ENGG1003 - Monday Week 8

Solving nonlinear algebraic equations

Steve Weller

University of Newcastle

26 April 2021

Last compiled: April 25, 2021 2:54pm +10:00

Lecture overview

1. Solving nonlinear algebraic equations pp. 175-176
    - general setting
    - problem: fluid level in measuring cup
2. Bisection method §7.4
3. Secant method §7.3
    - Newton's method
4. Extensions
    - bisection & secant methods: re-write as functions
    - timing code in Python
    - speed comparisons: bisection vs. secant

1) Solving nonlinear algebraic equations

- *linear* equations: $ax + b = 0$
  - solution $x = -b/a$

- *nonlinear* equations
  - quadratic $ax^2 + bx + c = 0$: solution $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
  - cubic and quartic (orders 3 and 4): exact solutions exist but are *very* complicated
  - quintic (order 5) equations: exact solutions *do not exist* in general, proving that needs *serious* mathematics

- most equations in engineering applications have no exact "pen and paper" solutions!

Numerical solutions to equations

"Far better an approximate answer to the right question...
than an exact answer to the wrong question"
—John Tukey

General problem: find x satisfying

$$f(x) = 0$$

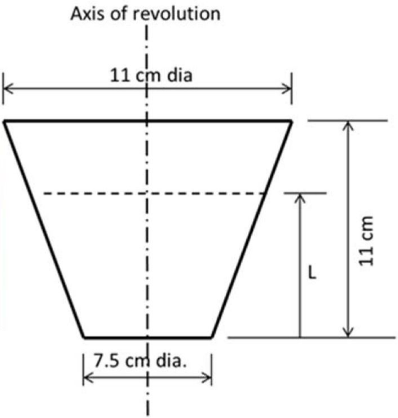
where $f(x)$ is a formula involving x
Example

$$f(x) = e^{-x}\sin(x) - \cos(x)$$

has solution $x = 7.85359326$ because

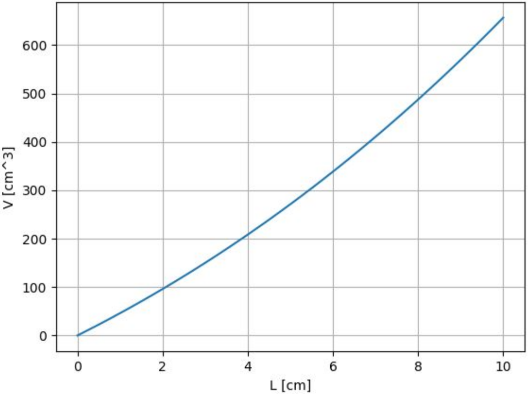
$$e^{-7.85359326}\sin(7.85359326) - \cos(7.85359326) = 0.000$$

Fluid level in truncated cone



- applications: water in dam, coal in conical hopper

Fluid level

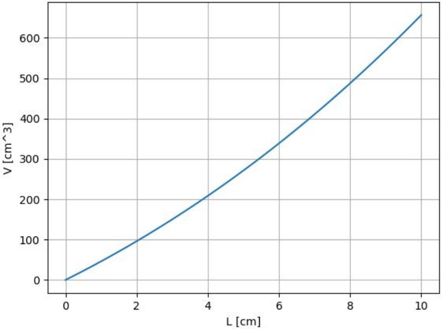


- volume V depends on depth L as follows:

$$V = 0.0268L^3 + 1.884L^2 + 44.15L$$

  - V (in millilitres, mL), L (in cm)

Fluid level



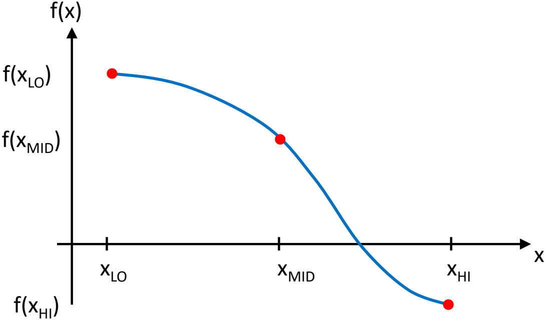
Q: depth L when cup holds 500 mL of water?

