《数值分析》课程关于编程实现方程的迭代解法的实验报告

姓名：张青龙

学号：1120172135

班级：08111702

目录

1. 实验目标

使用 Python 语言实现方程的迭代解法，支持埃特肯法、牛顿迭代法、牛顿下山法、单点弦截法、双点弦截法五种迭代解法。

在有余力的情况下，将迭代解法置于服务器中，并编写客户端与服务器连接，编写出美观、交互友好的可视化方程迭代解法界面。

2. 实验内容

2.1 实现迭代核心算法

五种迭代解法分别对应五种核心算法，这五种核心算法各有差异但是有共同之处。因此使用面向对象的思想，使五种迭代解法暴露出统一的接口，便于维护与扩展。

2.2 实现客户端服务端连接

在五种迭代解法暴露统一接口后，将其置于服务器中就变得容易起来。有余力的情况下编写服务器与客户端，搭建出可视化的方程求解界面。

3. 实现过程

3.1 确定项目结构与算法框架

当我们站在用户及《数值分析》课堂学生的角度思考，如果我想要一个程序来帮助我模拟手算，这个程序应该有什么输入呢？

在仔细研究和学习《数值分析》相关课程内容后，我总结出了方程迭代求解的步骤（以第 2 章埃特肯法的讲解视频第 7 分 30 秒及 19 分 30 秒出现的例题为例）：

（1）初步确认解的范围。如例题中首先判断了解的范围位于[0.5, 0.6]之间；

（2）使用解的范围粗略判断收敛性。如例题中使用解的范围确定了 φ’(x)

小于 1，即可收敛。此处特殊的地方是，使用计算机来判断完全正确的收敛性有困难（至少在这次实验中无法准确判断收敛性，难以判断一阶导单调性），因此此实验中采用“估计”的方法判断收敛性；

（3）带入参数，使用核心迭代算法开始迭代。如例题中首先确定了 x0, y1, z1 的取值，使用埃特肯公式开始迭代。在（2）中提到，收敛性无法准确判断，因此无论上一步判断的收敛性如何都尝试进行迭代；

（4）终止迭代。首先确定想要的精度，即精确到小数点后几位，当解达到精度时停止迭代。此处需要考虑到不收敛的情况，因此设置一个允许迭代的最大次数，当大于这个次数还没有得到解时即认为方程不收敛，停止迭代。

在总结出迭代算法的基本流程与输入输出后，我们的算法框架就很清晰了。我们可以设计一个抽象类，其构造函数需要传入迭代函数 $\phi(x)$（或原函数 f(x)，取决于迭代方法）、解范围、解精度，其需要提供估计是否收敛的函数、迭代的函数，如下：

```
class EquationSolution:
    '''
    fn_str: 迭代函数（或原函数）的字符串形式，如 'x**3 - x - 1'
    solution_range: 解范围，如 [0.6, 1.5]
    accurate_digits: 解精度，如 5
    '''
    def __init__(self, fn_str, solution_range, accurate_digits):
        pass

    '''
    判断方法是否收敛。
    '''
    def _estimate_is_convergent(self):
        pass

    '''
    迭代核心方法。
    '''
    def run(self):
        pass
```

每一个具体的核心算法类都需要继承此基类，然后实现相应函数即可。

3.2 使用 Python 进行数学计算的细节

3.2.1 将字符串形式的函数转化为可计算、可求导的函数

为了程序的可扩展性（如后期增加与客户端的交互），我们将函数定为字符串形式，使得用户不需要每一次运行都手动的去修改源码来达到修改求解函数的目的。

那么问题来了，字符串是不能直接运行的，怎么将其转化为可计算、可传参的函数呢？

经过查阅资料后，我了解到 Python 进行科学计算的库 Sympy，其使用方法非常贴合我们的需求，例：

```
from sympy import *

# 通过下面两行语句创建了sympy函数，可直接进行求值、求导等操作
x = Symbol('x')
fx = 5*x + 4

# 求x为5时的fx的值
y = fx.evalf(subs={x: 5})
print(y)

# 求x为5时的fx的一阶导值
diff_fx = diff(fx, x)
diff_y = diff_fx.evalf(subs={x: 5})
print(diff_y)
```

