****

**“数值分析”课程实验报告**

**姓 名： 乔富康**

**学 号： 235011034**

**专 业： 测绘科学与技术**

**授课教师： 徐宇锋**

**学 期： 2023秋**

目 录

[目 录 I](#_Toc18668)

[1 项目1 插值法 1](#_Toc7321)

[1.1 背景简介 1](#_Toc23866)

[1.2 数学模型 1](#_Toc10832)

[1.3 求解算法(算法名称) 1](#_Toc4179)

[1.3.1 算法过程 1](#_Toc18407)

[1.3.2 相关参数 2](#_Toc12285)

[1.3.3 计算结果 2](#_Toc10292)

[1.4 分析与讨论 3](#_Toc22357)

[2 项目2 函数逼近与离散数据拟合 4](#_Toc18192)

[2.1 背景简介 4](#_Toc29498)

[2.2 数学模型 4](#_Toc25758)

[2.3 求解算法(算法名称) 6](#_Toc14069)

[2.3.1 算法过程 6](#_Toc29656)

[2.3.2 相关参数 6](#_Toc27435)

[2.3.3 计算结果 7](#_Toc24110)

[2.4 分析与讨论 9](#_Toc4449)

[3 项目3 数值积分 10](#_Toc19843)

[3.1 背景简介 10](#_Toc26777)

[3.2 数学模型 10](#_Toc30424)

[3.3 求解算法(算法名称) 11](#_Toc970)

[3.3.1 算法过程 11](#_Toc29523)

[3.3.2 相关参数 11](#_Toc26958)

[3.3.3 计算结果 11](#_Toc25228)

[3.4 分析与讨论 12](#_Toc10866)

[4 项目4 线性方程组的直接解法 13](#_Toc10316)

[4.1 背景简介 13](#_Toc7372)

[4.2 数学模型 13](#_Toc26902)

[4.3 求解算法(算法名称) 14](#_Toc27399)

[4.3.1 算法过程 14](#_Toc17466)

[4.3.2 相关参数 14](#_Toc8511)

[4.3.3 计算结果 14](#_Toc32579)

[4.4 分析与讨论 15](#_Toc14639)

[5 项目5 线性方程组的迭代解法 16](#_Toc13813)

[5.1 背景简介 16](#_Toc3249)

[5.2 数学模型 16](#_Toc31557)

[5.3 求解算法(算法名称) 16](#_Toc2404)

[5.3.1 算法过程 17](#_Toc31460)

[5.3.2 相关参数 17](#_Toc5757)

[5.3.3 计算结果 17](#_Toc26712)

[5.4 分析与讨论 18](#_Toc26114)

[6 项目6 非线性方程的迭代解法 19](#_Toc12224)

[6.1 背景简介 19](#_Toc29958)

[6.2 数学模型 19](#_Toc15782)

[6.3 求解算法(算法名称) 19](#_Toc23198)

[6.3.1 算法过程 19](#_Toc7834)

[6.3.2 相关参数 20](#_Toc17320)

[6.3.3 计算结果 20](#_Toc31848)

[6.4 分析与讨论 20](#_Toc10650)

[7 项目7 矩阵特征值和特征向量 21](#_Toc28192)

[7.1 背景简介 21](#_Toc1263)

[7.2 数学模型 21](#_Toc31055)

[7.3 求解算法(算法名称) 21](#_Toc32758)

[7.3.1 算法过程 21](#_Toc22044)

[7.3.2 相关参数 22](#_Toc5383)

[7.3.3 计算结果 22](#_Toc21794)

[7.4 分析与讨论 23](#_Toc7017)

[8 项目8 常微分方程数值解法 24](#_Toc32061)

[8.1 背景简介 24](#_Toc8308)

[8.2 数学模型 24](#_Toc26792)

[8.3 求解算法(算法名称) 24](#_Toc5809)

[8.3.1 算法过程 24](#_Toc25626)

[8.3.2 相关参数 25](#_Toc29443)

[8.3.3 计算结果 26](#_Toc10184)

[8.4 分析与讨论 29](#_Toc20519)

[9 附录一 主要关键代码 30](#_Toc372)

[9.1 项目1 插值法 30](#_Toc13057)

[9.2 项目2 函数逼近与离散数据拟合 32](#_Toc16367)

[9.3 项目3 数值积分 36](#_Toc7830)

[9.4 项目4 线性方程组的直接解法 37](#_Toc29627)

[9.5 项目5 线性方程组的迭代解法 42](#_Toc23722)

[9.6 项目6 非线性方程的迭代解法 43](#_Toc30130)

[9.7 项目7 矩阵特征值与特征向量 44](#_Toc18509)

[9.8 项目8 常微分方程数值解法 46](#_Toc27604)

[10 附录二 查重报告 49](#_Toc4580)

# 项目1 插值法

## 背景简介

许多实际问题都是用函数来表示某种内在规律的数量关系，其中相当一部分是通过观测得到的。只能给出一系列点的函数值，希望给定一个简单函数使得,这样确定的则为插值函数。比较常用的就是拉格朗日插值函数，首先要进行构造基函数。同时为了插值节点的增减方便又引出了牛顿插值多项式，利用均差表生成。

## 数学模型

得到n次插值基函数为

则由定义可知

均差可表示为

## 求解算法(算法名称)

（1）拉格朗日插值法以及基函数构建

（2）均差表的生成

### 算法过程

（1）构造基函数；再根据拉格朗日插值法计算

（2）根据公式输出均差表

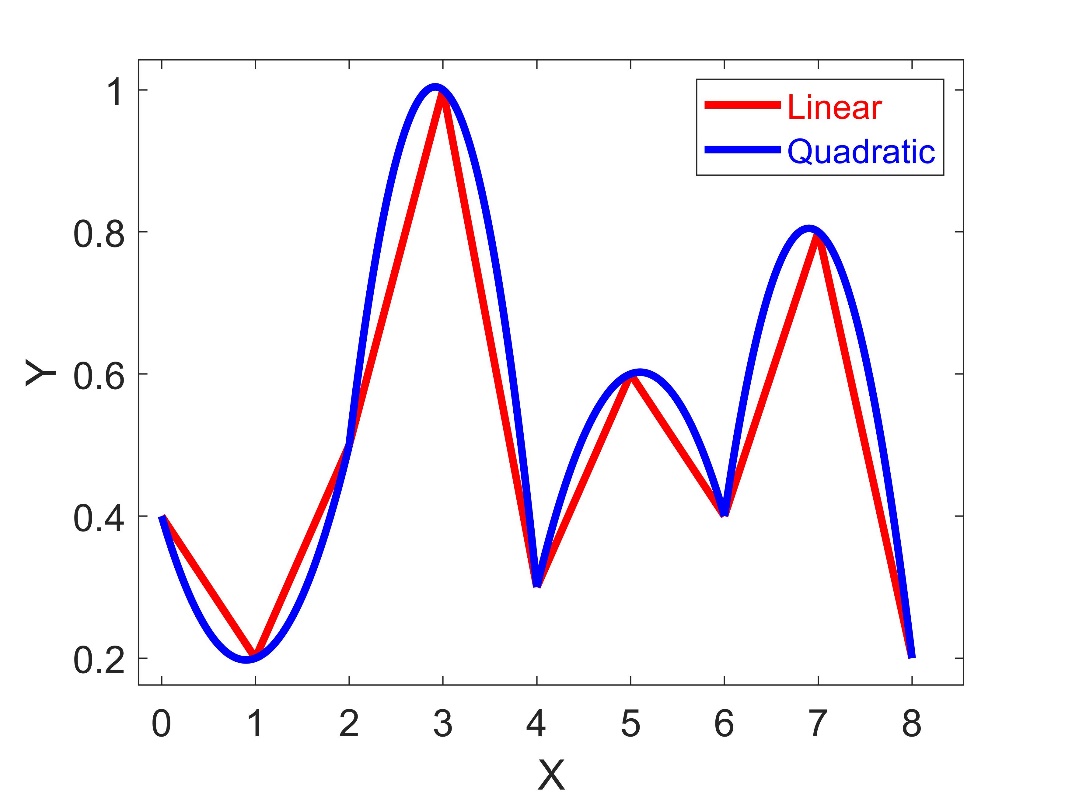
### 相关参数

x\_data、y\_data、x、y示例数据；

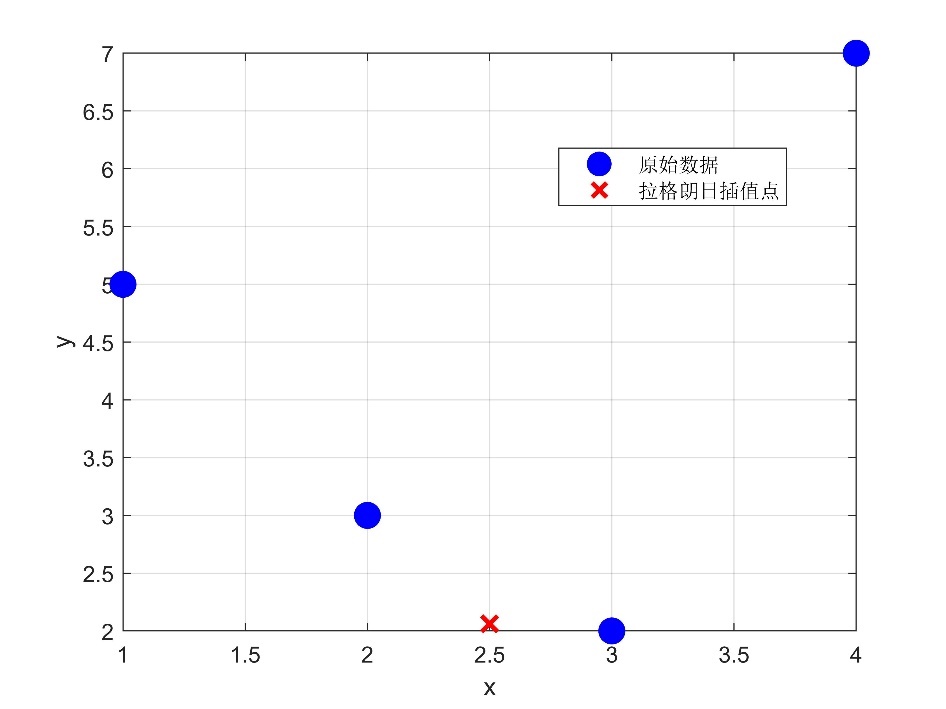
x\_interp 插值结点；

difftable 均差表

### 计算结果

****

**图1 线性、二次插值基函数**

****

**图2 拉格朗日点插值**



**图3 均差表**

## 分析与讨论

拉格朗日插值是利用基函数方法构造的插值多项式，在理论上较为重要但是在计算上不太方便。基函数方法是将插值问题规划为特定条件下容易实现的插值问题。后面针对均差表构造牛顿插值多项式运用较为广泛。

