Root Finding

COS 323

Why Root Finding?

- Solve for x in any equation:

$$f(x) = b \text{ where } x = ?$$

$$\rightarrow \text{ find root of } g(x) = f(x) - b = 0$$

  - Might not be able to solve for x directly

$$\text{e.g., } f(x) = e^{-0.2x}\sin(3x-0.5)$$

  - Evaluating f(x) might itself require solving a differential equation, running a simulation, etc.

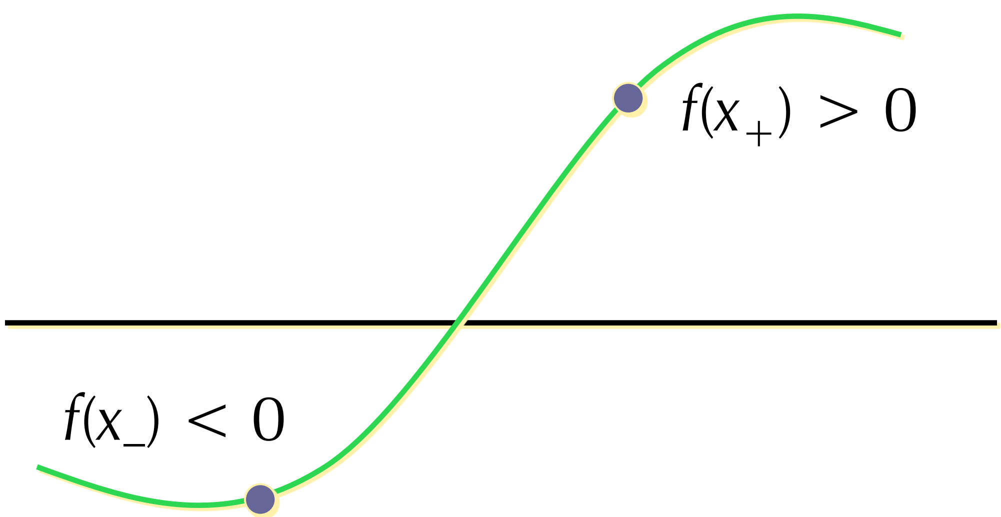
Why Root Finding?

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

- Focus on finding *real* roots

1-D Root Finding

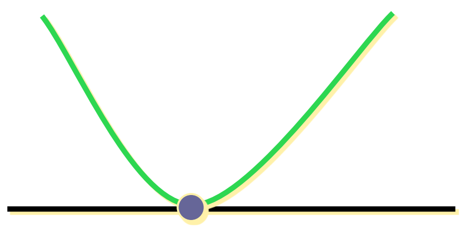
- Given some function, find location where $f(x)=0$
- Need:
  - Starting position x_0, hopefully close to solution
  - Ideally, points that *bracket* the root



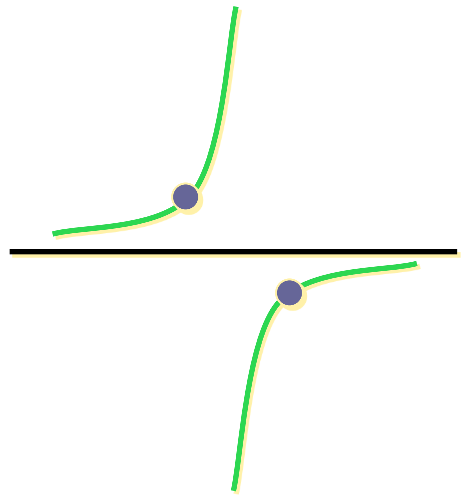
1-D Root Finding

- Given some function, find location where $f(x)=0$
- Need:
  - Starting position x_0, hopefully close to solution
  - Ideally, points that *bracket* the root
  - Well-behaved function

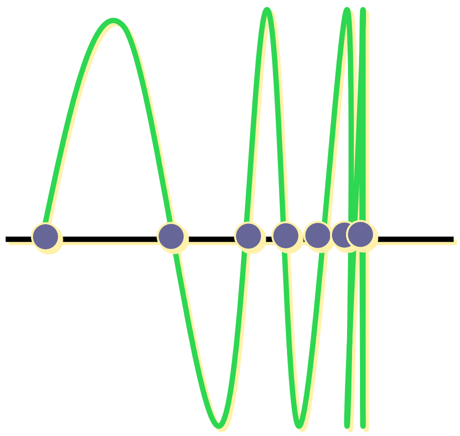
What Goes Wrong?



