038)
(2.60, 72.5855986060) (3.00, 156.2251982758)
When N = 10, the result is

(1.20, 0.8663791120) (1.40, 2.6197405205) (1.60, 5.7198952795)
(1.80, 10.7920175975) (2.00, 18.6808523645) (2.20, 30.5215981354)
(2.40, 47.8323658327) (2.60, 72.6345035377) (2.80, 107.6088519912)
(3.00, 156.2982574429)
When N = 20, the result is
(1.10, 0.3459102873) (1.20, 0.8666216927) (1.30, 1.6071813477)
(1.40, 2.6203113059) (1.50, 3.9676018980) (1.60, 5.7208793242)
(1.70, 7.9637717926) (1.80, 10.7935017836) (1.90, 14.3229357276)
(2.00, 18.6829265677) (2.10, 24.0249894197) (2.20, 30.5243558898)
(2.30, 38.3834586600) (2.40, 47.8359047809) (2.50, 59.1510038275)
(2.60, 72.6389257808) (2.70, 88.6565733309) (2.80, 107.6142643893)
(2.90, 129.9833331157) (3.00, 156.3047718808)

Question2_2

When N = 5, the result is
(1.40, -1.5539889981) (1.80, -1.3836172899) (2.20, -1.2934015269)
(2.60, -1.2375401579) (3.00, -1.1995479585)
When N = 10, the result is
(1.20, -1.7142451805) (1.40, -1.5555228848) (1.60, -1.4545197492)
(1.80, -1.3845945063) (2.00, -1.3333158561) (2.20, -1.2941026606)
(2.40, -1.2631447989) (2.60, -1.2380836212) (2.80, -1.2173808733)
(3.00, -1.1999905397)
When N = 20, the result is
(1.10, -1.8333328294) (1.20, -1.7142851698) (1.30, -1.6249995002)
(1.40, -1.5555551111) (1.50, -1.4999996057) (1.60, -1.4545451028)
(1.70, -1.4166663505) (1.80, -1.3846150982) (1.90, -1.3571425958)
(2.00, -1.3333330933) (2.10, -1.3124997783) (2.20, -1.2941174411)
(2.30, -1.2777775857) (2.40, -1.2631577147) (2.50, -1.2499998307)
(2.60, -1.2380950784) (2.70, -1.2272725761) (2.80, -1.2173911609)
(2.90, -1.2083331969) (3.00, -1.1999998699)

Question3_1

When N = 5, the result is
(0.20, 1.7600000000) (0.40, 8.8133333333) (0.60, 43.6800000000)
(0.80, 217.2933333333) (1.00, 1084.3200000000)
When N = 10, the result is
(0.10, 0.1227777778) (0.20, 0.0792592593) (0.30, 0.1047530864)
(0.40, 0.1665843621) (0.50, 0.2538614540) (0.60, 0.3629538180)
(0.70, 0.4926512727) (0.80, 0.6425504242) (0.90, 0.8125168081)
(1.00, 1.0025056027)
When N = 20, the result is
(0.05, 0.1275520833) (0.10, 0.0569466146) (0.15, 0.0401570638)
(0.20, 0.0466734823) (0.25, 0.0650546392) (0.30, 0.0910100730)
(0.35, 0.1229308607) (0.40, 0.1602136561) (0.45, 0.2026322044)

(0.50, 0.2501016600) (0.55, 0.3025902058) (0.60, 0.3600859105)
(0.65, 0.4225842998) (0.70, 0.4900836957) (0.75, 0.5625834692)
(0.80, 0.6400833843) (0.85, 0.7225833524) (0.90, 0.8100833405)
(0.95, 0.9025833360) (1.00, 1.0000833343)