# 项目2 函数逼近与离散数据拟合

## 背景简介

20世纪初龙格（Runge）给出了一个等距结点插值多项式不收敛于

的例子，一般总认为的次数越高逼近的精度越好，由于高次插值出现龙格现象，事实并非如此。但若用切比雪夫多项式零点插值却可避免龙格现象，可保证整个区间上收敛。

在函数的最佳平方逼近中，常常使得误差平方和最小，如此可以最佳的拟合曲线，由一般的最小二乘逼近到加权平方和的最小二乘逼近，仅仅需要知道系数便可以对离散数据拟合。

## 数学模型

切比雪夫多项式在区间上有个零点,和个极值点（包括端点），这两组点称为切比雪夫点，其刚好是单位圆周上等距分布点的横坐标。利用切比雪夫点做插值可使误差最小化。

如果函数只在一组离散点上给出，求一个函数与所给的数据拟合，误差，的平方和最小：

其中

一般地如果考虑权函数，它表示不同点的数据比重不同，，在加权意义下的平方和最小：

考虑如下的多元函数：

由多元函数取极值的必要条件可知

若记

上述式子可以改成其中

则法方程为

要使得法方程有唯一解则需要矩阵非奇异，而前面直接用函数族进行拟合，将导致系数矩阵随着的增大而迅速病态，利用正交多项式将改善这一情况。

此时法方程的解为

按照如下的递推式获得基于离散数据点的正交多项式序列：

这样生成的是首项系数为1的次多项式，利用正交性可求得

用正交多项式的线性组合作最小二乘曲线拟合，只需要根据递推公式求出每一个具体的正交多项式，系数由下式计算

则

## 求解算法(算法名称)

（1）验证龙格现象

（2）曲线拟合的最小二乘法，包括线性、二次多项式和正交多项式

### 算法过程

（1）首先画出区间[-5,5]上的的图像；其次画出利用等距节点拉格朗日插值绘图；最后根据切比雪夫节点进行插值。将三张图进行对比。

（2）首先给出数据x、f并用matlab最小二乘线性拟合绘图；其次进行带权线性拟合，与不带权拟合对比并输出表达式p1；最后进行带权抛物线拟合，与不带权拟合对比并输出表达式p2。

### 相关参数

lagrange\_interpolation表示拉格朗日插值法；

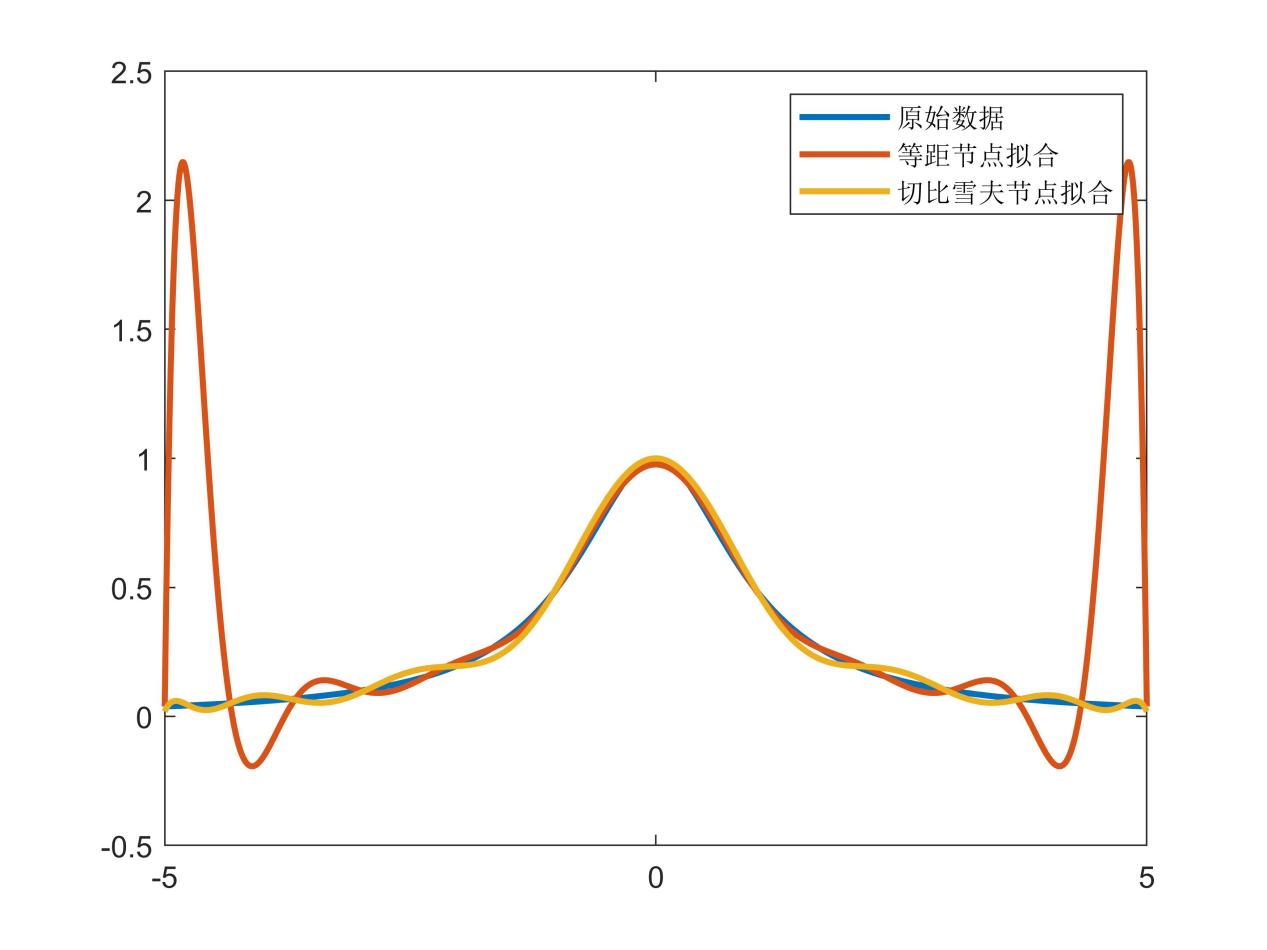
x11为等距插值节点；

x21、x\_chep为切比雪夫插值节点；

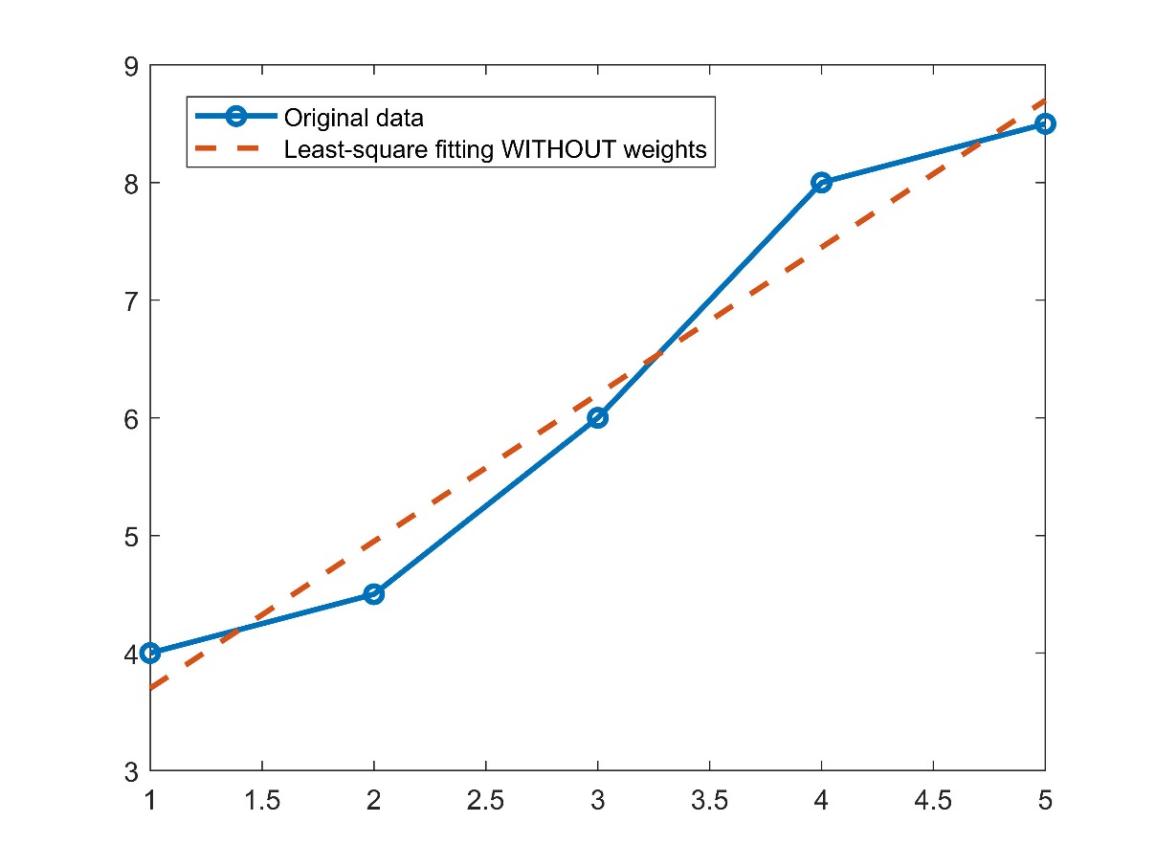
1. B为最小二乘拟合系数公式；

p1、p2为表达式.

