Scientific Computing: Solving Nonlinear Equations

Aleksandar Donev
Courant Institute, NYU

donev@courant.nyu.edu

1 Course MATH-GA.2043 or CSCI-GA.2112, Fall 2020

October 8th, 2020
1. Basics of Nonlinear Solvers
2. One Dimensional Root Finding
3. Systems of Non-Linear Equations
Outline

1. Basics of Nonlinear Solvers
2. One Dimensional Root Finding
3. Systems of Non-Linear Equations
Simplest problem: **Root finding** in one dimension:

\[ f(x) = 0 \text{ with } x \in [a, b] \]

Or more generally, solving a **square system of nonlinear equations**

\[ f(x) = 0 \Rightarrow f_i(x_1, x_2, \ldots, x_n) = 0 \text{ for } i = 1, \ldots, n. \]

There can be no closed-form answer, so just as for eigenvalues, we need **iterative methods**.

Most generally, starting from \( m \geq 1 \) **initial guesses** \( x^0, x^1, \ldots, x^m \), iterate:

\[ x^{k+1} = \phi(x^k, x^{k-1}, \ldots, x^{k-m}). \]
Consider one dimensional root finding and let the actual root be $\alpha$, $f(\alpha) = 0$.

A sequence of iterates $x^k$ that converges to $\alpha$ has order of convergence $p \geq 1$ if as $k \to \infty$

$$\frac{|x^{k+1} - \alpha|}{|x^k - \alpha|^p} = \frac{|e^{k+1}|}{|e^k|^p} \to C = \text{const},$$

where the constant $C$ is a convergence factor, $C < 1$ for $p = 1$.

A method should at least converge linearly ($p = 1$), that is, the error should at least be reduced by a constant factor every iteration, for example, the number of accurate digits increases by 1 every iteration.

A good method for root finding converges quadratically ($p = 2$), that is, the number of accurate digits doubles every iteration!
A good initial guess is extremely important in nonlinear solvers!

Assume we are looking for a unique root \( a \leq \alpha \leq b \) starting with an initial guess \( a \leq x_0 \leq b \).

A method has local convergence if it converges to a given root \( \alpha \) for any initial guess that is sufficiently close to \( \alpha \) (in the neighborhood of a root).

A method has global convergence if it converges to the root for any initial guess.

General rule: Global convergence requires a slower (careful) method but is safer.

It is best to combine a global method to first find a good initial guess close to \( \alpha \) and then use a faster local method.
Conditioning of root finding

\[ f(\alpha + \delta \alpha) \approx f(\alpha) + f'(\alpha)\delta \alpha = \delta f \]

\[ |\delta \alpha| \approx \left| \frac{\delta f}{f'(\alpha)} \right| \quad \Rightarrow \quad \kappa_{abs} = \left| f'(\alpha) \right|^{-1}. \]

- The problem of finding a **simple root** is well-conditioned when \( |f'(\alpha)| \) is far from zero.
- Finding **roots with multiplicity** \( m > 1 \) is **ill-conditioned**:

\[ |f'(\alpha)| = \cdots = \left| f^{(m-1)}(\alpha) \right| = 0 \quad \Rightarrow \quad |\delta \alpha| \approx \left[ \frac{|\delta f|}{|f^m(\alpha)|} \right]^{1/m} \]

- Note that finding **roots of algebraic equations** (polynomials) is a separate subject of its own that we skip.
Outline

1. Basics of Nonlinear Solvers
2. One Dimensional Root Finding
3. Systems of Non-Linear Equations
The bisection and Newton algorithms
Bisection

- First step is to **locate a root** by searching for a **sign change**, i.e., finding $a^0$ and $b^0$ such that
  \[ f(a^0)f(b^0) < 0. \]
- The simply **bisect** the interval, for $k = 0, 1, \ldots$
  \[ x^k = \frac{a^k + b^k}{2} \]
  and choose the half in which the function changes sign, i.e., either $a^{k+1} = x^k$, $b^{k+1} = b^k$ or $b^{k+1} = x^k$, $a^{k+1} = a^k$ so that
  \[ f(a^{k+1})f(b^{k+1}) < 0. \]
- Observe that each step we need one **function evaluation**, $f(x^k)$, but only the sign matters.
- The **convergence is essentially linear** because
  \[ |x^k - \alpha| \leq \frac{b^k}{2^{k+1}} \Rightarrow \frac{|x^{k+1} - \alpha|}{|x^k - \alpha|} \leq 2. \]
Newton’s Method

