PCSC Project: Nonlinear Systems Solver

Yuqi Wang, Haojun Zhu

Programing Concepts in Scientific Computing (MATH-458), School of Basic Sciences, EPFL

1. Overview

This project is a course project of Programming Concepts in Scientific Computing (MATH-458) in EPFL. The aim of the project is to solve non linear systems by using numerical methods including bisection, fixed point (Aitken acceleration as well), chord, newton and modified newton method. Now the tool can support solving nonlinear equation with **only 1 variable** by **Bisection, Fixed Point, Aitken, Chord, Newton and Modified Newton**. The tool also supports solving nonlinear systems with **equal or less than 4 variables** by **Newton** and **Modified Newton**.

2. How to Compile

If you didn't clone the project yet. Please follow command below to download project. Note that use option "--depth=1" to avoid downloading large files.

```
git clone --depth=1 https://github.com/YQRickWang/PCSC_Non_Linear_Systems.git
cd PCSC_Non_Linear_Systems
cd Non_Linear_Systems
```

If you already download the project, please follow following commands to generate makefile from cmakelist and build the project (including compiling, linking).

```
mkdir build
cd build
cmake ..
make
```

If you want to run the program, type in followed command.

```
./Non_Linear_Systems
```

3. How to Use and Examples

When the compiling, linking and generating is done. The program is executive. In the executive program, user can choose three different ways to test. Each of them has input standard. **Please follow the standard to have proper results.**

3.1 Test from existing test cases

In the project, there are already some pre-defined test cases. Functions are defined in these test cases. For example, if we want to test "Test_B" with "Newton1D" and exit finally. Type following command. Instructions are made in the programs, and they are easy to follow.

3.2 Test from keyboard inputs

The tool also supports reading input from keyboard. User can type in any test cases into the program. For 1d, the variable name should be "x". For 2d, variable should be "x" and "y". For 3d, variables should be "x", "y" and "z". For 4d, variables should be "x", "y", "z" and "w". Higher dimension is not supported yet. The order of input should follow in order of dimension, functions, derivative functions (derivative for all variables in function 1 then function 2 ...) and fixed point function (only for 1D). Following is an example.

3.3 Test from reading txt files

Our tool can also test from existing txt files. The content in the file should following the standard. No extra lines are allowed in txt files. And all the files should be in the directory "txt_testfiles". The order of lines should follow in order of dimension, functions, derivative functions (derivative for all variables in function 1 then function 2 ...) and fixed point function (only for 1D). Following are two examples of test from txt files.

3.4 Test featuring the google unit test

We use the google to test all the available cases.

Due to the limit of the length of this report, you may find examples of these three modes for reading the testing function in our readme file.

4. List of Features

4.1 Interactive user experience

We provide a basic user interface to load existing test cases, read random equations from keyboard as well as read from txt files. This interactive interface helps user to switch between different approaches.

4.2 Multiple input approaches

As mentioned in previous part, the tool provides three input ways. The first is to load pre-defined test cases which are already defined in the program. The second is to read equations from keyboard. User can type any appropriate equation or system of equations (less than or equal to 4 variables). The program will parse these inputs automatically and give proper results. The third one is reading from txt files. The input flow is same as the second approach. But these equations are already defined in txt files.

4.3 Applicable for both single equation and systems of equations

Our program is suitable for solving nonlinear equation with one variable by Aitken, Newton, bisection and etc. What's more, solving system of equations by newton and modified newton method is also supported.

5. Project Structure

- txt_files (contains all txt test cases)
  - test1
  - …
- test (contains all test classes)
  - src
    - all corresponding sources files
  - TestBase.h
  - RandomTest.h
  - Test_A.h
  - …
- nonlinear (contain major classes for project)
  - src
    - all corresponding source files
  - NonLinearEquation.h
  - NonLinearSolver.h
  - …
- lib
  - external libraries (exprtk and googletest)
- CMakeList.txt
- **…**

6. List of Tests (Validated)

6.1 Test from existing test cases

In total, we have 4 pre-defined test cases in the form of test classes: 2 cases of single nonlinear equations and 2 cases of systems of equations:

Test_A: f(x) = exp(x) – 1, the solution is x = 0;

Test_B: f(x) = exp(x) – x – 2, the solution is x1 = -1.8414 and x2 = 1.14619;

Test_C: f1(x, y) = 2 * x ^ 2 – y + 1, f2(x, y) = x + 2 * y – 6, the solution is x = 0.882782 and y = 2.55861;

Test_D: f1(x, y) = x ^ 3 + y – 1, f2(x, y) = x – y ^ 3 – 1, the solution is x = 1.0 and y = 0.0

6.2 Test from keyboard input

It depends on the input entered by the user.

6.3 Test from txt files

It contains 6 different nonlinear equation or system of equations test cases. You may find more information about them by compiling our program. Otherwise, you may check the Readme.md file.

7. Limitations and Problems

7.1 User Interface is very basic

At present, our user interface is simple and naïve, which requires a lot of user operations. And the input should be very careful to avoid problems.

7.2 Algorithms are not stable enough

There are two unstable parts in our program. The first is that our program lacks ability detecting inappropriate input. This may cause unexpected error.

Another part is related to the implementation of algorithms. In our implementation, especially Newton method for nonlinear systems (2 variables or more), we are using some algorithms to solve linear systems. We implement such algorithm by ourselves. As far, these implementations are not stable enough to consider general problems. Also, we implement matrix operations by ourselves, which may also encounter some problems. We have to deal with the allocated space carefully to avoid memory leak.

7.3 Only support less than (or equal to) 4 variables

Now, our tool only supports nonlinear systems equal to (or less than) 4, which involves hard-coding. We still need a way to develop more dynamic programs.

8. TODOs and perspectives

From the limitations and problems of projects, we may develop a more sophisticated user interface (even GUI) to make it more user friendly. Also, we can use external libraries to increase stability of our algorithms. For example, we can use “eigen” to help use solve linear systems problem even the representation of matrix.

What’s more, we can also deal with exceptions and unexpected errors to make our program more stable.