6.S083 / 18.S190 Spring 2020: Problem set 5

Submission deadline: Tuesday May 11, 2020 at 10:00pm EDT.

We started off this course by looking at some real-world *data* on the COVID-19 epidemic. Most of the course has been about how to define and implement mathematical and computational *models*.

Models have *parameters*, e.g. the rate of recovery from infection. Where do the values of these parameters come from? Ideally we extract them from data. The goal of this problem set, then, is to put everything together by *fitting* a model to data.

In order to keep things as simple as possible, we will use data that we generated from the spatial model in Problem Set 4, rather than real-world data, and we will fit the simplest SIR model (the ODE version). But the same basic ideas apply in the more general context.

There are several ways to fit a function to data, but all in the end involve some form of **optimization**, usually **minimization** of a special function called a **loss function**, which measures how far we currently are away from our goal. This forms the basis of the vast and key field of **machine learning**. Indeed, machine learning is just "fitting a model to some data by optimizing over the parameters". In this problem set we will build up the tools we need to find the parameters in the SIR equations that best fit the data.

We emphasize that this material is intended to be pedagogical; we are not suggesting that the specific techniques here are the ones you should use for an actual calculation, but the underlying ideas are what are important.

This is a capstone problem set that replaces two problem sets (two weeks' work) and hence is a little longer than usual.

Exercise 1: Simulating the SIR differential equations

Recall that the ordinary differential equations (ODEs) for the SIR model are as follows:

$$\begin{aligned}\dot{S} &= -\beta SI \\ \dot{I} &= \beta SI - \gamma I \\ \dot{R} &= \gamma I\end{aligned}$$

where we use the notation $\dot{S} := \frac{dS}{dt}$ for the derivative of S with respect to time. S represents the *proportion* (fraction) of the population that is susceptible.

We will use the simplest possible method to simulate these, namely the **Euler method**. The Euler method is *not* a good method to solve ODEs accurately. But for our purposes it is good enough.

For an ODE with a single variable, $\dot{x} = f(x)$, the Euler method takes steps of length h in time. If we have an approximation x_n at time t_n it gives us an approximation for the value x_{n+1} of x at time $t_{n+1} = t_n + h$ as

$$x_{n+1} = x_n + h\, f(x_n)$$

In the case of several functions S, I and R, we must use a rule like this for *each* of the differential equations within a *single* time step to get new values for *each* of S, I and R at the end of the time step in terms of the values at the start of the time step. In Julia we can write `S` for the old value and `S_new` for the new value.

1. Implement a function `euler_SIR(β, γ, S0, I0, R0, h, T)` that integrates these equations with the given parameter values and initial values, with a step size h up to a final time T.

   It should return vectors of the trajectory of S, I and R, as well as the collection of times at which they are calculated.

2. Run the SIR model with $\beta = 0.1, \gamma = 0.05$, and $S_0 = 0.99, I_0 = 0.01$ and $R_0 = 0$ for a time $T = 300$ with $h = 0.1$. Plot S, I and R as a function of time t.

3. Do you see an epidemic outbreak (i.e. a rapid growth in number of infected individuals, followed by a decline)? What happens after a long time? Does everybody get infected?

4. Make an interactive visualization in which you can vary β and γ. What relation should β and γ have for an epidemic outbreak to occur?

Exercise 2: Numerical derivatives

For fitting we need optimization, and for optimization we will use *derivatives*. So in this question we see one method for calculating derivatives numerically.

1. Define a function `deriv` that takes a function $f : \mathbb{R} \to \mathbb{R}$, a number a and an optional h with default value 0.001, and calculates the finite difference approximation

   $$f'(a) \simeq \frac{f(a+h) - f(a)}{h}$$

   of the derivative $f'(a)$.

2. Write a function `tangent_line` that calculates the tangent line to f at a point a. It should also accept a value of x at which it will calculate the height of the tangent line. Use the function `deriv` to calculate $f'(a)$.

3. Make an interactive visualization of the function $f(x) = x^3 - 2x$, showing the tangent line at a point a and allowing you to vary a. Make sure that the line is indeed visually tangent to the graph!

4. Write functions `∂x(f, a, b)` and `∂y(f, a, b)` that calculate the **partial derivatives** $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ at (a, b) of a function $f : \mathbb{R}^2 \to \mathbb{R}$ (i.e. a function that takes two real numbers and returns one real).

