18.330 Problem set 2 (spring 2020)

Submission deadline: 11:59pm on Monday, February 17

Exercise 1: Fixed-point iteration In this question we will find the roots of the function

$$f(\alpha, x) = x^3 - \alpha x + \sqrt{2}$$

by using fixed-point iterations.

1. Define a Julia function for f and draw it; make an interactive visualization as α changes. [Make sure to fix the plot limits so as to be able to see what's going on.]
2. When α varies, the number of real roots can change. For which approximate value of α does the number of real roots change? How many real roots are there?
3. When $\alpha = 2.5$, approximately where are the roots? From now on fix α to this value.

The simplest thing we can try is $g(x) = x + f(\alpha, x)$, since then a fixed point of g is a root of f.

4. Plot the function $g(x)$ and the function $y = x$. Taking into account the results stated in the slides, which root do you expect to be able to calculate by iterating g? Fix an initial condition and show that it does converge there.
5. What is the rate of convergence to that root?
6. What happens with the initial condition $x = 1.1$? Why?
7. Draw a cobweb diagram to illustrate this behaviour.
8. By using algebraic transformations, find fixed-point iterations g that converge to the other two roots.

   There are two alternative approaches you can take here. The first is to find other iteration schemes, $x = h(x)$ by algebraically rearranging $f(\alpha, x) = 0$ to isolate an x on one side. The second is to introduce the generalized function $g(c, x) := x + cf(x)$. Make an interactive plot of $g(c, x)$ and $y = x$ as c varies. What do you notice about the slope of $g(c, x)$ at the fixed points as c changes? Can you use this to change the stabilities of the fixed points in the iteration scheme?
9. (**Extra credit**) Call α_c the **critical value** of α at which the number of roots changes. [This is called a **bifurcation point**.] Find α_c numerically.

Exercise 2: Defining a type for Polynomials Although we have already used polynomials, we haven't had a way to say "this object is a polynomial". For this we need to **define a new type**!

1. Define a type `Polynomial` to represent a polynomial. It should have fields `degree` and `coefficients`.
2. Write a **constructor** function with the same name (`Polynomial`) that accepts a vector of coefficients and builds a `Polynomial` object whose degree it automatically calculates.
3. Write a `show` method to display the polynomial nicely.
4. Write a function to evaluate the polynomial at a point `x`, as follows:

   ```
   function (p::Polynomial)(x)
       # fill in
   end
   ```

   Then if you have an object `p` of type polynomial, writing e.g. `p(3)` will evaluate that polynomial at the value 3.
5. Write a function `derivative` that takes a polynomial and returns a new polynomial that is its derivative.
6. Julia contains a module `Test` (in the standard library – no installation required) for testing that code is correct.

   Write a few tests of the functionality you have defined using tests of the form

   ```
   @test a == b
   ```

   E.g. to test the sum of two `Polynomial`s you can write

   ```
   @test Polynomial([1, 2]) + Polynomial([3, 4]) == Polynomial([4, 6])
   ```

   When you run these tests, you should see the message `Test passed`.

   To create the tests, do the calculations by hand.

Exercise 3: Newton method

1. Write a function `newton` to implement the Newton method. It should take a function f, its derivative f' and an initial guess x_0. It should terminate when the residual is less than some tolerance, or when the number of iterations is too large.
2. Write a specialised method of `newton` for polynomials that accepts an object of type `Polynomial`, written `p::Polynomial` and calculates its derivative automatically.

3. Taking the same polynomial as in exercise 1, confirm that the Newton method has quadratic convergence by verifying numerically the defining limit.

4. Can you find different roots by taking different initial conditions x_0? How will you know if/when you have found all of the roots like this?

The Newton method works fantastically well when the starting point is close enough to a root, but can also behave very badly, as follows.

5. How many roots does the function $f(z) = z^3 - 1$ have in the complex plane $\mathbb{C}$? Where are they?

6. Calculate those roots using the Newton method with *complex* starting points $a + ib$ forming a square grid in the complex plane. Store the imaginary part of the resulting root (or final value, if no root is reached) in a matrix.

7. Plot the matrix with the `heatmap` function and plot the true roots as points. What kind of object are you seeing? What does this imply for the Newton method? What happens close to the roots?

Exercise 4: Aberth method Newton's method only finds one root at a time. The Aberth method calculates them all at once! Given estimates (z_k) for all roots of a polynomial it calculates a new estimate via $z_k = z_k - w_k$, where

$$w_k = \frac{\frac{p(z_k)}{p'(z_k)}}{1 - \frac{p(z_k)}{p'(z_k)} \cdot \sum_{j \neq k} \frac{1}{z_k - z_j}}.$$

In order to guarantee convergence the initial guess must be chosen in a special way. Cauchy's bound tells us that all the roots of a polynomial $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ must have absolute value less than

$$U = 1 + \max\left(\left|\frac{a_{n-1}}{a_n}\right|, \left|\frac{a_{n-2}}{a_n}\right|, \ldots, \left|\frac{a_0}{a_n}\right|\right)$$

and a lower bound of

$$L = \frac{1}{1 + \max\left\{\left|\frac{a_n}{a_0}\right|, \left|\frac{a_{n-1}}{a_0}\right|, \ldots, \left|\frac{a_1}{a_0}\right|\right\}}$$

The starting points should be chosen so that their absolute value lies in between these bounds.

1. Implement a function `poly_bounds` that takes in a `Polynomial` and returns L and U.
2. Use `poly_bounds` to write a function `initialize_points` that finds a suitable starting group of points. Do this by choosing ρs sampled from a uniform distribution over $[L, U]$ and θs sampled from a uniform distribution over $[0, 2\pi]$. Then calculate $z = \rho e^{i\theta}$.

   [Hint: The `rand()` function generates uniform random numbers between 0 and 1. Use `rand` to write a function `uniform(a, b)` to generate uniform numbers between a and b.]
3. Implement the Aberth method for a polynomial p which takes in values from `initialize_points` and terminates either after a certain number of iterations or when a certain tolerance is met. Test your function for the polynomials we have considered so far.
4. Make an interactive visualization of the progress over time on some random polynomials of degree 10 or 20.
5. Find numerically the order of convergence of the method.
6. Try polynomials with multiple roots, such as $(x^2 + 1)^3$. What happens?