SYSTEMS DEVELOPMENT FOR COMPUTATIONAL SCIENCE

LECTURE 10

Fabian Wermelinger
Harvard University
CS107 / AC207

Tuesday, October 4th 2022

LAST TIME

- Class methods, static methods and instance methods
- Python modules
- Python packages and the Python package index (PyPI)
- Build a Python package and publish on https://test.pypi.org/

TODAY

Main topics: ***Derivatives and their role in Science, Newton's method***

Details:

- Towards automatic differentiation
- Linearization of Euler equations as an example for the Jacobian
- Newton's method

AGENDA CHECK:

- Milestone M1B is due today → first touch with GitHub Actions to prepare your project repository for task automation using continuous integration (CI). ***Many teams already did it, great!*** 😄

INTRODUCTION AND MOTIVATION

Differentiation is one of the most important operations in science.

- Finding extrema of functions and determining zeros of functions are central to optimization.
- Linearization of non-linear equations requires a prediction for a change in a small neighborhood which involves derivatives.
- *Numerically* solving differential equations forms a cornerstone of modern science and engineering and is intimately linked with predictive science.

INTRODUCTION AND MOTIVATION

Euler equations: a system of *partial differential equations* (PDEs) to describe compressible ***fluid motion*** in the form of *conservation laws*:

- Conservation of *mass* ρ
- Conservation of *momentum* $\rho\mathbf{u}$
- Conservation of *energy* E

The $\partial/\partial t$ and $\partial/\partial x$ are differential operators that describe the change in time and space of the *conserved* quantities ρ, ρu and E (mass, momentum and energy, respectively).

The Euler Equations:

$$\frac{\partial \rho}{\partial t} + \frac{\partial}{\partial x}(\rho u) = 0$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial}{\partial x}(\rho u^2 + p) = 0$$

$$\frac{\partial E}{\partial t} + \frac{\partial}{\partial x}\big((E + p)u\big) = 0$$

INTRODUCTION AND MOTIVATION

- The Euler equations in the previous slide are highly *non-linear*. If we were to ***linearize*** the equations around a certain point q, we would need to find a so called ***Jacobian*** J of $f(q) \in \mathbb{R}^3$ (the *flux function*), where the input $q = [\rho, \rho u, E]^\intercal \in \mathbb{R}^3$ are the conserved variables.
- We can then find the best linear approximation to $f(q + \Delta q)$ by projecting the Jacobian in the direction of a small change Δq, i.e., $f(q + \Delta q) \approx f(q) + J(q) \cdot \Delta q$. The Jacobian contains the first derivatives $J_{ij} = \partial f_i / \partial q_j$ and is a 3×3 matrix for this example.

INTRODUCTION AND MOTIVATION

- A very frequent occurrence in science requires the scientist to find the zeros of a function $y = f(x)$. The input to the function is an m -dimensional vector $x \in \mathbb{R}^m$ and the function returns an n-dimensional vector $y \in \mathbb{R}^n$. We denote this mathematically as

$$f(x) : \mathbb{R}^m \mapsto \mathbb{R}^n.$$

- This expression is read: the function $f(x)$ maps from $\mathbb{R}^m$ to $\mathbb{R}^n$.

EXAMPLE: NON-LINEAR SYSTEM

Consider the following system of non-linear equations:

$$x_1 {x_2}^3 + \ln({x_3}^2) = \sin(x_1 x_2 x_3)$$

$$x_1 + x_2 + \tan(x_3) = \frac{1}{x_1 x_2 x_3}.$$

We define the vector:

$$x = [x_1, x_2, x_3]^\intercal = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},$$

and say $x \in \mathbb{R}^3$ (the vector x is in $\mathbb{R}^3$, a 3-dimensional vector space).

Following the notation from above $\rightarrow m = 3$.

EXAMPLE: NON-LINEAR SYSTEM

Put everything on one side and define the function:

$$f(x) = \begin{bmatrix} x_1 {x_2}^3 + \ln({x_3}^2) - \sin(x_1 x_2 x_3) \\ x_1 + x_2 + \tan(x_3) - \dfrac{1}{x_1 x_2 x_3} \end{bmatrix} \tag{1}$$

The ***vector function*** $f(x)$ has only *two* components and maps an input $x \in \mathbb{R}^3$ to a smaller vector in $\mathbb{R}^2$. We write:

$$f(x) : \mathbb{R}^3 \mapsto \mathbb{R}^2.$$

If we plug-in numbers, say $x = [1, 2, 1]^\intercal$, we can evaluate the non-linear system *at the point* x:

$$f\left(\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 7.09 \\ 4.06 \end{bmatrix}$$

NEWTON'S METHOD

Often we need to find an x for which $f(x) = 0$. This is *not so difficult for a linear system*, but *for a non-linear system it can be a major challenge*.

