Lab 1

Newton's Method and Basins of Attraction

Lab Objective: *Understand Newton's Method. Understand the definition of a basin of attraction.*

Newton's method finds the zeros of functions; that is, Newton's method finds $\overline{x}$ such that $f(\overline{x}) = 0$. This method can be used to optimize functions (find their maxima and minima). For example, it can be used to find the zeros of the first derivative.

Newton's Method

Newton's method begins with an initial guess x_0. Then recursively define a sequence by

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

In other words, Newton's method approximates a function by its tangent line, and then uses the zero of the tangent line as the next guess for x_n (see Figure **??**).

The sequence $\{x_n\}$ will converge to the zero $\overline{x}$ of f if

1. f, f', and f'' exist and are continuous,
2. $f'(\overline{x}) \neq 0$, and
3. x_0 is "sufficiently close" to $\overline{x}$.

In applications, the first two conditions usually hold. However, if $\overline{x}$ and x_0 are not "sufficiently close," Newton's method may converge very slowly, or it may not converge at all.

Newton's method is powerful because given the three conditions above, it converges quickly. In these cases, the $\{x_n\}$ converge to the actual root quadratically, meaning that the maximum error is squared at every iteration.

Let us do an example with $f(x) = x^2 - 1$. We define $f(x)$ and $f'(x)$ in Python as follows.

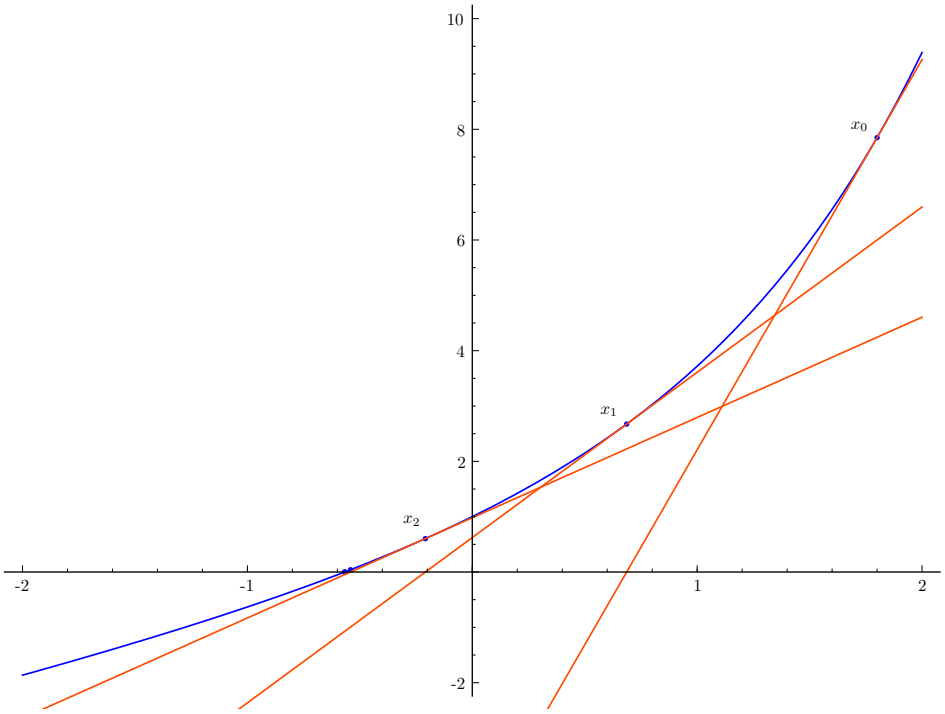

Figure 1.1: An illustration of how one iteration of Newton's method works.

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> f = lambda x : x**2 - 1
>>> Df = lambda x : 2*x
```

Now we set $x_0 = 1.5$ and iterate.

```
>>> xold = 1.5
>>> xnew = xold - f(xold)/Df(xold)
>>> xnew
1.0833333333333333
```

We can repeat this as many times as we desire.