输出如下：

```
29.0000000000000
5.00000000000000
```

结合到我们的程序中，我们只需要将字符串形式的函数使用 Python 的 eval 函数来执行即可，例：

```
x = Symbol('x')
fx_str = '5*x + 4'
fx = eval(fx_str)
```

那么需要注意的是，我们所给的字符串形式的函数必须遵循 Sympy 的格式，即自然常数表示为 E，圆周率表示为 PI，次方表示为**等。

3.3 迭代核心算法

3.3.1 埃特肯法

埃特肯法需要提供的函数是迭代函数 $\phi(x)$，因此构造函数中传入的 fn_str 代表迭代函数而不是原函数。

第一步，我们需要实现_estimate_is_convergent 函数。其基本思想就是判断迭代函数的一阶导在解范围内是否小于 1。我们在此处很难判断其是否永远小于 1（即我们很难利用编程判断一阶导的单调性），因此我们只计算一阶导在解范围的两个端点上的值。如果这两个值都小于 1，那么就猜测其一阶导在解范围内一直小于 1，且估计方法收敛。

接下来是核心迭代算法。我们取 x0 为解范围的起点，然后便可以计算 y1, z1。然后判断 y1 与 x0 的差的绝对值与 z1 与 y1 的差的绝对值是否小于 10 的 -accurate_digits 次方（即如果传入的 accurate_digits 为 5，则判断是否小于 1e-5），如果小于，迭代结束，返回结果；否则重复上述步骤，使用埃特肯公式继续计算 xn, yn+1, zn+1……

需要注意的是，我们不管_estimate_is_convergent 函数输出的是 True 还是 False 都会尝试进行迭代，仅仅在控制台输出上告知用户我们估计收敛还是不收敛，因此在迭代的时候需要控制次数。如果迭代次数超过了设定的阈值还没有得到解，则判断其不收敛，停止迭代。

3.3.2 牛顿迭代法

牛顿迭代法需要提供的函数是原函数 f(x)，因此构造函数中传入的 fn_str 代表原函数。

同上，我们先实现_estimate_is_convergent 函数。依据课程内容，牛顿迭代法的收敛条件苛刻且判断麻烦，我们在此同样只能“估计”其收敛性。

（1）我们假定传入的原函数存在二阶导；

（2）计算解范围端点函数值乘积是否小于 0（即条件中的 f(a)f(b) > 0）；

（3）计算一阶导在解范围端点的函数值乘积是否大于 0（即使用 f’(a)f’(b) > 0 来推测 f’(x) != 0）；

（4）计算二阶导在解范围端点的函数值乘积是否大于 0（即使用 f’’(a)f’’(b) > 0 来推测 f’’(x)不变号）；

（5）计算原函数与二阶导在解范围起点的取值乘积是否大于 0（即使用 f(a)f’’(a) > 0 来推测初值 x0 满足 f(x0)f’’(x0) > 0）；

如果上述条件均为真，则判断收敛。

在迭代函数上面牛顿迭代法的实现较为简单。牛顿迭代法只需要 xn 与 xn+1 两个变量，如果两者之差绝对值小于阈值则迭代终止。第一次迭代时取初值为解范围起点，然后代入牛顿迭代法公式中求 xn+1，判断是否达到终止条件，若没有则将 xn+1 的值赋给 xn，再次进行计算，直到找到解或者迭代次数超过阈值。

3.3.3 牛顿下山法

牛顿下山法需要提供的函数是原函数 f(x)，因此构造函数中传入的 fn_str 代表原函数。

课程内容中并没有提到牛顿下山法的收敛条件，于是此处的 _estimate_is_convergent 函数就不需要实现。

我们在每一次迭代的开始根据公式计算出 xn（初值为解范围的起点）, f(xn), yn+1, xn+1（xn+1 与 yn+1 一开始是相等的，因为因子 λ=1），然后判断 xn 与 xn+1 差的绝对值是否小于阈值，若小于则迭代终止，否则开始判断是否符合下山条件，即|f(xn)| > |f(xn+1)|。若不符合，则重复减小因子 λ 直到下山条件符合为止，将 xn+1 的值赋给 xn，开始下一次迭代。如此往复直到到达迭代终止条件。

