Chapter 8

Lab 8: Roots and Integrals

8.1 Introduction

In this lab, we will implement numerical alogirthms for root-finding and integration. We will apply the root finding algorithm to a classic problem from quantum mechanics: the bound-state energy levels of particle in finite potential well. We will apply numerical integration to the calculation of the periods of a simple pendulum.

8.2 Roots of a Linear Function

It is usually best to start simple when developing code, so we'll develop our algorithms using a simple linear function with a root at $x = 3$.

△ **Jupyter Notebook Exercise 8.1:** Implement $f(x) = 5\,(x - 3)$ as a python function

```
def f(x):
    # your code ...
```

Calculate the derivative of f and implement it as the function:

```
def fp(x):
    # your code ...
```

Check you code with:

```
for x in [-1,3,4]:
    print(f(x), fp(x))
```

which should return:

```
-20 5
0 5
5 5
```

.

8.3 Bisection Method

In the bisection method, we start from values a_1 and b_1 where $f(a_1) \cdot f(b_1) \leq 0$. This condition insures that the range $[a_1, b_1]$ contains at least one root. The bisection method halves the interval with each iteration, chosing the half that contains at least one root. Given a_n and b_n, we calculate:

$$c_n = (a_n + b_n)/2$$

we then determine:

$$a_{n+1} = \begin{cases} a_n & f(a_n) \cdot f(c_n) \leq 0 \\ c_n & \text{(otherwise)} \end{cases}$$

and

$$b_{n+1} = \begin{cases} c_n & f(a_n) \cdot f(c_n) \leq 0 \\ b_n & \text{(otherwise)} \end{cases}$$

△ **Jupyter Notebook Exercise 8.2:** Implement the python function

```
def bisection(f, a, b):
    # your code
    return a, b # updated values
```

which, given the function f=$f(x)$ and interval defined by a=a_n b=b_n, returns the smaller interval defined by a=a_{n+1} and b=b_{n+1} using the bisection algorithm. Test your code with:

```
print(bisection(f,0,4))
print(bisection(f,3,4))
```

which should have output:

```
(2.0, 4)
(3, 3.5)
```

8.4 Newton's Method

We can use Newton's method to find the roots of a function $f(x)$ if we know its derivative $f'(x)$. From our current best estimate for the root x_n we calculate a better estimate as:

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

△ **Jupyter Notebook Exercise 8.3:** Implement the python function:

```
def newton(f,fp,x):
    # your code
    return x # updated value
```

which, given the function f=$f(x)$, it's derivative fp=$f'(x)$, and the current best estimate for the root x=x_n, returns the improved estimate x_{n+1}. Test your code as:

```
for x in [-100,5,1000]:
    print(newton(f,fp,x))
```

which should return the values 3, 3, and 3. Why does one single iteration of Netwon's method find the root in this case?

8.5 Secant Method

When we do not know the derivative a function, we can use the secant method. The secant method provides an improved estimate for the root x_{n+1} from the previous two estimates x_n and $x_n - 1$ which are used to estimate the derivative and then apply Newton's method:

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

△ **Jupyter Notebook Exercise 8.4:** Implement the python function:

```
def secant(f,a,b):
    # your code here
    return b,c
```

which, given the function f=$f(x)$, and two previous estimates for the root a=x_{n-1} and b=x_n, returns the updated estimates $b = x_n$ and $c = x_{n+1}$ using the secant method. Test your code as:

```
print(secant(f,0,4))
print(secant(f,5,3))
```

which should return

```
(4, 3.0)
(3, 3.0)
```

8.6 Roots of a Quadratic Function

In this section, we will test your root finding algorithms on a quadratic equation:

$$g(x) = (x - 1)(x - 4) \tag{8.1}$$

△ **Jupyter Notebook Exercise 8.5:** Define a python function `g(x)` which returns the value $g(x)$. Calculate the derivative $g'(x)$ analytically, and define `gp(x)` which returns $g'(x)$. Plot $g(x)$ and $g'(x)$ for $0 < x < 5$. Be sure to include axis labels and a legend.

△ **Jupyter Notebook Exercise 8.6:** Apply five iterations of your function `bisection` to the range (0,2.4). After each iteration, print out a, b, $g(a)$, and $g(b)$. Did you approach the correct root?

△ **Jupyter Notebook Exercise 8.7:** Starting from the value x=2.4, apply five iterations of your function `newton`. After each iteration, print x and $g(x)$ Did you approach a root?

△ **Jupyter Notebook Exercise 8.8:** Starting from estimates a=0 and b=2.5, apply five iterations of your function `secant`. After each iteration, print out a, b, $g(a)$, and $g(b)$. Did you approach the correct root?