Question3_2

When N = 5, the result is
(0.20, 5.1973381062) (0.40, 25.3761707044) (0.60, 125.4868152611)
(0.80, 625.3120955171) (1.00, 3123.7951509472)
When N = 10, the result is
(0.10, 0.4331389965) (0.20, 0.3096604680) (0.30, 0.3323246667)
(0.40, 0.4014139713) (0.50, 0.4830743415) (0.60, 0.5654352797)
(0.70, 0.6439890045) (0.80, 0.7167223471) (0.90, 0.7824991512)
(1.00, 0.8405257206)
When N = 20, the result is
(0.05, 0.4249785186) (0.10, 0.2404562221) (0.15, 0.2021684390)
(0.20, 0.2184386634) (0.25, 0.2548116511) (0.30, 0.2982910222)
(0.35, 0.3439285511) (0.40, 0.3897953364) (0.45, 0.4350961733)
(0.50, 0.4794626229) (0.55, 0.5226880879) (0.60, 0.5646286384)
(0.65, 0.6051659863) (0.70, 0.6441937626) (0.75, 0.6816125255)
(0.80, 0.7173280379) (0.85, 0.7512507634) (0.90, 0.7832958132)
(0.95, 0.8133830538) (1.00, 0.8414372689)

Question3_3

When N = 5, the result is
(0.20, 0.2986462128) (0.40, 0.9272198700) (0.60, 2.8354773389)
(0.80, 10.7108853309) (1.00, 47.9414463816)
When N = 10, the result is
(0.10, 0.1120551091) (0.20, 0.2451165144) (0.30, 0.4017780967)
(0.40, 0.5840969566) (0.50, 0.7938220530) (0.60, 1.0324183053)
(0.70, 1.3010149884) (0.80, 1.6003210120) (0.90, 1.9305210338)
(1.00, 2.2911569231)
When N = 20, the result is
(0.05, 0.0525950400) (0.10, 0.1104089863) (0.15, 0.1737093905)
(0.20, 0.2427490009) (0.25, 0.3177716916) (0.30, 0.3990135525)
(0.35, 0.4867020696) (0.40, 0.5810544891) (0.45, 0.6822757725)
(0.50, 0.7905562930) (0.55, 0.9060693250) (0.60, 1.0289683441)
(0.65, 1.1593841417) (0.70, 1.2974217528) (0.75, 1.4431571960)
(0.80, 1.5966340222) (0.85, 1.7578596710) (0.90, 1.9268016338)
(0.95, 2.1033834233) (1.00, 2.2874803507)

2.结论：

通过对比利用四阶龙格-库塔(*Runge – Kutta*)方法所求解的等距节点的数值解和

准确解，发现其近似程度相对较高，且当区间中所取点的个数越多，也即步长较小时，其数值解与准确解的逼近程度越大，故可以利用四阶龙格-库塔($Runge-Kutta$)方法来求解带有初值的常微分方程的数值解问题，从而解决一些不需要确定解析解的实际问题

3.讨论：

思考题 1

(1)数值解和解析解大致上相同，且随着给定区间中取点个数的增加，其数值解与解析解的近似程度越大

(2)出现原因：一方面由于计算机存在一定的字长限制，所以会存在一定的舍入误差，在进行数值运算时，又会产生一定的截断误差，故解析解和数值解之间在客观上存在一定差异；另一方面，四阶龙格-库塔($Runge-Kutta$)方法，虽然阶数较高，但是并不能完全拟合原函数，只能在数值上逼近

思考题 2

N越大越精确，由于N越大，其计算的步长越小，在一个较小的区间内可以很好地拟合原函数，从而在进行四阶龙格-库塔($Runge-Kutta$)方法计算时，其误差会相对较小，且更加近似于解析解，故N越大，一般来说所求得的数值解越精确。但N过大时，会导致程序的空间代价和时间代价变得不可接受，所以需要找到一个满足合适精度范围下，且精度相对较高的N

思考题 3

N较小时，会导致数值解会严重偏离解析解。由于N较小，所以在进行四阶龙格-库塔($Runge-Kutta$)方法计算时，所设置的步长较大，而在这个步长范围内，函数取值情况被忽略，导致求解数值解的过程中，不能很好地考虑到函数的具体取值情况，从而使得求解精度下降，同时在越靠近区间右边界时，由于龙格-库塔方法的迭代性质，误差会不断积累，所求得的数值解同解析解之间的误差也会被逐渐放大

