**利用二分法和牛顿公式求解方程的根**

姓名：岳宇轩 学号：19020011038 序号：14

实验目的：

**1.实现牛顿公式，并分别找到收敛和发散的例子；**

**2.在相同精度和相同条件下，比较二分法与牛顿公式的迭代次数；**

实验原理：

1. **二分法：**

函数f(x)在区间[a,b]内单调连续，且f(a)f(b)<0，根据连续函数的性质可知方程在区间[a,b]内一定有唯一的实根。通过不断地把函数f(x)的零点所在的区间一分为二，使区间的两个端点逐步逼近根，进而得到根的近似值。

1. **牛顿公式：**

对于方程

f(x)=0

已知它的近似根xk，则函数f(x)在点xk附近可用一阶泰勒展开式

p(x)=f(xk)+f’(xk)(x-xk)

来近似，因此方程f(x)=0可近似的表示为p(x)=0。p(x)=0是一个线性方程，容易求根，因此取p(x)=0的根作为f(x)=0的新的近似根，记xk+1，则有



即为牛顿公式。

实验过程：

1. **实现牛顿公式，并分别找到收敛和发散的例子**

选取函数：



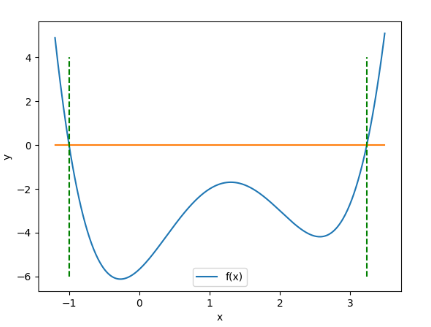
则有



带入牛顿公式可得迭代式为:



画出f(x)的函数图：



求解f(x)=0的两个根，两个根分别是x =-1和x = 3.2348365。为了验证牛顿公式的局部收敛性，我们分别选择x0 =4和x0 =2，作为牛顿公式的起始点进行迭代求根（误差不超过10-3)。

**2.在相同精度和相同条件下，比较二分法与牛顿公式的迭代次数；**

对于上述f(x)=0方程的根3.2348365，取二分初始区间为[2, 4]，牛顿迭代的初始值X0设为4，以保证公平性。精度设置：（误差不超过10-3)。

代码实现：

**全局变量设置：**

#define MAX\_TIMES 100 //牛顿迭代的最大迭代次数

#define e 0.001 //精度，即最小误差

#define ROOT 3.2348365 //根的真值

**f(x）函数以及f’(x)函数**

/// f(x) = 5/6x^4 - 4x^3 + 23/6x^2 + 3x - 17/3

double f(double x) {

return (5.0 / 6.0) \* pow(x, 4) - 4 \* pow(x, 3)

+ (23.0 / 6.0) \* pow(x, 2) + 3 \* pow(x, 1) - (17.0 / 3.0);

}

/// f'(x) = 10/3x^3 - 12x^2 + 23/3x + 3

double f\_d(double x) {

return (10.0 / 3.0) \* pow(x, 3) - 12 \* pow(x, 2)

+ (23.0 / 3) \* pow(x, 1) + 3;

**二分法：**

void Binary() {

int count = 0; //记录二分次数

double left = 2.0; //二分初始区间左

double right = 4.0; // 二分初始区间右

double mid;

printf("二分:\n");

printf(" k\t xk\n");

for (int i = 1; i <= MAX\_TIMES && (right - left) >= e; i++) {

mid = (left + right) / 2.0;

if (f(left) \* f(mid) < 0) {

right = mid;

}

else {

left = mid;

}

printf("%3d %f\n", i, left); //最终取区间左值为结果

count++;

}

printf("\n二分法的迭代次数为:%d\n\n", count); //输出二分次数

}

**牛顿公式：**

void Newton() {

int count = 0; //记录迭代次数

double x0 = 4.0; //迭代初值

double x1;

printf("牛顿公式:\n");

printf(" k\t xk\n");

for (int i = 1; i <= MAX\_TIMES; i++) {

x1 = x0 - f(x0) / f\_d(x0); //牛顿公式

printf("%3d %f\n", i, x1);

count++;

if (fabs(x1 - x0) < e) { //收敛精度达到要求

break;