3.3.4 单点弦截法

单点弦截法需要提供的函数是原函数 f(x)，因此构造函数中传入的 fn_str 代表原函数。

单点弦截法的收敛条件判断也比较复杂，这里采用了与估计牛顿迭代法收敛性类似的方法：

（1）假定原函数二阶导存在；

（2）计算原函数在解范围两端点的值乘积，判断是否小于 0（即 f(a)f(b) < 0）；

（3）计算一阶导在解范围端点的函数值乘积是否大于 0（即使用 f'(a)f'(b) > 0 来推测 f'(x) != 0）；

（4）计算二阶导在解范围端点的函数值乘积是否大于 0（即使用 f''(a)f''(b) > 0 来推测 f''(x)不变号）；

（5）取不动点 x0 为解范围的起点或终点，分别判断 f(x0)f''(x0)是否大于 0，若其中一个大于 0 即可；

（6）解范围起点终点是否相异；

若以上条件均满足，则估计收敛。

迭代函数的一开始先取不动点（依据上述条件中的（5）），之后的迭代与牛顿迭代法相似，只是将牛顿迭代法中的导数计算替换成了两点连线的斜率，因此不再赘述。

3.3.5 双点弦截法

双点弦截法需要提供的函数是原函数 f(x)，因此构造函数中传入的 fn_str 代表原函数。

双点弦截法的收敛估计条件为：

（1）计算原函数在解范围两端点的函数值乘积是否小于 0（即 f(a)f(b) < 0）；

（2）计算一阶导在解范围两端点的函数值乘积是否大于 0（即通过计算 f'(a)f'(b) > 0 来估计 f'(x) != 0）；

若上述条件为真，则估计收敛。

迭代方法的实现也比较简单。取解范围两端点为 x0, x1，按照公式计算出 x2, x3…每一次迭代的结束之前判断是否可以终止迭代，方法同上。

3.4 从用户角度再次考虑

上面我们已经实现了核心算法，这意味着我们已经能成功的得到迭代结果了。但是对于学习《数值分析》的同学来说，光看个结果不够啊，能不能将解题的步骤也展示出来呢？

这其实并不难，我们只需要在关键的地方将关键的变量输出即可。因此我们在基类中增加一个_record_step()函数，用来打印及保存关键步骤的信息（保存是为了后续的前后端交互做准备）。

举个例子，我们在埃特肯算法的_estimate_is_convergent 函数中使用 _record_step 函数打印并记录 φ'(a)及 φ'(b)的值，让用户知道，程序是怎么估计收敛性的。接着在迭代函数中打印并记录每一次迭代中的 xn, yn+1, zn+1 的值，让用户知道清晰的过程。