- Bisection is a slow but sure method. It uses no information about the value of the function or its derivatives.
- Better convergence, of order $p = (1 + \sqrt{5})/2 \approx 1.63$ (the golden ratio), can be achieved by using the value of the function at two points, as in the secant method.
- Achieving second-order convergence requires also evaluating the function derivative.
- **Linearize the function** around the current guess using Taylor series:

  $$f(x^{k+1}) \approx f(x^k) + (x^{k+1} - x^k)f'(x^k) = 0$$

  $$x^{k+1} = x^k - \frac{f(x^k)}{f'(x^k)}$$
Convergence of Newton’s method

Use Taylor series with remainder and divide by \( f'(x^k) \neq 0 \):

\[
\exists \xi \in [x_n, \alpha] : \quad f(\alpha) = 0 = f(x^k) + (\alpha - x^k)f'(x^k) + \frac{1}{2} (\alpha - x^k)^2 f''(\xi) = 0,
\]

\[
\left[ x^k - \frac{f(x^k)}{f'(x^k)} \right] - \alpha = -\frac{1}{2}(\alpha - x^k)^2 \frac{f''(\xi)}{f'(x^k)}
\]

\[
x^{k+1} - \alpha = e^{k+1} = -\frac{1}{2} (e^k)^2 \frac{f''(\xi)}{f'(x^k)}
\]

which shows second-order convergence

\[
\frac{|x^{k+1} - \alpha|}{|x^k - \alpha|^2} = \frac{|e^{k+1}|}{|e^k|^2} = \frac{f''(\xi)}{2f'(x^k)} \rightarrow \frac{f''(\alpha)}{2f'(\alpha)}
\]
For convergence we want $|e^{k+1}| < |e^k|$ so we want

$$|e^k| \left| \frac{f''(\alpha)}{2f'(\alpha)} \right| < 1 \quad \Rightarrow \quad |e^k| < \left| \frac{2f'(\alpha)}{f''(\alpha)} \right|$$

Newton’s method converges quadratically if we start sufficiently close to a simple root, more precisely, if

$$|x^0 - \alpha| = |e^0| \lesssim \left| \frac{2f'(\alpha)}{f''(\alpha)} \right|.$$

This is just a rough estimate, not a precise bound!

A robust but fast algorithm for root finding would safeguard Newton’s method with bisection: Eventually we will accept all Newton steps once close to the root, so we will get quadratic convergence, but also be guaranteed to converge to a root.
fixed-point iteration

\[ f(x) = 0 \quad \Rightarrow \quad x = f(x) + x = \phi(x) \]

- Then we can use **fixed-point iteration**

\[ x^{k+1} = \phi(x^k) \]

- whose fixed point (limit), if it converges, is \( x \to \alpha \). Taylor series estimate:

\[
x^{k+1} = \alpha + e^{k+1} \approx \phi(\alpha) + \phi'(\alpha) (x^k - \alpha) = \alpha + \phi'(\alpha)e^k \quad \Rightarrow
\]

\[ e^{k+1} \approx \phi'(\alpha)e^k \quad \Rightarrow \quad \text{we want} \quad |\phi'(\alpha)| < 1. \]

- It can be proven that the fixed-point iteration converges if \( \phi(x) \) is a **contraction mapping**:

\[
|\phi'(x)| \leq K < 1 \quad \forall x \in [a, b]
\]

[If \( \phi(x) \) is **Lipschitz continuous** with Lipschitz constant \( L < 1 \).]
A good library function for root finding has to implement careful termination criteria.

An obvious option is to terminate when the residual becomes small

$$|f(x^k)| < \varepsilon,$$

which works for very well-conditioned problems, $|f'(\alpha)| \sim 1$.

Another option is to terminate when the increment becomes small

$$|x^{k+1} - x^k| < \varepsilon.$$

For example, for fixed-point iteration this test would stop at step $k$:

$$x^{k+1} - x^k = e^{k+1} - e^k \approx \left[1 - \phi'(\alpha)\right] e^k \Rightarrow |e^k| \approx \frac{\varepsilon}{\left[1 - \phi'(\alpha)\right]},$$

so we see that the increment test works for rapidly converging iterations, i.e., when $|1 - \phi'(\alpha)|$ is not small.
In practice

A robust but fast algorithm for root finding would combine (safeguard) bisection with Newton’s method: Given a current bisection interval \([a, b]\), if \(x^{k+1} \in (a, b)\) then accept Newton step, otherwise just set \(x^{k+1} = (a + b)/2\). Take new bisection interval to be either \([a, x^{k+1}]\) or \([x^{k+1}, b]\) the same way as in bisection where we always use \(x^{k+1} = (a + b)/2\).