```
>>> xold = xnew
>>> xnew = xold - f(xold)/Df(xold)
>>> xnew
1.0032051282051282
```

We have already computed the root 1 to two digits of accuracy.

Problem 1.

Implement Newton's method with the following function.

```
def Newtons_method(f, x0, Df, iters=15, tol=1e-5):
    '''
    Use Newton's method to approximate a zero of a function.
    Inputs:
        f (function): A function handle. Should represent a function from R to R.
        x0 (float): Initial guess.
        Df (function): A function handle. Should represent the derivative of f.
        iters (int): Maximum number of iterations before the function returns. Defaults to 15.
        tol (float): The function returns when the difference between successive approximations is less than tol.
    Returns:
        A tuple (x, converged, numiters) with
        x (float): the approximation for a zero of f;
        converged (bool): a Boolean telling whether Newton's method converged;
        numiters (int): the number of iterations the method computed.
    '''
```

Problem 2. 1. Run `Newtons_method()` on $f = \cos(x)$ with $x_0 = 1$ and $x_0 = 2$. How many iterations are required to acheive five digits of accuracy (tolerance of 10^{-5})?

2. Newton's method can be used to find zeros of functions that are hard to solve for analytically. Plot $f(x) = \frac{sin(x)}{x} - x$ on $[-4, 4]$. Note that this function can be made continuous on this domain by defining $f(0) = 1$. Use your function `Newtons_method()` to compute the zero of this function to seven digits of accuracy.

3. Run `Newtons_method()` on $f(x) = x^9$ with $x_0 = 1$. How many iterations are required to get five digits of accuracy? Why is it so slow?

4. Run `Newtons_method()` on $f(x) = x^{1/3}$ with $x_0 = .01$. What happens and why? Hint: The command `x**(1/3)` will not work when `x` is negative. Here is one way to define the function $f(x) = x^{1/3}$ in NumPy.

```
f = lambda x: np.sign(x)*np.power(np.abs(x), 1./3)
```

Problem 3. (Optional) Modify the function `Newtons_method()` in Problem 1 so that the argument `Df` defaults to `None`. If no derivative is passed to the function, use the centered coefficients method of Lab **??** to compute the derivative.

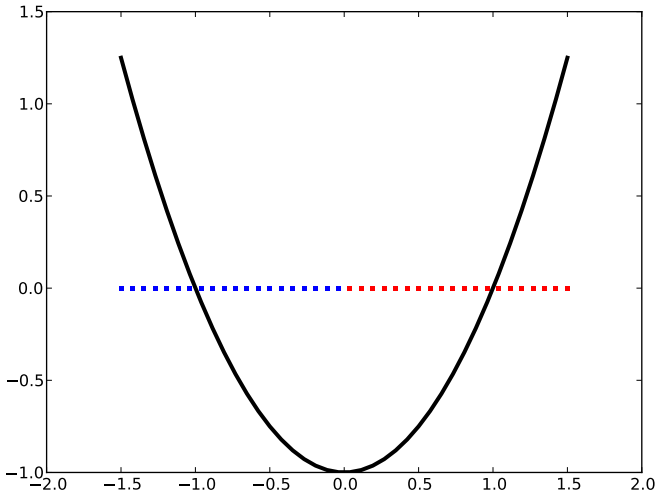

Figure 1.2: The plot of $f(x) = x^2 - 1$ along with some values for x_0. When Newton's method is initialized with a blue value for x_0 it converges to -1; when it is initialized with a red value it converges to 1.

Basins of Attraction: Newton Fractals

When $f(x)$ has many roots, the root that Newton's method converges to depends on the initial guess x_0. For example, the function $f(x) = x^2 - 1$ has roots at -1 and 1. If $x_0 < 0$, then Newton's method coverges to -1; if $x_0 > 0$ then it converges to 1 (see Figure 1.2). We call the regions $(-\infty, 0)$ and $(0, \infty)$ *basins of attraction*.

When f is a polynomial of degree greater than 2, the basins of attraction are much more interesting. For example, if $f(x) = x^3 - x$, the basins are depicted in Figure 1.3.

We can extend these examples to the complex plane. Newton's method works in arbitrary Banach spaces with slightly stronger hypotheses (see Chapter 7 of Volume 1), and in particular it holds over $\mathbb{C}$.

Let us plot the basins of attraction for $f(x) = x^3 - x$ on the domain $\{a + bi \mid (a, b) \in [-1.5, 1.5] \times [-1.5, 1.5]\}$ in the complex plane. We begin by creating a 700×700 grid of points in this domain. We create the real and imaginary parts of the points separately, and then use `np.meshgrid()` to turn them into a single grid of complex numbers.