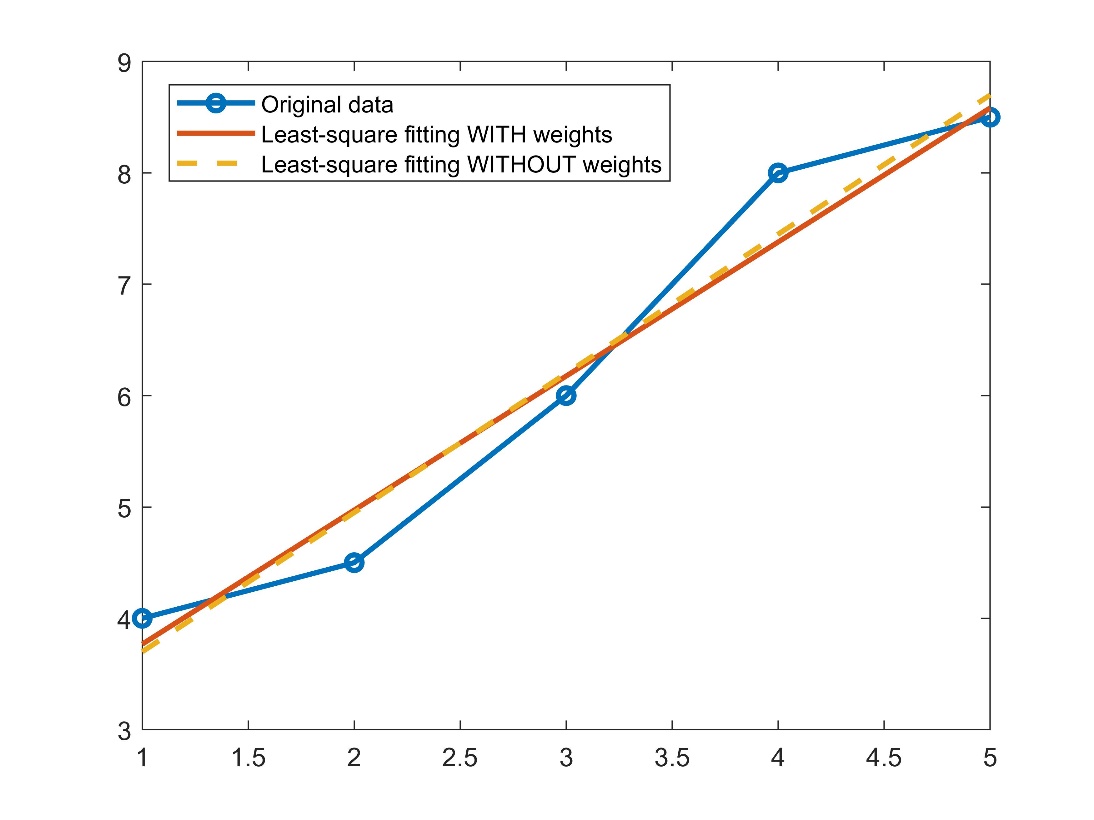
### 计算结果

****

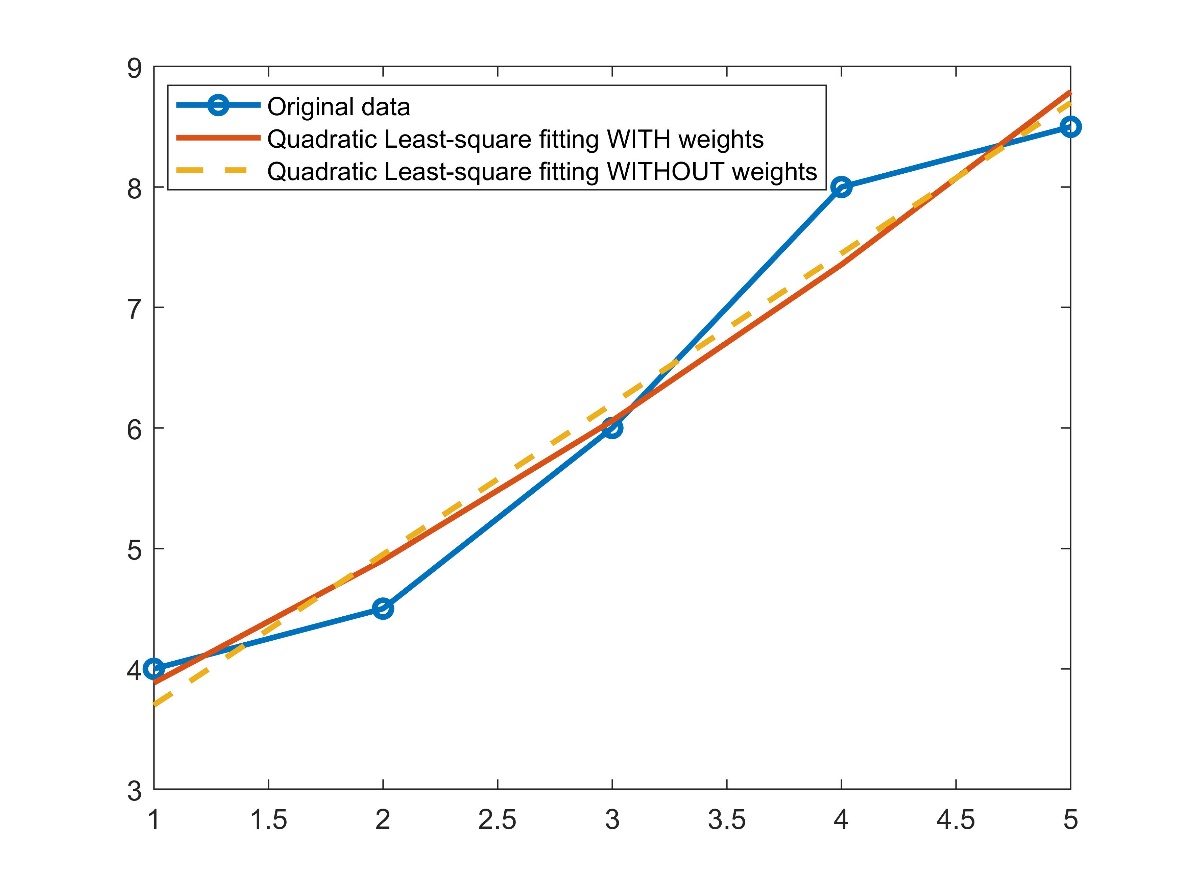
**图1 龙格现象，切比雪夫节点**

****

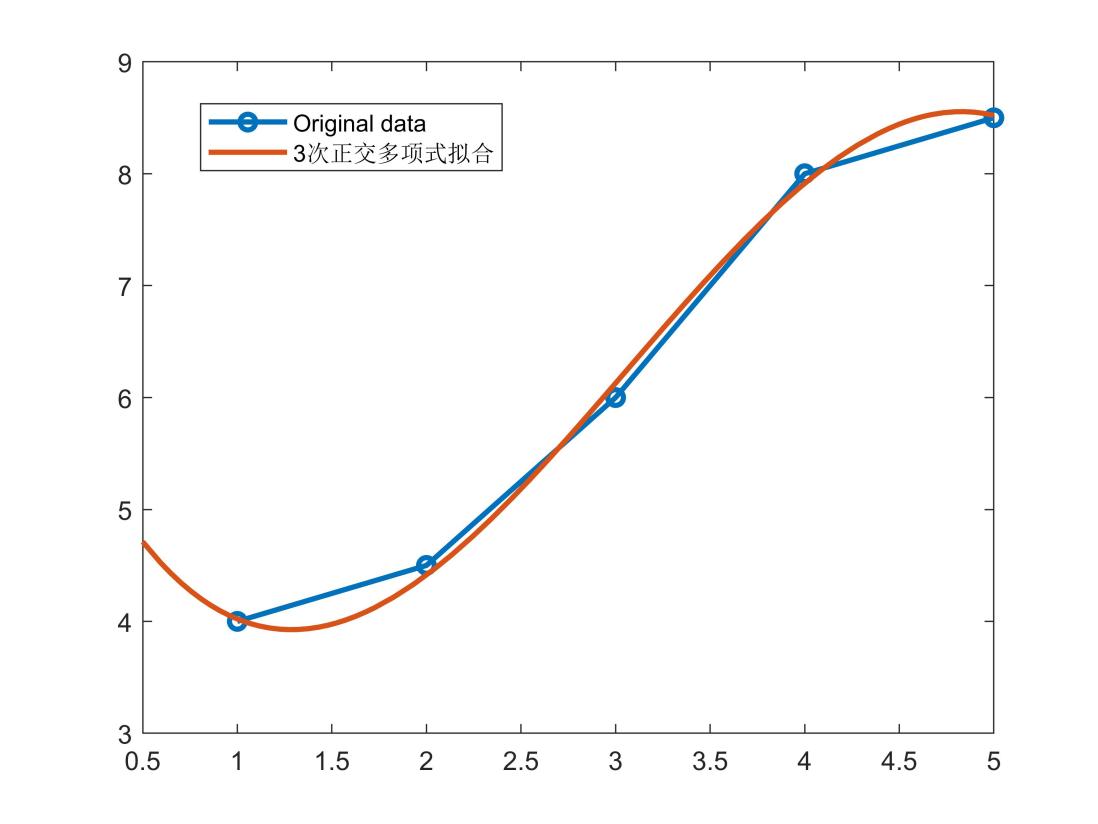
**图2 最小二乘拟合**

****

**图3 最小二乘加权线性拟合对比**

****

**图4 最小二乘加权抛物线多项式拟合对比**

****

**图5 3次正交多项式拟合**

线性拟合表达式p1 =

(277\*x)/108 + 65/54

抛物线拟合表达式p2 =

(13214500859753\*x^2)/4398046511104+ (57011714032821\*x)/70368744177664 + 19546873382691/281474976710656

## 分析与讨论

经过此次实验，对函数逼近和离散值拟合有了更加深刻的了解。实验对比了加权和不加劝的最小二乘拟合，并用线性、二次多项式、正交多项式对示例数据进行拟合，看出随着拟合次数越高，给定了权的数值对应考虑带权内积，最小二乘是给定曲线之后的拟合，因此先确定曲线类型十分重要。

# 项目3 数值积分

## 背景简介

实际问题中经常使用到数值积分，有些数值方法例如微分方程和积分方程的求解也都和微分计算有着密切的关系。对于某些积分，只要找到其原函数，便可以根据牛顿莱布尼兹公式求积，但是在实际过程中，这种求积方法往往有困难，其原函数不能用初等函数表示，因此有必要研究积分的数值计算问题。

## 数学模型

复合梯形公式：

将划分为等分，分点，在每个子区间上采用梯形公式

记

复合辛普森公式

将划分为等分，分点，在每个子区间采用辛普森公式，若记，则

记

## 求解算法(算法名称)

复合梯形求积

复合辛普森求积公式

龙贝格求积外推技巧

### 算法过程

（1）给出被积函数*f*（*x*）、区间[*a*，*b*]端点*a*，*b*和等分数*n；*

（2）根据公式计算，设*n*=6；

（3）画图看三个积分的收敛速度，从1到20。

### 相关参数

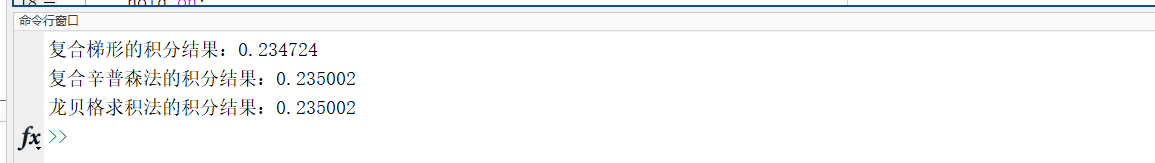
N=6；等分数

compositeTrapezoidal 复合梯形

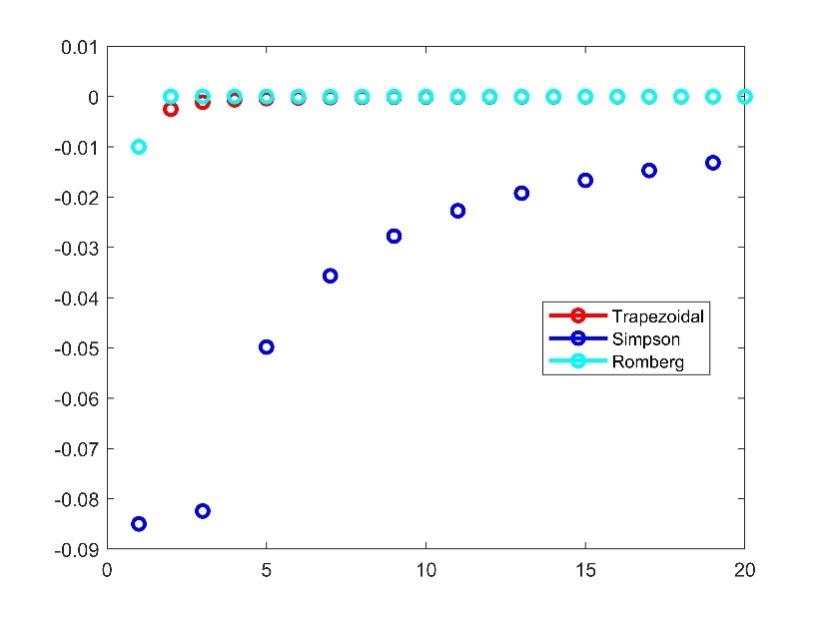
compositeSimpson复合辛普森

rombergIntegration龙贝格求积

### 计算结果



**图1 n=6 积分结果**

****

**图2 三种方法收敛速度**

## 分析与讨论

由公式可得三种积分公式，同时对比三种积分方法的收敛速度和精度。复合辛普森求积公式是一种计算积分近似值的重要方法，而龙贝格求积方法将精度进一步提高，收敛速度也进一步加快。