4. 输入、输出测试

下列测试均使用教学视频中出现过的例题进行测试。

4.1 埃特肯法

4.1.1 埃特肯法教学视频第 19 分 30 秒，例 2.6

题目：用埃特肯法解 x=e^(-x)。

可得 fn_str 为’E**(-x)’，解范围 solution_range 为[0.5, 0.6]，解精度为 5。

测试代码为：

```
AitkenSolution('E**(-x)', [0.5, 0.6], 5, True).run()
```

（注：第四个参数为是否输出解题过程，后同）

测试输出为：

```
2020-04-12 15:19:10,528 - INFO - ϕ(x) = exp(-x), x ∈ [0.5, 0.6]
2020-04-12 15:19:10,528 - INFO - Atiken Iteration:
2020-04-12 15:19:10,533 - INFO - ϕ'(x) = -exp(-x)
2020-04-12 15:19:10,534 - INFO - ϕ'(0.5): -0.606530659712633
2020-04-12 15:19:10,535 - INFO - ϕ'(0.6): -0.548811636094027
2020-04-12 15:19:10,536 - INFO - estimated f(x) is convergent, start iteration
2020-04-12 15:19:10,536 - INFO - x0 = 0.5
2020-04-12 15:19:10,536 - INFO - y1 = 0.606530659712633
2020-04-12 15:19:10,537 - INFO - z1 = 0.545239211892605
2020-04-12 15:19:10,635 - INFO - x1 = 0.567623876410920
2020-04-12 15:19:10,636 - INFO - y2 = 0.566870794767815
2020-04-12 15:19:10,636 - INFO - z2 = 0.567297855542999
2020-04-12 15:19:10,637 - INFO - x2 = 0.567143314105580
2020-04-12 15:19:10,637 - INFO - y3 = 0.567143276970872
2020-04-12 15:19:10,638 - INFO - z3 = 0.567143298031573
2020-04-12 15:19:10,639 - INFO - |y3 - x2| < 1e-05 and |z3 - y2| < 1e-05, iteration break
2020-04-12 15:19:10,639 - INFO - the solution is 0.56714 (accurate to 5 digits after decimal point)
```

可以看到，我们的程序得到了与标准答案一致的结果，且解题步骤输出清晰明了，完全符合预期。

4.2 牛顿迭代法

4.2.1 牛顿迭代法教学视频第 6 分钟，例 2.10

题目：按切线法求 x=e^(-x)的根。

可得 fn_str 为’x*e**x – 1’，解范围为[0.5, 0.6]，解精度为 5。

测试代码为：

```
NetwonSolution('x*E**x - 1', [0.5, 0.6], 5, True).run()
```

测试输出为：

```
2020-04-12 15:30:40,635 - INFO - f(x) = x*exp(x) - 1, x ∈ [0.5, 0.6]
2020-04-12 15:30:40,635 - INFO - Netwon Iteration:
2020-04-12 15:30:40,636 - INFO - f'(x) = x*exp(x) + exp(x)
2020-04-12 15:30:40,645 - INFO - a = 0.5, b = 0.6
2020-04-12 15:30:40,645 - INFO - f(a) = -0.175639364649936, f(b) = 0.0932712802343053
2020-04-12 15:30:40,645 - INFO - f'(a) = 2.47308190605019, f'(b) = 2.91539008062481
2020-04-12 15:30:40,645 - INFO - f''(a) = 4.12180317675032, f''(b) = 4.73750888101532
2020-04-12 15:30:40,646 - INFO - estimated f(x) is not convergent, but try iterating
2020-04-12 15:30:40,646 - INFO - x0 = 0.5
2020-04-12 15:30:40,647 - INFO - x1 = 0.571020439808422
2020-04-12 15:30:40,648 - INFO - x2 = 0.567155568744114
2020-04-12 15:30:40,649 - INFO - x3 = 0.567143290533261
2020-04-12 15:30:40,650 - INFO - x4 = 0.567143290409784
2020-04-12 15:30:40,650 - INFO - |x3 - x4| < 1e-05, iteration break
2020-04-12 15:30:40,650 - INFO - the solution is 0.56714 (accurate to 5 digits after decimal point)
```

（注：exp(x)即代表 E**x，我们输入的 E**x 被自动转化了）

可以看到，我们的程序判断出此方法并不收敛（不符合定理 2.5 的第 4 条：f(x0)f”(x0) > 0），但是最后还是成功收敛并得到正确的答案。为什么呢？仔细回顾课程内容，我们可以知道定理 2.5 其实是充分条件。所以这也是我为什么选择在程序判断不收敛也尝试继续迭代的原因之一。

