实验一 误差分析与数值稳定性

# 设为精确值，为的一个近似值，有两个量： 计算和的绝对误差和相对误差。试问哪个误差更大？

答：

将绝对误差记作，将相对误差记作.

对于有：

x1 = 3.100; a1=3.000;

absErr1 = x1 - a1

absErr1 = 0.1000

relErr1 = absErr1 / x1

relErr1 = 0.0323

对于有：

x2 = 310.0; a2=300.0;

absErr2 = x2 - a2

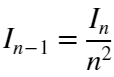
absErr2 = 10

relErr2 = absErr2 / x2

relErr2 = 0.0323

综上，与的相对误差一致，但的绝对误差比大。

# 设递归序列，现有两种算法：

1. 给出，利用，求解
2. 给出，利用，求解

# 试问那种算法是数值稳定的？给出讨论过程。

答：

如果使用算法1：

将算法命名为 ，函数详见结尾，

I6 = funcIn(0, 0.0235, 5)

I1=0.0235

I2=0.094

I3=0.846

I4=13.536

I5=338.4

如果使用算法2：

将算法命名为 ，函数详见结尾，

Iminus1 = funcInMinus(5, 338.4)

I4=13.536

I3=0.846

I2=0.094

I1=0.0235

I0=0.0235

就计算结果而言，二者没有明显差异。在此区间内，两种算法都是稳定的。

# 编程实现秦九韶算法，并用于多项式求解运算。

1. 在；；处的值
2. 在；；处的值

答：

秦九韶算法又称为Hornor法则，此处将算法函数命名为 ，详见结尾处。

对于问题1，可通过以下代码求解：

coef1 = [1 0 3 0 -2 6];

res\_1\_1 = polyHornor(coef1, 1.1)

res\_1\_1 = 9.4035

res\_1\_2 = polyHornor(coef1, 1.2)

res\_1\_2 = 11.2723

res\_1\_3 = polyHornor(coef1, 1.3)

res\_1\_3 = 13.7039

对于问题2，可通过以下代码求解：

coef2 = [3 -1 2 0 1];

res\_2\_1 = polyHornor(coef2, 2)

res\_2\_1 = 49

res\_2\_2 = polyHornor(coef2, 3)

res\_2\_2 = 235

res\_2\_3 = polyHornor(coef2, 4)

res\_2\_3 = 737

此实时脚本中使用的函数：

function res = funcIn(n, nValue, endN)

n = n+1;

if n ~= (endN + 1)

res = (n^2)\*nValue;

disp("I"+n+"="+res);

res = funcIn(n, res, endN);

end

end

function res = funcInMinus(n, nValue)

if n ~= 0

res = nValue / (n^2);

n = n - 1;

disp("I"+n+"="+res);

res = funcInMinus(n, res);

end

end

function res = polyHornor(coef, point)

res = coef(1);

for i=1:1:(length(coef)-1)

res = res\*point + coef(i+1);

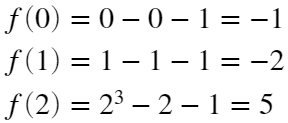
end

end

实验二 非线性方程的求解(一)

# 用逐步搜索法求方程的一个有根区间，要求有根区间范围不得超过0.1

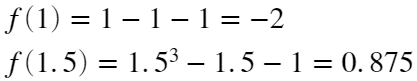
从开始以步长为 1 向右搜寻：



说明区间为一个有根区间。

区间范围超过，继续缩小区间。

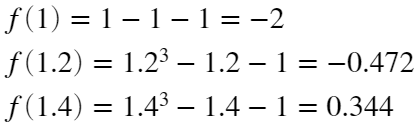
从开始以步长为 0.5 向右搜寻：



说明区间为一个有根区间。

区间范围超过，继续缩小区间。

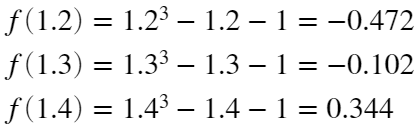
从开始以步长为 0.2 向右搜寻：



说明区间为一个有根区间。

区间范围超过，继续缩小区间。

从开始以步长为 0.1 向右搜寻：



说明区间为一个有根区间。

符合区间范围要求。

则此题的一个解为：区间

可以定义一种逐步搜索法的函数，如结尾处所示。

则可由如下脚本解决该问题，虽然效率不及上述计算过程。

clear;

f = inline('x^3-x-1','x');

