

MATH-458 Programming Concepts in Scientific Computing: Nonlinear Solvers

Michael Jaquier Alexander Lorkowski

Contents

1	Installation	1
2	List of Features	1
3	Usage	1
3.1	Command Line Flags	1
3.2	Examples	2
4	Tests	2
5	TODO	2
6	Algorithms	3
6.1	Bisection Method	3
6.2	Newton/Quasi-Newton Methods	3
6.3	Fixed Point Iteration	4
6.4	Aitken Method	4

1 Installation

This program requires C++11 minimum.

Step 1: Clone the repository.

```
$ git clone https://c4science.ch/diffusion/1726/pcsc-nonlinear-systems-project.git
```

Or

```
$ git clone ssh://git@c4science.ch/diffusion/1726/pcsc-nonlinear-systems-project.git
```

Step 2: Use CMAKE to prepare the source code for compilation.

```
$ cmake -DCMAKE_BUILD_TYPE= -G "CodeBlocks - Unix Makefiles"
```

Step 3: Compile the code

```
$ make
```

2 List of Features

This program includes a family of Nonlinear solvers. Precisely, the following five methods are available: the Aitken Method, the Bisection Method, the Chord Method, and the Newton Method. Description of each algorithm in detail are present in Section 6.

After compiling the program via *make*, it can be run by typing,

```
$ ./RunMain
```

into a terminal followed by command line options as described in Section 3.

We provide a few example files to demonstrate the formatting necessary to load systems of equations.

3 Usage

3.1 Command Line Flags

Scalar Usage:

```
./RunMain -m %s -e '%s' -d '%s' -xi %d -nmax %i -t %d -v %b -l %d -u %d -mod %i
```

System of Equations Usage:

```
./RunMain -f %s -j '%s' -xv '%s' -m %s -nmax %i -t %d -v %b -mod %i
```

Scalar Nonlinear Solver Requirement:

-m,--method	Specify Non-Linear Solver
-e,--expression	Mathematical expression to solve enclosed in ''
-d,--derivative	The derivative to solve enclosed in '' [only for the Newton]

System of Equations Requirement:

-f	File containing the system of equations
-j	File containing the jacobian of the system of equations
-f	File containing the initial vector

Optional:

-h,--help	Show this help message
-xi	Initial guess of the solution [default: 0.0]

-nmax	Maximum number of iterations [default: 1000]
-t	The convergence tolerance [default: 0.001]
-v	print all intermediate calculations [default: false]
-l	The lower bound of the search interval [default: -1.0]
-u	The upper bound of the search interval [default: 1.0]
-mod	The multiplicity of the root of the equation

Non-Linear Solvers:

aitken	Aitken Method
bisection	Bisection Method
chord	Chord Method
fixedpoint	Fixed Point Method
newton	Newton Method

3.2 Examples

Aitken Method:

```
./RunMain -m aitken -e '2x-5'
```

Fixed Point Method:

```
./RunMain -m fixedpoint -e 'cos(x)'
```

Bisection Method:

```
./RunMain -m bisection -e '2x-5' -l -5 -u 10
```

Newton Method with an equation and derivative:

```
./RunMain -m Newton -e 'x^2-5' -d '2x'
```

Newton Method for system of equations:

```
./RunMain -m Newton -f Equations.txt -j Jacobian.txt -xv initialVector.txt
```

4 Tests

The program will test all Algorithms, and sub-tools utilized to ensure correct results. The tests can be initiated by running them via:

```
$ ./RunFunctionTests
```

5 TODO

1. Expand system of equations to the Secant method.
2. Expand system of equations to the Bisection method.
3. Expand system of equations beyond 3D.
4. Implement intelligent starting value(s) if not provided by user
5. Implement intelligent value variance to overcome zero derivative
6. Implement GUI interface for inputs

6 Algorithms

6.1 Bisection Method

The bisection method is based on Theorem 1.

Theorem 1. (*Zeros of a continuous functions*)

Given a continuous function $f : [a, b] \rightarrow \mathbb{R}$, such that $f(a)f(b) < 0$, then \exists at least an $\alpha \in (a, b)$ such that $f(\alpha) = 0$.