# 项目4 线性方程组的直接解法

## 背景简介

在自然科学和工程技术中很多问题的解决常常归结为解线性代数方程组，关于线性方程组的数值解法一般有两种：直接法和迭代法。而直接法就是经过有限的算数运算，可求得线性方程组的精确解，主要包括高斯消元法、列主元法、矩阵三角分解法。

## 数学模型

高斯消元法

1. 消元计算
2. 回代计算

矩阵LU分解

计算U的第r行，L的第r列元素

求解

## 求解算法(算法名称)

高斯消元法

LU分解法

### 算法过程

1. 通过消元、回代计算出方程的解

（2） 通过矩阵的三角分解简化计算方程组

### 相关参数

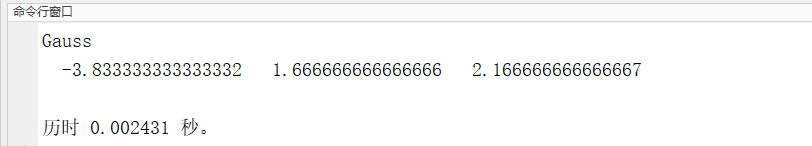
LU\_composition表示矩阵的LU分解

LU表示通过三角分解解方程

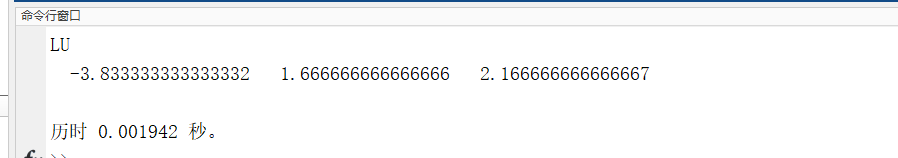
Gauss表示高斯法

Time\_Contrast表示方程解算的时间对比

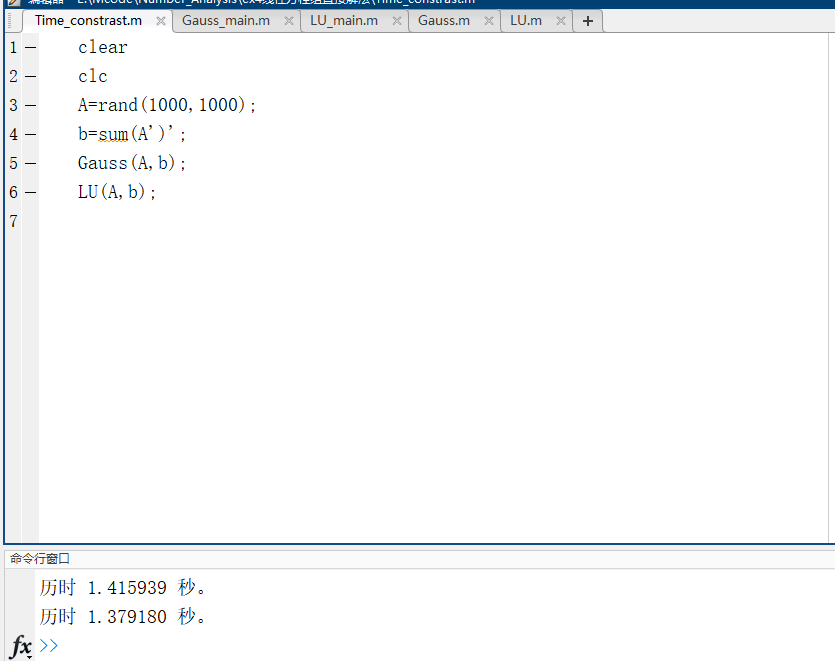
### 计算结果



**图1 高斯消元法求解**



**图2 LU分解法求解**



**图3 时间对比**

## 分析与讨论

解线性方程组的直接解法使用有限步的算术运算即可求得方程组的精确解，且仅受舍入误差的影响，对比发现针对大型矩阵LU分解的计算速度比Gauss分解法快。

# 项目5 线性方程组的迭代解法

## 背景简介

迭代解法通常是利用系数矩阵中含有大量零元素，对于由工程技术中产生的大型稀疏矩阵方程组利用迭代法解是合适的。迭代法就是用某种极限过程去逐步逼近线性方程组精确解的方法，在计算过程中保持原始系数矩阵不变的优点，但存在收敛性和收敛速度的问题。主要包括雅可比迭代法、高斯赛德尔迭代法、超松弛迭代法和共轭梯度法。

## 数学模型

(1)雅可比迭代

系数矩阵 分解为三部分

其中

(2)高斯赛德尔迭代

取分裂矩阵为的下三角部分，即 所以得到

其中为的高斯赛德尔迭代法的迭代矩阵。

## 求解算法(算法名称)

Jacobi迭代

Seidel迭代

### 算法过程

设置初始值[0,0,0]，根据矩阵分解，分别带入公式计算迭代数值，当小于阈值的时候停止，x第一个分量值结算对比速度如下图2所示。

### 相关参数

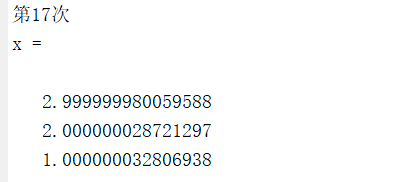
A为系数矩阵

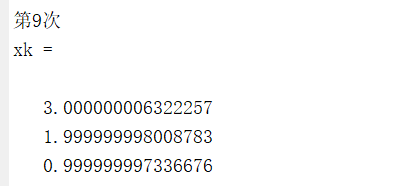
b为常数项

x0为初始值

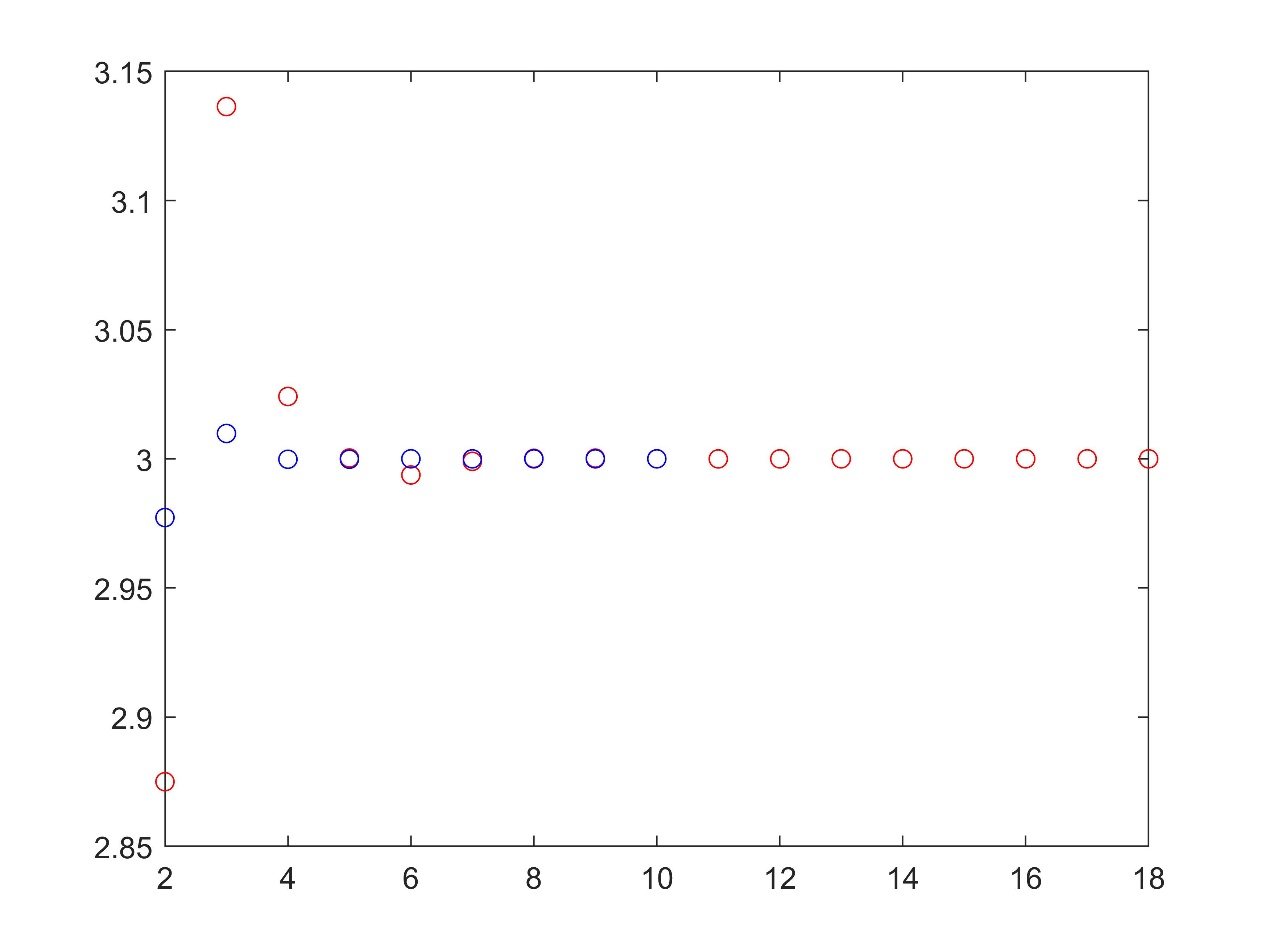
eps为误差阈值

### 计算结果





**图1 方程计算结果**



**图2 迭代速度对比 蓝色为高斯赛德尔迭代 红色为雅可比迭代**

## 分析与讨论

可以发现高斯赛德尔（GS）迭代法收敛速度比雅可比迭代收敛更快，大约5次就达到了精度。迭代法由存储空间小、程序简单的特点，是解决大型稀疏方程组的有效方法。

# 项目6 非线性方程的迭代解法

## 背景简介

非线性是实际问题中经常出现的。很多熟悉的线性模型都是在一定条件下由非线性条件简化得到的，为了得到更加符合实际的解，往往需要研究非线性模型。从线性到非线性是一个质的变化，方程的性质由本质不同，求解方法也有很大差别。本项目主要讨论牛顿法和不同点迭代法解非线性方程的过程。

