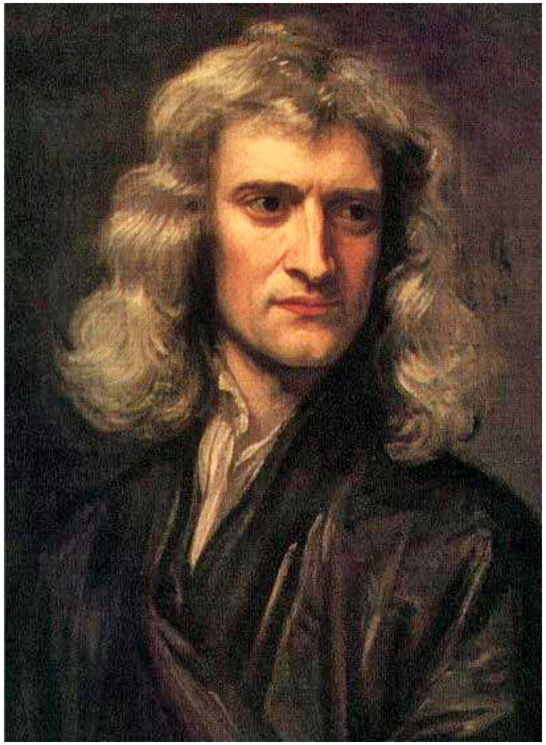
TDT4127 Programming and Numerics
Week 42

Newton's method in multiple dimensions

Learning goals

- Goals
  - Solving nonlinear systems of equations
  - Algorithm:
    - ***Newton's method for systems***
- Curriculum
  - Exercise set 7
  - Programming for Computations - Python
    - Ch. 6.6



Newton's method

- Week 38-39: Newton's method for scalar equations:

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

- There is a natural extension to multiple dimensions
  - Topic of this week's lecture
- Will only cover the formulation of it, not theory around

Systems of equations

- A system of (nonlinear) equations:

$$
\begin{aligned}
f_0(x_0, x_1, \dots, x_n) &= 0 \\
f_1(x_0, x_1, \dots, x_n) &= 0 \\
&\vdots \\
f_n(x_0, x_1, \dots, x_n) &= 0
\end{aligned}
$$

- Unlike linear systems, we **cannot** say more about the structure of the f_i, and can't write it in matrix-vector form.
- We **can** write the system more compactly with vectors:

$$
\boldsymbol{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \qquad \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_0(\boldsymbol{x}) \\ f_1(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{bmatrix} = \mathbf{0}
$$

Newton's method for systems

- We want to solve the nonlinear system of equations
$$\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0}$$
- What is the trick we've been using all along?
  - That's right – linearization!
- Idea: Exchange the nonlinear system of equations with a linear system, and solve
$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}$$
- Step 1: Find an approximate linear system $\boldsymbol{g}(\boldsymbol{x})$

Linear approximation

- In the 1D case, Taylor's theorem gives a linear approximation:

$$f(x) \approx f\left(x^k\right) + f'\left(x^k\right)\left(x - x^k\right)$$

- In several dimensions, Taylor's theorem also gives a linear approximation, using partial derivatives:

- $$f_j(\boldsymbol{x}) \approx f_j\left(\boldsymbol{x}^k\right) + \frac{\partial f_j}{\partial x_0}\left(\boldsymbol{x}^k\right)\left(x_0 - x_0^k\right) + \frac{\partial f_j}{\partial x_1}(\boldsymbol{x}^k)(x_1 - x_1^k) + \cdots + \frac{\partial f_j}{\partial x_n}(\boldsymbol{x}^k)(x_n - x_n^k)$$

NTNU

Linear approximation

- So, each equation is approximated by

$$g_0(\boldsymbol{x}) = b_0 + a_{00}(x_0 - x_0^k) + a_{01}(x_1 - x_1^k) + \cdots + a_{0n}(x_n - x_n^k)$$
$$g_1(\boldsymbol{x}) = b_1 + a_{10}(x_0 - x_0^k) + a_{11}(x_1 - x_1^k) + \cdots + a_{1n}(x_n - x_n^k)$$
$$\vdots$$
$$g_n(\boldsymbol{x}) = b_n + a_{n0}(x_0 - x_0^k) + a_{n1}(x_1 - x_1^k) + \cdots + a_{nn}(x_n - x_n^k)$$

where

$$b_j = f_j(\boldsymbol{x}^k), \qquad a_{jl} = \frac{\partial f_j}{\partial x_l}(\boldsymbol{x}^k)$$

- This is a linear system!

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{b} + A(\boldsymbol{x} - \boldsymbol{x}^k)$$

Newton's method for systems

- This is a linear system!