Newton’s method requires first-order derivatives so often other methods are preferred that require function evaluation only. Examples include secant method (based on linear interpolation) or inverse quadratic interpolation (fit a parabola through three past points \((f(x_i), x_i), i = 1, 2, 3\), and evaluate for zero argument to give a new point).

Matlab’s function \(fzero\) combines bisection, secant and inverse quadratic interpolation and is “fail-safe”. See, for example, “Brent’s method” on Wikipedia.
Find zeros of $a \sin(x) + b \exp(-x^2/2)$

% f=@mfile uses a function in an m-file

% Parameterized functions are created with:
\[ a = 1; \quad b = 2; \]
\[ f = @(x) \quad a \cdot \sin(x) + b \cdot \exp(-x^2/2) \quad ; \quad \% \quad \text{Handle} \]

\texttt{figure(1)}
\texttt{ezplot(f,[-5,5]); grid}

\texttt{x1=fzero(f, [-2,0])}
\texttt{[x2,f2]=fzero(f, 2.0)}

\begin{align*}
x1 & = -1.227430849357917 \\
x2 & = 3.155366415494801 \\
f2 & = -2.116362640691705e-16
\end{align*}
Figure of $f(x)$

$$a \sin(x) + b \exp(-x^2/2)$$
Outline

1. Basics of Nonlinear Solvers
2. One Dimensional Root Finding
3. Systems of Non-Linear Equations
It is convenient to focus on one of the equations, i.e., consider a scalar function $f(x)$.

The usual Taylor series is replaced by

$$f(x + \Delta x) = f(x) + g^T (\Delta x) + \frac{1}{2} (\Delta x)^T H (\Delta x)$$

where the gradient vector is

$$g = \nabla_x f = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right]^T$$

and the Hessian matrix is

$$H = \nabla_x^2 f = \left\{ \frac{\partial^2 f}{\partial x_i \partial x_j} \right\}_{ij}$$
We are after solving a **square system of nonlinear equations** for some variables $\mathbf{x}$:

$$f(\mathbf{x}) = 0 \Rightarrow f_i(x_1, x_2, \ldots, x_n) = 0 \text{ for } i = 1, \ldots, n.$$

The first-order Taylor series is

$$f(\mathbf{x}^k + \Delta \mathbf{x}) \approx f(\mathbf{x}^k) + [\mathbf{J}(\mathbf{x}^k)] \Delta \mathbf{x} = 0$$

where the Jacobian $\mathbf{J}$ has the gradients of $f_i(\mathbf{x})$ as rows:

$$[\mathbf{J}(\mathbf{x})]_{ij} = \frac{\partial f_i}{\partial x_j}$$
Newton’s Method for Systems of Equations

- It is much harder if not impossible to do globally convergent methods like bisection in higher dimensions!
- A good initial guess is therefore a must when solving systems, and Newton’s method can be used to refine the guess.
- The basic idea behind Newton’s method is to linearize the equation around the current guess:

\[ f(x^k + \Delta x) \approx f(x^k) + [J(x^k)] \Delta x = 0 \]

\[ [J(x^k)] \Delta x = -f(x^k) \text{ but denote } J \equiv J(x^k) \]

\[ x^{k+1} = x^k + \Delta x = x^k - J^{-1}f(x^k). \]

- This method requires computing a whole matrix of derivatives, which can be expensive or hard to do (differentiation by hand?)!
Convergence of Newton’s method

- Near the root the Jacobian and Hessian don’t change much so just approximate $J \approx J(\alpha)$ and $H \approx H(\alpha)$.
- Next order term in Taylor series indicates error

$$f(x^k) = f(\alpha) + Je^k + \frac{1}{2} (e^k)^T He^k = Je^k + \frac{1}{2} (e^k)^T He^k \implies$$

$$e^{k+1} = x^{k+1} - \alpha = e^k - J^{-1}f(x^k) = \frac{1}{2} J^{-1} (e^k)^T He^k$$

- Newton’s method converges **quadratically** if started sufficiently close to a root $\alpha$:

$$\|e^{k+1}\| \leq \frac{\|J^{-1}\| \|H\|}{2} \|e^k\|^2$$

- Newton’s method converges fast **if the Jacobian $J(\alpha)$ is well-conditioned**.
- Newton’s method requires solving **many linear systems**, which can be expensive for many variables.
For large systems one can use so called quasi-Newton methods to estimate derivatives using finite-differences and to speed up by using rank-1 matrix updates (see Woodbury formula in homework 2):

