

**数值分析上机作业**

|  |  |
| --- | --- |
| 姓名： |  |
| 班级： |  |
| 学号： |  |
| 日期： | 2020年11月13日 |

|  |  |
| --- | --- |
| 题目I | P20第17题：舍入误差与有效数 |
| 题目II | P56第20题：Newton迭代法 |
| 题目III | P126第40题：列主元Gauss消去法 |

# 目录

[题目I 1](#_Toc56116090)

[1.1 算法 1](#_Toc56116091)

[1.2 程序 1](#_Toc56116092)

[1.3 输出结果 2](#_Toc56116093)

[1.4 结果分析及感悟 2](#_Toc56116094)

[题目II 3](#_Toc56116095)

[2.1 算法 3](#_Toc56116096)

[2.2 程序 3](#_Toc56116097)

[2.3 输出结果 4](#_Toc56116098)

[2.4 结果分析及感悟 6](#_Toc56116099)

[题目III 7](#_Toc56116100)

[3.1 算法 7](#_Toc56116101)

[3.2 程序 7](#_Toc56116102)

[3.3 输出结果 8](#_Toc56116103)

[3.4 结果分析及感悟 9](#_Toc56116104)

# 题目I

设，其精确值为.

1. 编制按从小到大的顺序，计算的通用程序；
2. 编制按从大到小的顺序，计算的通用程序；
3. 按两种顺序分别计算,并指出有效位数；（编制程序时用单精度）
4. 通过本上机你明白了什么？

## 1.1 算法

使用计算机程序，分别编写使用精确值公式、按从小到大顺序累加以及按从大到小顺序累加的程序。在编写以上各个函数的时候，所有变量、参数、返回值等的精度均选用单精度（float），以放大精度对计算方法的影响。编写完3个函数之后，编写主程序，根据输入值分别使用3个函数计算结果，并进行对比。使用C语言，可以方便地控制所使用的变量的精度，且本题没有复杂的矩阵计算，使用C语言编程并不复杂。

## 1.2 程序

#include <stdio.h>

float accumulate\_Sn(int n) { //求和计算Sn

    int i;

    float sum = 0;

    for (i = 2; i < n; i ++) { // 从2开始到n

        sum += 1.0f / ((float)i \* (float)i - 1);

    }

    return sum;

}

float accumulate\_Sn\_reverse(int n) { //反向求和计算Sn

    int i;

    float sum = 0;

    for (i = n; i >= 2; i --) { // 从n开始反向累加到2

        sum += 1.0f / ((float)i \* (float)i - 1);

    }

    return sum;

}

float calculate\_Sn(int n) { //直接计算Sn

    return 0.5f \* (1.5f - 1.0f / (float)n - 1.0f / ((float)n + 1));

}

int main() {

    int n;

    while(1) {

        printf("Please enter the value of n: ");

        scanf\_s("%d", &n);

        printf("Sn by accumulating: %f\r\n", accumulate\_Sn(n));

        printf("Sn by reverse accumulating: %f\r\n", accumulate\_Sn\_reverse(n));

        printf("Sn by direct calculation: %f\r\n\r\n", calculate\_Sn(n));

    }

}

## 1.3 输出结果

Please enter the value of n: 100

Sn by accumulating: 0.739949

Sn by reverse accumulating: 0.74005

Sn by direct calculation: 0.740049

Please enter the value of n: 10000

Sn by accumulating: 0.749852

Sn by reverse accumulating: 0.7499

Sn by direct calculation: 0.7499

Please enter the value of n: 1000000

Sn by accumulating: 0.749852

Sn by reverse accumulating: 0.749999

Sn by direct calculation: 0.749999

## 1.4 结果分析及感悟

从程序的输出结果可以看出，不同的算法之间结果确实存在误差，且若以公式直接计算为基准，从大到小顺序计算出来的结果明显更加准确，这是因为数值计算中存在“大数吃小数”的问题：由于计算机计算的截断特点，从大到小的计算会导致较小数的有效位被忽略掉。

通过本上机题，我明白了看似等价的计算，在计算机计算过程中仍然会存在不同。因此，在未来编写计算机数值计算程序时，要预先了解是否存在类似“大数吃小数”的问题，避免计算精度影响最终的解。

