Lab 16

# Newton's Method and Basins of Attraction

**Lab Objective:** *Use Newton's Method to find zeros of a function. Determine where an initial point will converge to based on basins of attraction.*

Newton's method finds the roots of functions; that is, it finds $\overline{x}$ such that $f\left(\overline{x}\right) = 0$. This method can be used in optimization to determine where the maxima and minima occur. 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 $x_0$. Successive approximations of the root are found with the following recursive sequence:

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

In other words, Newton's method approximates the root of a function by finding the x-intercept of the tangent line at $(x_n, f(x_n))$ (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 sequence $\{x_n\}$ converges 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.

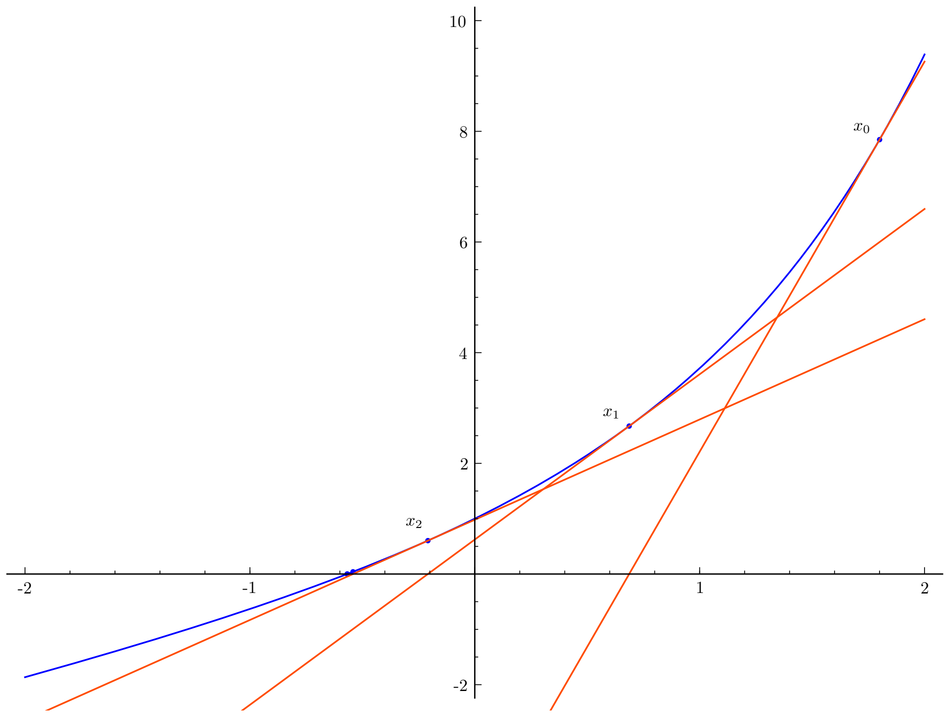

Figure 16.1: An illustration of how two iterations of Newton's method work.

```
>>> 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 a function that accepts the following parameters: a function $f$, an initial x-value, the derivative of the function $f$, the number of iterations of Newton's method to perform that defaults to 15, and a tolerance that defaults to $10^{-6}$. The function returns when the difference between successive approximations is less than the tolerance or the max number of iterations has been reached.

**Problem 2.**

1. 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.

2. 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.** Suppose that an amount of $P_1$ dollars is put into an account at the beginning of years $1, 2, ..., N_1$ and that the account accumulates interest at a fractional rate $r$. (For example, $r = .05$ corresponds to 5% interest.) Suppose also that, at the beginning of years $N_1 + 1, N_1 + 2, ..., N_1 + N_2$, an amount of $P_2$ dollars is withdrawn from the account and that the account balance is exactly zero after the withdrawal at year $N_1 + N_2$. Then the variables satisfy the following equation:

$$P_1[(1+r)^{N_1} - 1] = P_2[1 - (1+r)^{-N_2}].$$

If $N_1 = 30, N_2 = 20, P_1 = 2000$, and $P_2 = 8000$, use Newton's method to determine $r$. (From Atkinson Page 118)

## Backtracking

There are times when Newton's method may not converge due to the fact that the step from $x_n$ to $x_{n+1}$ was too large and the zero was stepped over completely. This was seen in Problem 2 when using $x_0 = .01$ to find the zero of $f(x) = x^{1/3}$. In that example, Newton's method did not converge since it stepped over the zero of the function, produced $x_1 = -.02$ and each iteration got increasingly more negative. To combat this problem of overstepping, backtracking is a useful tool. Backtracking is simply taking a fraction of the full step from $x_n$ to $x_{n+1}$. Define Newton's Method with the recursive sequence:

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

and the vector version of Newton's Method as:

$$x_{n+1} = x_n - \alpha Df(x_n)^{-1} f(x_n).$$

Previously, we have used $\alpha = 1$ in Newton's method. Backtracking uses $\alpha < 1$ in the above sequences and allows us to take a fraction of the step when the step size is too big.

**Problem 4.** 1. Modify your `Newtons_method()` function so that it accepts a parameter $\alpha$ that defaults to 1 to allow backtracking.

2. Find an $\alpha < 1$ so that running `Newtons_method()` on $f(x) = x^{1/3}$ with $x_0 = .01$ converges. (See Problem 2). Return the results of `Newtons_method()`.

**Problem 5.** 1. Create a `Newtons_vector()` function that performs Newton's method on vectors.

2. Bioremediation involves the use of bacteria to consume toxic wastes. At steady state, the bacterial density $x$ and the nutrient concentration $y$ satisfy the system of nonlinear equations

$$\gamma xy - x(1 + y) = 0,$$

$$-xy + (\delta - y)(1 + y) = 0,$$

where $\gamma$ and $\delta$ are parameters that depend on various physical features of the system. For this problem, assume the typical values $\gamma = 5$ and $\delta = 1$, for which the system has solutions at $(x, y) = (0, 1), (0, -1)$, and $(3.75, .25)$. Solve the system using Newton's method and Newton's method with backtracking. (Find an initial point where using $\alpha = 1$ converges to either $(0, 1)$ or $(0, -1)$ and using $\alpha < 1$ converges to $(3.75, .25)$). Use `matplotlib` to demonstrate the tracks used to find the solution. (See Figure 16.2) Hint: use starting values within the rectangle

$$(x, y) : -.25 \leqslant x \leqslant .25, -.25 \leqslant y \leqslant .25.$$

(Adapted from problem 5.19 of M. T. Heath, Scientific Computing, an Introductory Survey, 2nd edition, McGraw?Hill, 2002 and the Notes of Homer Walker)

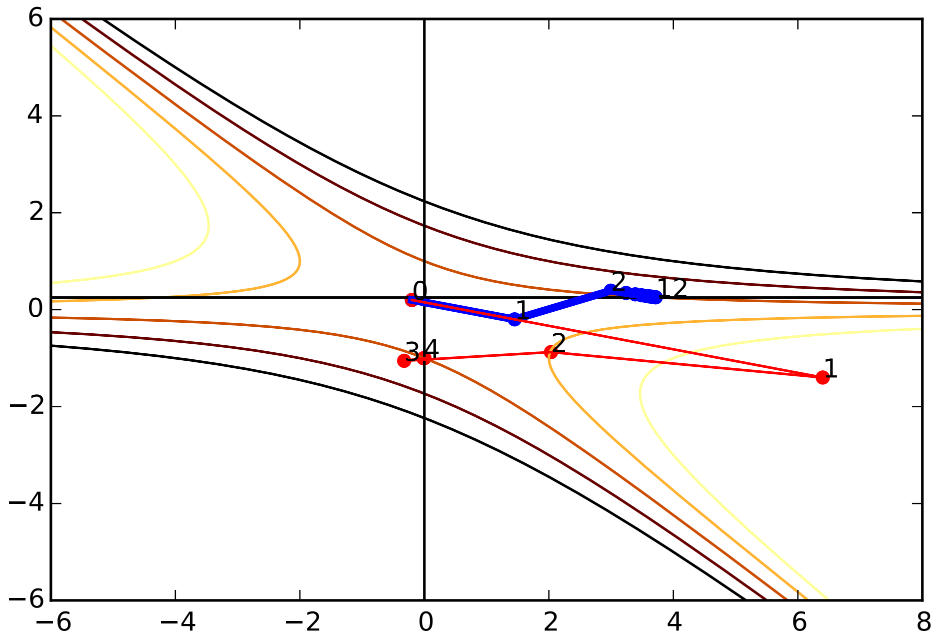

Figure 16.2: Starting at the same initial value results in convergence to two different solutions. The red line converges to $(0, -1)$ with $\alpha = 1$ in 4 iterations of Newton's method while the blue line converges to $(3.75, .25)$ with $\alpha < 1$ in 12 iterations .

## 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 converges to -1; if $x_0 > 0$ then it converges to 1 (see Figure 16.3). 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 16.4.

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 $700 \times 700$ 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
```