8.7 Particle in a Finite Potential Well

Suppose a particle of mass m and energy E is located in a finite potential well of form:

$$V(x) = \begin{cases} V_0 & x \leq -L/2 \\ 0 & -L/2 < x < L/2 \\ V_0 & L/2 \leq x \end{cases}$$

We are going to consider the interesting situation where $E < V_0$, that is, when the particle is bound by the potential. For simplicity, we will also assume the particle is in a symmetric state, such that wave function ψ has the property $\psi(-x) = \psi(x)$. In this case, the solutions to the Schrodinger Equation satisfy the transcendental equation:

$$v \tan v = \sqrt{v_0^2 - v^2} \tag{8.2}$$

where

$$v = L\sqrt{\frac{m\,E}{2\hbar}}$$

and

$$v_0^2 = \frac{m\,L^2\,V_0}{2\hbar}$$

are both dimensionless quantities, related to the particle energy E and the potential energy of the well V_0. If you haven't yet encountered this essential problem from quantum mechanics, you can just take my word about Equation 8.2 for now.

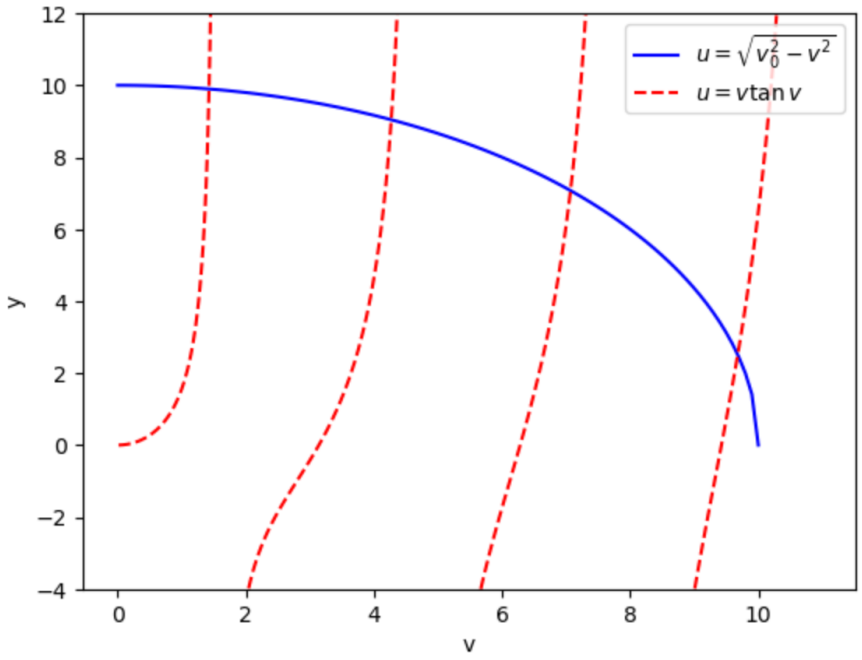

Figure 8.1: Graphical solution to Equation 8.2 for $v_0 = 10$

Most quantum mechanical textbooks suggest that the solutions to Equation 8.2 may be obtained "graphically", by plotting $u = v\tan v$ and $u = \sqrt{v_0^2 - v^2}$ and observing where they intersect. This

approach is illustrated in Fig. 8.1 for $v_0 = 10$. You can see from the plot that there are four places where these curves intersect. In a classical system, the particle could have any $E < V_0$, but in quantum mechanical system, the particle may have only one of the four energy values corresponding to the four intersection points. The number of interesections, and where they occur, depends on the particular value of v_0.

△ **Jupyter Notebook Exercise 8.9:** Reproduce Fig. 8.1. If you attempt to plot the dashed red lines all at once, you will introduce superflous vertical lines from when $\tan v$ switches from $+\infty$ to $-\infty$. To avoid this, plot each continuous region of $u = v \tan v$ separately, in the regions $(-\pi/2, \pi/2)$, then $(\pi/2, 3\pi/2)$, and so on. You may find it useful to remove problematic end points from a numpy array by slicing them off like this `x=x[1:-1]`. Instead of the math formulas, your legend may use the simpler labels LHS and RHS , which refer to the left hand side and right hand side of Equation 8.2. Note: you will have to use 'r-' for each tangent plot to keep it from using different colors. Also, only put a label on the first one, so you don't get a bunch of separate legend entries.