实验报告四 牛顿（Newton）迭代法

第一部分：问题分析 *（描述并总结出实验题目）*

本次实验主要要求实现牛顿(*Newton*)迭代法的程序编写以及分析不同迭代格式、迭代初值选择、迭代区间对于非线性方程求解过程中的迭代速度、收敛性判断的影响等一系列问题。在此次实验中，通过给定非线性方程以及一系列参数如迭代精度ξ、最大迭代次数N、迭代初值x_0等，据此利用牛顿迭代格式$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$，通过不断迭代，在未达到迭代最大次数$N$时进行不断迭代，直到迭代过程中得到的$x_k$，满足$|f(x_k)| < \xi_1$，即满足预设条件，将$x_k$作为该非线性方程的近似零点。此次实验，一方面锻炼了同学的代码编写和程序设计能力，另一方面通过编写代码和分析结果，更进一步理解了牛顿(*Newton*)迭代法相关理论，提高了问题分析和思考能力。

第二部分：数学原理

求解非线性方程$f(x) = 0$的根x^*时，利用牛顿(*Newton*)迭代法计算公式

$$x_0 = \alpha$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

一般而言，牛顿迭代法在满足一定条件下会具有局部收敛性，而为保证其迭代格式收敛，对于充分小的$\delta > 0$，$\alpha \in O(x^*, \delta)$，如果$x^*$是$f(x)$的单根，即满足$f(x) \in C^2[a,b], f(x^*) = 0, f'(x^*) \neq 0$，那么牛顿迭代法在根$x^*$的充分小的邻域，即$\alpha \in O(x^*, \delta)$时，由牛顿迭代法计算的$\{x_n\}$收敛于$x^*$，且其收敛速度是 2 阶的；如果$x^*$是$f(x)$的重根，即满足$f(x) \in C^m[a,b],\ f(x^*) = f'(x^*) = \cdots = f^{(m-1)}(x^*) =$

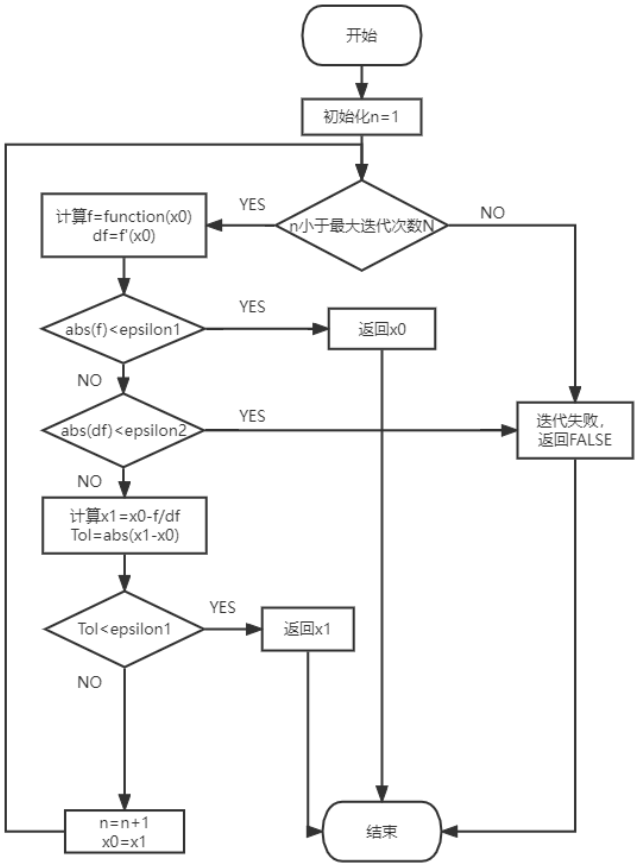
0，$f^{(m)}(x^*) \neq 0 (m > 1)$，那么牛顿迭代法在根$x^*$的充分小的邻域，即$\alpha \in O(x^*, \delta)$时，由牛顿迭代法计算的$\{x_n\}$收敛于$x^*$，且其收敛速度是 1 阶的