# 题目II

1. 给定初值x0及容许误差ε，编制Newton法解方程f(x)=0根的通用程序。
2. 给定方程,易知其有三个根x1※=，x2※=0，x3※=。

①由Newton方法的局部收敛性可知存在＞0，当x0∈（，），Newton迭代序列收敛于根x2※ ，试确定尽可能大的；

②试取若干个初始值，观察当x0∈（-∞，-1），（-1，），（，），（，1），（1，+∞）时，Newton序列是否收敛以及收敛于哪一个根。

1. 通过本上机题，你明白了什么？

## 2.1 算法

首先，编写两个函数，分别代表f(x)与f’(x)。然后，依据牛顿迭代方法，编写牛顿迭代法解方程的通用程序。程序中首先为迭代变量赋初值，然后使用while循环，让程序在达到误差要求前一直进行迭代运算。为了满足第(1)问和第(2)问的要求，牛顿迭代法函数可以开启或关闭计算过程中的打印信息。

编写计算最大值的程序时，x从0开始以一个固定的步进值增加，每次增加后都进行牛顿迭代，第一个结果不为0的x值即为最大的。

计算f(x)在x0各个范围的收敛性及收敛的根时，在题目所给的范围中，各选数个数值，使用牛顿迭代法函数进行求解，并将详细步骤打印到控制台上，即可观察f(x)在各个x范围内的敛散性及最终迭代值。

为了确保最高的计算精度，本题程序中出现的所有变量、参数与返回值均使用双精度（double）型变量。

## 2.2 程序

#include <stdio.h>

#include <math.h>

double f(double x) { //输入方程

    return x \* x \* x / 3 - x;

}

double df(double x) { //输入方程的导数

    return x \* x - 1;

}

//牛顿法解方程f(x)=0的通用程序

double newton\_iteration(double x0, double max\_error,

double (\*f) (double), double (\*df) (double),  //f与df的函数指针

int enalbe\_print) {

    double x1 = x0;

    double x2 = x1 - f(x1) / df(x1); //牛顿迭代法公式

int k = 0;

    if (enalbe\_print) {

        printf(" k        x(k)\r\n", k, x1);

    }

    while (fabs(x1 - x2) >= max\_error) { //进行迭代 直到满足最大误差要求

        if (enalbe\_print) {

            printf("%3d  %13.10f\r\n", k, x1);

        }

        x1 = x2;

        x2 = x1 - f(x1) / df(x1);

        k++;

    }

    return x1;

}

double get\_maxium\_delta(double step) {

    double x = step;

double y = 0;

    do {

        x += step;

        y = newton\_iteration(x, 1e-5, f, df, 0);

    } while (fabs(y) <= step);

    return x;

}

double input\_values[] = {-20,  -10,    -5,      -1.1, //(-inf, -1)

                       -0.99, -0.9, -0.85, -0.774599, //(-1, -delta)

                      -0.774, -0.1,   0.1,     0.774, //(-delta, delta)

                    0.774559, 0.85,   0.9,      0.99, //(delta, 1)

                         1.1,    5,    10,        20};//(1, inf)

int main() {

    double res;

    int i;

    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(1)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n\r\n");

    res = get\_maxium\_delta(1e-5); //(1) 求收敛于x2的最大delta值

    printf("Max delta: %.10f\r\n\r\n", res);

    printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(2)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n\r\n");

    for (i = 0; i < sizeof(input\_values) / sizeof(double); i++) {

        newton\_iteration(input\_values[i], 1e-5, f, df, 1); //(2) 输入不同初始值 观察收敛性及收敛的根

        printf("\r\n\r\n");

    }

}

## 2.3 输出结果

(1)

Max delta: 0.7746000000

(2)

x0∈（-∞，-1）时：

|  |  |  |  |
| --- | --- | --- | --- |
| k x(k) | k x(k) | k x(k) | k x(k) |
| 0 -20.0000000000  1 -13.3667502089  2 -8.9613225229  3 -6.0495468647  4 -4.1463281301  5 -2.9349334547  6 -2.2136048174  7 -1.8541260399  8 -1.7431364747  9 -1.7321556694 | 0 -10.0000000000  1 -6.7340067340  2 -4.5905702249  3 -3.2128401249  4 -2.3716525794  5 -1.9229810688  6 -1.7571748201  7 -1.7325795665 | 0 -5.0000000000  1 -3.4722222222  2 -2.5241804370  3 -1.9960683633  4 -1.7766182124  5 -1.7336735825 | 0 -1.1000000000  1 -4.2253968254  2 -2.9840686821  3 -2.2410506211  4 -1.8654706910  5 -1.7451216056  6 -1.7321962046 |