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
$$x_{half} = \tfrac{1}{2}\,(x_+ + x_-)$$
and evaluate $f(x_{half})$
- If positive, $x_+ \leftarrow x_{half}$ else $x_- \leftarrow x_{half}$
- Stop when x_+ and x_- close enough
- If function is continuous, this *will* succeed in finding *some* root

Bisection

- Very robust method
- Convergence rate:
  - Error bounded by size of $[x_+ \ldots x_-]$ interval
  - Interval shrinks in half at each iteration
  - Therefore, error cut in half at each iteration:

$$|\varepsilon_{n+1}| \leq \tfrac{1}{2} |\varepsilon_n|$$

  - This is called “linear convergence”
  - One extra bit of accuracy in x at each iteration

Faster Root-Finding

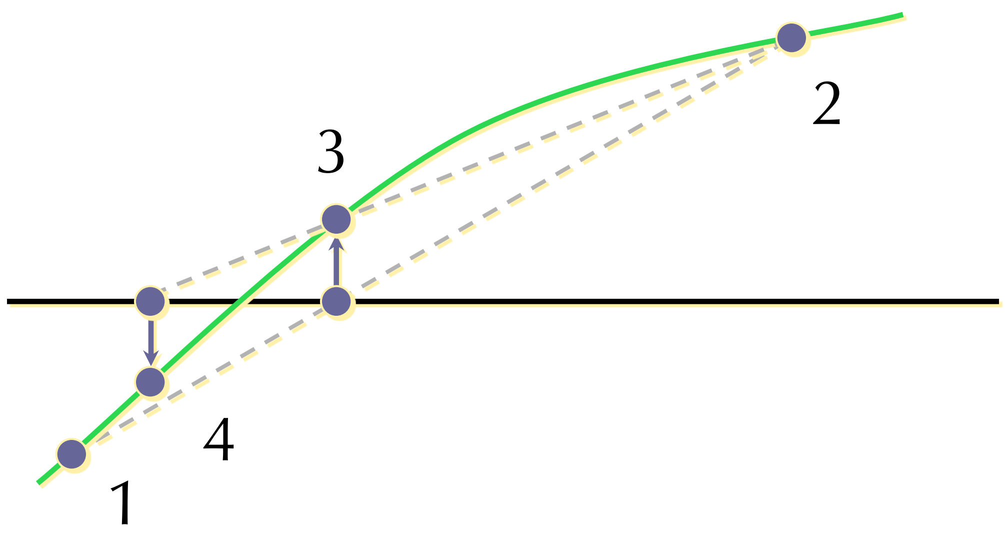
- Fancier methods get super-linear convergence
  - Typical approach: model function locally by something whose root you can find exactly
  - Model didn't match function exactly, so iterate
  - In many cases, these are less safe than bisection

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

- Simple extension to bisection: interpolate or extrapolate through two most recent points