```
>>> xreal = np.linspace(-1.5, 1.5, 700)
>>> ximag = np.linspace(-1.5, 1.5, 700)
>>> Xreal, Ximag = np.meshgrid(xreal, ximag)
>>> Xold = Xreal+1j*Ximag
```

Recall that `1j` is the complex number i in NumPy. The array `Xold` contains 700^2 complex points evenly spaced in the domain.

We may now perform Newton's method on the points in `Xold`.

```
>>> f = lambda x : x**3-x
```

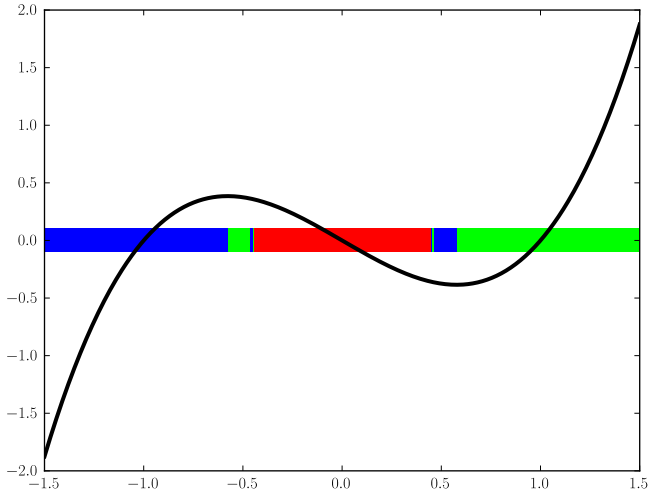

Figure 1.3: The plot of $f(x) = x^3 - x$ along with some values for x_0. Blue values converge to -1, red converge to 0, and green converge to 1.

```
>>> Df = lambda x : 3*x**2 - 1
>>> Xnew = Xold - f(Xold)/Df(Xold)
```

After iterating the desired number of times, we have an array `Xnew` whose entries are various roots of $x^3 - x$.

Finally, we plot the array `Xnew`. The result is similar to Figure 1.4.

```
>>> plt.pcolormesh(Xreal, Ximag, Xnew)
```

Notice that in Figure 1.4, whenever red and blue try to come together, a patch of green appears in between. This behavior repeats on an infinitely small scale, producing a fractal. Because it arises from Newton's method, this fractal is called a *Newton fractal*.

Newton fractals tell us that the long-term behavior of the Newton method is extremely sensitive to the initial guess x_0. Changing x_0 by a small amount can change the output of Newton's method in a seemingly random way. This is an example of *chaos*.

Problem 4.

Complete the following function to plot the basins of attraction of a function.

```
def plot_basins(f, Df, roots, xmin, xmax, ymin, ymax, numpoints=100, iters=15, colormap='brg'):
    '''Plot the basins of attraction of f.

    INPUTS:
```

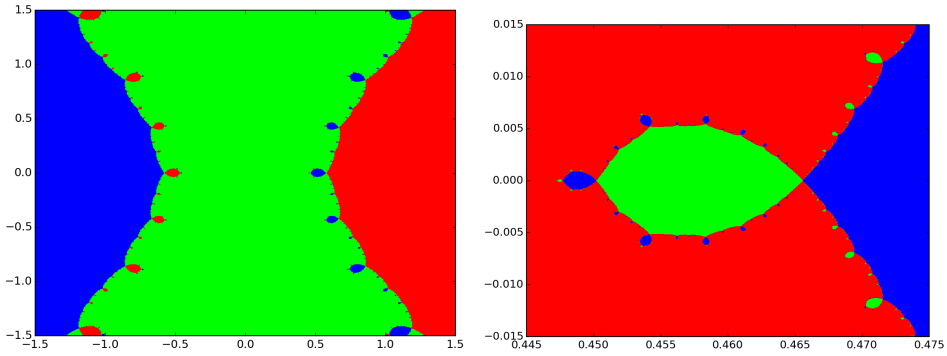

Figure 1.4: Basins of attraction for $x^3 - x$ in the complex plane. The picture on the right is a close-up of the figure on the left.