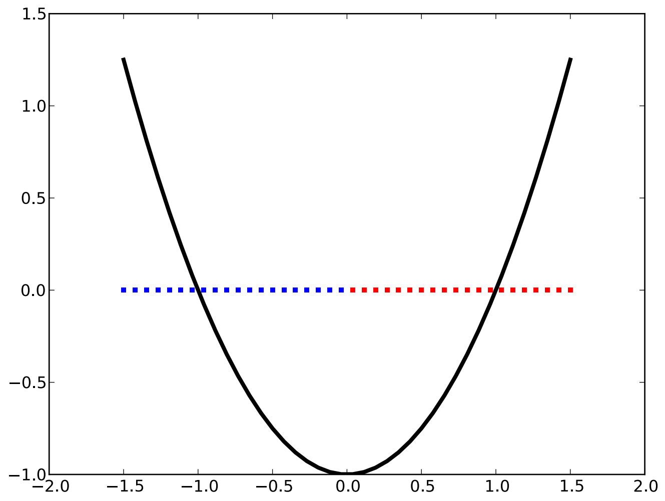

Figure 16.3: The plot of $f(x) = x^2 - 1$ along with some values for $x_0$. When Newton's method is initialized with a blue value for $x_0$ it converges to -1; when it is initialized with a red value it converges to 1.

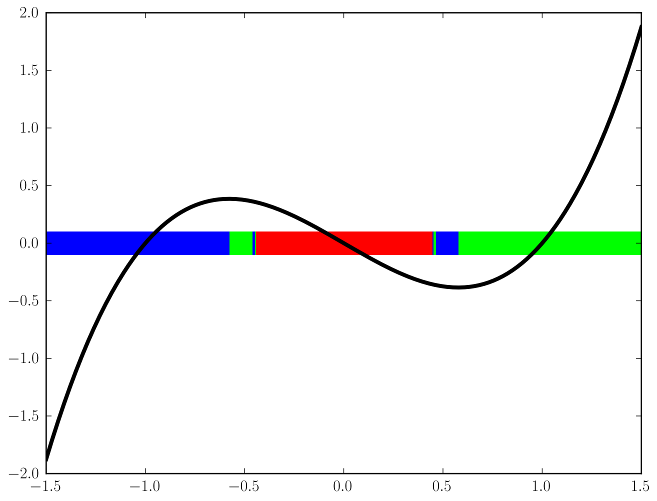

Figure 16.4: The plot of $f(x) = x^3 - x$ along with some values for $x_0$. Blue values converge to $-1$, red converge to 0, and green converge to 1.

