Root Finding

COS 323
Reminder

• Sign up for Piazza
• Assignment 0 is posted, due Tue 9/25
Last time..

• Floating point numbers and precision
• Machine epsilon
• Sources of error
• Sensitivity and conditioning
• Stability and accuracy
• Asymptotic analysis and convergence order
Today

• Root finding definition & motivation
• Standard techniques for root finding
  – Algorithms, convergence, tradeoffs
• Example applications of Newton’s Method
• Root finding in > 1 dimension
1-D Root Finding

- Given some function, find location where \( f(x) = 0 \)
Why Root Finding?

• Solve for \( x \) in any equation: \( f(x) = b \) where \( x = ? \)  
  \[ \rightarrow \text{ find root of } g(x) = f(x) - b = 0 \]

  – Might not be able to solve for \( x \) directly  
    e.g., \( f(x) = e^{-0.2x}\sin(3x-0.5) \)

  – Evaluating \( f(x) \) might itself require solving a differential equation, running a simulation, etc.
Why Root Finding?

• Engineering applications: Predict dependent variable (e.g., temperature, force, voltage) given independent variables (e.g., time, position)

• Focus on finding real roots
Bracket-Based Methods

• Given:
  – Points that \textit{bracket} the root
  – A \textit{well-behaved} function

• Can always find \textit{some} root

\[ f(x_+) > 0 \]
\[ f(x_-) < 0 \]
What Goes Wrong?

Tangent point: very difficult to find

Singularity: brackets don’t surround root

Pathological case: infinite number of roots – e.g. \( \sin(1/x) \)
Bisection Method

• Given points $x_+$ and $x_-$ that bracket a root, find
  \[ x_{\text{half}} = \frac{1}{2} (x_+ + x_-) \]
  and evaluate $f(x_{\text{half}})$

• If positive, $x_+ \leftarrow x_{\text{half}}$  else  $x_- \leftarrow x_{\text{half}}$

• Stop when $x_+$ and $x_-$ close enough

• If function is continuous, this \textit{will} succeed in finding \textit{some} root
Error Convergence of Iterative Methods

• **(Absolute) error bound** $\varepsilon_n$ at step $n$:
  
  $\varepsilon_n$ bounds $|x_{estimated \ at \ step \ n} - x_{true}|$

• **Convergence**: describes how $\varepsilon_{n+1}$ relates to $\varepsilon_n$

• **Linear convergence**:
  
  $|\varepsilon_{n+1}| = c |\varepsilon_n|$ for some $c \in (0, 1)$

• **Superlinear convergence**:
  
  $|\varepsilon_{n+1}| = c |\varepsilon_n|^q$ for some $c \in (0, 1), \ q > 1$
Linear:

Superlinear:

Sublinear:
Bisection Error Convergence

• Very robust method: guaranteed to find root!

• Convergence rate:
  – Error bounded by size of \([x_+... \ x_-]\) interval
  – Interval shrinks in half at each iteration
  – So, error bound cut in half at each iteration:
    \[ |\varepsilon_{n+1}| = \frac{1}{2} |\varepsilon_n| \]
  – Linear convergence!
  – One extra bit of accuracy in \(x\) at each iteration
Faster Root-Finding

• Fancier methods get super-linear convergence
  – Typical approach: model function locally by something whose root you can find exactly
  – Model didn’t match function exactly, so iterate
  – In many cases, these are less safe than bisection
Secant Method

• Interpolate or extrapolate through two most recent points
Secant Method Convergence

• Faster than bisection:
  \[ |\varepsilon_{n+1}| = c |\varepsilon_n|^{1.6} \]

• Faster than linear: number of correct bits multiplied by 1.6

• Drawback: only true if \textit{sufficiently close} to a root of a \textit{sufficiently smooth} function
  – Does not guarantee that root remains bracketed
False Position Method

- Similar to secant, but guarantees bracketing

- Stable, but linear in bad cases
False Position Failure
Other Interpolation Strategies

- Ridders’ method: fit exponential to $f(x_+), f(x_-)$, and $f(x_{\text{half}})$
- Van Wijngaarden-Dekker-Brent method: inverse quadratic fit to 3 most recent points if within bracket, else bisection
- Both of these \textit{safe} if function is nasty, but \textit{fast} (super-linear) if function is nice
Demo
Newton-Raphson