4.3 牛顿下山法

4.3.1 牛顿下山法教学视频第 9 分 47 秒，例 2.12

题目：用牛顿下山法求 f(x) = x^3 – x – 1 = 0 的根。

测试代码：

```
NetwonDownHillSolution('x**3 - x - 1', [0.6, 1.5], 5, True).run()
```

测试输出为：

```
2020-04-12 15:44:24,785 - INFO - f(x) = x**3 - x - 1, x ∈ [0.6, 1.5]
2020-04-12 15:44:24,785 - INFO - Netwon Down-Hill Iteration:
2020-04-12 15:44:24,796 - INFO - f'(x) = 3*x**2 - 1
2020-04-12 15:44:24,797 - INFO - x0 = 0.6
2020-04-12 15:44:24,797 - INFO - f(x0) = -1.38400000000000
2020-04-12 15:44:24,797 - INFO - x1 = y1 = 17.9000000000000
2020-04-12 15:44:24,798 - INFO - λ = 1/2, x1 = 9.25000000000001
2020-04-12 15:44:24,799 - INFO - λ = 1/4, x1 = 4.92500000000000
2020-04-12 15:44:24,800 - INFO - λ = 1/8, x1 = 2.76250000000000
2020-04-12 15:44:24,801 - INFO - λ = 1/16, x1 = 1.68125000000000
2020-04-12 15:44:24,802 - INFO - λ = 1/32, x1 = 1.14062500000000
2020-04-12 15:44:24,803 - INFO - x1 = 1.14062500000000
2020-04-12 15:44:24,803 - INFO - f(x1) = -0.656642913818358
2020-04-12 15:44:24,803 - INFO - x2 = y2 = 1.36681366159280
2020-04-12 15:44:24,804 - INFO - x2 = 1.36681366159280
2020-04-12 15:44:24,804 - INFO - f(x2) = 0.186639718200228
2020-04-12 15:44:24,804 - INFO - x3 = y3 = 1.32627980400832
2020-04-12 15:44:24,805 - INFO - x3 = 1.32627980400832
2020-04-12 15:44:24,805 - INFO - f(x3) = 0.00667040146993017
2020-04-12 15:44:24,805 - INFO - x4 = y4 = 1.32472022563606
2020-04-12 15:44:24,806 - INFO - x4 = 1.32472022563606
2020-04-12 15:44:24,806 - INFO - f(x4) = 9.67387688381062e-6
2020-04-12 15:44:24,807 - INFO - x5 = y5 = 1.32471795724954
2020-04-12 15:44:24,807 - INFO - |x4 - x5| < 1e-05, iteration break
2020-04-12 15:44:24,807 - INFO - the solution is 1.32472 (accurate to 5 digits after decimal point)
```

可以看到，程序的测试输出与正确答案一致，并且详细的展示了“下山”的过程。

4.4 单点弦截法

4.4.1 弦截法教学视频第 9 分 30 秒，例 2.13

题目：用单点弦截法求 f(x) = xe^x – 1 = 0 的根。

测试代码：

```
SingleSecantSolution('x*E**x - 1', [0.5, 0.6], 5, True).run()
```

测试输出为：

```
2020-04-12 15:50:19,163 - INFO - f(x) = x*exp(x) - 1, x ∈ [0.5, 0.6]
2020-04-12 15:50:19,163 - INFO - Single Secant Iteration:
2020-04-12 15:50:19,173 - INFO - a = 0.5, b = 0.6
2020-04-12 15:50:19,173 - INFO - f(a) = -0.175639364649936, f(b) = 0.0932712802343053
2020-04-12 15:50:19,173 - INFO - f'(a) = 2.47308190605019, f'(b) = 2.91539008062481
2020-04-12 15:50:19,173 - INFO - f''(a) = 4.12180317675032, f''(b) = 4.73750888101532
2020-04-12 15:50:19,173 - INFO - estimated f(x) is convergent, start iteration
2020-04-12 15:50:19,173 - INFO - x0 = 0.6
2020-04-12 15:50:19,173 - INFO - x1 = 0.5
2020-04-12 15:50:19,174 - INFO - x2 = 0.565315140174367
2020-04-12 15:50:19,175 - INFO - x3 = 0.567094633483845
2020-04-12 15:50:19,177 - INFO - x4 = 0.567141996189781
2020-04-12 15:50:19,178 - INFO - x5 = 0.567143255985542
2020-04-12 15:50:19,179 - INFO - |x4 - x5| < 1e-05, iteration break
2020-04-12 15:50:19,179 - INFO - the solution is 0.56714 (accurate to 5 digits after decimal point)
```