x0∈（-1，）时：

|  |  |  |  |
| --- | --- | --- | --- |
| k x(k) | k x(k) | k x(k) | k x(k) |
| 0 -0.9900000000  1 32.5058291457  2 21.6910813342  3 14.4915209495  4 9.7072379892  5 6.5409059090  6 4.4649659279  7 3.1338395799  8 2.3260746019  9 1.9023031006  10 1.7524783955  11 1.7324025123 | 0 -0.9000000000  1 2.5578947368  2 2.0129154848  3 1.7816615329  4 1.7340488445 | 0 -0.8500000000  1 1.4753753754  2 1.8194435475  3 1.7379691099  4 1.7320809013 | 0 -0.7745990000  1 0.7746106540  2 -0.7746805832  3 0.7751003575  4 -0.7776261845  5 0.7930439968  6 -0.8960486959  7 2.4334596439  8 1.9519278837  9 1.7643723922  10 1.7329177972 |

x0∈（，）时：

|  |  |  |  |
| --- | --- | --- | --- |
| k x(k) | k x(k) | k x(k) | k x(k) |
| 0 -0.7740000000  1 0.7710269677  2 -0.7535428201  3 0.6600467532  4 -0.3396982623  5 0.0295419584  6 -0.0000172031 | 0 -0.1000000000  1 0.0006734007 | 0 0.1000000000  1 -0.0006734007 | 0 0.7740000000  1 -0.7710269677  2 0.7535428201  3 -0.6600467532  4 0.3396982623  5 -0.0295419584  6 0.0000172031 |

x0∈（，1）时：

|  |  |  |  |
| --- | --- | --- | --- |
| k x(k) | k x(k) | k x(k) | k x(k) |
| 0 0.7745590000  0 0.7745990000  1 -0.7746106540  2 0.7746805832  3 -0.7751003575  4 0.7776261845  5 -0.7930439968  6 0.8960486959  7 -2.4334596439  8 -1.9519278837  9 -1.7643723922  10 -1.7329177972 | 0 0.8500000000  1 -1.4753753754  2 -1.8194435475  3 -1.7379691099  4 -1.7320809013 | 0 0.9000000000  1 -2.5578947368  2 -2.0129154848  3 -1.7816615329  4 -1.7340488445 | 0 0.9900000000  1 -32.5058291457  2 -21.6910813342  3 -14.4915209495  4 -9.7072379892  5 -6.5409059090  6 -4.4649659279  7 -3.1338395799  8 -2.3260746019  9 -1.9023031006  10 -1.7524783955  11 -1.7324025123 |

x0∈（1，+∞）时：

|  |  |  |  |
| --- | --- | --- | --- |
| k x(k) | k x(k) | k x(k) | k x(k) |
| 0 1.1000000000  1 4.2253968254  2 2.9840686821  3 2.2410506211  4 1.8654706910  5 1.7451216056  6 1.7321962046 | 0 5.0000000000  1 3.4722222222  2 2.5241804370  3 1.9960683633  4 1.7766182124  5 1.7336735825 | 0 10.0000000000  1 6.7340067340  2 4.5905702249  3 3.2128401249  4 2.3716525794  5 1.9229810688  6 1.7571748201  7 1.7325795665 | 0 20.0000000000  1 13.3667502089  2 8.9613225229  3 6.0495468647  4 4.1463281301  5 2.9349334547  6 2.2136048174  7 1.8541260399  8 1.7431364747  9 1.7321556694 |

## 2.4 结果分析及感悟

上述结果显示，在各个区间内，迭代序列均收敛，且在（-∞，-1）区间内收敛于，即x1※；在（-1，）区间内收敛于，即x3※；在（，）区间内收敛于0，即x2※；在（，1）区间内收敛于，即x1※；在（1，+∞）区间内收敛于，即x3※。

通过本上机题，我明白了在使用Newton法求解多根方程的根时，迭代序列会在某个区间上收敛于某一个根，因此使用Newton法有一定的区间限制。在一个区间上，迭代结果可能会局部收敛于不同的根。

Newton迭代法对于初值的选择要求较高，因此，在牛顿迭代时可先使用简单迭代法或者试凑法，寻找粗略的根的范围，再进行Newton迭代。

# 题目III

对于某电路的分析，归结为求解线性方程组RI=V。

32 -13 0 0 0 -10 0 0 0

-13 35 -9 0 -11 0 0 0 0

0 -9 31 -10 0 0 0 0 0

R= 0 0 0 -30 57 -7 0 -5 0

0 0 0 0 -7 47 -30 0 0

0 0 0 0 0 -30 41 0 0

0 0 0 0 -5 0 0 27 -2

0 0 0 -9 0 0 0 -2 29

VT=[-15，27，-23，0，-20,12，-7,7,10]T

1. 编制解n阶线性方程组Ax=b的列主元Gauss消去法的通用程序；
2. 用所编程序解线性方程组RI=V，并打印出解向量，保留5位有效数字；
3. 在本编程之中，你提高了那些编程能力。

