Chapter 11

Root Finding

11.1 Motion with air resistance

Consider a ball of mass m, tossed straight up into the air. Let y denote the height of the ball above ground level. The ball's velocity is $\dot{y}$, where the dot denotes a derivative with respect to time t. The force of gravity acting on the ball is $F_{grav} = -mg$, where g is the acceleration due to gravity. Air resistance also acts on the ball, exerting a force in the direction opposite to the velocity. For low velocities, this force can be approximated as $F_{air} = -mk\dot{y}$, where k is a positive constant. (The factor of m is included for later convenience.)

Newton's second law for the ball is $F_{grav} + F_{air} = m\ddot{y}$, where $\ddot{y}$ is the ball's acceleration. This yields a simple second order differential equation

$$\ddot{y} + k\dot{y} + g = 0 \ , \tag{11.1}$$

for the height y as a function of time t. Standard techniques can be used to derive the solution,

$$y(t) = \left(v_0/k + g/k^2\right)\left(1 - e^{-kt}\right) - (g/k)t \ , \tag{11.2}$$

where v_0 is the initial velocity. The ball is launched from ground level, so the initial height is $y(0) = 0$.

Exercise 11.1a

Use SymPy to check the result (11.2) by substituting $y(t)$ into the differential equation (11.1). Also verify that the initial velocity is $\dot{y}(0) = v_0$.

How long does the ball remain in the air? In other words, for what value of time t (in addition to $t = 0$) is the height equal to zero? The answer is found by setting $y(t) = 0$ in Eq. (11.2) and solving for t. However, it's not possible to solve for t analytically in terms of standard functions like powers, trig functions, logarithms, *etc.* We must resort to numerical methods to find the time in air.

One way to solve $y(t) = 0$ numerically is to start with an initial guess, then increase (or decrease) the guess in small steps until the function $y(t)$ changes sign. Since $y(t)$ has opposite signs for the last two guesses, the function $y(t)$ must vanish for some value of t between those guesses. We can approximate the time where $y(t)$ vanishes by taking the average of the last two guesses.

Exercise 11.1b

Let $g = 9.8$, $k = 0.2$ and $v_0 = 35.0$ (in SI units) in Eq. (11.2). Write a Python program that uses the algorithm described above to solve $y(t) = 0$. Start with an initial guess close to 0 and use a time step size of 0.00001. Use your result to determine how fast the ball is moving when it reaches the ground.

11.2 The general problem

We are often faced with the problem of finding the solution (or solutions) of an algebraic equation of the form $f(x) = 0$. The air resistance problem has this form, but with a simple change of variable names (y instead of f, t instead of x.)

The solutions of $f(x) = 0$ are called the "roots" of the function $f(x)$. In this chapter we develop numerical techniques for finding roots. The goal is to find a root to within some tolerance ϵ. That is, the root should be accurate to within $\pm\epsilon$.

The straightforward method for root finding, described in the previous section, is usually inefficient. To obtain an accurate answer you must use a very small step size, and this requires very many steps (unless your initial guess is really good). There are better methods.

11.3 The bisection method

Bisection is a simple method of finding the roots of a continuous function $f(x)$. Start by assuming a root exists between $x = x_L$ and $x = x_R$. If $f(x_L)$ and $f(x_R)$ have opposite signs, then there must be at least one root between

x_L and x_R. Next, compute the midpoint of the interval $x_M = (x_L + x_R)/2$. Use this value to replace either x_L or x_R, depending on the relative signs of $f(x_L)$, $f(x_M)$ and $f(x_R)$. Repeat this process until x_L and x_R are close enough that the root (which must be between x_L and x_R) can be reasonably approximated by the midpoint $x_M = (x_L + x_R)/2$. The idea is depicted in Fig. 11.1.

