

数值分析 Chapter 1 非线性方程 编程作业

樊睿

12435044@zju.edu.cn

数学科学学院
浙江大学

2024 年 9 月 18 日

作业 A D 要求

- 将非线性方程求解器统一封装为一个抽象类EquationSolver，并定义虚函数solve.
- 将本章介绍的三种非线性方程解法（二分、牛顿、割线）分别从这个抽象类继承.
- 用三种方法分别求解给定的方程.



回顾：虚函数和继承

我们在《数据结构与算法》课程中学习过虚函数和继承的概念。

如果一个基类的某个成员函数为虚函数（加`virtual`修饰），则这个基类可以继承出子类，在子类中可以重载该虚函数。这样做的好处是：当定义基类的指针或引用时，如果它明确指向了某个子类对象，则调用该虚函数时会直接调用子类重载过的虚函数。

如果基类的某个虚成员函数定义为纯虚函数（例如`virtual void solve() = 0`），则不能直接定义该类的对象，只能定义由它继承的类的对象。

具体语法可详见 c++ 官网<https://en.cppreference.com/w/>

“共性” 和 “差异”

在设计类的继承关系时，我们一般会将**功能相近，有共同属性但又不完全相同**的一些类设计成同一个基类的不同子类。将它们的“共性”定义为基类的成员，“差异”定义为子类的成员。

本题中的非线性方程求解器的共性是什么？三个求解器的唯一共性就是它们均要求解一个一元函数的零点，其他参数均不同。

因此可以考虑将这个一元函数作为基类的成员，而其他参数（例如二分法的 $a, b, \delta, \epsilon, M$ ）作为子类的成员。

函数类

对其他参数，我们用 `int`，`double` 等常规类型即可输入和存储。但如何将一个函数**作为参数**传到求解器中？

我们在《C 程序设计专题》课程中学习过函数指针的概念。可以将函数在测试程序中定义好，然后用函数指针传进求解器中。

但函数指针的结构过于简单，无法表示同一类的函数（例如：所有 n 次多项式）或者高度关联的函数（例如：函数与它的导数）。因此我们一般会定义一个**函数类**。

函数类

例如下面我们定义了一个函数基类，然后定义了一个子类并进行实例化。

```

1  class Function {
2  public:
3      virtual double operator() (double x) const = 0;
4      virtual double derivative(double x) const {...}
5  };
6  class F1 : public Function {
7  public:
8      virtual double operator() (double x) const {...}
9  };
    
```

思考：如何定义多项式函数类？这个问题在第 2 章编程作业中将至关重要。

求解器类示例代码

```

1  class EquationSolver{
2  protected:
3      const Function & F;
4  public:
5      EquationSolver(const Function& F) : F(F) {}
6      virtual double solve() = 0;
7  };
8
9  class Bisection_Method : public EquationSolver {
10 private:
11     double a, b;
12     double eps, delta;
13     int Maxiter;
14 public:
15     Bisection_Method(const Function &F, double a, double b,
16                     double eps = 1e-7, double delta = 1e-6, int Maxiter = 50) :
17         EquationSolver(F), a(a), b(b), eps(eps), delta(delta), Maxiter(Maxiter) {}
18
19     virtual double solve() {...}
20 };
    
```

Problem-B 示例代码

```

1  #include "Function.hpp"
2  #include "EquationSolver.hpp"
3  #include <iostream>
4  #include <cmath>
5
6  const double Pi = acos(-1.);
7
8  class F1 : public Function {
9  public:
10     double operator() (double x) const {
11         return 1.0/x-tan(x);
12     }
13 };
14
15 void solve_f1() {
16     std::cout << "Solving  $x^{-1} - \tan x$  on  $[0, \pi/2]$ " << std::endl;
17     Bisection_Method solver_f1(F1(), 0, Pi/2);
18     double x = solver_f1.solve();
19     std::cout << "A root is: " << x << std::endl;
20 }
21
22 /* Type your code here */
23
24 int main() {
25     solve_f1();
26     /* Type your code here */
27     return 0;
28 }

```