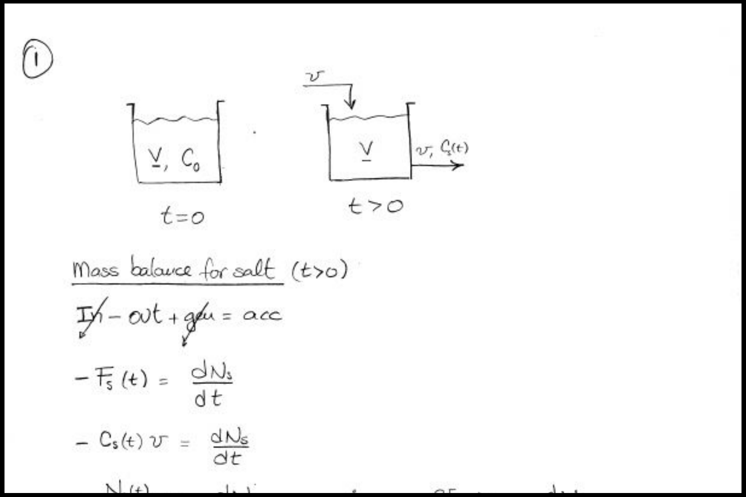
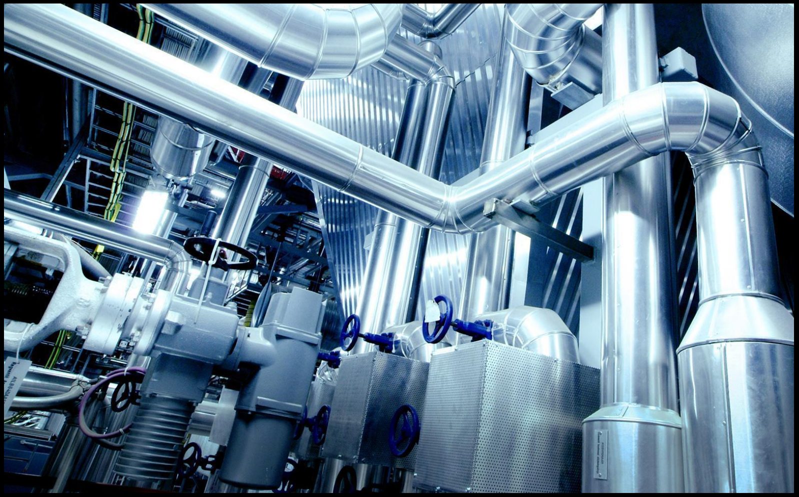




ChE352

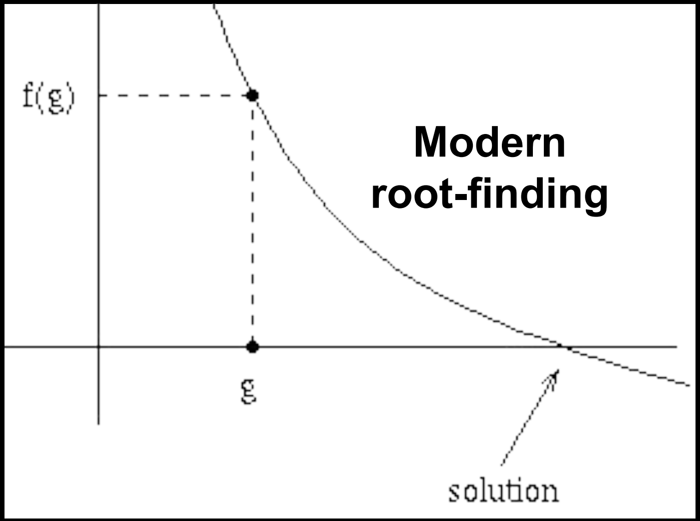
Numerical Techniques for Chemical Engineers