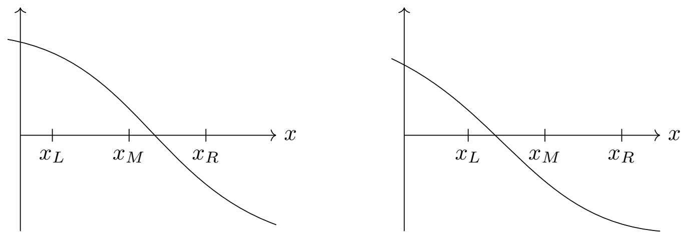

Fig. 11.1: The biscection method. The actual root is where the curve crosses the x–axis. Left panel: $f(x_M)$ has the same sign as $f(x_L)$, so x_M is smaller than the actual root. For the next iteration, x_M becomes the new x_L. Right panel: $f(x_M)$ has the same sign as $f(x_R)$, so x_M is larger than the actual root. For the next iteration, x_M becomes the new x_R.

Here is the step–by–step procedure:

(1) Choose values for x_L and x_R, with $x_L < x_R$.
(2) Verify that $f(x_L)$ and $f(x_R)$ have opposite signs. If not, adjust your choices for x_L and x_R.
(3) Compute the midpoint $x_M = (x_L + x_R)/2$.
(4) If $f(x_M)$ and $f(x_L)$ have the *same* sign, then x_M must be smaller than the root we're looking for. Replace the value of x_L with x_M.
(5) If $f(x_M)$ and $f(x_R)$ have the same sign, then x_M must be *larger* than the root we're looking for. Replace the value of x_R with x_M.
(6) Repeat steps 3, 4 and 5 until $x_R - x_L < 2\epsilon$, where ϵ is some chosen tolerance.
(7) Take the midpoint $(x_L + x_R)/2$ as your final answer. Since the actual root is between x_L and x_R, and $x_R - x_L < 2\epsilon$, the midpoint must be within $\pm\epsilon$ of the actual root.

The bisection method is robust, meaning it almost always succeeds in finding a root.

Exercise 11.3a

Use bisection to solve the equation

$$\exp(-x^2) - x = 0 \ ,$$

with a tolerance of $\epsilon = 0.0001$.

Exercise 11.3b

The bisection method can converge on a vertical asymptote, as well as a root. Consider the function $\tan(x) - x$. Plot this function and apply the bisection method with (initially) $x_L = 1$ and $x_R = 2$. Does this give a root or a vertical asymptote? Use bisection to find the other roots and vertical asymptotes in the domain $0 < x < 6$.

11.4 Newton's method

Newton's method (also known as the Newton–Raphson method) is much more efficient than the bisection method—typically it reaches the answer with a comparable tolerance in fewer steps. However, Newton's method is less robust than bisection, and it can be more difficult to apply because it requires knowledge of the derivative of $f(x)$.

Let x_{old} denote an initial guess for a root of $f(x)$. Consider the Taylor series expansion of $f(x)$ about x_{old}:

$$f(x) = f(x_{old}) + f'(x_{old})(x - x_{old}) + \cdots \ . \tag{11.3}$$

Here, $f'(x_{old})$ is the derivative of $f(x)$ evaluated at x_{old}. The unwritten terms in Eq. (11.3) are proportional to $(x - x_{old})^2$, $(x - x_{old})^3$, *etc.* Let's assume for the moment that x is an exact root of the function, so that $f(x) = 0$. If the initial guess x_{old} is close to the root x, then the unwritten terms in Eq. (11.3) will be small. Dropping these terms and setting $f(x) = 0$, Eq. (11.3) becomes

$$0 = f(x_{old}) + f'(x_{old})(x - x_{old}) \ . \tag{11.4}$$

Solving for x, we find

$$x = x_{old} - \frac{f(x_{old})}{f'(x_{old})} \ . \tag{11.5}$$

This equation gives us the result x for the root. This result is not exact, because we approximated Eq. (11.3) by dropping the higher order terms.

Rather than calling the result x, let's call it x_{new} and write Eq. (11.5) as

$$x_{new} = x_{old} - \frac{f(x_{old})}{f'(x_{old})} \ . \tag{11.6}$$

As noted above, x_{new} is only an approximation to the actual root. Nevertheless, x_{new} is usually a better approximation than our initial guess x_{old}.