## 数学模型

不动点迭代法将方程写成等价形式的形式 ，迭代法本质上是一个逐次逼近法，其基本思想是将隐式方程归结为一组显式的计算公式，本质上是一个逐步显式化的过程。

牛顿法实质上是一种线性化的思想，其基本思想将非线性方程逐步归结为某种线性方程求解。

## 求解算法(算法名称)

牛顿法

不动点迭代法

### 算法过程

不动点迭代法：选择初始值；使用迭代公式进行计算迭代；检查迭代序列是否收敛，若收敛则当前的是方程的近似解。

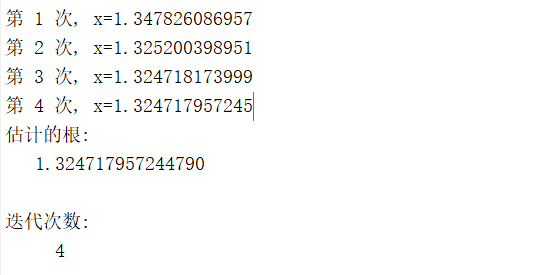
牛顿法迭代：可以近似地表示为，若记其根为，则计算公式为因此其迭代函数为

### 相关参数

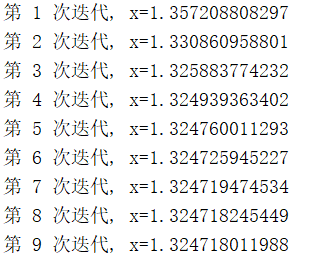
budong: 不动点迭代

newton\_method 牛顿法迭代

### 计算结果



**图1 牛顿法迭代**



**图2 不动点法迭代**

## 分析与讨论

由上述实验可知，牛顿法在迭代法中比较实用，且在单根附近二阶收敛，但是应用时要选取较好的初始近似才能保证迭代收敛。因此为了克服这些缺点可使用牛顿下山法、斯特芬森方法也是值得重视的算法，由于篇幅限制此项目不再赘述。

# 项目7 矩阵特征值和特征向量

## 背景简介

矩阵特征值的作用在数值分析中的作用很大，因此寻找求解矩阵特征值的方法很重要，常见的几种方法：幂法、反幂法及QR算法。前两种是迭代法，只求模最大与模最小的特征值及特征向量，本项目主要针对这个算法编写程序。

## 数学模型

**幂法**

设实矩阵有一个完备的特征向量组，有n个线性无关的特征向量，已知A的主特征值是实根且满足条件，求对应的特征向量。

**反幂法**

用来计算矩阵按模最小的特征值及其特征向量，也可以用来计算对应一个给定的近似特征值的特征向量。

## 求解算法(算法名称)

幂法

反幂法

### 算法过程

幂法

由A构造序列

则有

反幂法

迭代公式为

则有

即

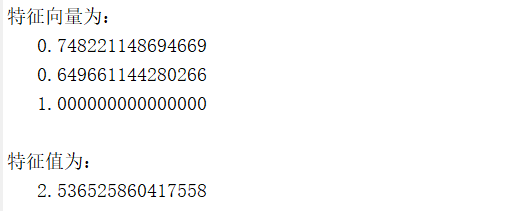
### 相关参数

A B为输入的矩阵

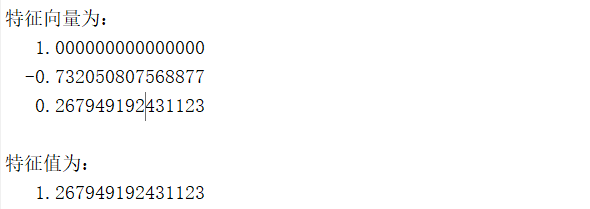
u0为初始值

n为迭代次数

### 计算结果



**图1 幂法结果**



**图2 反幂法结果**

## 分析与讨论

幂法和反幂法均为迭代方法，可以在一定程度上估计矩阵的特征值和特征向量。幂法主要用于估计最大特征值，而反幂法主要用于估计最小特征值。在实际应用中，可以根据具体问题的需求选择适合的方法，或者结合多种方法进行更全面的特征值估计。

# 项目8 常微分方程数值解法

## 背景简介

科学技术中很多问题都可用常微分方程的定解问题来描述，主要有初值问题和边值问题两大类，常微分方程的解对工程应用也有很大影响。主要用欧拉法、改进欧拉法和龙格库塔方法为例。

## 数学模型

常微分方程初值的数学模型是求，使之满足

且满足Lipschitz条件，存在唯一的连续可微解

## 求解算法(算法名称)

欧拉法

改进欧拉法

4阶龙格-库塔法

### 算法过程

欧拉法

记在处的值是精准解，用均差近似代替其导数得到

令为的近似值，将上述两个近似式写成等式，整理后

分别为显式Euler和隐式Euler。

改进欧拉法

采取上述两式子的算数平均，得到初值问题的梯形法。

或

龙格-库塔

Euler可以看作是Taylor展开后的前两项，然而在实际情况求导数是比较麻烦的，龙格-库塔方法的基本思想是用区间上若干点的导数作线性组合得到平均斜率四阶方法公式如下：