Professor Stevenson

Lecture 6

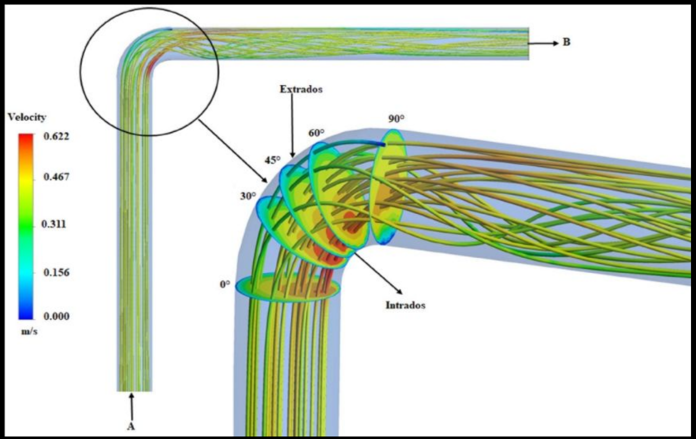
Root finding

- Origin: Egypt, ~1700 B.C.E.
- Solve any algebraic equation, even equations with no analytic solutions (which is most of them)
- Common in ChemE
- We will learn two methods:
  1. Bisection (simple, safe)
  2. Newton-Raphson (fast)



Example: Fluid Mechanics

Churchill and Zajic (2002): equation for friction factor f in a pipe as a function of Reynolds number Re



Polynomial + log: no analytic solution

$$\sqrt{\frac{2}{f}} = 3.2 - 227\frac{\sqrt{\frac{2}{f}}}{0.5\,\mathrm{Re}} + 2500\left(\frac{\sqrt{\frac{2}{f}}}{0.5\,\mathrm{Re}}\right)^2 + \frac{1}{0.436}\ln\left(\frac{0.5\,\mathrm{Re}}{\sqrt{\frac{2}{f}}}\right)$$

Given the Reynolds number Re, how would you find the friction factor f?

Finding the Friction Factor from Re

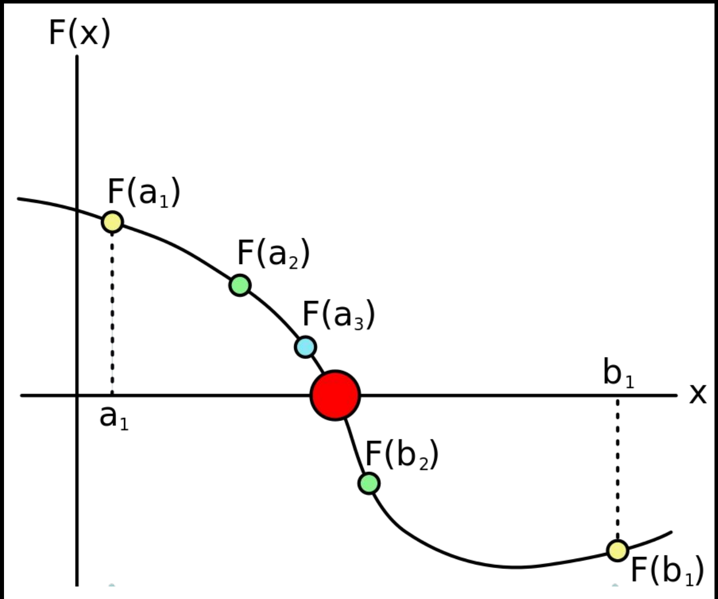
First we rearrange the Churchill-Zajic equation:

$$f(x) = 3.2 - 227\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}} + 2500\left(\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}}\right)^2 + \frac{1}{0.436}\ln\left(\frac{0.5\,\mathrm{Re}}{\sqrt{\frac{2}{x}}}\right) - \sqrt{\frac{2}{x}}$$

Why is this form easier?

Does a Root Exist?

- We want to find p such that f(p) = 0
- Don't try to find p if it doesn't exist!



Why f(p) = 0, not the more general f(p) = K?

How do we know there will be a root p?

How many values of p might exist?

Intermediate Value Theorem

Existence of an intermediate value:

$$\forall f \in C[a,b], \quad \forall K : f(a) < K < f(b)$$

$$\exists c \in (a,b) : f(c) = K$$

Plug in K = 0 and c = p (root finding):

$$if: \quad f \in C[a,b], f(a) < 0, and \ f(b) > 0,$$

$$then: \quad p \in (a,b) \ exists \ such \ that \ f(p) = 0$$

(Note that we <u>can flip a and b</u>; IVT is still valid)

We Must Prove That a Root Exists

$$f(x) = 3.2 - 227\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}} + 2500\left(\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}}\right)^2 + \frac{1}{0.436}\ln\left(\frac{0.5\,\mathrm{Re}}{\sqrt{\frac{2}{x}}}\right) - \sqrt{\frac{2}{x}}$$

1. Is f(x) continuous for the Churchill-Zajic eqn.?
2. Can our bounds, a & b, be negative?
3. Should f(a) or f(b) ever be equal to zero?
4. Is f(a) always less than f(b)?