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
>>> 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

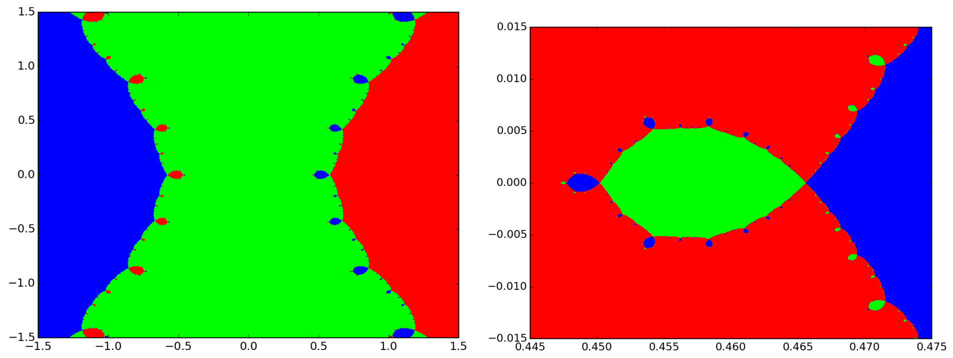

Figure 16.5: Basins of attraction for $x^3 - x$ in the complex plane. The picture on the right is a close-up of the figure on the left.

are various roots of $x^3 - x$.

Finally, we plot the array `Xnew`. The result is similar to Figure 16.5.

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

Notice that in Figure 16.5, 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 6.** 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:
    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, and $-1$.)

**Problem 7.** 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 16.6.
(Hint: the roots of $f(x) = x^3 - 1$ are the third roots of unity: 1, $-\frac{1}{2} + \frac{\sqrt{3}}{2}i$, and $-\frac{1}{2} - \frac{\sqrt{3}}{2}i$.)

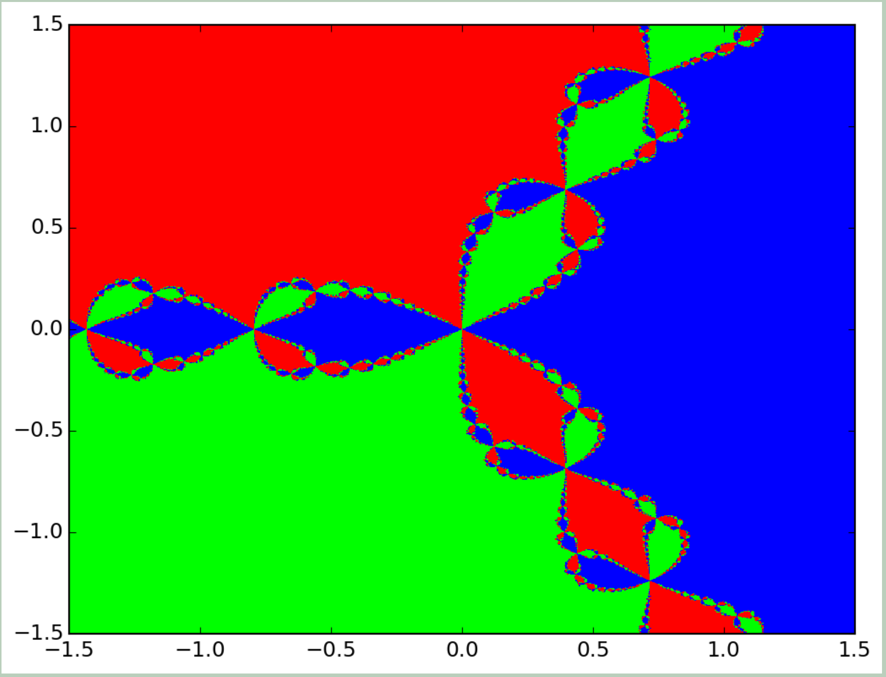

Figure 16.6: Basins of attraction for $x^3 - 1$.