```
    f       - A function handle. Should represent a function
            from C to C.
    Df      - A function handle. Should be the derivative of f.
    roots   - An array of the zeros of f.
    xmin, xmax, ymin, ymax - Scalars that define the domain
            for the plot.
    numpoints - A scalar that determines the resolution of
            the plot. Defaults to 100.
    iters   - Number of times to iterate Newton's method.
            Defaults to 15.
    colormap - A colormap to use in the plot. Defaults to 'brg'.
    '''
```

You can test your function on the example $f(x) = x^3 - x$ above.

When the function `plt.pcolormesh()` is called on a complex array, it evaluates only on the real part of the complex numbers. This means that if two roots of `f` have the same real part, their basins will be the same color if you plot directly using `plt.pcolormesh()`.

One way to fix this problem is to compute `Xnew` as usual. Then iterate through the entries of `Xnew` and identify which root each entry is closest to using the input `roots`. Finally, create a new array whose entries are integers corresponding to the indices of these roots. Plot the array of integers to view the basins of attraction. Hint: The roots of $f(x) = x^3 - x$ are $[0, 1, -1]$.

Problem 5. Run `plot_basins()` on the function $f(x) = x^3 - 1$ on the domain $\{a + bi \mid (a, b) \in [-1.5, 1.5] \times [-1.5, 1.5]\}$. The resulting plot should look like Figure 1.5. Hint: the roots of $f(x) = x^3 - 1$ are $[1, -1j^{1/3}, 1j^{2/3}]$.

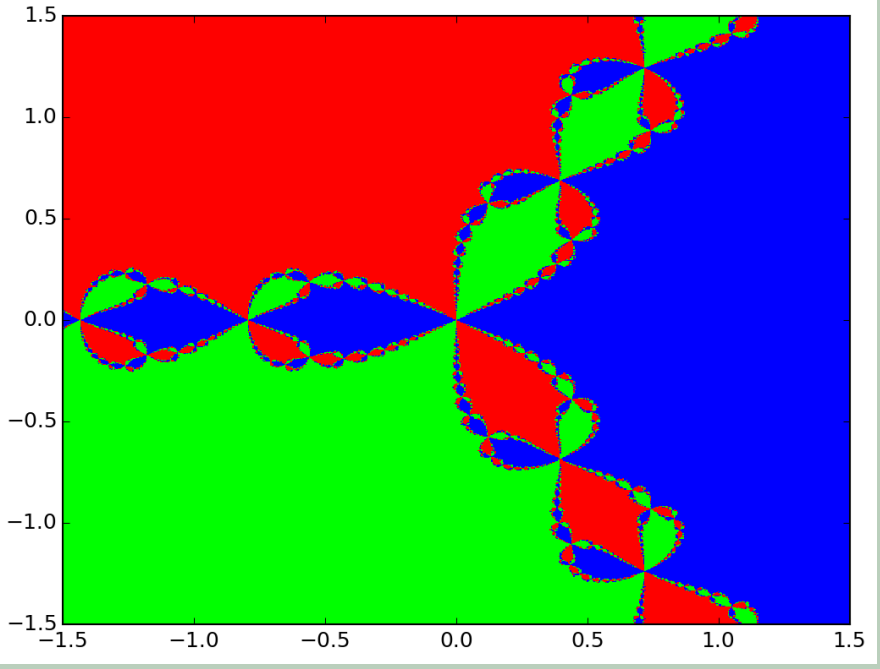

Figure 1.5: Basins of attraction for $x^3 - 1$.