- Approximate the Jacobian with another matrix $\tilde{J}^k$ and solve $\tilde{J}^k d = f(x^k)$.
- Damp the step by a step length $\alpha_k \lesssim 1$,
  $$x^{k+1} = x^k + \alpha_k d = x^k + \Delta x^k.$$  
- Update Jacobian by a low-rank update that ensures the secant condition
  $$f(x^{k+1}) - f(x^k) = \tilde{J}^{k+1} \Delta x^k.$$  
- An example is the (recall Woodbury formula from Homework 2!) rank-1 update in Broyden’s method:
  $$\tilde{J}^{k+1} = \tilde{J}^k + \left( f(x^{k+1}) - \left( f(x^k) + \tilde{J}^k \Delta x^k \right) \right) \frac{(\Delta x^k)^T}{\|\Delta x^k\|_2^2}.$$
Continuation methods

- To get a good initial guess for Newton’s method and ensure that it converges fast we can use **continuation methods** (also called **homotopy methods**).
- The basic idea is to solve
  \[
  \tilde{f}_\lambda(x) = \lambda f(x) + (1 - \lambda) f_a(x) = 0
  \]
  instead of the original equations, where \(0 \leq \lambda \leq 1\) is a parameter.
- If \(\lambda = 1\), we are solving the original equation \(f(x) = 0\), which is hard because we do not have a good guess for the initial solution.
- If \(\lambda = 0\), we are solving \(f_a(x) = 0\), and we will assume that this is **easy to solve**. For example, consider making this a linear function,
  \[
  f_a(x) = x - a,
  \]
  where \(a\) is a vector of parameters that need to be chosen somehow.
  One can also take a more general \(f_a(x) = Ax - a\) where \(A\) is a matrix of parameters, so that solving \(f_a(x) = 0\) amounts to a linear solve which we know how to do already.
The basic idea of continuation methods is to start with $\lambda = 0$, and solve $\tilde{f}_\lambda (x) = 0$. This gives us a solution $x_0$.

Then increment $\lambda$ by a little bit, say $\lambda = 0.05$, and solve $\tilde{f}_\lambda (x)$ using Newton’s method starting with $x_0$ as an initial guess. Observe that this is a good initial guess under the assumption that the solution has not changed much because $\lambda$ has not changed much.

We can repeat this process until we reach $\lambda = 1$, when we get the actual solution we are after:

- Choose a sequence $\lambda_0 = 0 < \lambda_1 < \lambda_2 < \cdots < \lambda_{n-1} < \lambda_n = 1$.
- For $k = 0$ solve $f_a (x_0) = 0$ to get $x_0$.
- For $k = 1, \ldots, n$, solve a nonlinear system to get $x_k$,

$$\tilde{f}_{\lambda_k} (x_k) = 0$$

using Newton’s method starting from $x_{k-1}$ as an initial guess.
Path Following

- Observe that if we change λ very slowly we have hope that the solution will trace a **continuous path of solutions**.
- That is, we can think of \( x(\lambda) \) as a continuous function defined on [0, 1], defined implicitly via

\[
\lambda f(x(\lambda)) + (1 - \lambda) f_a(x(\lambda)) = 0.
\]

- This rests on the assumption that this path will **not have turning points, bifurcate or wonder to infinity**, and that there is a solution for every λ.
- It turns out that by a judicious choice of \( f_a \) one can insure this is the case. For example, choosing a random \( a \) and taking \( f_a(x) = x - a \) works.
- The trick now becomes how to choose the sequence \( \lambda_k \) to make sure \( \lambda \) changes not too much but also not too little (i.e., not too slowly), see HOMPACK library for an example.
In practice

- It is much harder to construct general robust solvers in higher dimensions and some **problem-specific knowledge** is required.
- There is no built-in function for solving nonlinear systems in MATLAB, but the **Optimization Toolbox** has `fsolve`.
- In many practical situations there is some continuity of the problem so that a **previous solution can be used as an initial guess**.
- For example, **implicit methods for differential equations** have a time-dependent Jacobian $J(t)$ and in many cases the solution $x(t)$ evolves smoothly in time.
- For large problems specialized **sparse-matrix solvers** need to be used.
- In many cases derivatives are not provided but there are some techniques for **automatic differentiation**.
Root finding is well-conditioned for **simple roots** (unit multiplicity), ill-conditioned otherwise.

Methods for solving nonlinear equations are always iterative and the **order of convergence** matters: second order is usually good enough.

A good method uses a higher-order unsafe method such as **Newton method** near the root, but **safeguards** it with something like the **bisection** method.

Newton’s method is second-order but requires derivative/Jacobian evaluation. In **higher dimensions** having a **good initial guess** for Newton’s method becomes very important.

**Quasi-Newton** methods can alleviate the complexity of solving the Jacobian linear system.