## 3.1 算法

程序首先定义了线性方程组的阶数，并输入了线性方程组以及方程组右端的向量。之后使用列主元高斯消去法，通过求每列的主元的函数并进行列消去，获得一个上三角矩阵，最后回带x，对线性方程组RI=V求解。

由于本题存在复杂的矩阵计算，因此选择使用Python语言及numpy运算库，该库提供了对于矩阵的描述方法，并提供了大量关于矩阵的计算函数，大大降低了编程量。

## 3.2 程序

import numpy as np

row = 9

col = 9

R = np.array([[ 31, -13,  0,  0,  0, -10,  0,  0,  0],

              [-13,  35,  -9,  0, -11,  0,  0,  0,  0],

              [ 0,  -9,  31, -10,  0,  0,  0,  0,  0],

              [ 0,   0, -10,  79, -30,  0,  0,  0, -9],

              [ 0,  0,  0, -30,  57,  -7,  0, -5,  0],

              [ 0,  0,  0,  0,  -7,  47, -30,  0,  0],

              [ 0,  0,  0,  0,  0, -30,  41,  0,  0],

              [ 0,  0,  0,  0,  -5,  0,  0, 27, -2],

              [ 0,  0,  0,  -9,  0,  0,  0, -2, 29]], dtype='float')

V = np.array([[-15], [27], [-23], [0], [-20], [12], [7], [7], [10]], dtype='float')

# 按行合并A,B

m = np.concatenate((R, V), axis=1)

# 列主元高斯消去法得上三角

for r1 in range(row):

    max\_val = max(abs(m[r1:row, r1]))

    for r2 in range(r1 + 1, row):

        if abs(m[r2, r1]) == max\_val:

            m1 = np.copy(m[r1, :])

            m[r1, :] = m[r2, :]

            m[r2, :] = m1

    for r2 in range(r1 + 1, row):

        m[r2, :] = m[r2, :] + m[r1, :]\*(-m[r2, r1]/m[r1, r1])

print("(1) Gaussian elimination method's result:\r\n", np.around(m, 5))

# 回代求解x

x = np.zeros((row, 1), dtype='float')

for i in range(row-1, -1, -1):

    sum = 0.0

    for j in range(i+1, col):

        sum = sum + m[i, j] \* x[j]

    x[i] = (m[i, col]-sum)/m[i, i]

print('(2) Answer vector of I is:\r\n', np.around(x, 5))

## 3.3 输出结果

(1) Gaussian elimination method's result:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 | -13 | 0 | 0 | 0 | -10 | 0 | 0 | 0 | -15 |
| 0 | 29.54839 | -9 | 0 | -11 | -4.19355 | 0 | 0 | 0 | 20.70968 |
| 0 | 0 | 28.25873 | -10 | -3.35044 | -1.27729 | 0 | 0 | 0 | -16.6921 |
| 0 | 0 | 0 | 75.46127 | -31.1856 | -0.452 | 0 | 0 | -9 | -5.9069 |
| 0 | 0 | 0 | 0 | 44.602 | -7.17969 | 0 | -5 | -3.57799 | -22.3483 |
| 0 | 0 | 0 | 0 | 0 | 45.87319 | -30 | -0.78472 | -0.56154 | 8.49257 |
| 0 | 0 | 0 | 0 | 0 | 0 | 21.3807 | -0.51319 | -0.36724 | -1.44605 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 26.41308 | -2.42 | 4.6081 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 27.3895 | 7.94922 |

(2) Answer vector of I is:

|  |
| --- |
| 0.34544 |
| -0.71281 |
| -0.22061 |
| -0.4304 |
| 0.15431 |
| -0.05782 |
| 0.20105 |
| 0.29023 |

## 3.4 结果分析及感悟

列主元Gauss消去法最重要的就是如何通过找到最大主元，并交换行，如何进行消去，这需要复杂的循环嵌套。通过进行实际编程，将自己平时手工计算的步骤使用计算机实现后，感受到了计算机编程所需要的逻辑性以及计算机计算功能的强大，并且巩固了对Gauss消去法的理解。