Starting from $I_0 = [a, b]$, the bisection method generates a sequence of subintervals $I_k = [a(k), b(k)]$, $k \geq 0$, with $I_k \subset I_{k-1}$, $k \geq 1$, and enjoys the property that $f(a(k))f(b(k)) < 0$.

Algorithm 1 Bisection Method

```

1:  $k \leftarrow 0$ ,  $a^{(0)} \leftarrow a$ ,  $b^{(0)} \leftarrow b$ ,  $x^{(0)} \leftarrow (a^{(0)} + b^{(0)})/2$ 
2: while  $x^{(k)} - x^{(k-1)} > \text{tol}$  and  $k < k_{\max}$  do
3:   if  $f(x^{(k-1)}) == 0$  then return  $\alpha = x^{(k-1)}$ 
4:   else
5:     if  $f(x^{(k-1)})f(a^{(k-1)}) < 0$  then
6:       set  $a^{(k)} = a^{(k-1)}$  and  $b^{(k)} = x^{(k-1)}$ 
7:     if  $f(x^{(k-1)})f(b^{(k-1)}) < 0$  then
8:       set  $a^{(k)} = x^{(k-1)}$  and  $b^{(k)} = b^{(k-1)}$ 
9:     set  $x^{(k)} = (a^{(k)} + b^{(k)})/2$ 
10: return  $\alpha = x^{(k)}$ 

```

6.2 Newton/Quasi-Newton Methods

Assuming that $f \in C^0(I)$ and is differentiable in the interval $I = (a, b) \subseteq \mathbb{R}$, the equation of a tangent line to the curve $(x, f(x))$ at coordinate $x^{(k)}$, where $x^{(k)} \in I$, is $y(x) = f(x^{(k)}) + df(x^{(k)})(x - x^{(k)})$. If we assume $y(x^{(k)}) = 0$, $x^{(k+1)}$ can be computed from Eq. 1.

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q^{(k)}} \quad (1)$$

with $q^{(k)}$ determining the method as Newton (Eq. 2) or Chord (Eq. 3):

$$q^{(k)} = \frac{df(x^{(k)})}{dx} \quad (2)$$

$$q^{(k)} = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad (3)$$

Algorithm 2 General quasi-Newton Method

```

1:  $k \leftarrow 0$ , set initial guess  $x^{(0)}$ 
2: while  $x^{(k)} - x^{(k-1)} > \text{tol}$  and  $k < k_{\max}$  do
3:    $x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q^{(k)}}$ 
4:    $k = k + 1$ ;
5: return  $\alpha = x^{(k)}$ 

```

6.3 Fixed Point Iteration

For a given $f : [a,b] \rightarrow \mathbb{R}$, the problem can be transformed from $f(x) = 0$ into an equivalent problem $x - \phi(x) = 0$. The auxiliary function $\phi : [a,b] \rightarrow \mathbb{R}$ has to be chosen in such a way that $\phi(\alpha) = \alpha$ whenever $f(\alpha) = 0$. Finding the fixed points of the mapping ϕ thus results in finding the roots of the original equation $f(x)$, hence the name fixed point iteration.

Algorithm 3 Fixed point iterations

```
1:  $k \leftarrow 0$ , set initial guess  $x^{(0)}$ 
2: while  $x^{(k)} - x^{(k-1)} > \text{tol}$  and  $k < k_{\max}$  do
3:    $x^{(k+1)} = \phi(x^{(k)})$ 
4:    $k = k + 1$ ;
5: return  $\alpha = x^{(k)}$ 
```

6.4 Aitken Method

Aitken's method provides a way to accelerate the convergence of iterative methods for finding the roots of a function. The general algorithm is shown below.

Algorithm 4 Aitken Method

```
1:  $k \leftarrow 0$ , set initial guess  $x^{(0)}$ 
2: Calculate  $x^{(1)}$  using any iterative method.
3: while  $x^{(k)} - x^{(k-1)} > \text{tol}$  and  $k < k_{\max}$  do
4:   Calculate  $x^{(k+2)}$ .
5:   Modify  $x^{(k+2)}$  using  $x^{(k+2)} = x^{(k)} - \frac{(x^{(k+1)} - x^{(k)})^2}{(x^{(k)} - 2x^{(k+1)} + x^{(k+2)})}$ 
6:    $k = k + 1$ ;
7: return  $\alpha = x^{(k)}$ 
```
