ENGG1003 - Monday Week 8

Solving nonlinear algebraic equations

Steve Weller

University of Newcastle

26 April 2021

Last compiled: April 24, 2021 4:36pm +10:00

Lecture overview

1. Solving nonlinear algebraic equations pp. 175-176
   - general setting
   - two problems: flight time, fluid level
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

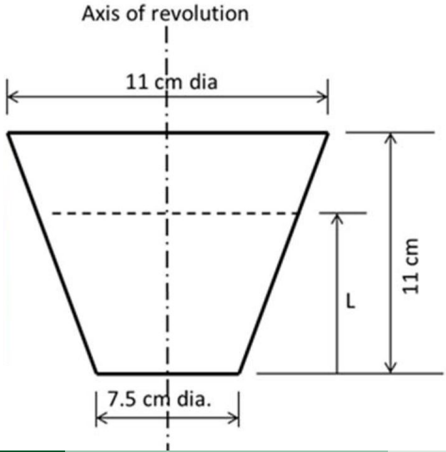
$$e^{-7.85359326}\sin(7.85359326) - \cos(7.85359326) = 0.000$$

Flight time

- one more time!

Fluid level

image of measuring cup
Engineering applications: water in dam, coal in stockpile



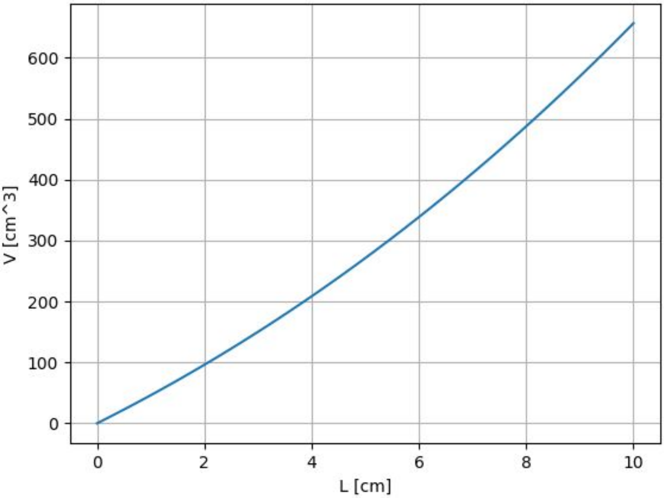
Fluid level

- volume V (in millilitres, mL) depends on depth L (in cm) as follows:

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

- plot V vs L
- link to proof: volumes of solids of revolution (needs calculus, MATH1110)
  `https://www.sjsu.edu/me/docs/hsu-Chap`

Fluid level



Fluid level

- Question: depth L when cup holds 500 mL of water?
- solve $f(L) = 0$ where

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

2) Bisection method

- basic idea: visualisation

Bisection method: pseudocode

```
INPUT: function f
       endpoint values xLO, xHI
       tolerance TOL
CONDITIONS: xLO < xHI
       f(xLO)<0 and f(xHI)>0  or  f(xLO)>0 and f(xHI)<0

xMID = (xLO + xHI)/2
WHILE |f(xMID)| > TOL
  IF f(xMID) is same sign as f(xLO)
    # case A
    set xLO = xMID
  ELSE
    # case B
    set xHI = xMID
  ENDIF
  xMID = (xLO + xHI)/2
END WHILE
```

Bisection method: Python code

bisection.py

```python
import numpy as np

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

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

- code commentary
- simulation results
- live demo

3) Secant method

- basic idea: visualisation

- secant method: key equations

Secant method: Python code

secant.py

```
import numpy as np

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

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)))
```

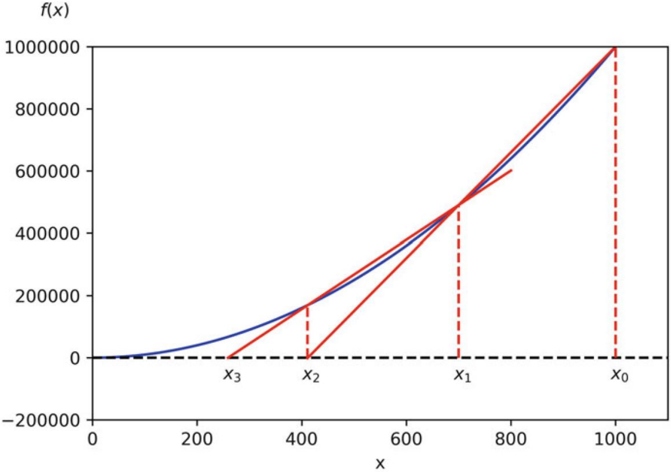
Secant method: simulation results

- code commentary
- simulation results
- live demo

Newton's method

- aka Newton–Raphson method
- discussion of derivatives, and how they're needed in Newton's method
- we won't consider Newton's method in this course, as can't assume knowledge of calculus
- secant as approximation to Newton's method
- Newton's method is *really* popular

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

- code commentary
- simulation results
- 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)))
```

Secant method as a function

- code commentary
- simulation results
- 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 comparisons: bisection vs. secant

- live demo `bisectionvssecant.py`

```
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 $f(x)$