18.330 Problem set 3 (spring 2020)

Submission deadline: 11:59pm on Tuesday, February 25

Exercise 1: Algorithmic differentiation

1. Complete the implementation of the `Dual` type from class, including derivatives for subtraction, division, and integer powers. For $/$ you may assume that the denominator is not zero.

   Add methods for each function to e.g. add a real number (type `::Real`) to a `Dual`. Treat a real number as having derivative 0.

2. Put these definitions in a file `dual.jl`. This may be included with `include("dual.jl")` from Juno or Jupyter, or from another file.

   (You must make sure the files are in the same directory, or give the path to the file on your machine.)

3. For functions f such as `exp`, we define the action of the function on a `Dual` using

$$f(a + b\epsilon) = f(a) + \epsilon\, b f'(a)$$

   Use this to define `exp`, `sin` and `log` on `Dual` numbers.

4. Write a function `derivative` that takes a function f and a point a, and uses `Dual` numbers to calculate the derivative $f'(a)$ of f at a. It should return *only* the derivative.

5. Define a function to calculate the directional derivative of a function $f : \mathbb{R}^n \to \mathbb{R}$ in a direction **v**.

6. Use this to write a function `partial` to calculate the ith partial derivative of a function.

7. Write functions to calculate the gradient and Jacobian of a function $f : \mathbb{R}^n$.

8. Write a few tests of the functions you have written. Use simple functions for which you can do the calculations by hand to write the tests.

Exercise 2: Newton for higher-dimensional functions

1. Implement the multidimensional Newton method. It should take a function f and an initial guess x_0.

   You should preferably use your own function from [Exercise 1] for calculating the Jacobian. You may also use instead the function `ForwardDiff.jacobian` from the `ForwardDiff.jl` package.

You should first do a basic check that f does the right thing by checking if $f(x_0)$ is of the same dimension as x_0.

Iterate until the residual is small enough or until some maximum number of iterations is reached. To check the residual, use the `norm` function from the standard library `LinearAlgebra` package. This calculates the "size" (length) of a vector.

The function should return a pair `(flag, r)`, where `flag` is a Boolean that indicates if a root was found, and `r` is the location of the root (if one was found).

2. Consider the nonlinear system

$$x^2 + y^2 = 3 \tag{1}$$

$$\left(\frac{x}{2}\right)^2 + (y-a)^2 = 1 \tag{2}$$

Make a plot of the two functions for $a = 0.5$. How many roots are there?

3. From your plot, identify by eye approximately where the top right root is. Use the Newton method to find it accurately.

Find the order of convergence of the method in this case. (As usual you probably need `BigFloat` to get a good result.)

4. Make a function `all_roots` that searches for all roots in a given rectangular box. (The size of the box should be given as arguments to the function). It should keep a list of the *unique* roots found so far (i.e. without duplicates). Use a tolerance to decide if two roots are the same or not.

You may use a grid search or a random search inside the box.

5. Use `all_roots` to find all solutions of the system.

6. Make an interactive visualization where you repeat the above for different values of a. You should redraw the curves and draw all roots you find on top, for each value of a.

Which qualitative change occurs and why? How could you think of going about finding numerically the critical value of a for which this happens? You do not need to do so, just think about what you *would* do.

Extra credit if you actually do so.

Exercise 3: A more complicated example Consider the system

$$(x+3)(y^3-7)+18=0 \tag{3}$$
$$\sin(y\exp(x)-1)=0 \tag{4}$$

(taken from here).

1. Draw plots of the two functions.
2. Find as many roots as you can in the region $[-5,5]^2$. How many are there?
3. Plot them on top of the graph. Did you find them all?

Exercise 4: Lorenz equations The Lorenz equations are a system of 3 ordinary differential equations that give a "simple" model of convection in the atmosphere. They are famous since their solutions form beautiful patterns and are an early example of the identification of **chaos** in deterministic dynamical systems:

$$\frac{\mathrm{d}x}{\mathrm{d}t}=\sigma(y-x), \tag{5}$$
$$\frac{\mathrm{d}y}{\mathrm{d}t}=x(\rho-z)-y, \tag{6}$$
$$\frac{\mathrm{d}z}{\mathrm{d}t}=xy-\beta z. \tag{7}$$

We will take the standard parameter values: $\sigma=10$ and $\beta=8/3$.

We will come back to look at their dynamics later in the course. For now we will look only at the **stationary points**, which are obtained by setting all of the time derivatives equal to 0 and solving the resulting system of nonlinear equations. [If the system starts at a stationary point, it will remain there.]

[Note that for this particular system the roots may be found analytically, which you may use to check your work. But if you were to perturb the original system a bit, it would no longer be possible to find the roots analytically.]

1. Make a function `lorenz(x, y, z, σ, ρ, β)` and a method `lorenz(ρ)` that takes a vector `x` and uses the standard parameter values. for σ and β.
2. Make a function `all_roots3` that is a 3D version of `all_roots`. [Extra credit: Make a generic version that works in *any* number of dimensions.]
3. Use `all_roots3` to find all the stationary points for $\rho=2$. You should search in a box $[-5,5]^3$. How many stationary points are there?

The **stability** of a stationary point turns out to be given by the eigenvalues of the Jacobian matrix at that point. [A trajectory starting close to an unstable point will move away from that point.] We will see how to calculate eigenvalues of a matrix later in the course; for now, use the `eigvals` function from `LinearAlgebra`. Recall that the eigenvalues of a matrix are complex numbers in general.

A stationary point is **stable** if the real part of each eigenvalue is < 0, and **unstable** if any eigenvalue has real part > 0.

4. Write a function `maximum_stability(f, x)` that calculates the maximum of the real parts of all the eigenvalues, assuming that x is a stationary point of f.
5. Write a function that "follows" each stationary point found in [3.]: increase ρ by an increment $\delta\rho = 0.1$ and use the stationary points from the previous value of ρ as initial conditions for the Newton method for the new value of ρ. [This is an example of **numerical continuation**. It assumes that the number of stationary points will not change.]
6. Plot the maximum stability as a function of ρ between 2 and 25.
7. Use a suitable numerical method to accurately find the critical value ρ_c at which the stability of the stationary points changes.