- need to solve equation

$$500 = 0.0268L^3 + 1.884L^2 + 44.15L$$

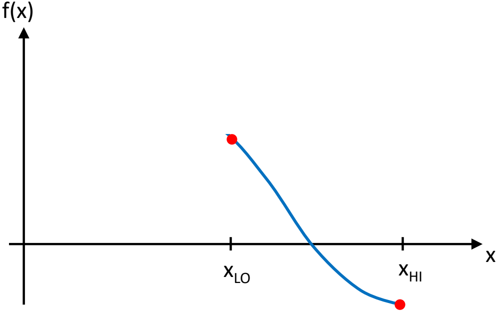
$$f(L) = 0.0268L^3 + 1.884L^2 + 44.15L - 500 = 0$$

2) Bisection method



- continuous function $f(x)$ on interval $[x_{\text{LO}}, x_{\text{HI}}]$, where value of f *changes sign* from x_{LO} to x_{HI}
- divide interval in two, $f(x) = 0$ in one sub-interval
- select sub-interval where sign of f changes & repeat

Bisection method



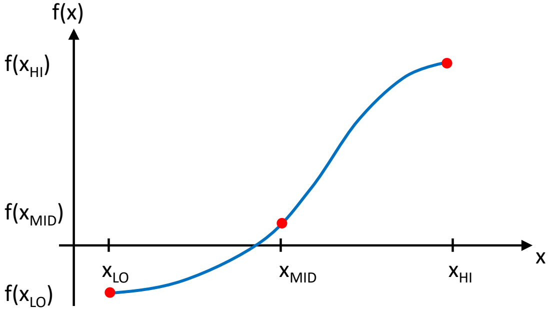
$$f(x_{\mathrm{MID}}) \times f(x_{\mathrm{LO}}) > 0$$

…select *upper sub-interval* by updating

$$x_{\mathrm{LO}} = x_{\mathrm{MID}}$$

…and repeat

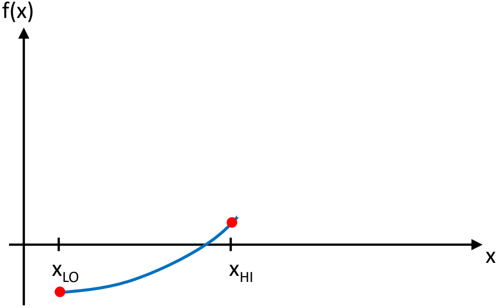
Bisection method



$$f(x_{\mathrm{MID}}) \times f(x_{\mathrm{LO}}) < 0$$

...select *lower sub-interval* by updating $x_{\mathrm{HI}} = x_{\mathrm{MID}}$

Bisection method



...and repeat, with new midpoint

$$x_{\mathrm{MID}} = (x_{\mathrm{LO}} + x_{\mathrm{HI}})/2$$

- each iteration halves the width of interval $[x_{\mathrm{LO}}, x_{\mathrm{HI}}]$
- continue until $|f(x_{\mathrm{MID}})| < \mathrm{tolerance}$, eg: 10^{-6}

Bisection method: Python code

bisection.py

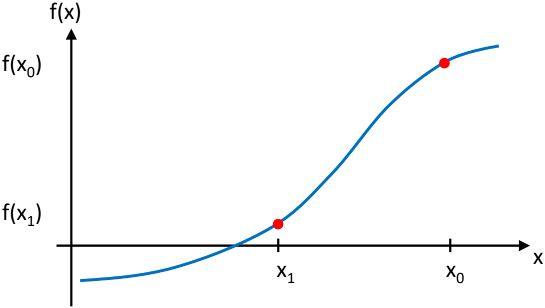
```
def f(L):
    return  0.0268*L**3 + 1.884*L**2 + 44.15*L - 500

eps = 1e-6
x_LO = 6
x_HI = 10

x_MID = (x_LO + x_HI)/2
itCnt = 0
while abs(f(x_MID)) > eps:
    if f(x_MID)*f(x_LO) > 0:
        x_LO = x_MID
    else:
        x_HI = x_MID
    x_MID = (x_LO + x_HI)/2
    itCnt += 1

print('Solution: {}'.format(x_MID))
print('Number of iterations: {}'.format(itCnt))
print('Check: f({:.8f}) = {:.8f}'.format(x_MID, f(x_MID)))
```