[lower, higher] = stepwise(f, 0, 10, 0.1)

lower =

1.300000000000000

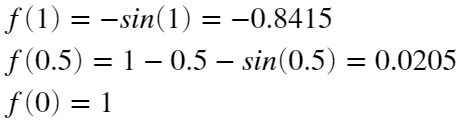
higher =

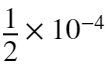
1.400000000000000

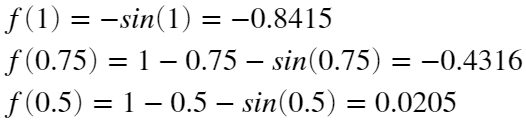
则此题的一个解为：区间

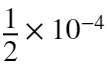
# 用二分法求解方程在区间内的一个实根，使得误差不大于

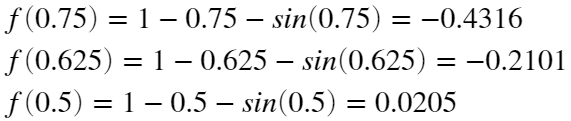
使用二分法，计算端点和中点的值：

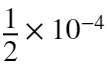


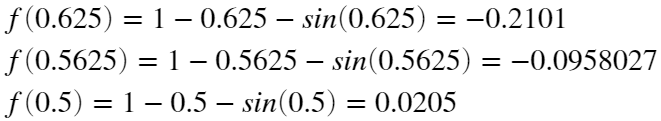
中点误差大于，继续二分：

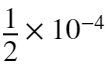


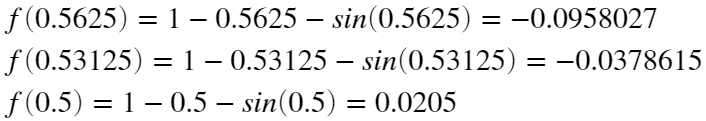
中点误差大于，继续二分：

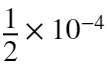


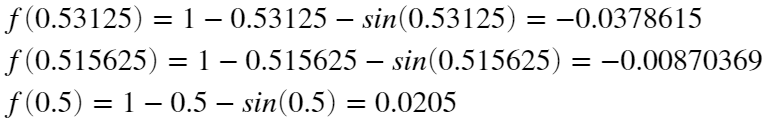
中点误差大于，继续二分：

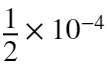


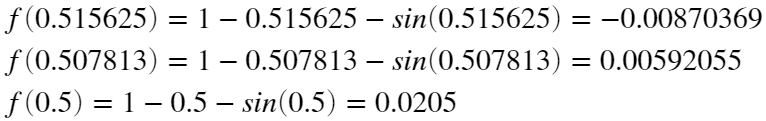
中点误差大于，继续二分：

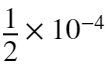


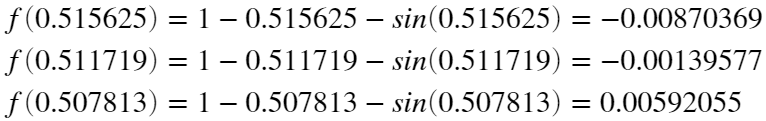
中点误差大于，继续二分：



中点误差大于，继续二分：



中点误差大于，继续二分：



计算过程过于复杂，余下省略。



可以由结尾处代码定义一种二分法求解方程的函数

则可由如下脚本解决该问题：

clear;

f = inline('1-x-sin(x)', 'x');

[res, err] = bisection(f, 0, 1, 5e-5)

res =

0.510986328125000

err =

2.414986223420179e-05

则此题结果为：

求得一个实根为，其误差为。

此实时脚本中使用的函数：

function [lower, higher] = stepwise(f, lower, higher, step)

% 逐步搜索

node = lower;

while node < higher

if f(node)\*f(node+step) < 0

lower = node;

higher = node + step;

break

end

node = node + step;

end

end

function [res, err] = bisection(f, lower, higher, err)

while f(lower)\*f(higher)<0

mid = (lower+higher)/2;

if f(lower)\*f(mid) < 0

higher = mid;

elseif f(higher)\*f(mid) < 0

lower = mid;

end

if abs(f(mid)) < err

break;

end

end

err = abs(f(mid));

res = mid;

end