Secant Method

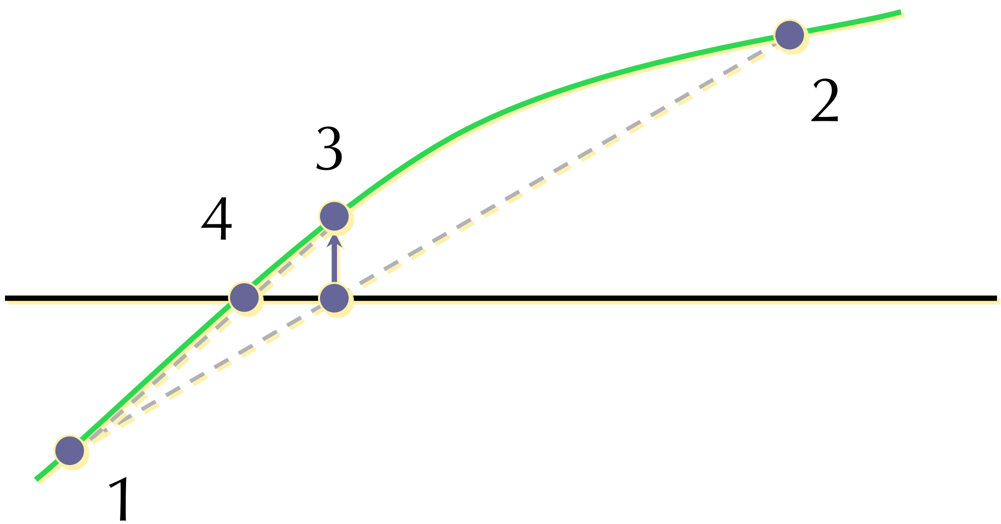
- Faster than bisection:

$$|\varepsilon_{n+1}| = \ const. \ |\varepsilon_n|^{1.6}$$

- Faster than linear: number of correct bits multiplied by 1.6
- Drawback: the above only true if sufficiently close to a root of a sufficiently smooth function
  - Does not guarantee that root remains bracketed

False Position Method

- Similar to secant, but guarantee bracketing



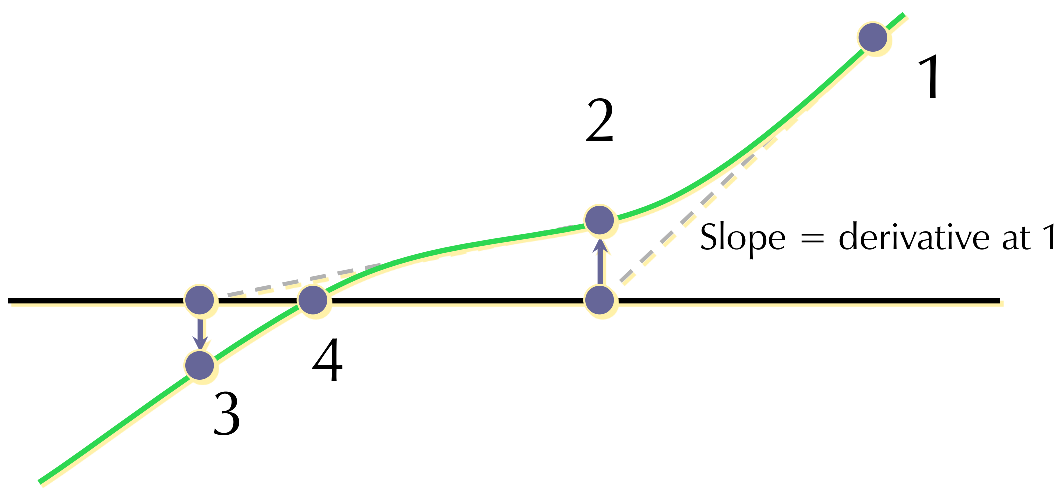
- Stable, but linear in bad cases

Other Interpolation Strategies

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

Newton-Raphson

- Best-known algorithm for getting *quadratic* convergence when derivative is easy to evaluate
- Another variant on the extrapolation theme



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

Newton-Raphson

- Begin with Taylor series

$$f(x_n+\delta) = f(x_n) + \delta f'(x_n) + \delta^2 \frac{f''(x_n)}{2} + \ldots \overset{want}{=} 0$$

- Divide by derivative (can't be zero!)