}

x0 = x1;

}

printf("\n牛顿公式的迭代次数为:%d\n\n", count);

}

**主函数调用：**

int main() {

Binary();

Newton();

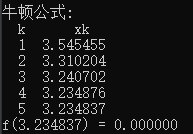
return 0;

}

实验结果

**1.实现牛顿公式，并分别找到收敛和发散的例子；**

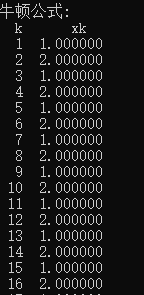
对于x0=4,结果如下：



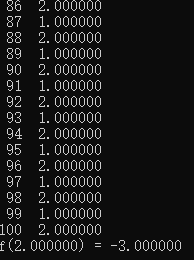
可以看到，牛顿公式经过5次迭代之后达到精度10^-3，输出f(xk)结果为0，表明实现了牛顿公式。

同时可以看到，对于初值x0=4，牛顿公式是收敛的。

对于x0=2，结果如下：

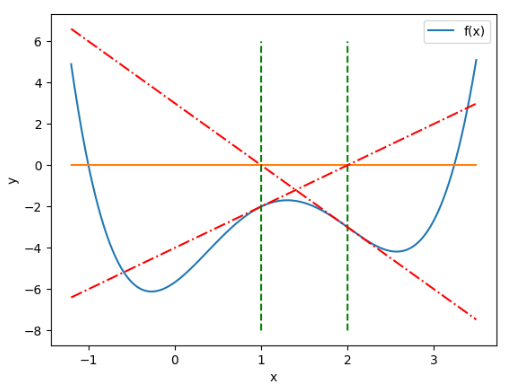


..........................



从图中可以看到，经过最大的迭代次数100之后，该迭代仍然没有收敛，因此对于初值x0=2时，牛顿公式是发散的。

**原因分析：**根据牛顿公式迭代的原理，xk+1实际上是取f(x)在xk处的切线与x轴的交点，那么可以画出如下图像：



根据f(x)和f’(x)，我们可以求出f(x)在x=1的切线方程为：



f(x)在x=2处的切线方程为：



切线在上图中用红线标出。

第一次迭代，对于初值x0=2，它的下一次迭代值x1就是f(x)在x=2处的切线与x轴交点处，将y=0带入f(x)在x=2处的切线方程可求出：x=1，所以第一次迭代结果x1=1.

第二次迭代，将y=0带入f(x)在x=1处的切线方程，可求出x=2，所以x2=2.

第三次迭代，将重复第一次迭代的过程。

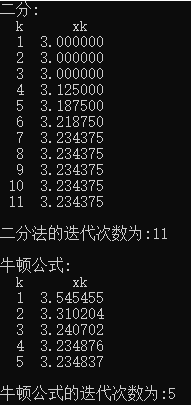
第四次迭代，将重复第二次迭代的过程。

......

可以发现，陷入了一个死循环之中，故对于初值x0=2的情况下，牛顿公式是不收敛的。

**2.在相同精度和相同条件下，比较二分法与牛顿公式的迭代次数**

实验结果如下：



可以看到，对于相同的精度要求（10^-3），二分法需要11次求解，而牛顿公式只需5次迭代。通过对比可知，牛顿公式收敛更快。

实验心得：

在这次的实验过程中，使用C++编写了二分法和牛顿公式的相关代码，增强了编程能力，也进一步理解了这两种求近似根的方法。

牛顿公式给了我们一种可以机械化求近似根的方式，不需要我们花费更多时间寻找适合的迭代函数p(，并且牛顿公式比二分法有更快的收敛速度，在精度一定的情况下，能更快地求解出近似根，但是牛顿公式是局部收敛的，也就是说对于某些函数f(x)与特定的xo牛顿公式不一定收敛于根x\*，所以在实际使用中可以用其他方法找出处在收敛区域内的xo，再使用牛顿公式求解近似根。