中北大学大数据学院实验报告

**课程名称： 数值分析 实验类型： 验证型**

**实验题目：**

班级：17070142学号：1707004716姓名：王浩 指导教师：韩慧妍 成绩：

|  |  |  |  |
| --- | --- | --- | --- |
| 实验地点 | 虚拟仿真实验室 | 实验时间 | 2019.12.2 |
| 1．实验目的  正确理解方程求根划界法和开放法，能够编程实现其中指定的方法，并且通过比较，分析出两类方法的优缺点。 | | | |
| 2. 实验任务  分别用二分法和牛顿法求方程在区间[2,3]内的根，观察两种方法的迭代次数，并说明原因。 | | | |
| 3. 相关知识  **二分法**  “二分法求根”的图片搜索结果"一般地，对于函数f(x),如果存在实数c,当x = c时，若f(c) = 0,那么把x = c叫做函数f(x)的零点。假定f(x)在区间(x,y)上连续，先找到a、b属于区间(x,y)，使f(a),f(b)异号，说明在区间(a,b)内一定有零点，然后求f[(a + b) / 2],现在假设f(a) < 0,f(b) > 0,a < b：   1. 如果f[(a + b) / 2] = 0，该点就是零点；如果f[(a + b) / 2] < 0,则在区间((a + b) / 2，b)内有零点，(a + b) / 2 => a，并重新从①开始继续寻找。 2. 中点函数值判断。如果f[(a + b) / 2] > 0，则在区间(a,(a + b) / 2)内有零点，(a + b) / 2 => b，并重新从①开始继续寻找。　这样就可以不断接近零点。   通过每次把f(x)的零点所在小区间收缩一半的方法，使区间的两个端点逐步迫近函数的零点，以求得零点的近似值。  **牛顿法**  如果根的初始估计值是xi，则可以在点[xi,f(xi)]处绘制一条与函数相切的切线，该切线与x轴的交点通常表示根的新估计值。    牛顿法正因为有此明显的几何意义，所以也叫切线法。若已知方程 的一个近似根，则函数在点附近可用一阶泰勒多项式来近似，因此方程可近似表示为设,则,取作为原方程新的近似根，然后将 作为代入上式。迭代公式为：。 | | | |
| 4. 实验内容与实验结果（要求画出两种方法的流程图，并给出实验运行结果图，要求程序结果先输出个人姓名和学号，再输出实验结果，程序提交电子版即可）    图1 二分法求根流程图 图2 牛顿法求根流程图    图3二分法初始区间 图4 二分法划分一次    图5二分法划分两次 图6 二分法划分四次    图7 二分法计算过程打印    图8牛顿切线法一次 图9牛顿切线法两次    图10 牛顿切线法计算过程  **二分法python代码**  **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** prettytable **import** PrettyTable title = PrettyTable([**"k"**,**r'$a\_k$'**,**"b\_k"**,**"x\_k"**,**"f(x\_k)的符号"**,**"b\_k-a\_k"**]) *# 定义表头* title.align[**"f(x\_k)的符号"**] = **"c"** *# 将符号列居中显示  # 设置原函数* **def** function(x):  **global** string  **return** eval(string)  *# 判断是否为实数根* **def** judge\_zero\_point(root1, root2):  *# 有两个根* **if** root1 == 0 **and** root2 == 0 **and** root1 != root2:  print(**"根为：{}和{}"**.format(root1, root2))  **elif** root1 == 0 **and** root2 == 0 **and** root1 == root2:  print(**"根为:{}"**.format(root1))  **return True  elif** root1 == 0:  print(**"根为:{}"**.format(root1))  **return True  elif** root2 == 0:  print(**"根为:{}"**.format(root1))  **return True** *# 二分法求根* **def** ordnaryDeal(left, right,k):  **global** string  mid = 0.5 \* (left + right)  left\_value = function(left)  right\_value = function(right)  mid\_value = function(mid)  plt.scatter(left,left\_value,c = **'red'**)  plt.scatter(right,right\_value,c = **'red'**)  signal = **"+" if** mid\_value > 0 **else "-"** *# 判断f(x\_k)的符号* title.add\_row([k, left, right, mid, signal, right - left]) *# 添加一行数据* k = k + 1 *# 标记行 + 1* **if** left\_value \* right\_value > 0:  print(**"此范围内没有实数根！请重新输入区域！"