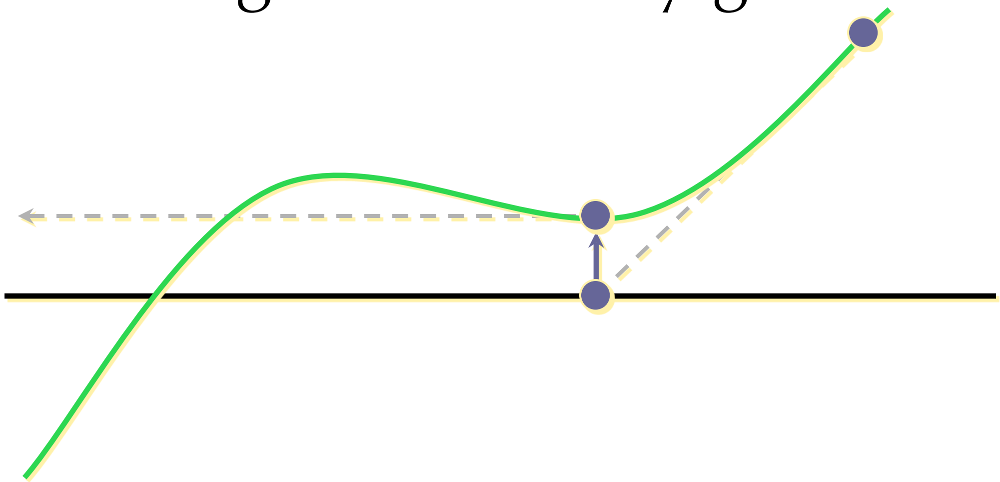
$$\frac{f(x_n)}{f'(x_n)} + \delta + \delta^2 \frac{f''(x_n)}{2f'(x_n)} = 0$$

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

$$\delta_{Newton} - \delta = \frac{f''(x_n)}{2f'(x_n)}\delta^2 \qquad \Rightarrow \qquad \varepsilon_{n+1} \sim \varepsilon_n{}^2$$

Newton-Raphson

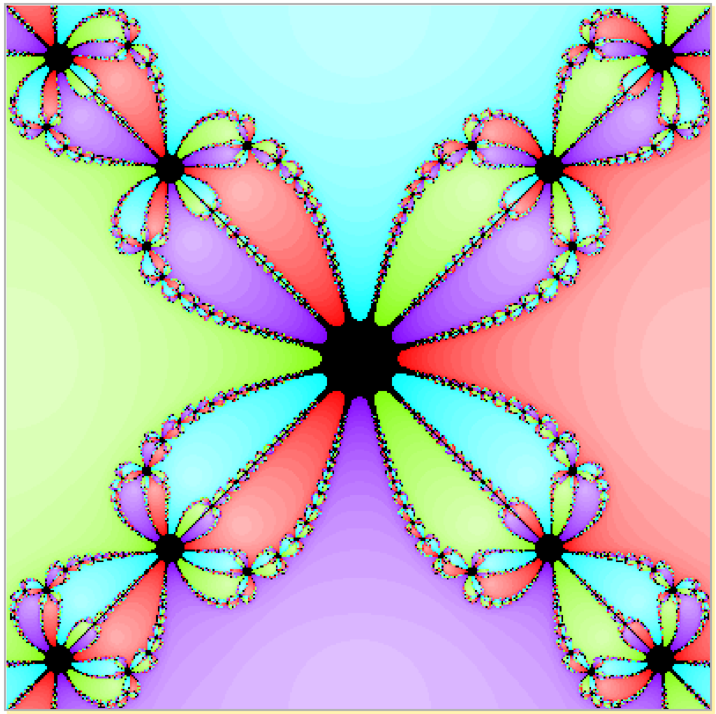
- Method fragile: can easily get confused



- Good starting point critical
  – Newton popular for "polishing off" a root found approximately using a more robust method

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



Popular 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

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

- "Divide and average" method

Reciprocal via Newton

- Division is slowest of basic operations
- On some computers, hardware divide not available (!): simulate in software