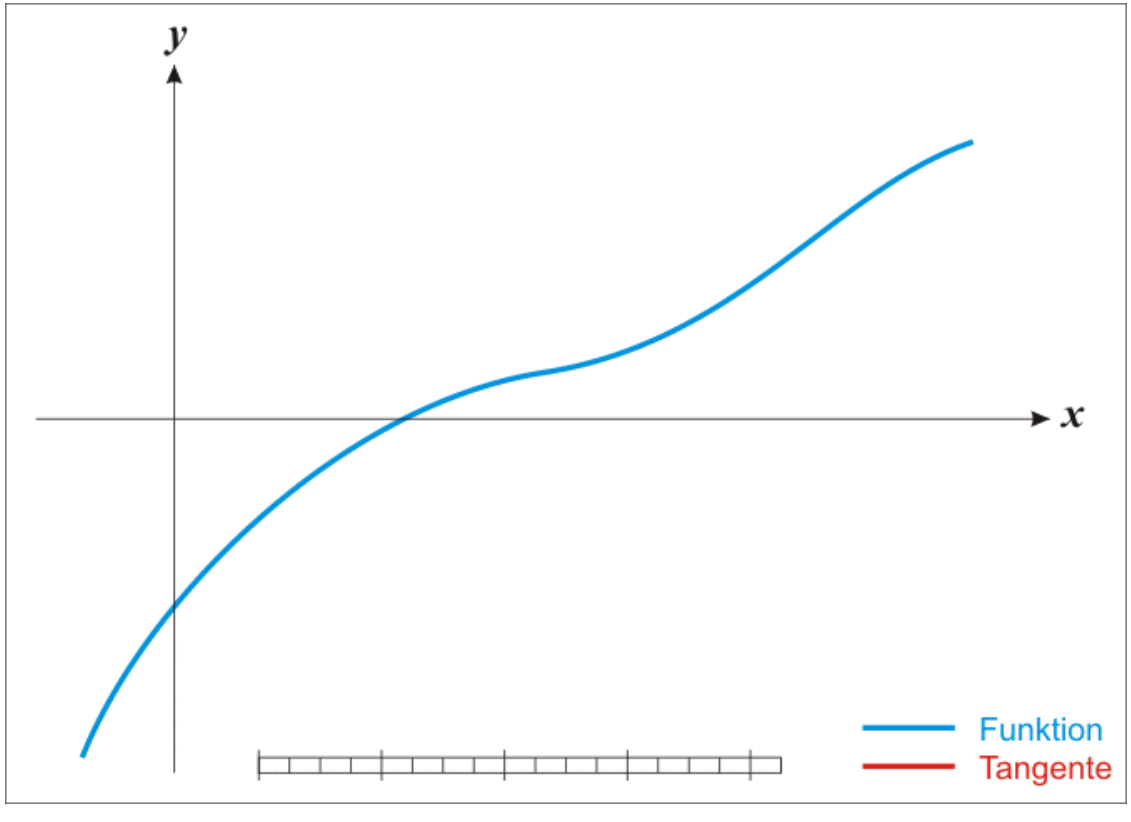
Newton's method is an algorithm with excellent convergence properties that allows us to find the ***roots*** x of a non-linear function f such that $f(x) = 0$. The x that satisfies this equation is called a ***root***.

- Finding the roots of a system $f(x)$ has important practical applications. Finding the roots of a system means solving the system of equations.
- Examples are ray casting algorithms where you need to find the intersection of light rays with objects. Another example finding object trajectories under the action of gravity.

DERIVATION OF NEWTON'S METHOD

The goal is to find $x \in \mathbb{R}^m$ such that $f(x) = 0$ for $f(x) \in \mathbb{R}^n$.

The algorithm in pictures:



(image taken from https://en.wikipedia.org/wiki/Newton%27s_method)

DERIVATION OF NEWTON'S METHOD

***Step 1:** choose an initial guess:*

Newton's method is an ***iterative*** method. We use the notation $x^{(k)}$ for the guess of the root at the k-th iteration. To start the algorithm you pick an *initial guess* at $k = 0$ for which most certainly $f\left(x^{(0)}\right) \neq 0$.

Newton's method is not guaranteed to converge! Convergence depends on a ***good initial guess*** which requires some intuition and experience. When the method does converge, a solution with high accuracy can be found within only few iterations.

DERIVATION OF NEWTON'S METHOD

***Step 2:* explore the neighborhood:**

We look at a point ***just a little beyond*** $x^{(k)}$. That is, we define the next iterate $x^{(k+1)}$ by the following relation:

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)},$$

where $\Delta x^{(k)} = x^{(k+1)} - x^{(k)}$.