Activity: Existence of a Root

For the Churchill-Zajic equation below:

1. Rewrite the function in the form f(y) = 0 with y = √(2/x), and Re = 20000
2. Write a Python function that returns f(y)
3. Find values a, b such that f is continuous on [a, b] and a root of f(y) exists between a & b

$$f\left(x\right)=3.2-227\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}}+2500\left(\frac{\sqrt{\frac{2}{x}}}{0.5\,\mathrm{Re}}\right)^{2}+\frac{1}{0.436}\ln\left(\frac{0.5\,\mathrm{Re}}{\sqrt{\frac{2}{x}}}\right)-\sqrt{\frac{2}{x}}$$

Answer: Existence of a Root

```
# Churchill-Zajic for Re = 20,000
# note, continuous for all positive values of y
def churchill_zajic(y):
    Re = 2e4
    return (3.2 - 227 * y / (0.5 * Re) +
            2500 * (y / (0.5 * Re))**2 +
            1 / 0.436 * np.log(0.5 * Re / y) - y)
```

Can we make this long equation nicer?

Answer: Existence of a Root

```
# Churchill-Zajic for Re = 20,000
# note, continuous for all positive values of y
def churchill_zajic(y):
    Re = 2e4
    yr = y / (0.5 * Re)
    return (3.2 - 227 * yr +
            2500 * yr**2 +
            1 / 0.436 * np.log(1 / yr) - y)
```

How is this form of the equation mathematically nicer?

Answer: Existence of a Root

```python
# test for roots of f from A to B
def find_root_bounds(f, A, B, step):
    for a in np.arange(A, B, step):
        for b in np.arange(a + step, B, step):
            if np.sign(f(a)) != np.sign(f(b)):
                return a, b
# we know churchill_zajic(y) is continuous for y > 0
bounds = find_root_bounds(churchill_zajic, 1, 100, 1)
print(bounds)  # gives 1, 18
print([churchill_zajic(y) for y in bounds])
```

How does this prove existence of a root?

Bisection for Root Finding

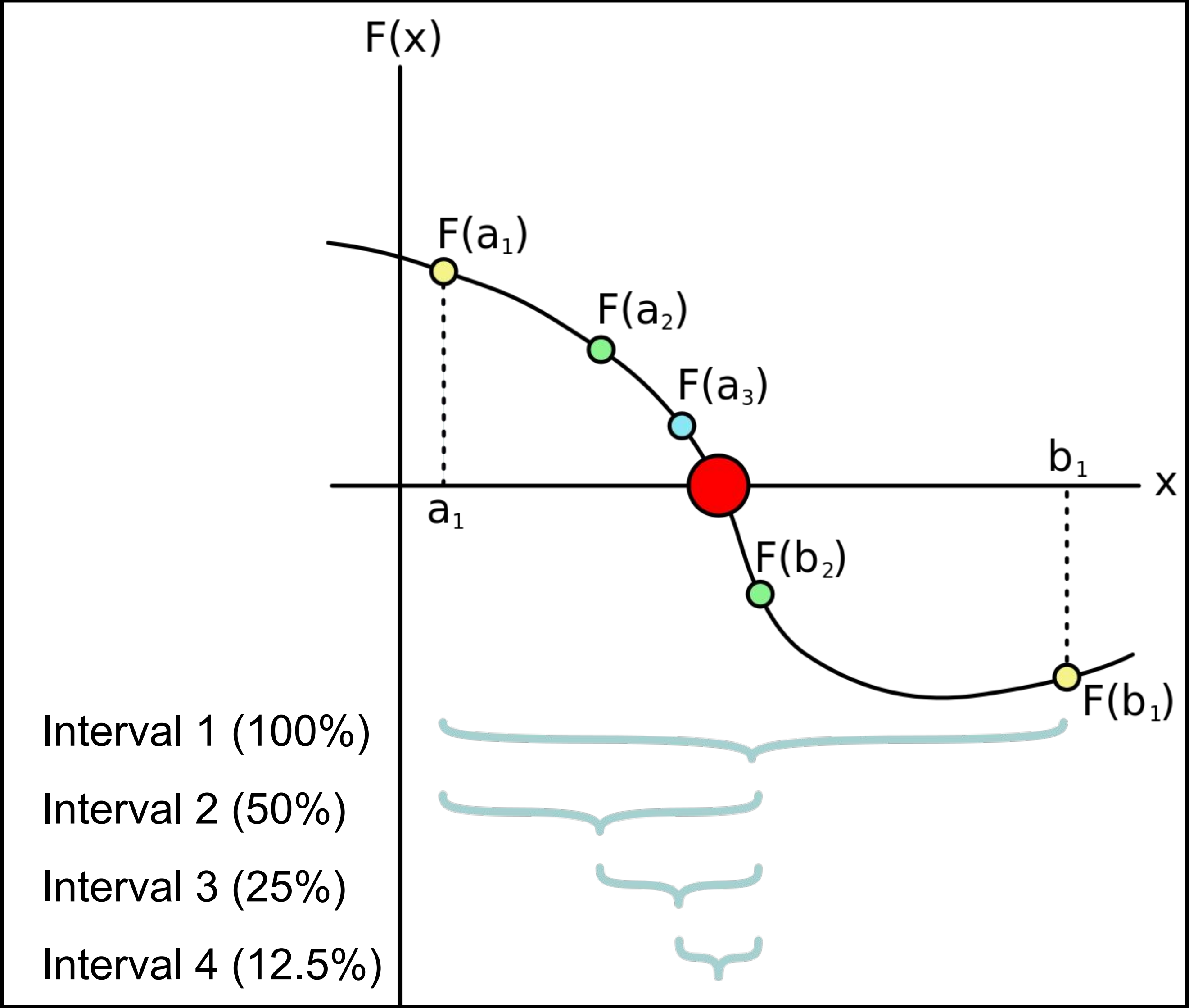
The bisection method finds p* such that $f(p^*) \approx 0$, on the interval [a, b], in this way:

1. Prove there is a root on the interval [a, b]
2. Cut the interval in half (aka bisect it)
3. Pick the half-interval that contains the root
4. If not done, go back to step 3

How do you pick the half with the root?

When are you done?

Bisection in pictures



Bisection in pseudocode

Inputs: function f, scalar a, and scalar b

1. If sign(f(a)) == sign(f(b)), raise an error
2. Set p = (a + b) / 2
3. If sign(f(a)) == sign(f(p)), set a = p, else b = p
4. If conditions are met, STOP
5. Go to Step 2

Outputs: p, f(p); or error if sign(f(a)) == sign(f(b))

Stopping conditions: $f(p) < \varepsilon$, or $|a-b| < \varepsilon$, or $|a-b| / |p| < \varepsilon$, or simply too many iterations

Convergence of Bisection

Each step, we reduce the uncertainty by half: ratio of error between adjacent steps is approximately a constant 0.5.

Not bad, but we can do better.

Is there more information about f(x) we can use to make a better method?

```
import time

print('Lecture paused')
time.sleep(300)
print('More information')
```

Use the slope f'(x)

How can we find a root using the following?

- Starting point x, y = p_0, $f(p_0)$
- Desired y = f(p) = 0
- Slope at p_0 = $f'(p_0)$ ← new information

Remember first-order Taylor series, any function can be approximated as a line:

$$f(p_1) = f(p_0) + f'(p_0)(p_1 - p_0)$$

By plugging in $f(p_1) = 0$, we can solve for p_1 and get a very nice estimate of p

Activity: solve for p_1

How can we find a root using the following?

- Starting point x, y = p_0, $f(p_0)$
- Desired y = f(p) = 0
- Slope at p_0 = $f'(p_0)$ ← new information

Remember first-order Taylor series, any function can be approximated as a line:

$$f(p_1) = f(p_0) + f'(p_0)(p_1 - p_0)$$

Plug in $f(p_1) = 0$

Solve for p_1

Solve for p_1

How can we find a root using the following?