### 相关参数

eulers 欧拉法

impeuler 改进欧拉法

rk4m 4阶龙格-库塔方法

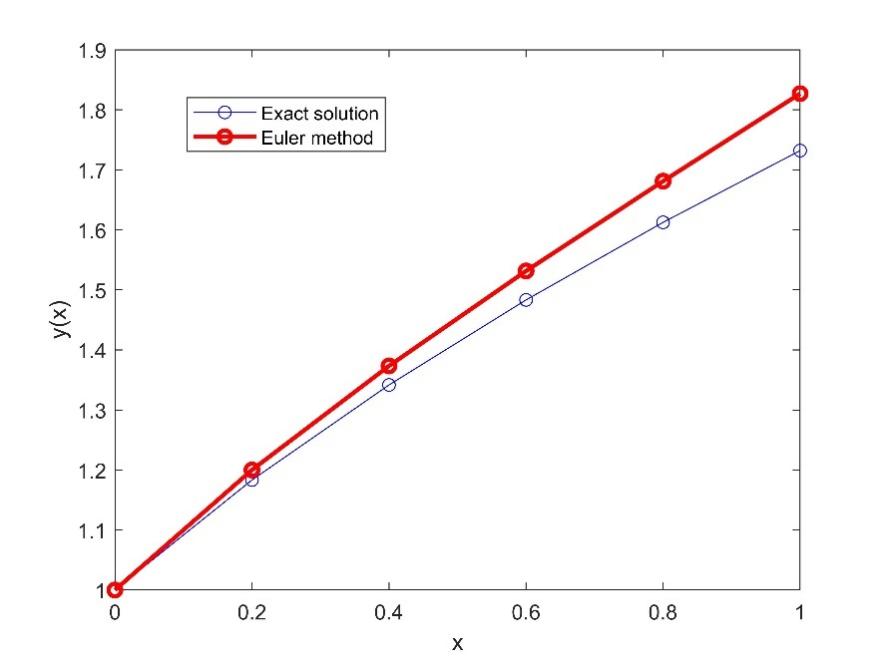
a b范围

h 步长

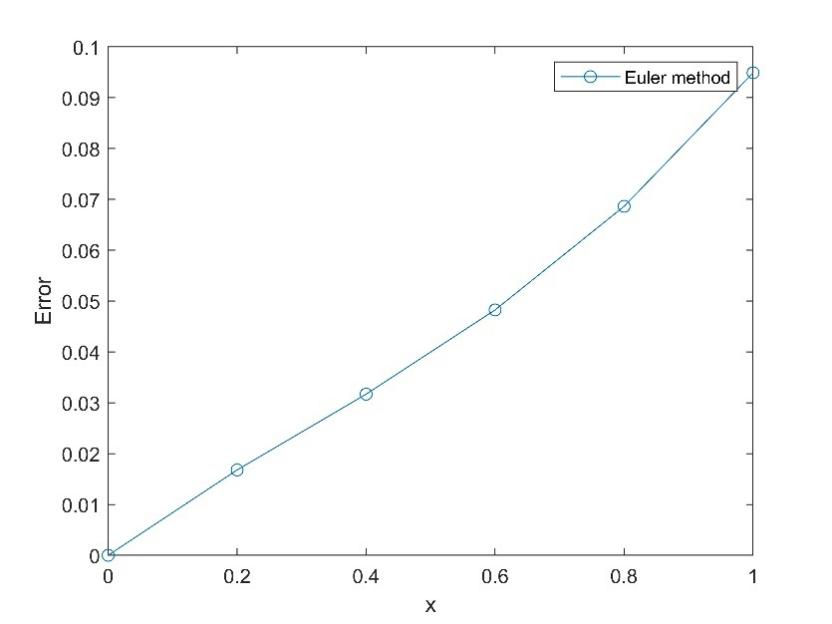
ya 初始值

f为微分方程

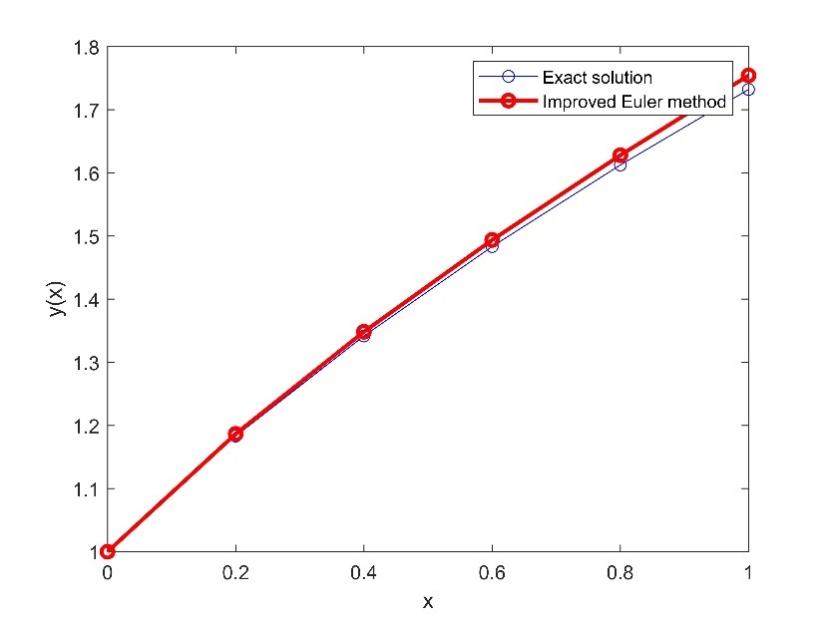
### 计算结果



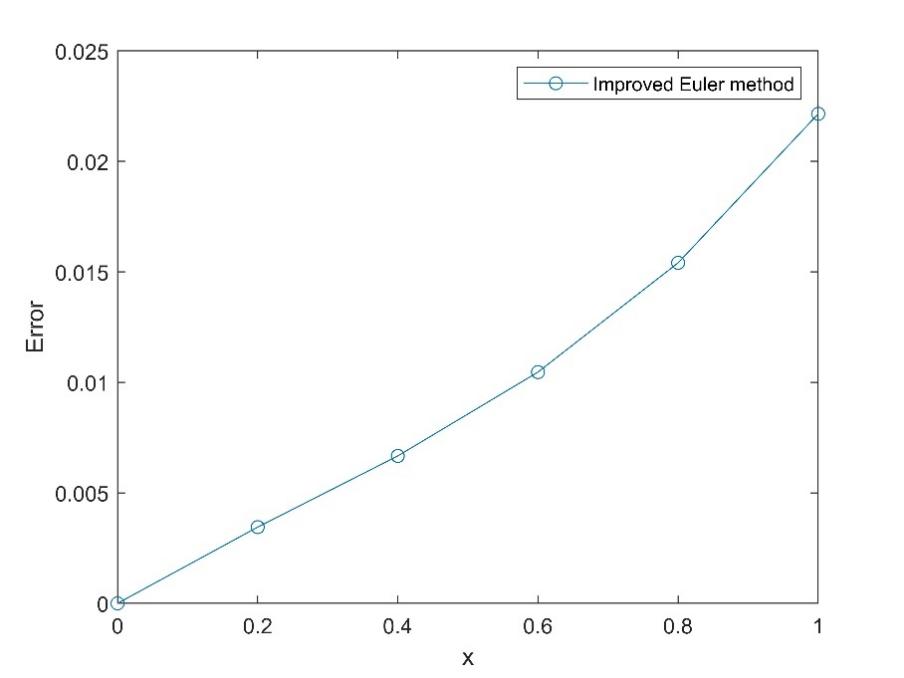
**图1 欧拉法数值解与精确解 步长0.2**



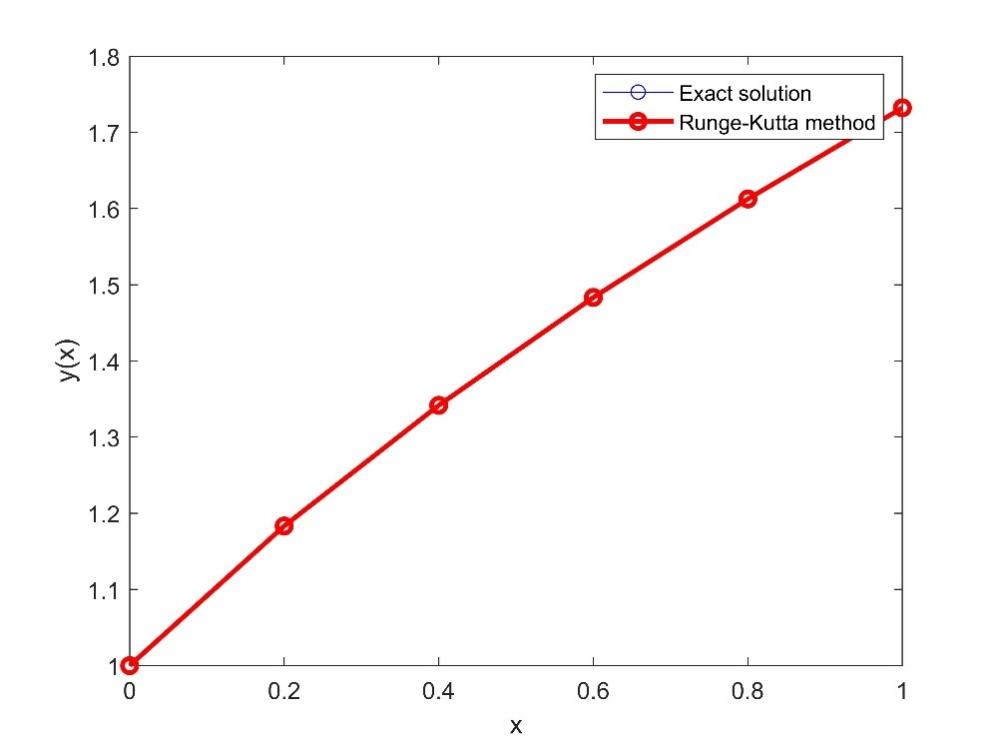
**图2 欧拉法误差**



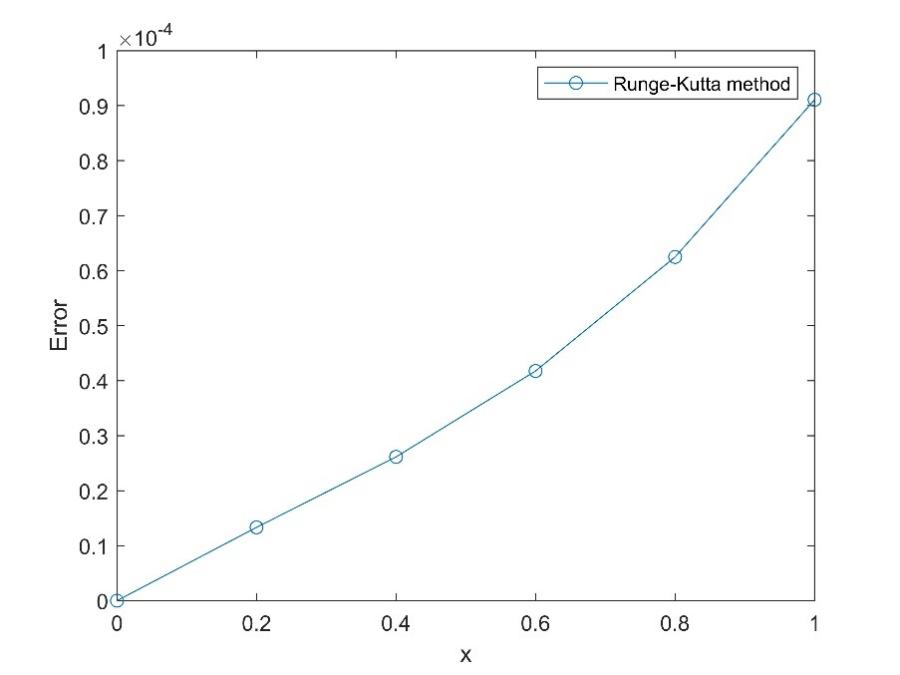
**图3 改进欧拉法数值解与精确解 步长0.2**



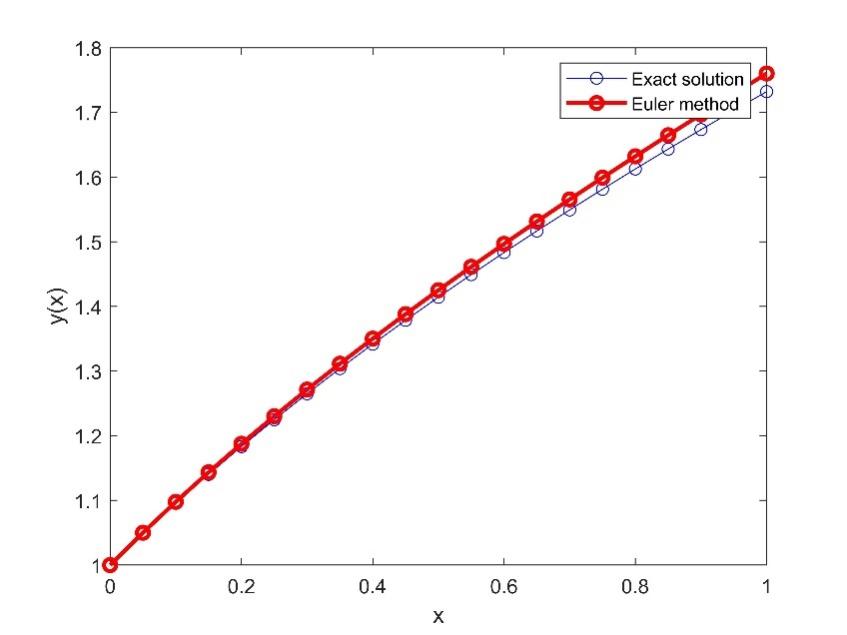
**图4 改进欧拉法数值解误差**

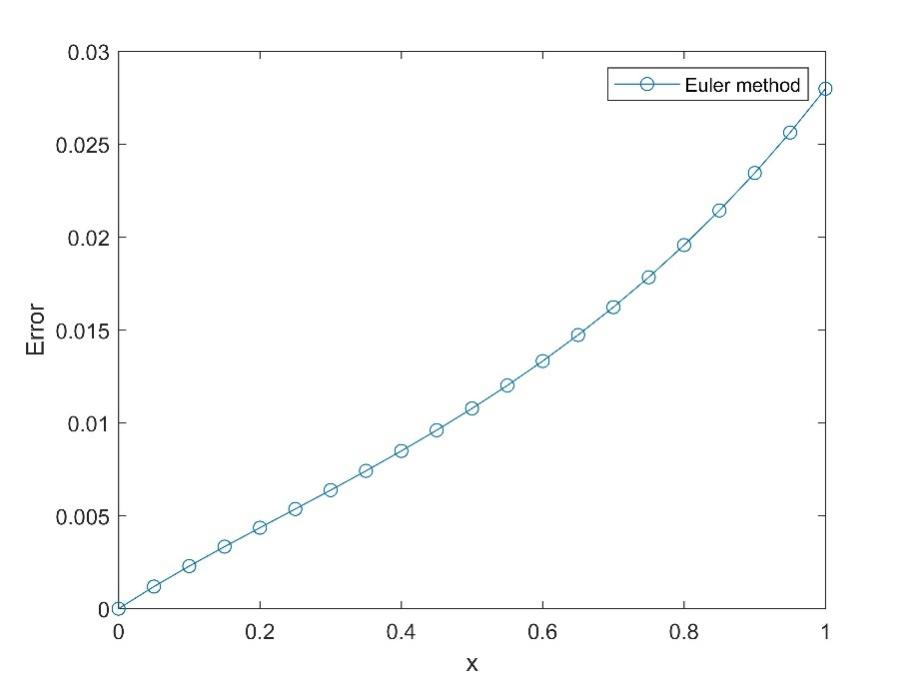


**图5 4阶龙格-库塔数值解与精确解 步长0.2**



**图6 4阶龙格-库塔数值解误差**



**图7 欧拉法数值解与精确解 步长0.05**

**图8 欧拉法数值解与精确解误差 步长0.05**

## 分析与讨论

可以看出，改进欧拉法和龙格-库塔方法较欧拉法解微分方程精度更高，且步长越小时，误差越小从最大0.1误差到0.03误差。

# 附录一 主要关键代码

## 项目1 插值法

线性、二次插值基函数

clear

X = [0 1 2 3 4 5 6 7 8];

Y = [0.4 0.2 0.5 1.0 0.3 0.6 0.4 0.8 0.2];

h = 0.01;

xQ=[];yQ=[];

for k = 1:length(X)-1

x = X(k):h:X(k+1);

lk = (x-X(k+1))/(X(k)-X(k+1));

lk1 = (X(k)-x)/(X(k)-X(k+1));

y = lk\*Y(k) + lk1\*Y(k+1);

h1=plot(x,y,'red','linewidth',3);

hold on;

axis([min(X) max(X) min(Y) max(Y)])

xlabel('X','fontsize',15);

ylabel('Y','fontsize',15);

set(gca,'xtick',min(X):max(X),'fontsize',15)

%pause(0.5);

hold on;

end

for k=1:2:length(X)-1

x=X(k):h:X(k+2); %每隔h取一个x

xQ=[xQ x];%定义全集x

Lk0=(x-X(k+1)).\*(x-X(k+2))/((X(k)-X(k+1)).\*(X(k)-X(k+2)) );

Lk1=(x-X(k)).\*(x-X(k+2))/((X(k+1)-X(k)).\*(X(k+1)-X(k+2)) );

Lk2=(x-X(k)).\*(x-X(k+1))/((X(k+2)-X(k)).\*(X(k+2)-X(k+1)) );

y=Lk0\*Y(k)+Lk1\*Y(k+1)+Lk2\*Y(k+2);

yQ=[yQ y];%定义全集x

%plot(x,Lk0,'r',x,Lk1,'b',x,Lk2,'black','linewidth',3);

h2=plot(x,y,'blue','linewidth',3);