Now iterate the process. Let the new approximation x_{new} become the old approximation x_{old} and apply the formula (11.6) again. With each iteration, the new x value is typically closer to the actual root than the old x value. To obtain an answer that is accurate to within some tolerance $\pm\epsilon$, keep iterating until $f(x_n + \epsilon)$ and $f(x_n - \epsilon)$ have opposite signs. This ensures that an actual root is between $x_n + \epsilon$ and $x_n - \epsilon$.

Newton's method is depicted in Fig. 11.2. The slope $f'(x_{old})$ is the "rise" $f(x_{old})$ over the "run" $x_{old} - x_{new}$; that is,

$$f'(x_{old}) = \frac{f(x_{old})}{x_{old} - x_{new}} \ . \tag{11.7}$$

This can be rearranged to give Eq. (11.6).

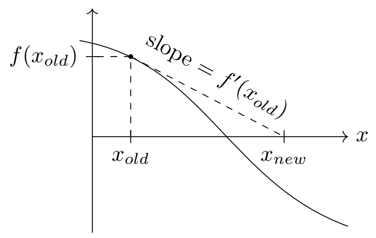

Fig. 11.2: For Newton's method, the new approximation x_{new} is obtained from x_{old} by assuming the function is a straight line with slope $f'(x_{old})$.

Newton's method will fail to find a root if any of the x_{old} values coincide with an extremum of $f(x)$. At an extremum, the derivative $f'(x_{old})$ vanishes and x_{new} becomes infinite.

Newton's method can fail in other ways. Consider, for example, the function $f(x) = (-3x^3 + x^2 + 15x + 3)/8$. If we choose $x_{old} = 1$ as our initial guess, the sequence of old and new estimates of the root will simply switch between 1 and -1. See Fig. 11.3.

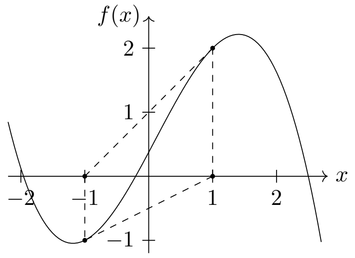

Fig. 11.3: Newton's method applied to the function $f(x) = (-3x^3 + x^2 + 15x + 3)/8$. The initial guess is $x_{old} = 1$. Since $f(1) = 2$ and $f'(1) = 1$, this yields $x_{new} = -1$. Iterating with $x_{old} = -1$, we have $f(-1) = -1$ and $f'(-1) = 1/2$, which gives $x_{new} = 1$.

Exercise 11.4a

Use Newton's method to find the root of $f(x) = x\ln(1 + x^2) + \cos x$ with a tolerance of $\epsilon = 10^{-5}$.

Exercise 11.4b

The equation

$$x^5 + tx^3 + x - t^3 = 0$$

defines x as a function of t; that is, $x(t)$. For a given value of t, we can use Newton's method to find the corresponding x. Using this strategy, write a code that will graph the function $x(t)$ for $-3.0 \leq t \leq 3.0$.

Exercise 11.4c

In the year AD 60, the Greek mathematician Hero of Alexandria described a method for computing $\sqrt{S}$, where S is a positive number. This is now called Heron's method. Start with an initial guess $x_{old} > 0$, then iterate the formula

$$x_{new} = \frac{1}{2}\left(x_{old} + \frac{S}{x_{old}}\right) .$$

As the number of iterations increases, x_{new} approaches $\sqrt{S}$.

- Show that Heron's method is equivalent to Newton's method for finding the positive root of $f(x) = x^2 - S$.
- Write a code that uses Heron's method to compute square roots with a tolerance of $\pm 10^{-8}$.

11.5 Execution time

You can find out how long your codes take to execute using the `time()` function from the `time` library. Begin by importing the library:

```
import time as tm
```

Add `begintime = tm.time()` at the beginning of your code, and `endtime = tm.time()` at the end. The execution time is `endtime - begintime`.

Exercise 11.5a

Use the `time` library to determine the execution time for your code from Exercise 11.1b.