$$\tfrac{a}{b} = a * \tfrac{1}{b}$$

$$f(x) = \tfrac{1}{x} - b = 0$$

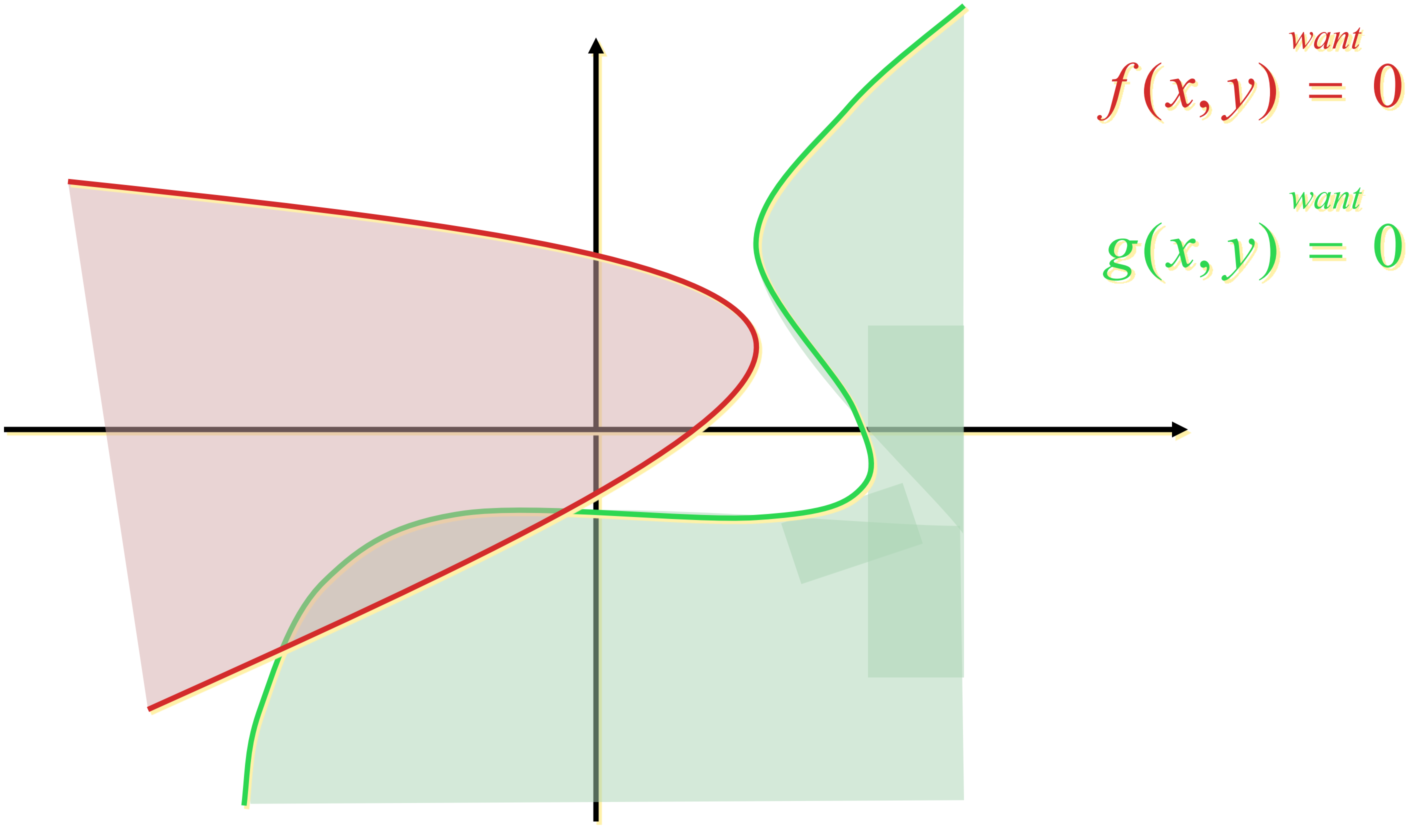
$$f'(x) = -\tfrac{1}{x^2}$$

$$x_{n+1} = x_n - \frac{\tfrac{1}{x} - b}{-\tfrac{1}{x^2}} = x_n\left(2 - bx_n\right)$$