可看到，程序输出与正确答案一致。

4.5 双点弦截法

4.5.2 弦截法教学视频第 13 分 35 秒，例 2.14

题目：用双点弦截法求 f(x) = xe^x – 1 = 0 的根。

测试代码：

```
DoubleSecantSolution('x*E**x - 1', [0.5, 0.6], 5, True).run()
```

测试输出为：

```
2020-04-12 15:55:18,563 - INFO - f(x) = x*exp(x) - 1, x ∈ [0.5, 0.6]
2020-04-12 15:55:18,563 - INFO - Double Secant Iteration:
2020-04-12 15:55:18,571 - INFO - a = 0.5, b = 0.6
2020-04-12 15:55:18,572 - INFO - f(a) = -0.175639364649936, f(b) = 0.0932712802343053
2020-04-12 15:55:18,572 - INFO - f'(a) = 2.47308190605019, f'(b) = 2.91539008062481
2020-04-12 15:55:18,572 - INFO - estimated f(x) is convergent, start iteration
2020-04-12 15:55:18,572 - INFO - x0 = 0.5
2020-04-12 15:55:18,572 - INFO - x1 = 0.6
2020-04-12 15:55:18,577 - INFO - x2 = 0.565315140174367
2020-04-12 15:55:18,578 - INFO - x3 = 0.567094633483845
2020-04-12 15:55:18,579 - INFO - x4 = 0.567143363314904
2020-04-12 15:55:18,580 - INFO - x5 = 0.567143290406878
2020-04-12 15:55:18,580 - INFO - |x4 - x5| < 1e-05, iteration break
2020-04-12 15:55:18,581 - INFO - the solution is 0.56714 (accurate to 5 digits after decimal point)
```

可以看到，程序输出与正确答案一致。

5. 实验分析、结论及心得

在认真学习、理解课程内容的基础上，我们清晰的理清了五种迭代算法的输入、流程、输出，并将其转化为了对用户友好的过程输出，基本达到了预期的效果。在实验中也让我对算法的细节有了更深的理解，比如一定要注意某些定理只是充分条件等等。

我认为《数值分析》这门课应该更注重于实践，把大量时间浪费在繁杂的手算上并无太大用处。比如后面的差商表、差分表，实际上随便写几行代码就能打印出表的内容；拉格朗日插值的基函数计算也是随便写几行代码就能解决的事，但是如果手工去按计算器将会浪费大量的时间。毕竟《数值分析》本就是为计算机计算而服务的课程，我们应该抓住重点，在加强自身计算能力、通过考试的同时，也应该抓住这门课的内涵，学会编程实现，让这门课给我们带来更大的用处。

```javascript
const xr = [0, 0.2, 0.4, 0.6, 0.8];
const yr = [1.00000, 1.22140, 1.49182, 1.82212, 2.22554];

const table = [[...yr]];
const cs = false;

for(let i = 0; i < xr.length - 1; i++) {
  const temp = [];
  const last = table[table.length - 1];
  for(let j = 0; j < last.length - 1; j++) {
    if(cs) {
      const a = (last[j + 1] - last[j]).toFixed(5);
      const b = (xr[j + 1 + i] - xr[j]).toFixed(5);
      temp.push((a / b).toFixed(5));
    } else {
      temp.push((last[j + 1] - last[j]).toFixed(5));
    }
  }
  table.push(temp);
}

console.log(table);
```