Bisection method: simulation results

- lines 1–2: function f, want L such that $f(L) = 0$
- line 4: tolerance 10^{-6}
- lines 9 & 16: count loop iterations
- live demo

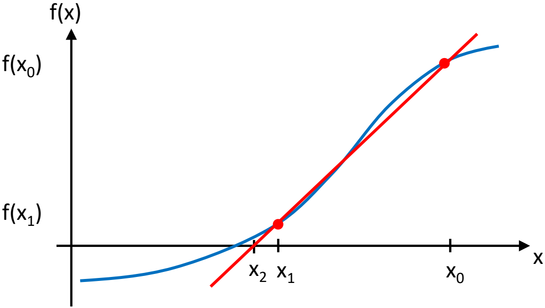
```
Solution: 8.15660098195076
Number of iterations: 26
Check: f(8.15660098) = -0.00000099
```

3) Secant method



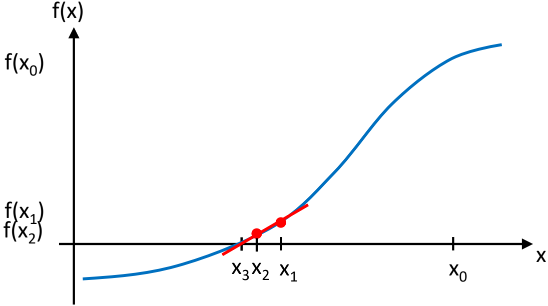
- start with two points $(x_0, f(x_0))$ and $(x_1, f(x_1))$
  - red dots
  - $f(x_0)$ and $f(x_1)$ do *not* necessarily have opposite signs

Secant method



- *secant* is line through $(x_0, f(x_0))$ and $(x_1, f(x_1))$
- define x_2 as point where secant intersects x-axis

Secant method



...and repeat, with x_3 defined as point where secant through $(x_1, f(x_1))$ and $(x_2, f(x_2))$ intersects x-axis

Secant method equations

Equation of secant connecting $(x_0, f(x_0))$ & $(x_1, f(x_1))$:

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \cdot (x - x_1) + f(x_1)$$

Solving for intersection of secant with x-axis:

$$x_2 = x_1 - f(x_1) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

Secant method equations

Equation of secant connecting $(x_0, f(x_0))$ & $(x_1, f(x_1))$:

$$y = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \cdot (x - x_1) + f(x_1)$$

Solving for intersection of secant with x-axis:

$$x_2 = x_1 - f(x_1) \cdot \frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

$$x_3 = x_2 - f(x_2) \cdot \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

$$x_4 = x_3 - f(x_3) \cdot \frac{x_3 - x_2}{f(x_3) - f(x_2)}$$

$$\vdots$$

Secant method: Python code

secant.py

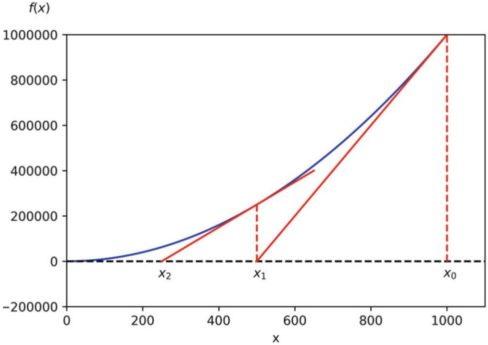
```python
def f(L):
    return  0.0268*L**3 + 1.884*L**2 + 44.15*L - 500

eps = 1e-6
x0 = 6
x1 = 10
itCnt = 0         # iteration counter
while abs(f(x1)) > eps:
    # line (=secant) through (x0,f(x)) and (x1,f(x1)) intersects
    # horizontal axis at (x,0)
    x = x1 - f(x1)*((x1 - x0)/(f(x1) - f(x0)))
    x0 = x1
    x1 = x
    itCnt += 1

print('Solution: {}'.format(x))
print('Number of iterations: {}'.format(itCnt))
print('Check: f({:.8f}) = {:.8f}'.format(x,f(x)))
```

Secant method: simulation results

- lines 12–13: this update simpler to code than $\{x_0, x_1\} \rightarrow x_2$, $\{x_1, x_2\} \rightarrow x_3$, $\{x_2, x_3\} \rightarrow x_4$, ...
- live demo

```
Solution: 8.156600987863818
Number of iterations: 4
Check: f(8.15660099) = -0.00000052
```

Newton's method



- choose initial estimate x_0
- x_1 is intersection of *tangent line* through $(x_0, f(x_0))$
- x_2 is intersection of tangent line through $(x_1, f(x_1))$

...