- Need only subtract and multiply

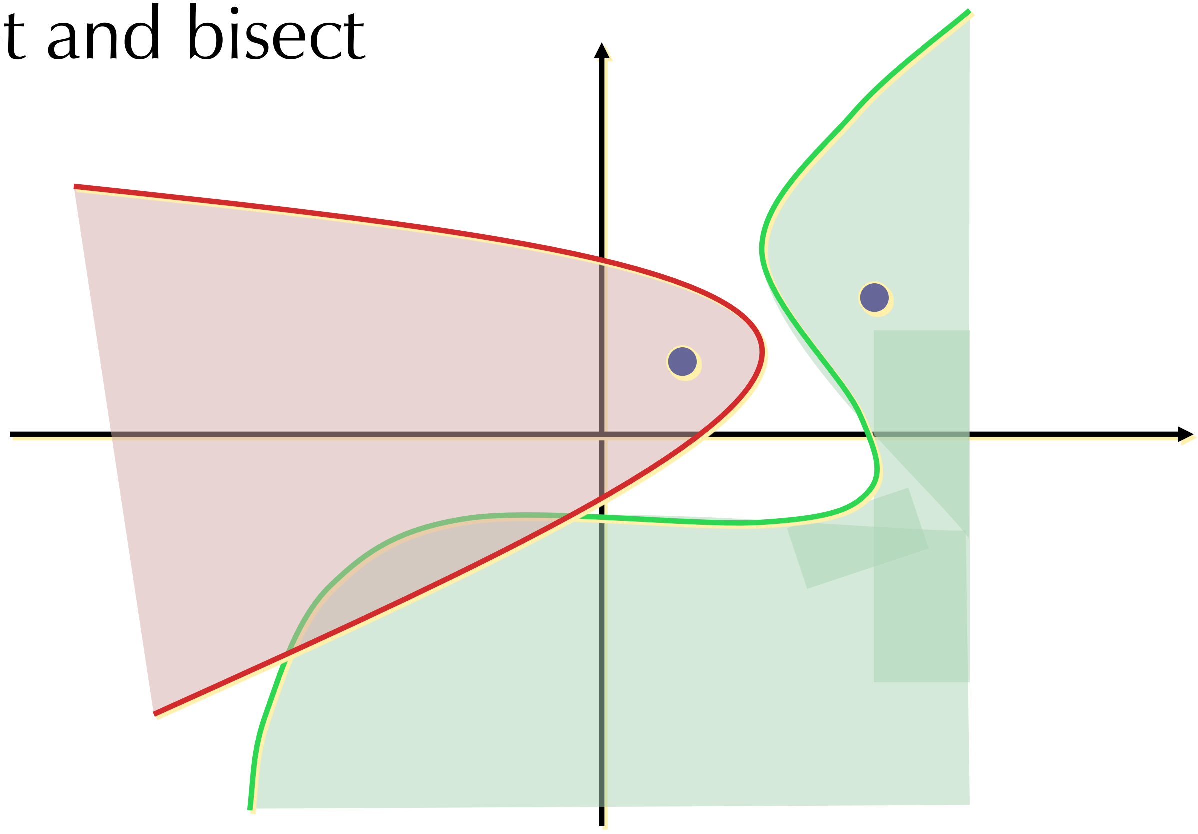
Rootfinding in >1D

- Behavior can be complex: e.g. in 2D



Rootfinding in >1D

- Can't bracket and bisect



- Result: few general methods

Newton in Higher Dimensions

- Start with

$$f(x,y) \overset{want}{=} 0$$
$$g(x,y) \overset{want}{=} 0$$

- Write as vector-valued function

$$\mathbf{f}(\mathbf{x}_n) = \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

Newton in Higher Dimensions

- Expand in terms of Taylor series

$$\mathbf{f}(\mathbf{x}_n + \boldsymbol{\delta}) = \mathbf{f}(\mathbf{x}_n) + \mathbf{f}'(\mathbf{x}_n)\,\boldsymbol{\delta} + \ldots \overset{want}{=} 0$$

- $\mathbf{f}'$ is a Jacobian

$$\mathbf{f}'(\mathbf{x}_n) = \mathbf{J} = \begin{pmatrix} \frac{\partial \mathbf{f}}{\partial x} & \frac{\partial \mathbf{f}}{\partial y} \end{pmatrix}$$

Newton in Higher Dimensions

- Solve for **δ**

$$\boldsymbol{\delta} = -\mathbf{J}^{-1}(\mathbf{x}_n)\,\mathbf{f}(\mathbf{x}_n)$$

- 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 **δ**