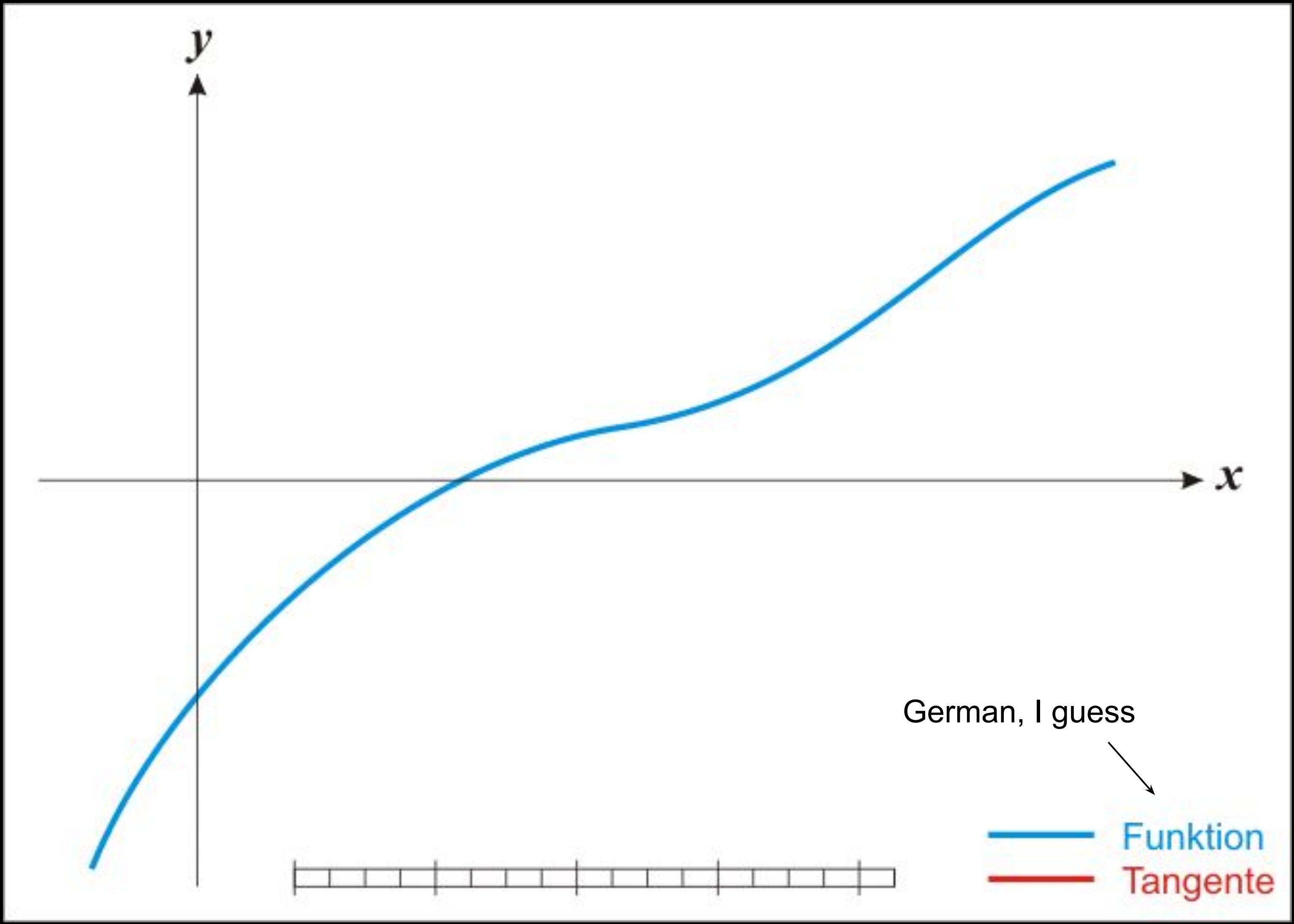
- Starting point x, y = p_0, $f(p_0)$
- Desired y = f(p) = 0
- Slope at p_0 = $f'(p_0)$ ← new information

Remember first-order Taylor series, any function can be approximated as a line:

$$f(p_1) = f(p_0) + f'(p_0)(p_1 - p_0)$$

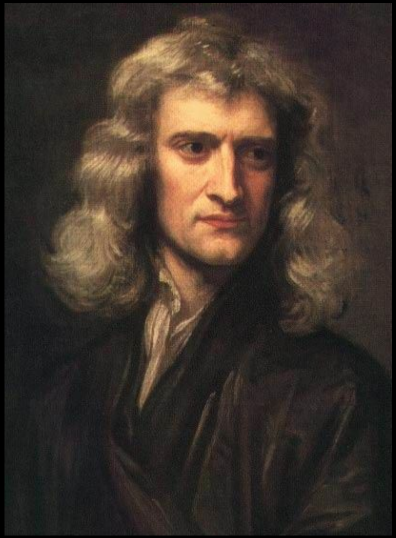
$$\boxed{p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}}$$