%pause(0.5);

%legend('linear', 'quadratic');

legend([h1, h2], {'\color{red}Linear', '\color{blue}Quadratic'});

end

拉格朗日插值

% 示例数据

x\_data = [1, 2, 3, 4];

y\_data = [5, 3, 2, 7];

% 要进行插值的点

x\_interp = 2.5;

% 使用拉格朗日插值函数计算插值结果

y\_interp = lagrange\_interpolation(x\_data, y\_data, x\_interp);

% 显示结果

fprintf('在 x = %.2f 处的拉格朗日插值结果为 y = %.2f\n', x\_interp, y\_interp);

% 绘制原始数据和插值结果

figure;

plot(x\_data, y\_data, 'o', 'MarkerSize', 10, 'MarkerFaceColor', 'b', 'MarkerEdgeColor', 'b', 'LineWidth', 2);

hold on;

plot(x\_interp, y\_interp, 'rx', 'MarkerSize', 10, 'LineWidth', 2);

xlabel('x');

ylabel('y');

legend('原始数据', '拉格朗日插值点', 'Location', 'best');

grid on;

% 定义拉格朗日插值函数

function result = lagrange\_interpolation(x\_data, y\_data, x)

n = length(x\_data);

result = 0;

for i = 1:n

term = y\_data(i);

for j = 1:n

if i ~= j

term = term \* (x - x\_data(j)) / (x\_data(i) - x\_data(j));

end

end

result = result + term;

end

end

均差表

clear

clc

x=[0.4 0.55 0.65 0.8 0.9 1.05];

y=[0.41075 0.57815 0.69675 0.88811 1.02652 1.25382];

n=length(x);

difftable=zeros(n,n);

difftable(:,1)=y;

for j = 2:n

for i = j:n

difftable(i, j) = (difftable(i, j-1) - difftable(i-1, j-1)) / (x(i) - x(i-j+1));

end

end

fprintf('xk\t');

fprintf('f(xk)\t\t');

fprintf('一阶均差\t\t');

fprintf('二阶均差\t\t');

fprintf('三阶均差\t\t');

fprintf('四阶均差\t\t');

fprintf('五阶均差\t\t');

fprintf('\n');

% 打印差商表

for i = 1:n

fprintf('x%d\t', i-1);

for j = 1:i

fprintf('%.6f\t', difftable(i, j));

end

fprintf('\n');

end

## 项目2 函数逼近与离散数据拟合

问题1代码

clear

clc

min=-5;max=5;

n=15;%分成15份

%画出原本的图像

x=min:0.01: max;

y=1./(1+x.^2);

plot(x,y,'LineWidth',2);

hold on

%等距节点

x11=min:(max-min)/n: max;

y11=1./(1+x11.^2);

f11=lagrange\_interpolation(x11, y11, x);

plot(x,f11,'LineWidth',2);

%插值节点选切比雪夫节点

for k=0:1:n-1

t(k+1)=cos(pi\*(2\*n-1-2\*k)/2/n);

x\_chep(k+1)=5\*t(k+1);

end

% x=min:0.01:max;

x21=x\_chep;

y21=1./(1+x21.^2);

f21=lagrange\_interpolation(x21, y21, x);

plot(x,f21,'LineWidth',2);

legend('原始数据','等距节点拟合','切比雪夫节点拟合');

% 定义拉格朗日插值函数

%x为插值结点

function result = lagrange\_interpolation(x\_data, y\_data, x)

n = length(x\_data);

result = 0;

for i = 1:n

temp = y\_data(i);

for j = 1:n

if i ~= j

temp = temp .\* (x - x\_data(j)) / (x\_data(i) - x\_data(j));

end

end

result = result + temp;

end

end

问题2代码

close all

clear

clc

x=[1 2 3 4 5];

f=[4 4.5 6 8 8.5];

%最小二乘拟合 用线性模型

plot(x,f,'-o','LineWidth',2);

hold on;

p = polyfit(x,f,1);

f1 = polyval(p,x);

plot(x,f1,'--','LineWidth',2)

legend1=legend('Original data','Least-square fitting WITHOUT weights');

set(legend1,'Location','best');

%% 线性、带权拟合

figure();

plot(x,f,'-o','LineWidth',2);

hold on;

w=[2 1 3 1 1];%m=4,n=1

A=zeros(2);b=zeros(2,1);

A(1,1)=sum(w);

A(1,2)=sum(w.\*x);

A(2,1)=sum(w.\*x);

A(2,2)=sum(w.\*x.\*x);

b(1)=sum(w.\*f);

b(2)=sum(w.\*x.\*f);

a=pinv(A)\*b;

s1=a(1)+a(2)\*x;

plot(x,s1,'LineWidth',2);

plot(x,f1,'--','LineWidth',2);

legend('Original data','Least-square fitting WITH weights','Least-square fitting WITHOUT weights');

p1 = poly2sym(a);

p1

%% 抛物线 、带权拟合

figure();

plot(x,f,'-o','LineWidth',2);

hold on;

w=[2 1 3 1 1];%m=4,n=2

n=2;

A=zeros(n+1);b=zeros(n+1,1);

A(1,1)=sum(w);

A(1,2)=sum(w.\*x);

A(1,3)=sum(w.\*x.\*x);

A(2,1)=A(1,2);

A(2,2)=sum(w.\*x.\*x);

A(2,3)=sum(w.\*x.\*x.\*x);

A(3,1)=A(1,3);

A(3,2)=A(2,3);

A(3,3)=sum(w.\*x.^4);

b(1)=sum(w.\*f);

b(2)=sum(w.\*x.\*f);

b(3)=sum(w.\*x.^2.\*f);

a=pinv(A)\*b;

s2=a(1)+a(2)\*x+a(3)\*x.^2;

plot(x,s2,'LineWidth',2);

plot(x,f1,'--','LineWidth',2);

legend('Original data','Quadratic Least-square fitting WITH weights','Quadratic Least-square fitting WITHOUT weights');

p2 = poly2sym(a);

p2

正交多项式拟合

close all

clear

clc

%P76 正交多项式作最小二乘拟合

%数据准备

xi=[1 2 3 4 5];

yi=[4 4.5 6 8 8.5];

wi=[2 1 3 1 1];

plot(xi,yi,'-o','LineWidth',2);

hold on;

n = 3; %拟合函数阶数

x = sym('x');

m = length(xi);

P=zeros(n,m); %构建矩阵用于存放原始点处P的值

for i=1:m %首先将P的第一行赋值为1

P(1,i)=1;

end

for k=2:n+1 %根据公式计算

alpha(k)=sum(wi(i).\*xi.\*P(k-1,:).^2)/sum(wi(i).\*P(k-1,:).^2);

if k>2

beta(k)=sum(wi(i).\*P(k-1,:).^2)/sum(wi(i).\*P(k-2,:).^2);

end

for i=1:m %计算在数据点xi(i)处Pk处的值

if k>2

P(k,i)=(xi(i)-alpha(k))\*P(k-1,i)-beta(k)\*P(k-2,i);

else

P(k,i)=(xi(i)-alpha(k))\*P(k-1,i);

end

end

end

for k=1:n+1 %计算a\*

astar(k)=sum(wi(i).\*yi.\*P(k,:))/sum(wi(i).\*P(k,:).^2);

end

p=cell(n,1); %利用元胞数组存放P

p{1}=1;

for k=2:n+1 %求出P1到Pk的表达式

if k>2

p{k}=(x-alpha(k))\*p{k-1}-beta(k)\*p{k-2};

else

p{k}=(x-alpha(k))\*p{k-1};

end

end

y = 0;

for k=1:n+1 %得出最终的拟合函数表达式

y=y+astar(k)\*p{k};

end

%画图

% plot(xi,yi,'--','LineWidth',2); hold on;

fplot(y, [0.5,5],'LineWidth',2);hold on;

legend('Original data','3次正交多项式拟合');

## 项目3 数值积分

clear

clc

f = @(x) x./(4+x.^2);

%f = @(x) sin(x)/x;

a=1;b=2;n=6;

result\_trapezoidal = compositeTrapezoidal(f, a, b, n);

fprintf('复合梯形的积分结果：%f\n', result\_trapezoidal);

result\_simpson = compositeSimpson(f, a, b, n);

fprintf('复合辛普森法的积分结果：%f\n', result\_simpson);

result\_romberg = rombergIntegration(f, a, b, n);

fprintf('龙贝格求积法的积分结果：%f\n', result\_romberg);

true= 0.235001814622868;

for n\_interval=1:1:20

error1= compositeTrapezoidal(f, a, b, n\_interval)-true;

error2= compositeSimpson(f, a, b, n\_interval)-true;

error3= rombergIntegration(f, a, b, n\_interval)-true;

plot(n\_interval,error1,'r-o','LineWidth',2);

hold on;

plot(n\_interval,error2,'b-o','LineWidth',2);

hold on;

plot(n\_interval,error3,'c-o','LineWidth',2);

legend('Trapezoidal','Simpson','Romberg');

end

function result = compositeTrapezoidal(f, a, b, n)

h = (b - a) / n;

x = a:h:b;

result = (h / 2) \* (f(a) + 2 \* sum(f(x(2:end-1))) + f(b));

end

function result = compositeSimpson(f, a, b, n)

h = (b - a) / n;

x = a:h:b;

%n 是子区间的数量，它必须是偶数以确保每个小区间都有三个点。

result = (h / 3) \* (f(a) + 4 \* sum(f(x(2:2:end-1))) + 2 \* sum(f(x(3:2:end-2))) + f(b));

end

function result = rombergIntegration(f, a, b, max\_iter)

R = zeros(max\_iter, max\_iter);

% 第一列，h=1 的情况

h = b - a;

R(1, 1) = (h / 2) \* (f(a) + f(b));

for i = 2:max\_iter

% 计算 h\_i

h = h / 2;

% 计算新的列

sum\_term = 0;

for k = 1:2^(i-2)

sum\_term = sum\_term + f(a + (2\*k-1) \* h);

end

R(i, 1) = 0.5 \* R(i-1, 1) + h \* sum\_term;

% 计算剩余的列

for j = 2:i

R(i, j) = R(i, j-1) + (R(i, j-1) - R(i-1, j-1)) / (4^(j-1) - 1);

end

end

% 返回最终结果

result = R(max\_iter, max\_iter);

end

## 项目4 线性方程组的直接解法

高斯消元和LU分解时间对比

clear

clc

A=rand(1000,1000);

b=sum(A')';

Gauss(A,b);

LU(A,b);

function [] = Gauss(A,b)

% Gauss消元计算

tic

n=length(b);%维数

zg=[A,b];%增广矩阵

for i=1:n-1%第一行到n-1行计算系数

for k=i+1:n%从第二行开始消元

m=zg(k,i)/zg(i,i);%计算乘数

%fprintf('第%d乘数大小：%d \n',i,m)

zg(k,i:n+1)=zg(k,i:n+1)-m\*zg(i,i:n+1);%公式

%fprintf('第%d列为主元消元后： \n',i)

%disp(zg);

end

end

%更新A、b的值

A=zg(1:n,1:n);

b=zg(1:n,n+1);

%回代计算

x(n)=b(n)/A(n,n);

for k=n-1:-1:1

x(k)=(b(k)-sum( A(k,k+1:n).\*x(k+1:n)))/A(k,k);%公式

end

% fprintf('X(1)：')