Determining the bound state energy levels of quantum system is of fundamental importance. For just one example, we are able to determine the chemical composition of stars using spectroscopy. The light frequencies which are readily absorbed by each element are determined from the difference in the bound state energy levels. In this section, we will use numerical techniques to accurately determine the values of v which satisfy Equation 8.2. This is equivalent to finding the bound-state energy levels. We will do so by finding the roots of the equation:

$$h(x) = v \tan v - \sqrt{v_0^2 - v^2} \tag{8.3}$$

△ **Jupyter Notebook Exercise 8.10:** Define a function:

```
def h(x):
    # your code
```

which implements $h(x)$ from Equation 8.3 for $v_0 = 10$. Also define a function for the analytical derivative $h'(x)$[1] as

```
def hp(x):
    # your code
```

Test these with

```
for v in [1,1.5,2,4.5]:
    print(np.around(h(v),2),np.around(hp(v),2))
```

This should return the values -8.39, 11.27, -14.17, and 11.94 for $h(x)$ and 5.08, 314.03, 9.57, and 106.41 for $h'(x)$

For each of the following exercises, you will find the third root; i.e. the one between $v = 6$ and $v = 8$, for the case where $v_0 = 10$ In each case, you will continue to iterate until $|h(x)| < .0001$. In each case, print the total number of iterations required and the final value of $h(x)$

[1] Numpy has no secant function so use $\frac{d}{dx} \tan x = \frac{1}{\cos^2 x}$.

△ **Jupyter Notebook Exercise 8.11:** Starting with $a = 6$ and $b = 8$, use the bisection method to find the root. Use $h(a)$ to test the value after each iteration.

△ **Jupyter Notebook Exercise 8.12:** Starting with $a = 6$ and $b = 8$, use the secant method to find the root. Use $h(a)$ to test the value after each iteration.

△ **Jupyter Notebook Exercise 8.13:** Starting with $x = 6$ and your definition of $h'(x)$, use Newton's method to find the root.

8.8 Trapezoid Method

The trapezoid method approximates the definite integral

$$I = \int_a^b f(x)\,dx$$

from the function evaluated at n evenly spaced points

$$I_T = \frac{h}{2}(f(a) + f(b)) + h\sum_{i=1}^{n-2} f(x_i)$$

where

$$h = \frac{b-a}{n-1}$$

and

$$x_i = a + h \cdot i; \quad 0 \leq i < n$$

The truncation error is:

$$I - I_T = \mathcal{O}(h^2).$$

Notice that the sum is over the interior points, which have twice the weight of the end points at a and b.

△ **Jupyter Notebook Exercise 8.14:** Implement the python function:

```
def trap(f,a,b,n):
    # your code
    return sum
```

Which returns the integral of f=$f(x)$ from a to b, using the trapezoid method from n evenly spaced points. Test your code by approximating

$$\int_0^\pi \sin(x)dx$$

with $n = 2, 3$ and 4

```
print(np.around(trap(np.sin,0,np.pi,2),2))
print(np.around(trap(np.sin,0,np.pi,3),2))
print(np.around(trap(np.sin,0,np.pi,4),2))
```

which should output the values 0.00, 1.57, and 1.81. Explain the zero value.

8.9 Iterative Trapezoid Method

In the iterative trapezoid method, we halve the spaces between points during each interation, so that:

$$h_0 = (b-a),\ h_1 = \frac{b-a}{2},\ \ldots\ ,\ h_m = \frac{b-a}{2^m}$$

Note that these correspond to $n = 2$, $n = 3 \rightarrow n = 2^m + 1$ in our function in the previous section.

We then define I_m as

$$I_m = \text{trap}(f, a, b, (2^m + 1))$$

△ **Jupyter Notebook Exercise 8.15:** Write an iterative routine

```
def itertrap(f,a,b,eps):
    # your code
    return I, m # Integral and iteration number
```

which starts at $m = 0$ and continues to call the iter routine with updated values of $n = 2^m - 1$ until

$$|I_m - I_{m-1}| < eps$$

It then returns the integral I and the final value of m.

Use this calculate the definite integral:

$$I = \int_0^{\pi} \sin(x)\, dx$$

to an accurace of $eps = 10^{-6}$. What is the final value? How many iterations did it take? (Remember, the number of iterations is equal to one more than the final value of m.

8.10 Period of a Pendulum

In this section we will consider the oscillation of a pendulum of length L in a graviational field with magnitude g. For small angles, the period of a pendulum oscillation is approximately a constant value:

$$T_0 = 2\pi\frac{L}{g}$$

For oscillations at larger angles, the period of the oscillation cannot be calculated analytically. However, it can be expressed as a definite integral:

$$\frac{T}{T_0} = \frac{2}{\pi}\int_0^{\pi/2} \frac{1}{\sqrt{1 - k^2 \sin u}}\, du \tag{8.4}$$

where:

$$k = \sin\frac{\theta_0}{2}$$

△ **Jupyter Notebook Exercise 8.16:** Estimate the ratio T/T_0 for $\theta_0 = 1$ using your iterative trapezoid routine. Iterate until the estimated truncation error is less than 10^{-6}. What is the final value of I? How many iterations did it take?