Newton's Method



Newton's Method

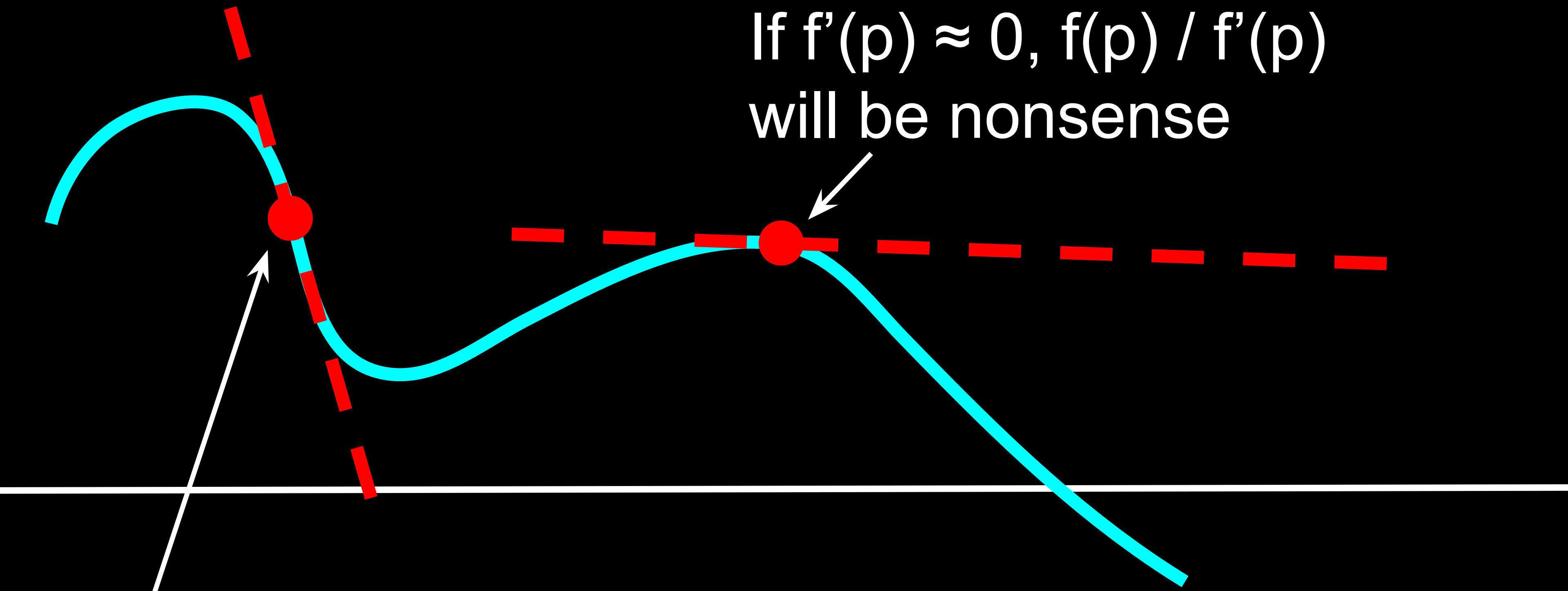
- Using the line defined by p_0, $f(p_0)$, $f'(p_0)$ to calculate your next guess p_1 is called Newton's Method or Newton-Raphson
- <u>Converges faster</u> than Bisection
- <u>Can fail</u> if you're unlucky
- How can it fail?



