6.S083 Fall 2019: Problem set 4

Submission deadline: Wednesday November 27, 2019 at 11:59pm.

Submit a Jupyter notebook with your solutions to `sandersd@mit.edu`. Make sure the name of the file contains your first name and last name.

Exercise 1: Numerical derivatives

1. Define a function `deriv` that takes a function f, a number a and an optional h with default value 0.001, and calculates the finite difference approximation $(f(a+h) - f(a))/h$ of the derivative $f'(a)$.
2. Make a function `tangent_line` that calculates the tangent line to f at a. It should return a function that takes in a value x and calculates the height of the tangent line at that value of x. Use the function `deriv` with a small value of h to calculate $f'(a)$.
3. Make a function that draws an interactive visualization of a function f, which allows you to vary the point a and the size h. Draw the graph of the function f, the tangent line at a, and the "secant line" that shows the line whose slope is calculated by the finite difference approximation.

   Use your interactive visualization with the function $f(x) = x^3 - 2x$.
4. To see how good the finite-difference approximation is, make a function `finite_diff_error`, which accepts a function `f`, its derivative `fp`, a point `a` and a size `h`, and calculates the error $\epsilon(h)$ that the finite-difference approximation makes.

   Draw the error on a log–log scale for 50 values of h ranging from 10^{-16} to 10^{0} that are *equally spaced* on the log scale, for the same function f as above. You will need to calculate its analytical derivative f' and provide that by hand to the function.

   What is the rate of decrease of $\epsilon(h)$ as a function of h for the larger values of h?
5. Repeat [4] for a "centered" finite-difference approximation, $(f(a+h) - f(a-h))/(2h)$. Draw the error on the same graph and compare the rates.

Exercise 2: 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. (These will return a new `Dual` number as the result.)

4. Write a function `derivative` that takes a function `f` and uses `Dual` numbers to calculate its derivative.

5. Recall that one way to define partial derivatives of a function $f : \mathbb{R}^2 \to \mathbb{R}$ is as the usual derivative of the function obtained when you fix the other argument.

   Use this to define functions `partial1` and `partial2` that calculate the partial derivatives of a function f at (a, b). You should use the function `derivative` from [4].

6. Define a function `partial` that takes also a value i and calculates the partial derivative in the ith direction of a function $f : \mathbb{R}^n \to \mathbb{R}$.

7. Define a function `gradient` that calculates the gradient ∇f of a function $f : \mathbb{R}^n \to \mathbb{R}$ at a vector $\mathbf{a} \in \mathbb{R}^n$.

8. Julia contains a module `Test` for testing that code is correct. In a function `dual_test.jl`, load the module and write tests of the functionality you have defined using tests of the form

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

   E.g. to test the sum of two `Dual` numbers you can write

```
@test Dual(1, 2) + Dual(3, 4) == Dual(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: Minimization by gradient descent

1. Define a function `minimize` that takes a function f and a starting point x_0 and carries out gradient descent with a step size η (take a default value

$\eta = 0.01$ or similar.

Recall that this is an iterative method in which you take steps to a new position in the direction opposite to the gradient.

You should make two methods of the function: one for 1D functions, when x_0 is a scalar, and one for functions $f : \mathbb{R}^n \to \mathbb{R}$, when x_0 is a vector.

2. Make a function that gives an interactive visualization of gradient descent in 1D for a function f. It should draw the function and should visualize a trajectory generated by the gradient descent algorithm over time from the given initial condition.

3. Apply your function to $f(x) = x^4 + 3x^3 - 3x + 5$.

   How can you find different minima?

4. Apply your function to the Himmelblau function $f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$. Visualize the trajectory using both contours (`contour` function) and a 2D surface (`surface` function).

   Can you find different minima?

   You can try to install the `PlotlyJS` and (if necessary) `ORCA` packages and activate it with `using Plots; plotlyjs()`. This will / should give an interactive 3D plot.

Exercise 4: Least squares fitting

1. Generate some fake data using the following piece of code:

```
import Random
Random.seed!(1);   # reproducible random numbers

xs = 0:0.05:5
ys = 1 .+ 0.5*(sqrt.(x) .+ 0.8 * rand(length(x))).^2
```

   Plot it.

2. Define a loss function $L(\mathbf{x}, \mathbf{y}, a, b)$ using the mean squared distance of the data from the straight line $y = ax + b$.

3. Make an interactive visualization in which you can vary a and b. You should plot the data, draw the straight line, and draw the vertical lines from the data points to the curve. You should also report (e.g. in the title) the corresponding value of the loss function.

4. Use your visualization to manipulate the parameters a and b until you find a minimum value of a and b.

5. Use your function `minimize` from the previous question to verify this result. Make an interactive visualization in which you see the straight line moving around, and simultaneously you draw the $L(a, b)$ surface and see the point moving around on it.