Exercise 11.5b

Let $g = 9.8$, $k = 0.2$ and $v_0 = 35.0$ (in SI units) in Eq. (11.2). Write a code using the bisection method to find the time in air for the ball. Your answer should have a tolerance of $\epsilon = 0.00001$. (That is, it should be accurate to within ± 0.00001.) Use `time` to find the execution time of your code.

Exercise 11.5c

Repeat the previous exercise, but replace the bisection method with Newton's method.

11.6 Multiple roots

What happens if the function $f(x)$ has more than one root?

The bisection method assumes that the initial values of x_L and x_R are chosen such that $f(x_L)$ and $f(x_R)$ have opposite signs. This implies an odd number of roots between x_L and x_R. If there is just one root in the interval $x_L < x < x_R$, then the bisection method will find that root. If there are multiple roots between x_L and x_R, your bisection code will probably find one of the roots, depending on how it is written.

Exercise 11.6a

Graph the cubic polynomial $f(x) = x^3 - 4x^2 + 2$. Use bisection to find all three roots to within a tolerance of $\epsilon = 0.0001$. What happens if there are two roots between the initial x_L and x_R? Three roots?

For a function with more than one root, Newton's method typically finds the root that is closest to the initial guess.

Exercise 11.6b

Graph the function $f(x) = \cos(2x) - x/2$ and estimate the values of the roots. Use Newton's method to find these roots to a tolerance of $\epsilon = 0.0001$. Show that Newton's method finds the closest root as long as your initial guess is within a few tenths of that root. Which root does Newton's method find when your initial guess is about half way between two roots? If your intitial guess is far away from all roots, does Newton's method always find the closest root?

11.7 SciPy optimize

The Python library `scipy.optimize` has built–in functions for root finding. The following code finds the roots of $f(x) = \cos(x) + \sin(x)$:

```
import scipy.optimize as so
import numpy as np

def fxn(x):
     return np.cos(x) + np.sin(x)

# Find root of fxn with initial guess 3 and tolerance 0.0001
rootinfo = so.root(fxn, 3, tol=0.0001)
print(f"root = {rootinfo.x[0]:.4f}")
```

The function `root()` returns a lot of information; here, the information is stored in a variable called `rootinfo`. The numerical approximation to the root is contained in the NumPy array `rootinfo.x`. In this example, `rootinfo.x[0]` is the first (and only) element in the array. Other information returned by `root()` includes `rootinfo.fun`, which is the value of

$f(x)$ at the approximate root and `rootinfo.message`, which tells whether or not the algorithm was successful in finding a root.

The output of this code is `root = 2.3562`. There are other roots which can be found using other initial guesses.

Exercise 11.7a

Consider the function

$$f(x) = \frac{1}{5}x^{3/2} + \frac{1}{9}x + 4\cos(x) - 3\ , \quad x \geq 0\ .$$

Write a code that will plot a graph of $f(x)$ and use the `scipy.optimize` function `root()` to find the roots. Experiment with different values for the tolerance.

The function `root()` can also be used to solve systems of equations. Consider, for example, the two equations

$$x^2 - y - 2 = 0\ , \tag{11.8a}$$

$$y^2 - x - 1 = 0\ , \tag{11.8b}$$

in two unknowns x and y. After importing `scipy.optimize` as `so`, this problem can be solved with the code

```
def fxns(variables):
    x,y = variables
    eq1 = x**2 - y - 2
    eq2 = y**2 - x - 1
    return [eq1,eq2]

# Find root using initial guess x=1, y=2 and tolerance 0.001
rootinfo = so.root(fxns, [1,2], tol=0.001)
print(rootinfo.x)
```

Exercise 11.7b

The system

$$x^3 - y + 1 = 0$$
$$x^2 + xy - y = 0$$

has two real solutions. Use `root()` to find them.

Exercise 11.7c

Use `root()` to solve

$$
\begin{aligned}
x - y^4/10 - z^4/10 = 0 \ , \\
y - x^4/10 - z^4/10 = 0 \ , \\
x^2 + y^2 + z^4/10 - 1 = 0 \ .
\end{aligned}
$$

There are two real solutions.