Edit this. Download. Other published documents...

Worksheet 5 - 3n+1 Conjecture

43 minutes ago by pub

The *3n+1* Conjecture

The $3n+1$ conjecture is an unsolved conjecture in mathematics. It is named after *Lothar Collatz*, who first proposed it in 1937. It is also known as the *Collatz conjecture*, as the *Ulam conjecture* (after Stanislaw Ulam), or as the *Syracuse problem*.

The *3n+1* operation

Consider the following operation on positive integers *n*.

- If *n* is even, then divide it by *2*.
- If *n* is odd, then multiply it by *3* and add *1*.

For example, if we apply this transformation to *6*, then we get *3* since *6* is even; and if we apply this operation to *11*, then we get *34* since *11* is odd.

Exercise: Write a function that implements this operation, and compute the images of *1, 2, ..., 100*.

Statement of the conjecture

If we start with *n=6* and apply this operation, then we get *3*. If we now apply this operation to *3*, then we get *10*. Applying the operation to *10* outputs *5*. Continuing in this way, we get a sequence of integers. For example, starting with *n=6*, we get the sequence

6, 3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, ...

Notice that this sequence has entered the loop $4 \mapsto 2 \mapsto 1 \mapsto 4$ The conjecture is

3n+1 conjecture: For every *n*, the resulting sequence will always reach the number *1*.

Exercise: Write a function that takes a positive integer and returns the sequence until it reaches *1*. For example, for *6*, your function will return [*6, 3, 10, 5, 16, 8, 4, 2, 1*]. Find the largest values in the sequences for **1, 3, 6, 9, 16, 27.**

(*Hint*: You might find a *while loop* helpful here. Below is a very simple example that repeatedly adds **2** to the variable **x** until **x** is no longer less than 7.)

```
x = 0
while x < 7:
    x = x + 2
print x
```

Exercise: Use the **line** command to plot the sequence for *27*.

Exercise: Write an **@interact** function that takes an integer *n* and plots the sequence for *n*.

Stopping Time

The number of steps it takes for a sequence to reach *1* is the *stopping time*. For example, the stopping time of *1* is *0* and the stopping time of *6* is *8*.

Exercise: Write a function that returns the stopping time of a poisitve integer n. Plot the stopping times for *1, 2, ..., 100* in a **bar chart**.

Exercise: Find the number less than 1000 with the largest stopping time. What is its stopping time? Repeat this for *2000, 3000, ..., 10000*.

Extension to Complex Numbers

Exercise: If n is odd, then $3n+1$ is even. So we can instead consider the operation that maps n to $\frac{n}{2}$, if n is even; and to $\frac{3n+1}{2}$, if n is odd.

$$f(z) = \frac{z}{2}\cos^2\left(z\frac{\pi}{2}\right) + \frac{(3z+1)}{2}\sin^2\left(z\frac{\pi}{2}\right).$$

Construct f as a symbolic function and use Sage to show that $f(n) = T(n)$ for all $1 \leq n \leq 1000$, where T is the $\frac{3n+1}{2}$-operator. Afterwards, argue that f is a smooth extension of T to the complex plane (you have to argue that applying f

to a positive integer has the same effect as applying T to that integer. You don't need Sage to do this, but it might offer you some insight!)

Exercise: Let $g(z)$ be the complex function:

$$g(z) = \frac{1}{4}(1 + 4z - (1 + 2z)\cos(\pi z)).$$

Construct g as a symbolic function, and show that f and g are equal.

Note: You can do this using a combination of Sage and Maxima. Sage wraps some of Maxima's functions, but not all. For example, in Sage you can write **f.trig_expand()**. If you want to use some of Maxima's commands, then you can do the following:

```
maxima(f).trigexpand().sage()
```

This command converts **f** into a Maxima object (via the command **maxima(f)**), then applies the Maxima function **trigexpand**, and then converts the result into a Sage object (via the method .**sage()**). To see the available Maxima commands, you can type: **maxima.<tab>**.

Exercise: Use the **command_plot** command to plot **g** in the domain $x = -5, ..., 5$ and $y = -5, ..., 5$

Exercise: Consider the composition $h_n(z) = (g \circ g \circ \cdots \circ g)$(where there are n copies of g in this composition). Use **complex_plot** and **graphics_array** to plot h_1, h_2, h_3, ..., h_6 on the domain $x = 1, ..., 5$ and $y = -0.5, ..., 0.5$.

(*Hint:* To speed things up or control the percision of the computations, you may want to replace **pi** in your equation with **CDF.pi()**. Type **CDF?** and **CDF.pi?** for more information.)

Exercise: Generate some *really nice* images of h_n that illustrate the fractal-like behaviour of h_n. (*Hint:* You may want to explore the **plot_points** and **interpolation** options for the **complex_plot** command.)