**)  **elif** right - left < 1e-4:  print(**"根的范围为：[{},{}]"**.format(left, right))  **return  elif** left\_value \* mid\_value < 0:  ordnaryDeal(left, mid,k)  **elif** right\_value \* mid\_value < 0:  ordnaryDeal(mid, right,k)  **elif** mid\_value == 0:  print(**"根为：{}"**.format(mid))  **return  if** \_\_name\_\_ == **'\_\_main\_\_'**:  **global** string  print(**"1707004716 王浩"**)  string = input(**'请输入表达式函数\n[符合python语法，例如：y==x\*\*2 ---> 只需要输入 x \*\* 2]\n:'**)  left = float(input(**"请输入左边界："**))  right = float(input(**"请输入右边界："**))   line = np.arange(left - 1.0, right + 1.0, 0.001) *# x坐标范围* plt.title(**"dichotomy"**) *# 图标标题* plt.xlabel(**"x"**) *# x轴标题* plt.ylabel(**"y"**) *# y轴标题* plt.plot(line, function(line)) *# 画出曲线* plt.grid(**True**) *# 显示网格* ordnaryDeal(left, right,0)   plt.show()  print(title) *# 打印数据* print()  **牛顿切线法python代码**  **import** numpy **as** np **import** matplotlib.pyplot **as** plt **from** prettytable **import** PrettyTable  title = PrettyTable([**"k"**,**"x\_k"**]) *# 定义表头* title.align[**"f(x\_k)的符号"**] = **"c"** *# 将符号列居中显示 # 原函数* **def** function(x):  **return** x \*\* 3 - 2 \* x - 5  *# 导数* **def** function1(x):  **return** 3 \* x \*\* 2 - 2  *# 点斜式求函数* **def** func(x0,f,x):  **return** f \* (x - x0) + function(x0)  **def** newtonian(num):  x0 = num  x1 = x0 - function(x0) / function1(x0)  first = [x0]  second = [x1]  plt.scatter(x0,function(x0))  line1 = np.arange(x1, x0, 0.001)  plt.plot(line1, func(x0, function1(x0), line1))  plt.scatter(x1, 0)  plt.scatter(x1,function(x1))  plt.plot([x1,x1],[0,function(x1)],color=**'red'**,linestyle=**'--'**)  count = 0  **while** np.abs(x1 - x0) > 1e-4:  x0 = x1  x1 = x0 - function(x0) / function1(x0)  title.add\_row([count,x0])  first.append(x0)  second.append(x1)  count += 1   print(**"该方程在{}附近的根为：{}"**.format(num,x1))  **return** x1  **if** \_\_name\_\_ == **'\_\_main\_\_'**:  print(**"1707004716 王浩"**)  left = float(input(**"请输入左边界："**))  right = float(input(**"请输入右边界："**))  number = float(input(**"请输入根附近的值："**))  line = np.arange(left, right, 0.001) *# x坐标范围* plt.title(**"newtonian"**) *# 图标标题* plt.xlabel(**"x"**) *# x轴标题* plt.ylabel(**"y"**) *# y轴标题* plt.plot(line, function(line)) *# 画出曲线* plt.grid(**True**) *# 显示网格* newtonian(number)  print(title)  plt.show() | | | |
| 5．总结  由上面的对二分法、牛顿法两种方法的两次实验结果，我们可以得出这样的结论：二分法要循环k=14次，牛顿法要迭代k=3次才能达到精度为的要求，而且方程的精确解经计算，为2.09455。计算量上二分法要大于牛顿法。由此可知，牛顿法的精确度要优越于二分法。两种方法相比来说，牛顿法不仅计算量少，而且精确度高。从而可知牛顿法收敛速度明显加快。二分法收敛虽然是速度最慢，但也有自己的优势，可常用于求精度不高的近似根。牛顿法是逐次逼近的方法,原理简单,但存在收敛性和收敛速度的问题。对与不同的题目，可以从这两种方法的优缺点考虑用哪一种方法比较好。 | | | |