Note: we have just introduced $\Delta x^{(k)}$ in our notation but *we do not know what its actual value should be* → ***We will need to find this value.***

DERIVATION OF NEWTON'S METHOD

Step 3: find a relationship between $f(x^{(k+1)})$ and $f(x^{(k)})$:

Since we are looking in the neighborhood of $x^{(k)}$, the main tool we need to establish this relationship is a Taylor series expansion:

$$f(y) = \sum_{\kappa=0}^{\infty} \frac{f^{(\kappa)}(x)}{\kappa!}(y-x)^{\kappa}.$$

(The notation $f^{(\kappa)}(x)$ means the κ-th derivative of f evaluated at x. It is a common mathematical notation and unrelated to $x^{(k)}$)

Apply a truncated Taylor series:

We substitute $y = x^{(k)} + \Delta x^{(k)}$ *(new guess for the root)* and $x = x^{(k)}$ *(current guess for the root)* and find:

$$f(x^{(k)} + \Delta x^{(k)}) = f(x^{(k)}) + \left.\frac{\partial f}{\partial x}\right|_{x=x^{(k)}} \Delta x^{(k)} + \text{h.o.t.}$$

DERIVATION OF NEWTON'S METHOD

Step 4: simplify:

As our derivation is based on an *iterative* correction to $x^{(k)}$, we can argue that we may omit the ***higher order terms (h.o.t.)*** in our previous result and therefore *approximate* the *exact* solution by performing few more iterations.

This simplifies to the following approximation:

$$f(x^{(k)} + \Delta x^{(k)}) \approx f(x^{(k)}) + \left.\frac{\partial f}{\partial x}\right|_{x=x^{(k)}} \Delta x^{(k)}$$

DERIVATION OF NEWTON'S METHOD

***Step 5:** add iteration criterion:*

We require a root for which $f(x^{(k+1)}) = 0$ which implies that $\Delta x^{(k)} = 0$ when converged *(note that in the previous step we have sacrificed accuracy for simplicity but we will still continue to use the '=' sign in the following).*

$$f(x^{(k)}) + \left.\frac{\partial f}{\partial x}\right|_{x=x^{(k)}} \Delta x^{(k)} = 0$$

This now allows us to solve for the unknown $\Delta x^{(k)}$.

DERIVATION OF NEWTON'S METHOD

***Step 6:** rearrange and interpret (scalar case):*

Although it was stated at the beginning that $x \in \mathbb{R}^m$ and the image $f(x) \in \mathbb{R}^n$, it was silently assumed that $f(x)$ is a ***single variate scalar function*** to keep the Taylor series simple. *In that case we can write the following iteration rule:*

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})},$$

where $f'(x^{(k)}) \neq 0$ is the first derivative $\partial f / \partial x$ evaluated at the approximated root $x^{(k)}$ of iteration k. *To start the iterations we need an initial guess* $x^{(0)}$.

DERIVATION OF NEWTON'S METHOD

The general multi-dimensional case:

In general we have the domain $x \in \mathbb{R}^m$ and the image $f(x) \in \mathbb{R}^n$.

We can no longer simply divide by $f'(x)$ because now the problem has a ***different structure***. In multiple dimensions the first order term in the Taylor series becomes $J(x^{(k)})\Delta x^{(k)}$ with $J(x^{(k)}) \in \mathbb{R}^{n \times m}$ the ***Jacobian*** of $f(x)$ evaluated at $x^{(k)}$ *(a $n \times m$ matrix with elements $\partial f_i / \partial x_j$)*.

Newton's method for arbitrary dimensions:

$$J(x^{(k)})\Delta x^{(k)} = -f(x^{(k)}),$$
$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)},$$

with $x^{(0)}$ some initial guess. In every iteration k, we must ***solve a linear system*** for the m unknown corrections needed to advance to $x^{(k+1)}$. Iterations are repeated until $|\Delta x^{(k)}| < \varepsilon$ where ε is a tolerance below which we assume the solution has converged.

DERIVATION OF NEWTON'S METHOD

A summary of what we just did:

- At the heart of the general Newton method is the Jacobian J.
- In order to use the algorithm, we need to compute J which means we must compute derivatives and *evaluate* them at a point $x^{(k)}$.
- We can obtain J in different ways:
  - compute the derivatives manually
  - with a software for symbolic math
  - automatic differentiation *(one test your project code should pass is an application of Newton's method)*
  - through a numerical approximation like Finite-Differences
- An accurate representation of J is key for good convergence behavior of Newton's method.

RECAP

- Importance of derivatives in Mathematics, Science, Technology and Engineering.
- Towards automatic differentiation
- Linearization of Euler equations as an example for the Jacobian
- Newton's method
- Computing derivatives in Science and engineering is fundamental.

Further reading:

- P. H.W. Hoffmann, *A Hitchhiker's Guide to Automatic Differentiation*, Springer 2015, doi:10.1007/s11075-015-0067-6 (You can access this paper through the Harvard network or find it in the class repository)
- Griewank, A. and Walther, A., *Evaluating derivatives: principles and techniques of algorithmic differentiation*, SIAM 2008, Vol. 105
- Newton's method: https://en.wikipedia.org/wiki/Newton%27s_method