$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$$

Newton-Raphson failure

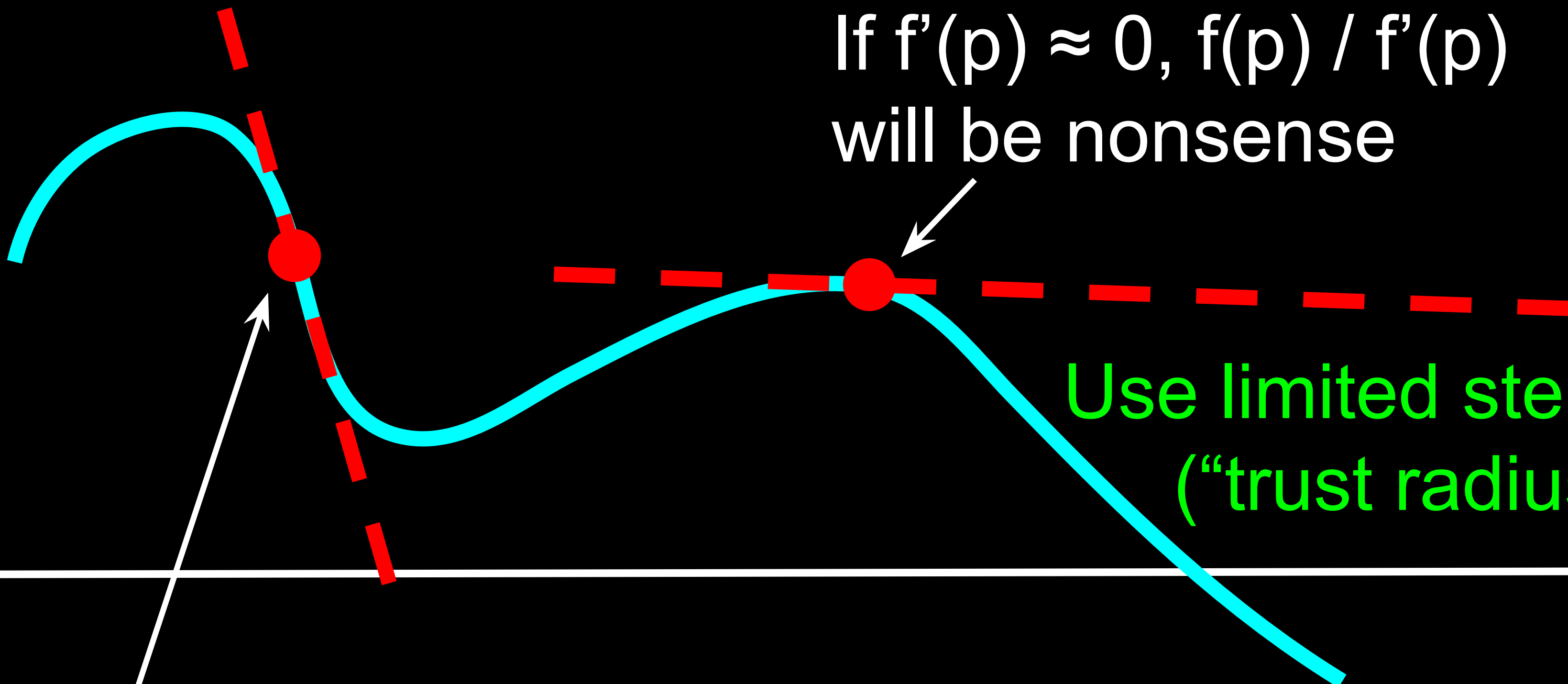
If $f'(p) \approx 0$, $f(p) / f'(p)$ will be nonsense



If p is near a max or min, not a root, we can get stuck

How could we address these problems?

Newton-Raphson failure



If $f'(p) \approx 0$, $f(p) / f'(p)$ will be nonsense

Use limited step size ("trust radius")

If p is near a max or min, not a root, we can get stuck

Use bounds p_{min}, p_{max}

After N iterations, try again with a new guess.

Can use bisection, finish with Newton ("polishing")

Quadratic vs Linear Convergence

An algorithm/sequence converges <u>linearly</u> if, for large n:

$$\left|\frac{p - p_{n+1}}{p - p_n}\right| \leq K$$

The ratio error_{n+1} / error_n is bounded by a constant K (for bisection, K = 0.5)

And <u>quadratically</u> if:

$$\left|\frac{p - p_{n+1}}{(p - p_n)^2}\right| \leq K$$

The ratio error_{n+1} / error_n^2 is bounded by a constant K (drop in error <u>accelerates</u>)

- Newton's Method converges quadratically (if at all), bisection linearly (every time)
- See F&B page 51 for more details

Bisection vs Newton-Raphson

	Bisection	Newton
Convergence?		
Always converges?		
Special conditions?		
Good when?		