% disp(x(1))

% fprintf('X(2)：')

% disp(x(2))

% fprintf('X(3)：')

% disp(x(3))

%disp(x)

toc

end

function [] = LU(A,b)

%LU计算解方程

tic

[L,U] = LU\_composition(A);

n=length(A);

%求解Ly=b Ux=y

y(1)=b(1);

for i=2:n

for k=1:i-1

b(i)=b(i)-L(i,k)\*y(k);

end

y(i)=b(i);

end

x(n)=y(n)/U(n,n);

for i=n-1:-1:1

for k=n:-1:i+1

y(i)=y(i)-U(i,k)\*x(k);

end

x(i)=y(i)/U(i,i);

end

disp("LU");

disp(x)

toc

end

function [L,U] = LU\_composition(A)

% 实现矩阵LU分解，L为单位下三角矩阵。U为上三角矩阵

n=length(A);

L=zeros(n);

U=zeros(n);

for i=1:n

L(i,i)=1;

end

%计算U、L

for r=1:n

if r==1

U(r,1:n)=A(r,1:n);

end

for i=r:n

U(r,i)=A(r,i)-sum(L(r,1:r-1).\*U(1:r-1,i)');

end

%fprintf('第%d次分解后U为： \n',r)

% disp(U)

for i=r+1:n

L(i,r)=(A(i,r)-sum(L(i,1:r-1).\*U(1:r-1,r)'))/U(r,r);

end

% fprintf('第%d次分解后L为： \n',r)

%disp(L)

end

end

高斯消元

clear

clc

b=[6;5;1];

A=[1,2,3;2,5,2;3,1,5];

Gauss(A,b);

function [] = Gauss(A,b)

% Gauss消元计算

tic

n=length(b);%维数

zg=[A,b];%增广矩阵

for i=1:n-1%第一行到n-1行计算系数

for k=i+1:n%从第二行开始消元

m=zg(k,i)/zg(i,i);%计算乘数

%fprintf('第%d乘数大小：%d \n',i,m)

zg(k,i:n+1)=zg(k,i:n+1)-m\*zg(i,i:n+1);%公式

%fprintf('第%d列为主元消元后： \n',i)

%disp(zg);

end

end

%更新A、b的值

A=zg(1:n,1:n);

b=zg(1:n,n+1);

%回代计算

x(n)=b(n)/A(n,n);

for k=n-1:-1:1

x(k)=(b(k)-sum( A(k,k+1:n).\*x(k+1:n)))/A(k,k);%公式

end

disp("Gauss");

disp(x)

toc

end

LU分解方程

clc

clear

% A=[1,2,3;2,5,2;3,1,5];

% b=[14;18;20];

b=[6;5;1];

A=[1,2,3;2,5,2;3,1,5];

LU(A,b);

function [] = LU(A,b)

%LU计算解方程

tic

[L,U] = LU\_composition(A);

n=length(A);

%求解Ly=b Ux=y

y(1)=b(1);

for i=2:n

for k=1:i-1

b(i)=b(i)-L(i,k)\*y(k);

end

y(i)=b(i);

end

x(n)=y(n)/U(n,n);

for i=n-1:-1:1

for k=n:-1:i+1

y(i)=y(i)-U(i,k)\*x(k);

end

x(i)=y(i)/U(i,i);

end

disp("LU");

disp(x)

toc

end

function [L,U] = LU\_composition(A)

% 实现矩阵LU分解，L为单位下三角矩阵。U为上三角矩阵

n=length(A);

L=zeros(n);

U=zeros(n);

for i=1:n

L(i,i)=1;

end

%计算U、L

for r=1:n

if r==1

U(r,1:n)=A(r,1:n);

end

for i=r:n

U(r,i)=A(r,i)-sum(L(r,1:r-1).\*U(1:r-1,i)');

end

%fprintf('第%d次分解后U为： \n',r)

% disp(U)

for i=r+1:n

L(i,r)=(A(i,r)-sum(L(i,1:r-1).\*U(1:r-1,r)'))/U(r,r);

end

% fprintf('第%d次分解后L为： \n',r)

%disp(L)

end

end

## 项目5 线性方程组的迭代解法

主函数

clear

clc

A = [8 -3 2;4 11 -1;6 3 12];

b = [20;33;36];

x0=[0,0,0]';

jacobi(A,b,x0,1.0e-7);

seidel(A,b,x0,1.0e-7);

Jacobi迭代法

function [x,n] = jacobi(A,b,x0,eps)

% 求线性方程组的Jacobi迭代法

D = diag(diag(A));%求A的对角矩阵

L = -tril(A,-1);%求A的下三角矩阵

U = -triu(A,1);%求A的上三角矩阵

B = D\(L+U);

f = D\b;

x = B\*x0+f;

n = 1;%迭代次数

while norm(x-x0)>=eps

x0 = x;

fprintf("第%d次",n);

x = B\*x0+f

n = n+1;

plot(n,x(1),'r-o');

hold on;

end

高斯赛德尔迭代法

function [xk,k] = seidel(A,b,x0,eps)

D=diag(diag(A));

U=triu(A,1);

L=tril(A,-1);

BG=-(D+L)\U;

fG=(D+L)\b;

xk=BG\*x0+fG;

k=1;

while norm(xk-x0)>=eps

x0=xk;

fprintf("第%d次",k);

xk=BG\*x0+fG

k=k+1;

plot(k,xk(1),'b-o');

hold on;

end

end

## 项目6 非线性方程的迭代解法

main函数

%牛顿迭代法

% 定义非线性方程函数和导数

f = @(x) x^3-x - 1;

df = @(x) 3 \* x^2-1;

% 初始猜测值

x0 = 1.5;

% 收敛容限和最大迭代次数

tolerance = 1e-6;

max\_iterations = 100;

% 调用牛顿法函数

[root, iterations] = newton\_method(f, df, x0, tolerance, max\_iterations);

% 显示结果

disp('估计的根:');

disp(root);

disp('迭代次数:');

disp(iterations);

newton\_method

function [root, iterations] = newton\_method(f, df, x0, tolerance, max\_iterations)

iterations = 1;

x = x0;

while iterations < max\_iterations

% 牛顿法迭代公式

x\_new = x - f(x) / df(x);

% 判断收敛

if abs(x\_new - x) < tolerance

root = x\_new;

fprintf('第 %d 次, x=%.12f\n', iterations, root);

return;

end

fprintf('第 %d 次, x=%.12f\n', iterations, x\_new);

x = x\_new;

iterations = iterations + 1;

end

end

clear

clc

% f(x)=x^3-x-1;

x=1.5; % 初始值

esp=1e-6; % 迭代终止条件

n=100; % 最大迭代次数

y=zeros(n, 1);

format long

for t=1:n

x=fun(x);

y(t)=x;

fprintf('第 %d 次迭代, x=%.12f\n', t, x);

if t>1

if abs(y(t)-y(t-1))<esp

break;

end

end

end

function x=fun(x)

x=(x+1).^(1./3); % p215迭代函数

end

## 项目7 矩阵特征值与特征向量

main

clear

clc

%% 输入参数

% 输入矩阵

A = [1 1 0.5;1 1 0.25;0.5 0.25 2];%P248例二

B=[2 1 0;1 3 1;0 1 4];%P253

% 输入初始值

u0 = [1;1;1];

%次数

n=50;

%% 采用幂法进行计算

power\_method(A,u0,n)

%% 采用反幂法进行计算

inverse\_power\_method(B,u0,n)

power\_method

function power\_method(A,u0,n)

%幂法

format long

v = A \* u0;

u = v / norm(v,inf);%无穷范数相当于最大值

i = 1;

for i=1:n

u0 = u;

v = A \* u0;

u = v / norm(v,inf);

end

lamda=norm(v,inf); % 特征值

fprintf('特征向量为：\n');

disp(u);

fprintf('特征值为： \n');

disp(lamda);

end

inverse\_power\_method

function inverse\_power\_method(A,u0,n)

%反幂法

p=1.2679;

I = eye(3,3);

v = pinv(A - p \* I) \* u0;

u = v / norm(v, inf);

for i=1:n

u0 = u;

v = pinv(A - p \* I) \* u0;

u = v / norm(v, inf);

end

lamda = p + 1 / norm(v, inf) ;

fprintf('特征向量为：\n');

disp(u);

fprintf('特征值为： \n');

disp(lamda)

end

## 项目8 常微分方程数值解法

Main函数

clear

clc

a=0;

b=1;

ya=1;

%h=0.2;

h=1/20;

f=@(x,y)y-x\*2/y;

eulers(f,a,b,ya,h);

impeuler(f,a,b,ya,h);

rk4m(f,a,b,ya,h);

eulers

function E=eulers(f,a,b,ya,h)

M=ceil((b-a)/h);

T=zeros(1,M+1);

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

Y(j+1)=Y(j)+h\*feval(f,T(j),Y(j));

end

E=[T' Y'];

%精确解

x=a:h:b;

u=sqrt(2\*x+1);

plot(x,u,'-bo');

hold on;

%数值解

plot(T,Y,'-ro','Linewidth',2);

ylabel('y(x)');

xlabel('x');

legend('Exact solution','Euler method');

%计算误差

error=abs(u-Y);

figure();

plot(x,error,'-o');

ylabel('Error');

xlabel('x');

legend('Euler method');

impeuler

function IME=impeuler(f,a,b,ya,h)

M=ceil((b-a)/h);

T=zeros(1,M+1);

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

Yj=Y(j)+h\*feval(f,T(j),Y(j));

Y(j+1)=Y(j)+h\*(feval(f,T(j),Y(j))+feval(f,T(j+1),Yj))/2;

end

IME=[T' Y'];

figure;

%标准解

x=a:h:b;

u=sqrt(2\*x+1);

plot(x,u,'-bo');

hold on;

%数值解

plot(T,Y,'-ro','Linewidth',2);

ylabel('y(x)');

xlabel('x');

legend('Exact solution','Improved Euler method');

%计算误差

error=abs(u-Y);

figure();

plot(x,error,'-o');

ylabel('Error');

xlabel('x');

legend('Improved Euler method');

rk4m

function RK4=rk4m(f,a,b,ya,h)

M=ceil((b-a)/h);

T=zeros(1,M+1);

Y=zeros(1,M+1);

T=a:h:b;

Y(1)=ya;

for j=1:M

f1=feval(f,T(j),Y(j));

f2=feval(f,T(j)+h/2,Y(j)+h\*f1/2);

f3=feval(f,T(j)+h/2,Y(j)+h\*f2/2);

f4=feval(f,T(j)+h,Y(j)+h\*f3);

Y(j+1)=Y(j)+h\*(f1+2\*f2+2\*f3+f4)/6;

end

RK4=[T' Y'];

figure;

%标准解

x=a:h:b;

u=sqrt(2\*x+1);

plot(x,u,'-bo');

hold on;

%数值解

plot(T,Y,'-ro','Linewidth',2);

legend('Exact solution','Runge-Kutta method');

ylabel('y(x)');

xlabel('x');

%计算误差

error=abs(u-Y);

figure();

plot(x,error,'-o');

ylabel('Error');

xlabel('x');

legend('Runge-Kutta method');

# 附录二 查重报告