第三部分：程序设计流程

主要代码：

```
def Newton(function, x0, epsilon1, epsilon2, N):
    n = 1
    x = sp.Symbol("x")
    while n <= N:
        F = function(x0)
        DF = sp.diff(function(x), x).evalf(subs={x: x0})
        if abs(F) < epsilon1:
            return x0
        if abs(DF) < epsilon2:
            return False
        x1 = x0 - F / DF
        Tol = abs(x1 - x0)
        if Tol < epsilon1:
            return x1
        n = n + 1
        x0 = x1
    return False
```

设计流程图：



第四部分：实验结果、结论与讨论

1.实验结果：

Question1_1

the approximate root is 0.739085

Question1_2

the approximate root is 0.588533

Question2_1

the approximate root is 0.567143

Question2_2

the approximate root is 0.566606

2.结论：

在本次实验中，利用牛顿迭代格式$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$来求解非线性方程的零点问题，通过编写程序得到的各个方程的零点均与给定初值较为接近，从一定程度上说明了一般来说，牛顿迭代法在零点附近具有局部收敛性，但是通过比较不同迭代格式的迭代速度，可以进一步得出不同的迭代格式可能得到在满足既定条件下的不同解，也可能具有不同的迭代速度甚至不同的敛散性

3.讨论：

思考题 1

确定初值的原则：满足函数在该初值的一阶导数绝对值小于 1

实际计算解决方案：选取的初值x_0满足，$x_0 \in [a,b]$（$[a,b]$为有根区间），同时使得$f''(x_0)f(x_0) > 0$

思考题 2

在实验 2 中（1）（2）的方程是等价的，但是所求解的近似解却不同，是因为对于函数的不同的迭代函数格式，在满足既定的返回条件下，可能会得到不同的解，不同的迭代速度，甚至不同的敛散性。故在构造迭代函数时，需要考虑到初值的收敛情况，同时通过比较不同的迭代格式，得到一个收敛速度相对较快的迭代函数，加速近似根的求解速度

实验报告五 高斯(Gauss)列主元消去法

第一部分：问题分析 *（描述并总结出实验题目）*

本次实验主要要求实现高斯(*Gauss*)列主元消去法的程序编写以及分析求解过程中出现的一系列问题。在此次实验中，需要求解线性方程组$Ax = b$的解，同时在求解过程中需判断方程组的解的情况（即线性方程组的奇异性），通过高斯(*Gauss*)列主元消去法，每次提取每列绝对值最大的元素，作为列主元对其他行进行消去处理，从一定程度上避免了普通高斯(*Gauss*)消去法，利用较小元素作为主元素而造成舍入误差严重扩散的问题，有利于提高方程组的求解精度。此次实验，一方面锻炼了同学的代码编写和程序设计能力，另一方面通过编写代码和分析结果，更进一步理解了高斯(*Gauss*)列主元消去法相关理论和计算步骤，提高了问题分析和思考能力。

第二部分：数学原理

主元选取：给定线性方程组$Ax = b$，进行$n-1$步消元，并记k为当前消元的次数。在每次消元前，取每列绝对值最大元素作为主元素，即寻找最小的正整数p，$k \le p \le n$和$|a_{pk}| = \max\limits_{k \le j \le n} |a_{jk}|$，如果$p \ne k$，则交换$p, k$两行

消去过程：

对$a_{ij}^{(0)} = a_{ij}, b_i^{(0)} = b_i$，以及$k = 1,2,\dots,(n-1)$，依次计算如下式子

$$\begin{cases} m_{ik} = a_{ik}^{(k-1)} / a_{kk}^{(k-1)} \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - m_{ik} a_{kj}^{(k-1)} \\ b_i^{(k)} = b_i^{(k-1)} - m_{ik} b_k^{(k-1)} \end{cases}$$

