ROB 101 - Fall 2021

Solutions of Nonlinear Equations

(Newton-Raphson for Vector Functions)

November 10, 2021

UNIVERSITY OF MICHIGAN

Learning Objectives

- Extend our horizons from linear equations to nonlinear equations.
- Appreciate the power of using algorithms to iteratively construct approximate solutions to a problem.
- Accomplish all of this without assuming a background in Calculus.

Outcomes

- Linear approximations of nonlinear functions.
- The Newton-Raphson algorithm for vector functions.

Recall: Linear Approximation of $f : \mathbb{R}^m \to \mathbb{R}^n$

Consider a function $f : \mathbb{R}^m \to \mathbb{R}^n$. We seek a means to build a linear approximation of the function near a given point $x_0 \in \mathbb{R}^m$.

- When $m = n = 1$, we can approximate a function by
$$f(x) \approx f(x_0) + \frac{df(x_0)}{dx}(x - x_0) =: f(x_0) + a(x - x_0).$$
- For the general case of $f : \mathbb{R}^m \to \mathbb{R}^n$, can we find an $n \times m$ matrix A such that
$$f(x) \approx f(x_0) + A(x - x_0).$$

Recall: Linear Approximation of $f : \mathbb{R}^m \to \mathbb{R}^n$

- A function $f : \mathbb{R}^m \to \mathbb{R}^n$ is called a vector-valued function.
- We compute its best linear approximation via the Jacobian $A := \frac{\partial f(x)}{\partial x}\Big|_{x=x_0}$ such that

$$f(x) \approx f(x_0) + A(x - x_0).$$

- The input is a vector such as $x \in \mathbb{R}^m$. If we use the standard basis of $\mathbb{R}^m$, we have

$$x = x_1 e_1 + x_2 e_2 + \cdots + x_n e_m = \sum_{i=1}^{m} x_i e_i.$$

- Then we can use a finite difference approximation to compute each column of $A = \begin{bmatrix} a_1^{\mathrm{col}} & \ldots & a_m^{\mathrm{col}} \end{bmatrix}$ as

$$\boxed{a_i^{\mathrm{col}} = \frac{\partial f(x_0)}{\partial x_i} = \frac{f(x_0 + he_i) - f(x_0 - he_i)}{2h}}$$

- We consider functions $f : \mathbb{R}^n \to \mathbb{R}^n$ and seek a root $f(x_0) = 0$.
- Note that the domain and range are both $\mathbb{R}^n$ and thus this is the nonlinear equivalent of solving a square linear equation $Ax - b = 0$.
- We recall that $\det(A) \neq 0$ was our magic condition for the existence and uniqueness of solutions to $Ax - b = 0$.

- Let x_k be our current approximation of a root of the function f.

- We write the linear approximation of f about x_k as

$$f(x) \approx f(x_k) + A(x - x_k), \quad A = \frac{\partial f(x_k)}{\partial x}.$$

Newton-Raphson for Vector Functions

- We want to chose x_{k+1} so that $f(x_{k+1}) = 0$.

- $f(x_{k+1}) = f(x_k) + A(x_{k+1} - x_k) = 0$.

- If $\det(A) \neq 0$, we could naively solve for x_{k+1}, giving us

$$\boxed{x_{k+1} = x_k - A^{-1} f(x_k).}$$

Remark

By now, you know well that explicitly inverting the Jacobian matrix A is not among desired methods for algorithmic implementations. We wish to find x_{k+1} rather than A^{-1}.

Let's break $x_{k+1} = x_k - A^{-1}f(x_k)$ for finding x_{k+1} into two steps.

- Define $\Delta x_k := x_{k+1} - x_k$. Then $x_{k+1} = x_k + \Delta x_k$.
- $f(x_{k+1}) = 0 \implies A\Delta x_k = -f(x_k)$.

To summarize:

1. Start with an initial guess x_0 ($k = 0$).
2. Solve the linear system $A\Delta x_k = -f(x_k)$.
3. Update the estimated root $x_{k+1} = x_k + \Delta x_k$.
4. Repeat (go back to 2) until convergence.

Damped Newton-Raphson Algorithm

Remark

In practice, if we take a full step using Δx_k, the algorithm might not converged. A solution is to use a step size to control how large each step should be

$$x_{k+1} = x_k + \epsilon \Delta x_k.$$

The step size $\epsilon > 0$ (α is a common notation too) can be fixed or found at each iteration using a method (often called "line search").

Find a root of $F : \mathbb{R}^4 \to \mathbb{R}^4$ near $x_0 = \begin{bmatrix} -2.0 & 3.0 & \pi & -1.0 \end{bmatrix}^\mathsf{T}$ for

$$F(x) = \begin{bmatrix} x_1 + 2x_2 - x_1(x_1 + 4x_2) - x_2(4x_1 + 10x_2) + 3 \\ 3x_1 + 4x_2 - x_1(x_1 + 4x_2) - x_2(4x_1 + 10x_2) + 4 \\ 0.5\cos(x_1) + x_3 - (\sin(x_3))^7 \\ -2(x_2)^2 \sin(x_1) + (x_4)^3 \end{bmatrix}.$$

Example

We used a symmetric difference approximation for the derivatives, with $h = 0.1$. Below are the first five results from the algorithm:

$$x_k = \begin{bmatrix} k=0 & k=1 & k=2 & k=3 & k=4 & k=5 \\ \\ -2.0000 & -3.0435 & -2.4233 & -2.2702 & -2.2596 & -2.2596 \\ 3.0000 & 2.5435 & 1.9233 & 1.7702 & 1.7596 & 1.7596 \\ 3.1416 & 0.6817 & 0.4104 & 0.3251 & 0.3181 & 0.3181 \\ -1.0000 & -1.8580 & -2.0710 & -1.7652 & -1.6884 & -1.6846 \end{bmatrix}$$

and

$$f(x_k) = \begin{bmatrix} k=0 & k=1 & k=2 & k=3 & k=4 & k=5 \\ \\ -39.0000 & -6.9839 & -1.1539 & -0.0703 & -0.0003 & -0.0000 \\ -36.0000 & -6.9839 & -1.1539 & -0.0703 & -0.0003 & -0.0000 \\ 2.9335 & 0.1447 & 0.0323 & 0.0028 & 0.0000 & -0.0000 \\ 15.3674 & -5.1471 & -4.0134 & -0.7044 & -0.0321 & -0.0001 \end{bmatrix}.$$

Let's switch to the Julia notebook.

Next Time

- Optimization
- Read Chapter 12 of ROB 101 Book