- Best-known algorithm for getting \textit{quadratic} convergence when derivative is easy to evaluate
- Quadratic: \# correct bits doubles each iteration!
  \[ |\varepsilon_{n+1}| = c |\varepsilon_n|^2 \]
- Another variant on the extrapolation theme

\[ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \]
Newton-Raphson convergence

• Begin with Taylor series

\[ f(x_n + \delta) = f(x_n) + \delta f'(x_n) + \delta^2 \frac{f''(x_n)}{2} + \ldots = 0 \]

• Divide by derivative (can’t be zero!)

\[ \frac{f(x_n)}{f'(x_n)} + \delta + \delta^2 \frac{f''(x_n)}{2 f'(x_n)} = 0 \]

\[ -\delta_{\text{Newton}} + \delta + \delta^2 \frac{f''(x_n)}{2 f'(x_n)} = 0 \]

\[ \delta_{\text{Newton}} - \delta = \frac{f''(x_n)}{2 f'(x_n)} \delta^2 \implies \varepsilon_{n+1} \sim \varepsilon_n^2 \]
Newton-Raphson

• Method fragile: can easily get confused

• Good starting point critical
  – Newton popular for “polishing off” a root found approximately using a more robust method

• Quadratic only for simple root
Newton-Raphson Convergence

- Can talk about “basin of convergence”: range of $x_0$ for which method finds a root
- Can be extremely complex: here’s an example in 2-D with 4 roots
Common Example of Newton: Square Root

• Let $f(x) = x^2 - a$: zero of this is square root of $a$

• $f'(x) = 2x$, so Newton iteration is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$x_{n+1} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2} \left( x_n + \frac{a}{x_n} \right)$$

• “Divide and average” method (~2000 B.C.)
Reciprocal via Newton

• Division is slowest of basic operations

• On some computers, hardware divide not available (!): simulate in software

\[
\frac{a}{b} = a \times \frac{1}{b} \\
f(x) = \frac{1}{x} - b = 0 \\
f'(x) = -\frac{1}{x^2} \\
x_{n+1} = x_n - \frac{\frac{1}{x} - b}{-\frac{1}{x^2}} = x_n \left(2 - bx_n\right)
\]

• Need only subtract and multiply
Rootfinding in >1D

• Behavior can be complex: e.g. in 2D

\[
f(x, y) = 0
\]
\[
g(x, y) = 0
\]
Rootfinding in $>1D$

- Can’t bracket and bisect
- Result: few general methods
Newton in Higher Dimensions

• Start with

\[ f(x, y) = 0 \]
\[ g(x, y) = 0 \]

• Write as vector-valued function

\[ f(x_n) = \begin{pmatrix} f(x, y) \\ g(x, y) \end{pmatrix} \]
Newton in Higher Dimensions

• Expand in terms of Taylor series

\[ f(x_n + \delta) = f(x_n) + f'(x_n) \delta + \ldots = 0 \]

• \( f' \) is a Jacobian

\[ f'(x_n) = J = \left( \frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y} \right) \]
Newton in Higher Dimensions

• 1-dimensional case: $\delta = \frac{f(x_n)}{f'(x_n)}$

• N-dimensional: Solve for $\delta$

$$\delta = -J^{-1}(x_n) \cdot f(x_n)$$

• Requires matrix inversion (we’ll see this later)
• Often fragile – must be careful
  – Keep track of whether error decreases
  – If not, try a smaller step in direction $\delta$
Recap: Tradeoffs

• Bracketing methods (Bisection, False-position)
  – Stable, slow

• Open methods (Secant, Newton)
  – Possibly divergent, fast
  – Newton requires derivative

• Hybrid methods (Brent)
  – Combine bracketing & open methods in a principled way
Practical notes

• Root-finding in Matlab:
  – fzero: For finding root of a single function
    Combines “safe” and “fast” methods
  – roots: For finding polynomial roots

• Excel:
  – Goal Seek: Drive an equation to 0 by adjusting 1 parameter
  – Solver: Can vary multiple parameters simultaneously, also minimize & maximize

• Tip: Plot your function first!!!