（做作业时为验证手算结果而编写的用于计算差商差分表的 JavaScript 代码）

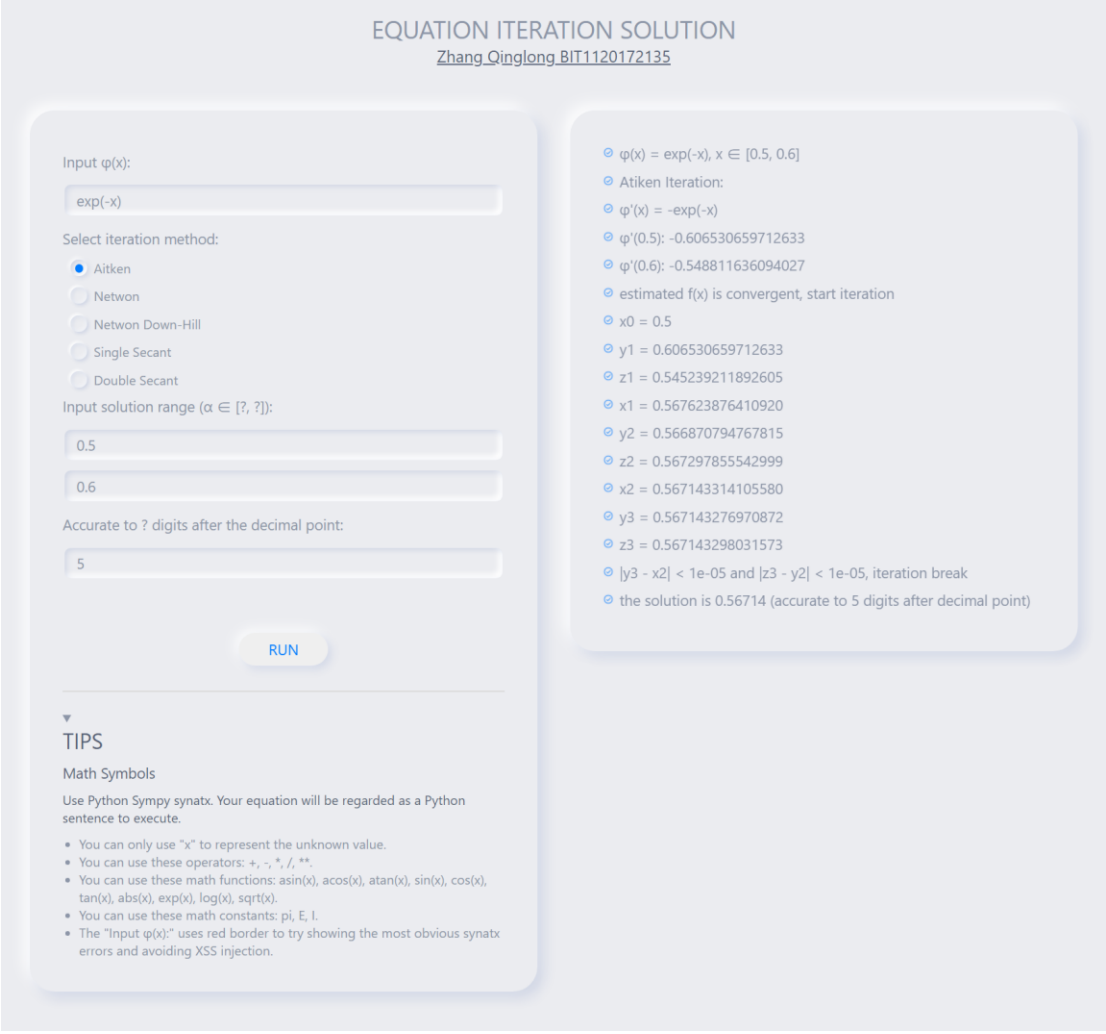
6. 附加任务

在实验目的中我写到，在有余力的情况下将编写服务端与客户端，打造出可视化的方程迭代解法平台。由于此项并不属于本课程内容，在此不介绍开发细节。

服务端为使用 Python Flask 搭建的简易服务器，得益于前期的面向对象设计，使得仅需要一个 Restful API 接口即可完成所有的前后端交互工作。

客户端为使用 React 打造的具有精美界面与设计感的可视化平台，设计灵感来源于今年大火的 neumorphism-ui 风格。

客户端效果图（压缩包内的 client.png）：



附录 程序运行方法

最简运行

即如本文测试中展示的那样，在控制台中输出步骤与结果。

以下命令均在/server 下运行。

（1）确保已安装 Python 3 版本（本人版本 3.7，但应该只要是 3 就行）；

（2）运行 pip install sympy 安装 Sympy 库；

（3）运行 python test.py 即可运行测试；

带客户端、服务端运行

（1）确保已安装 node.js, Python 3；

（2）在/web 文件夹下运行 npm install 安装依赖；

（3）在/server 文件夹下运行 pip install –r requirements.txt 安装依赖；

（4）在/server 文件夹下运行 python server.py 启动服务器；

（5）在/web 文件夹下运行 npm start 启动客户端（启动客户端后网页会自动在浏览器中打开；推荐使用 Chrome 浏览器，否则可能出现样式兼容性问题）；