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{b} + A\left(\boldsymbol{x} - \boldsymbol{x}^k\right)$$

- The matrix A is called the Jacobian of $\boldsymbol{f}$ and is often written $J_{\boldsymbol{f}}(\boldsymbol{x}^k)$. In general:

$$J_{\boldsymbol{f}}(\boldsymbol{y}) = \begin{bmatrix} \frac{\partial f_0}{\partial x_0}(\boldsymbol{y}) & \frac{\partial f_0}{\partial x_1}(\boldsymbol{y}) & \cdots & \frac{\partial f_0}{\partial x_n}(\boldsymbol{y}) \\ \frac{\partial f_1}{\partial x_0}(\boldsymbol{y}) & \frac{\partial f_1}{\partial x_1}(\boldsymbol{y}) & \cdots & \frac{\partial f_1}{\partial x_n}(\boldsymbol{y}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_0}(\boldsymbol{y}) & \frac{\partial f_n}{\partial x_1}(\boldsymbol{y}) & \cdots & \frac{\partial f_n}{\partial x_n}(\boldsymbol{y}) \end{bmatrix}$$

Newton's method for systems

- This is a linear system!

$$\mathbf{g}(\mathbf{x}) = \mathbf{b} + A(\mathbf{x} - \mathbf{x}^k)$$

- Note also that $\mathbf{b} = \mathbf{f}(\mathbf{x}^k)$ so we have, more precisely:

$$\mathbf{g}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^k) + J_{\mathbf{f}}(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k)$$

- We solve $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ in two steps:

1. Solve the linear system $J_{\mathbf{f}}(\mathbf{x}^k)\mathbf{y} = -\mathbf{f}(\mathbf{x}^k)$
2. Compute $\mathbf{x} = \mathbf{x}^k + \mathbf{y}$

Newton's method for systems

- We could also directly solve

$$\boldsymbol{f}\left(\boldsymbol{x}^k\right) + J_{\boldsymbol{f}}\left(\boldsymbol{x}^k\right)\left(\boldsymbol{x} - \boldsymbol{x}^k\right) = \boldsymbol{0}$$

  by writing

$$\boldsymbol{x} = \boldsymbol{x}^k - J_{\boldsymbol{f}}\left(\boldsymbol{x}^k\right)^{-\mathbf{1}} \boldsymbol{f}\left(\boldsymbol{x}^k\right)$$

- This formulation is a bit misleading, though – we don't want to actually compute $J_{\boldsymbol{f}}\left(\boldsymbol{x}^k\right)^{-\mathbf{1}}$, just solve the linear system! Hence the two-step formulation from last slide.

Newton's method for systems

- Solving $\boldsymbol{g}(\boldsymbol{x}) = \mathbf{0}$ does not give us the exact solution since $\boldsymbol{g}$ only approximates $\boldsymbol{f}$, but we get a method from it:

$$\boldsymbol{x}^{\boldsymbol{k+1}} = \boldsymbol{x}^k - J_{\boldsymbol{f}}\left(\boldsymbol{x}^k\right)^{\boldsymbol{-1}} \boldsymbol{f}\left(\boldsymbol{x}^k\right)$$

- Note the similarities with 1D-Newton:

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

- As with 1D-Newton, we require **stopping conditions**

Stopping conditions

- 1D Newton's method: Stop when

$$|x^{k+1} - x^{k}| < \delta, \text{ or } \left|f\left(x^{k+1}\right)\right| < \epsilon.$$

  …or a combination of the two

- Here: stop on reaching one or more of the following:
  - $|x_j^{k+1} - x_j^{k}| < \delta$ for all j
  - $\sqrt{(x_0^{k+1} - x_0^{k})^2 + (x_1^{k+1} - x_1^{k})^2 + \cdots + (x_n^{k+1} - x_n^{k})^2} < \delta$
  - $\left|f_j\left(\boldsymbol{x}^{\boldsymbol{k+1}}\right)\right| < \epsilon$ for all j
  - $\sqrt{f_0(\boldsymbol{x}^{\boldsymbol{k+1}})^2 + f_1(\boldsymbol{x}^{\boldsymbol{k+1}})^2 + \ldots + f_n(\boldsymbol{x}^{\boldsymbol{k+1}})^2} < \epsilon$
- We can pick and choose stopping conditions based on what seems reasonable for the problem.

Programming Newton's for systems

1. Write code for evaluating $J_{\boldsymbol{f}}(\boldsymbol{x}^k)$ and $\boldsymbol{f}(\boldsymbol{x}^k)$
2. Choose an initial guess $\boldsymbol{x}^{\boldsymbol{0}}$
3. Iterate until stopping condition is met:
   1. Solve the linear system $J_{\boldsymbol{f}}(\boldsymbol{x}^k)\boldsymbol{y} = -\boldsymbol{f}(\boldsymbol{x}^k)$
   2. Compute $\boldsymbol{x}^{\boldsymbol{k+1}} = \boldsymbol{x}^{\boldsymbol{k}} + \boldsymbol{y}$

Demonstration: newtonSkeleton.py

Summary

- We can generalize Newton's method to higher-dimensional equations
  - Relies on a linearization of the problem
  - Uses the Jacobian of the function we want to find a root of
- Newton's method for systems requires vectors and matrices, and each step requires solution of a linear system
- Implementation is best done using several subfunctions

Questions?