其中$i, j = k+1, k+2, \cdots, n$

回代过程：

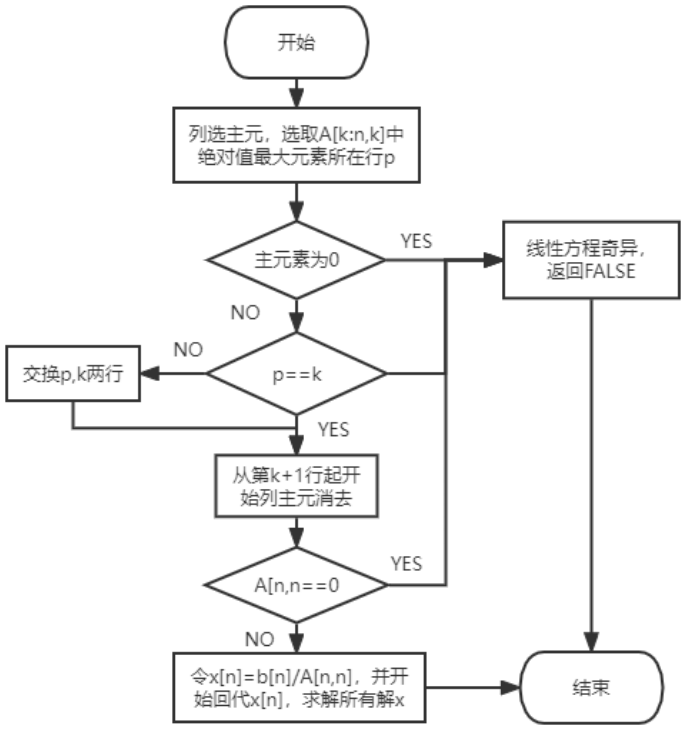
$$x_i = \left(b_i^{(i-1)} - \sum_{j=i+1}^{n} a_{ij}^{(i-1)} x_j \right) / a_{ii}^{(i-1)} (i = n, n-1, \cdots, 1)$$

第三部分：程序设计流程

主要代码：

```
def Gauss(A, b):
    n = b.size
    m = np.zeros((n, n))
    x = np.zeros(n)
    for k in range(n):
        item = np.max(abs(A[k:n, k]))
        p = np.argmax(abs(A[k:n, k])) + k
        if item == 0:
            return False
        if p != k:
            A[[k, p], :] = A[[p, k], :]
            b[[k, p]] = b[[p, k]]
        for i in range(k + 1, n):
            m[i, k] = A[i, k] / A[k, k]
        for i in range(k + 1, n):
            b[i] -= b[k] * m[i, k]
            for j in range(k, n):
                A[i, j] -= A[k, j] * m[i, k]
    if A[n - 1, n - 1] == 0:
        return False
    x[n - 1] = b[n - 1] / A[n - 1, n - 1]
    for k in range(n - 2, -1, -1):
        x[k] = b[k]
        for j in range(k + 1, n):
            x[k] -= A[k, j] * x[j]
        x[k] /= A[k, k]
    return x
```

设计流程图：



第四部分：实验结果、结论与讨论

1.实验结果：

Question1_1

the result is [1.000000000000003 1.000000000000002 0.999999999999997 0.999999999999999]

Question1_2

the result is [1.000000000000118 0.999999999999824 0.999999999999901 1.000000000000026]

Question1_3

the result is [0.999999999999993 1.000000000000069 0.999999999999855 1.000000000000085]

Question1_4

the result is [1.000000000000000 1.000000000000000 1.000000000000000 1.000000000000000]

Question2_1

the result is [0.953679106901772 0.320956845521104 1.078708075793238 -0.090108509539579]

Question2_2

the result is [0.516177297958542 0.415219472830135 0.109966102867889 1.036539223336201]

Question2_3

the result is [1.000000000000000 1.000000000000000 1.000000000000000]

Question2_4

the result is [1.000000000000000 1.000000000000000 1.000000000000000]

2. 结论：

利用高斯(*Gauss*)列主元消去法求解线性方程组$Ax = b$的解，同时在求解过程中也可以判断线性方程组$Ax = b$的奇异性。利用列选主元的高斯消去法，所得到的的线性方程组的解同精确解较为接近，一般情况下不会出现因主元素过小而导致的舍入误差严重扩散、不可控而带来的精度损失问题

3. 讨论：

通过不同的消元方法，一方面让我们了解到不同格式的消去方法的计算步骤，另一方面通过对比不同的消去方法，又可以发现不同消去法的优势以及需要改进的问题，并根据实际情况选择一种空间和时间性能较好、精确度较高的有效的消去方法