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:

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

• 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:

\[
\begin{align*}
  f_0(x_0, x_1, \ldots, x_n) &= 0 \\
  f_1(x_0, x_1, \ldots, x_n) &= 0 \\
  & \vdots \\
  f_n(x_0, x_1, \ldots, x_n) &= 0
\end{align*}
\]

• 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:

\[
\begin{align*}
  x &= \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \\
  f(x) &= \begin{bmatrix} f_0(x) \\ f_1(x) \\ \vdots \\ f_n(x) \end{bmatrix} = 0
\end{align*}
\]
Newton’s method for systems

• We want to solve the nonlinear system of equations
  \[ f(x) = 0 \]

• 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
  \[ f(x) \approx g(x) = 0 \]

• Step 1: Find an approximate linear system \( g(x) \)
Linear approximation

• In the 1D case, Taylor’s theorem gives a linear approximation:
  \[ f(x) \approx f(x^k) + f'(x^k)(x - x^k) \]

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

  \[ f_j(x) \approx f_j(x^k) + \frac{\partial f_j}{\partial x_0}(x^k)(x_0 - x_0^k) \]
  \[ + \frac{\partial f_j}{\partial x_1}(x^k)(x_1 - x_1^k) + \cdots + \frac{\partial f_j}{\partial x_n}(x^k)(x_n - x_n^k) \]
Linear approximation

• So, each equation is approximated by

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

where

\[ b_j = f_j(x^k), \quad a_{jl} = \frac{\partial f_j}{\partial x_l}(x^k) \]

• This is a linear system!

\[ g(x) = b + A(x - x^k) \]
Newton’s method for systems

• This is a linear system!

\[ g(x) = b + A(x - x^k) \]

• The matrix \( A \) is called the Jacobian of \( f \) and is often written \( J_f(x^k) \). In general:

\[
J_f(y) = \begin{bmatrix}
\frac{\partial f_0}{\partial x_0}(y) & \frac{\partial f_0}{\partial x_1}(y) & \ldots & \frac{\partial f_0}{\partial x_n}(y) \\
\frac{\partial f_1}{\partial x_0}(y) & \frac{\partial f_1}{\partial x_1}(y) & \ldots & \frac{\partial f_1}{\partial x_n}(y) \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial f_n}{\partial x_0}(y) & \frac{\partial f_n}{\partial x_1}(y) & \ldots & \frac{\partial f_n}{\partial x_n}(y)
\end{bmatrix}
\]
Newton’s method for systems

• This is a linear system!
  \[ g(x) = b + A(x - x^k) \]

• Note also that \( b = f(x^k) \) so we have, more precisely:
  \[ g(x) = f(x^k) + J_f(x^k)(x - x^k) \]

• We solve \( g(x) = 0 \) in two steps:

  1. Solve the linear system \( J_f(x^k)y = -f(x^k) \)
  2. Compute \( x = x^k + y \)
Newton’s method for systems

- We could also directly solve
  \[ f(x^k) + J_f(x^k)(x - x^k) = 0 \]
  by writing
  \[ x = x^k - J_f(x^k)^{-1}f(x^k) \]
- This formulation is a bit misleading, though – we don’t want to actually compute \( J_f(x^k)^{-1} \), just solve the linear system! Hence the two-step formulation from last slide.
Newton’s method for systems

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

$$x^{k+1} = x^k - J_f(x^k)^{-1} f(x^k)$$

• Note the similarities with 1D-Newton:

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

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

• 1D Newton’s method: Stop when
  \[ |x^{k+1} - x^k| < \delta, \text{ or } |f(x^{k+1})| < \varepsilon. \]
  …or a combination of the two

• Here: stop on reaching one or more of the following:
  
  - \[ |x_j^{k+1} - x_j^k| < \delta \text{ for all } j \]
  
  - \[ \sqrt{(x_0^{k+1} - x_0^k)^2 + (x_1^{k+1} - x_1^k)^2 + \cdots + (x_n^{k+1} - x_n^k)^2} < \delta \]
  
  - \[ |f_j(x^{k+1})| < \varepsilon \text{ for all } j \]
  
  - \[ \sqrt{f_0(x^{k+1})^2 + f_1(x^{k+1})^2 + \cdots + f_n(x^{k+1})^2} < \varepsilon \]

• 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 $J_f(x^k)$ and $f(x^k)$
2. Choose an initial guess $x^0$
3. Iterate until stopping condition is met:
   1. Solve the linear system $J_f(x^k)y = -f(x^k)$
   2. Compute $x^{k+1} = x^k + y$

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?