Bisection vs Newton-Raphson

	Bisection	**Newton**
Convergence?	Linear	Quadratic
Always converges?	Yes	**No**, if bad p_0 or if we hit $f'(p_n) \approx 0$
Special conditions?	Need the bounds a, b	Need a guess p_0, need to have $f'(x)$
Good when?	We need stability	We need accuracy & speed

Code for Newton-Raphson

```
def f_df(x):  # example function: f(x) = x**2 - 6
   return x**2 - 6, 2 * x  # return f(x) and f'(x)
rel_x_tolerance = 1e-4
n_max = 5
p0 = 1.0  # very simple guess
for i in range(n_max):
   f_p0, df_p0 = f_df(p0)  # get f(x) and f'(x)
   p = p0 - f_p0 / df_p0  # get new point p
   if abs((p - p0) / p) < rel_x_tolerance:
        break
   p0 = p  # try again from new point p
print(f'p**2 = {p**2}')  # p**2 = 6.0000000002
```

Activity: Newton-Raphson √6

Use Newton's Method to find sqrt(6) from an initial guess of $p_0 = 1$:

$$f(x) = x^2 - 6$$

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$$

At each step, record your guess p_n and your relative error bound abs((p_{n-1} - p_n) / p_n). Stop when the relative error bound is under 0.01.

Answer: Newton-Raphson by Hand

$$f(x) = x^2 - 6, \quad f'(x) = 2x \quad \rightarrow$$

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)} = 1 - \frac{(1)^2 - 6}{2(1)} = 3.5 \quad \left(error = \frac{|1 - 3.5|}{|3.5|} = 0.71 \right)$$

$$p_2 = p_1 - \frac{f(p_1)}{f'(p_1)} = 3.5 - \frac{(3.5)^2 - 6}{2(3.5)} = 2.607 \quad \left(error = \frac{|3.5 - 2.607|}{|2.607|} = 0.34 \right)$$

$$p_3 = 2.607 - \frac{(2.607)^2 - 6}{2(2.607)} = 2.454 \quad (error = 0.062)$$

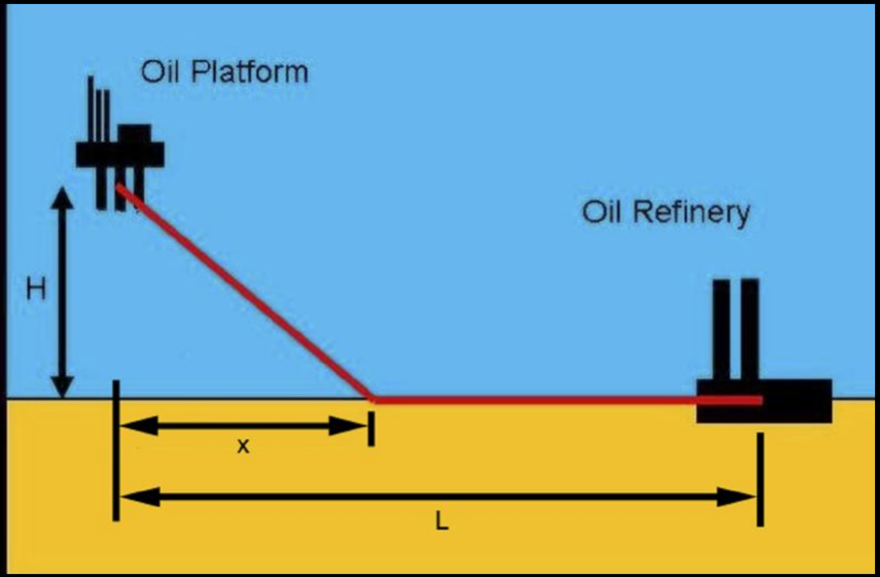
$$p_4 = 2.454 - \frac{(2.454)^2 - 6}{2(2.454)} = 2.449 \quad (error = 0.0019, \quad 4\ iterations)$$

Root Finding Implementations

1. For linear or quadratic, solve analytically
2. For polynomials of order > 2, use "roots" in numpy: numpy.roots(P) gives all the roots
3. For general nonlinear equations, use scipy.optimize: define a function f, then optimize.newton(f, guess) gives one root near your initial guess "guess"
   - What might scipy.optimize.newton do if function f doesn't return its derivative f'?

Summary and Problems

- Open Python Numerical Methods, go to Chapter 19.6: Summary and Problems
- Solve the problem beginning "*Consider the problem of building a pipeline...*"
- Note, same with an offshore wind turbine



All reading for next week: linear, spline, & Lagrange interpolation (PNM 17.1-4), numerical derivatives (PNM 20.1-2) & integrals (PNM 21.1-3)