Newton's method

- also known as *Newton–Raphson method*
- Newton's method much more popular than bisection or secant methods
- calculation of "tangent lines" requires *derivative* of function $f(x)$
    - therefore needs *calculus* (eg: MATH1110)
    - beyond assumed knowledge for ENGG1003, won't consider Newton's method in this course
- secant method can be considered as an approximation to Newton's method

4) Extensions

bisection_fn.py

```
def f(L):
    return L**3 + 70.3*L**2 + 1647.39*L - 18656.72

def my_bisection(f, x_LO, x_HI, tol):
    x_MID = (x_LO + x_HI) / 2
    itCnt = 0
    while abs(f(x_MID)) > tol:
        if f(x_MID) * f(x_LO) > 0:
            x_LO = x_MID
        else:
            x_HI = x_MID
        x_MID = (x_LO + x_HI) / 2
        itCnt += 1
    return x_MID, itCnt

x, numIt = my_bisection(f, 6, 10, 1e-6)

print('Solution: {}'.format(x))
print('Number of iterations: {}'.format(numIt))
print('Check: f({:.8f}) = {:.8f}'.format(x,f(x)))
```

Bisection method as a function

- line 4: `my_bisection` function takes function `f` as first argument

  ... also pass in x_{LO}, x_{HI} and convergence tolerance

- line 14: function returns approximate solution & iteration count

- line 16: call `my_bisection` function with four arguments

- live demo

Secant method as a function

secant_fn.py

```python
def f(L):
    return L**3 + 70.3*L**2 + 1647.39*L - 18656.72

def my_secant(f, x0, x1, tol):
    itCnt = 0
    while abs(f(x1)) > tol:
        x = x1 - f(x1) * ((x1 - x0) / (f(x1) - f(x0)))
        x0 = x1
        x1 = x
        itCnt += 1
    return x1, itCnt

x, numIt = my_secant(f, 6, 10, 1e-6)

print('Solution: {}'.format(x))
print('Number of iterations: {}'.format(numIt))
print('Check: f({:.8f}) = {:.8f}'.format(x,f(x)))
```

- live demo

Timing code in Python

- often useful to measure time taken to perform calculations; easy in Python!
- start by importing `time` module:

```
import time
```

- function `time.perf_counter()` returns value of a clock
  - `float` value (in seconds)
- elapsed time is *difference* between two successive calls

```
tStart = time.perf_counter()
xB, numItB = my_bisection(f, 6, 10, 1e-6)
tStop = time.perf_counter()
tBisect = tStop - tStart
```

Speed comparison: bisection vs. secant

- live demo `bisectionvssecant.py`
- code in #lecturecode

```
Solution (bisection): 8.15660098195076
Number of iterations (bisection): 26
Check: f(8.15660098) = -0.00000099
Run-time (bisection): 6.166e-05 seconds

Solution (secant): 8.156600987863818
Number of iterations (secant): 4
Check: f(8.15660099) = -0.00000052
Run-time (secant): 1.257e-05 seconds

Secant method is 4.9 times as fast as bisection method
```

Lecture summary

- Solving nonlinear algebraic equations
- Bisection method
- Secant method
  - Newton's method
- Extensions

More information

- Newton's method in textbook §7.2
  - needs *differentiation* from calculus (MATH1110)
  - in particular: need expression for *tangent lines* to function $f(x)$, written as $f'(x)$
- "optimised" versions of bisection and secant methods in textbook §7.3 and §7.4
  - maximise speed of computation by minimising number of function evaluations $f(x)$
- volume of truncated cone based on volumes of solids of revolution (needs calculus, MATH1110) https://bit.ly/3sOsaj4