Lecture 4, Sep 20, 2023

Numerical Root Finding

- Root finding involves finding x^* given f such that $f(x^*) = 0$
  - One application is optimization problems where we find the roots of the derivative
- Regularizing assumptions are used to make the problem well-posed; some common assumptions are, in increasing order of restrictiveness:
  - Continuity: f is continuous if $f(x) \to f(z)$ as $x \to z$ for all z
  - Lipschitz continuity: f is Lipschitz continuous if there exists a constant c such that $|f(x) - f(z)| \leq c|x - z|$ for all $x, z \in \mathbb{R}$
    - This is like saying that the derivative is bounded
    - e.g. $\frac{1}{x}$ is not Lipschitz continuous
  - Differentiability: f is differentiable if $f'(x)$ exists for all x; $f \in C^m$ if m derivatives exist and are continuous; $f \in C^\infty$ if all derivatives exist and are continuous
- Generally, the more assumptions we can make, the faster our algorithm becomes

Definition

The *convergence rate* of an algorithm is the rate at which the error decreases; formally, the convergence rate (aka order) of an algorithm is r if

$$\lim_{k \to \infty} \frac{|E_{k+1}|}{|E_k|^r} = C$$

where C is a constant.

- The first algorithm we will look at is *bisection*, which relies on the intermediate value theorem; if we have a continuous f and two points, where f is positive at one test point and negative at the other, we know there must be a root between the test points
  - The algorithm:
    1. Initialization: start with bracketing guesses l_0, r_0 such that $f(l_0)f(r_0) < 0$ (that is, $\operatorname{sgn} f(l_0) \neq \operatorname{sgn} f(r_0)$)
       - These points are typically picked based on some prior knowledge of the function
    2. Iterative update: bisect the current interval by computing $c = \frac{l_k + r_k}{2}$
       - If $f(c) \cdot f(l_k) < 0$, then $r_{k+1} = c, l_{k+1} = l_k$
       - If $f(c) \cdot f(r_k) < 0$, then $r_{k+1} = r_k, l_{k+1} = c$
       - In other words, change the bound that has the same sign as c to be c
    3. Iterate until $|r_k - l_k| < \epsilon$ and return c
  - For bisection $E_k = |r_k - l_k|$, which is decreased by a factor of $\frac{1}{2}$ for each iteration, therefore $\frac{|E_{k+1}|}{|E_k|} = \frac{1}{2}$ and so bisection has linear convergence (order 1)
  - In summary, bisection is easy to implement and guaranteed to converge on any continuous f, but finding the initial bracket l_0, r_0 can be challenging
- With more assumptions, we can use fixed-point iteration
  - Assume that there exists some update algorithm g such that $g(x^*) = x^*$, so $f(x^*) = g(x^*) - x^* = 0$; this update function will bring us closer to the root if it converges
    - This update is problem-dependent, but has to be Lipschitz continuous for convergence
    - Example: for $f(x) = e^{\frac{1}{2}x} - x - 2$
      - $g_1(x) = e^{\frac{1}{2}x} - 2$
      - $e^{\frac{1}{2}x} = x + 2 \implies \frac{1}{2}x = \ln(x + 2) \implies -x + 2\ln(x + 2) = 0$
      - $g_2(x) = 2\ln(x + 2)$

  - The algorithm:
    1. Start with an initial guess x_0
    2. Update $x_{k+1} = g(x_k)$
    3. Iterate until $|x_{k+1} - x_k| < \epsilon$, then return x_{k+1}
  - This is called a "fixed-point" algorithm because if $x_k = x^*$ then all $x_{k+n} = x^*$
  - $E_{k+1} = |x_{k+1} - x^*|$
    $= |g(x_k) - g(x^*)| \leq c|x_k - x^*|$
    $= cE_k$
    $= c^k E_0$
    * Convergence requires that $c < 1$ when x is close to x^* (i.e. g is not too steep near the root), otherwise it diverges
    * Convergence is linear but can be quadratic if $g'(x) \approx 0$ (e.g. Newton's method)
  - In summary, fixed-point iteration requires choosing a problem-dependent $g(x)$, which must be Lipschitz continuous with $c < 1$ close to x^*
- Newton's method is a form of fixed-point iteration which assumes $f \in C^1$ (continuously differentiable up to the first derivative), which has quadratic convergence
  - We can do a Taylor expansion around x_k to get $f(x) \approx f(x_k) + f'(x_k)(x - x_k)$, then $f(x) = 0 \implies f(x_k) + f'(x_k)(x - x_k) = 0 \implies x = x_k - \dfrac{f(x_k)}{f'(x_k)}$
  - Therefore Newton's method is equivalent to a fixed-point iteration with $g(x) = x - \dfrac{f(x)}{f'(x)}$ (i.e. the update step is $x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)}$)
  - Newton's method is exact for a linear function; the more linear a function is, the faster the convergence; if the function is highly nonlinear, Newton's method converges slowly or not at all
  - Assuming f is differentiable, then we can show $E_{k+1} = \left| g'(x^*)E_k + \dfrac{1}{2}g''(x^*)E_k^2 \right| + O(E_k^3)$
    * For Newton updates, $g'(x) = \dfrac{f(x)f''(x)}{f'(x)^2}$
    * Assuming x^* is a single root, then $f'(x) \neq 0$ close to x^* and so $g'(x) \approx 0$ when close to x^*; this means the $g'(x^*)$ term disappears and the convergence rate becomes quadratic
  - In summary, Newton's method assumes continuous differentiability, in return for quadratic convergence for single roots with $f'(x^*) \neq 0$; however it requires computing the derivative at each step which could be expensive
- If we don't know the actual derivative, we can use the secant method
  - Instead of the derivative we use * $f'(x_k) \approx \dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$
    * $f(x_{k-1})$ can be recycled from the previous iteration to minimize cost
  - Convergence rate is difficult to analyze, but it is between linear and quadratic
- Hybrid methods that combine one or more methods also exist
- Note the overall speed of a root finding algorithm depends both on the rate of convergence and the cost per iteration; they must be balanced to achieve a fast algorithm
  - The difference between linear and quadratic convergence can be huge in robotics applications, but rate of convergence is not everything