   Recall that $\frac{\partial f}{\partial x}$ is the derivative of the single-variable function $g(x) := f(x, b)$ obtained by fixing the value of y to b.

   You should use **anonymous functions** for this. These have the form `x -> x^2`, meaning "the function that sends x to x^2".

5. Write a function `gradient(f, a, b)` that calculates the **gradient** of a function f at the point (a, b), given by the vector $\nabla f(a, b) := (\frac{\partial f}{\partial x}(a, b), \frac{\partial f}{\partial y}(a, b))$.

Exercise 3: Minimization using gradient descent

In this exercise we will use **gradient descent** to find local **minima** of (smooth enough) functions.

To do so we will think of a function as a hill. To find a minimum we should "roll down the hill".

1. Write a function `gradient_descent_1d(f, x0)` to minimize a 1D function, i.e. a function $f : \mathbb{R} \to \mathbb{R}$.

   To do so we notice that the derivative tells us the direction in which the function *increases*. So we take steps in the *opposite* direction, of a small size η (eta).

   Use this to write the function starting from the point `x0` and using your function `deriv` to approximate the derivative.

   Take a reasonably large number of steps, say 100, with $\eta = 0.01$.

   What would be a better way to decide when to end the function?

2. Write an interactive visualization showing the progress of gradient descent on the function

$$f(x) = x^4 + 3x^3 - 3x + 5$$

   Make sure that it does indeed get close to a local minimum.

How can you find different local minima?

3. Write a function `gradient_descent_2d(f, x0, y0)` that does the same for functions $f(x, y)$ of two variables.

   Multivariable calculus tells us that the gradient $\nabla f(a, b)$ at a point (a, b) is the direction in which the function *increases* the fastest. So again we should take a small step in the *opposite* direction.

4. Apply this 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. (But don't spend time on it if it doesn't immediately work.)

Exercise 4: Learning parameter values

In this exercise we will apply gradient descent to fit a simple function $y = f_{a,b}(x)$ to some data given as pairs (x_i, y_i). Here a and b are **parameters** that appear in the form of the function f. We want to find the parameters that **best fit** to the given data.

To do so we need to define what "best" means. We will define a measure of the distance between the function and the data, given by a **loss function**, which itself depends on the values of a and b. Then we will *minimize* the loss function over a and b to find those values that minimize this distance, and hence are "best" in this precise sense.

The iterative procedure by which we gradually adjust the parameter values to improve the loss function is often called **machine learning** or just **learning**, since the computer is "discovering" information in a gradual way, which is supposed to remind us of how humans learn. [It doesn't, and it's not.]

1. Load the data from the file `some_data.csv` into vectors `xs` and `ys`.
2. Plot the data. What does it remind you of?
3. Let's try to fit a gaussian (normal) distribution. Its PDF with mean μ and standard deviation σ is

$$f_{\mu,\sigma}(x) := \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

   Define this function as `f(x, μ, σ)`.

4. Define a **loss function** to measure the "distance" between the actual data and the function. It will depend on the values of μ and σ that you choose:

$$\mathcal{L}(\mu, \sigma) := \sum_i [f_{\mu,\sigma}(x_i) - y_i]^2$$

5. Use your `gradient_descent` function to find a local minimum of $\mathcal{L}$, starting with initial values $\mu = 0$ and $\sigma = 1$.

   What are the final values for μ and σ that you find?

6. Modify the gradient descent function to store the whole history of the values of μ and σ that it went through.

   Make an interactive visualization showing how the resulting curve $f_{\mu,\sigma}$ evolves over time, compared to the original data.

   ("Time" here corresponds to the iterations in the gradient descent function.)

Exercise 5: Putting it all together – fitting an SIR model to data

In this exercise we will fit the (non-spatial) SIR ODE model from Exercise 1 to the (spatial) data you generated in Problem Set 4. If we are able to find a good fit, that might (or might not) constitute evidence that space "does not matter" too much for the dynamics of these models. If the fit is not so good, perhaps there is an important effect of space. (As usual in statistics, and indeed in modelling in general, we should cautious of making categorical claims.)

This fitting procedure will be different from that in Exercise 4, however: we no longer have an explicit form for the function that we are fitting! So what should we do?

We will try to find the parameters β and γ for which *the output of the ODEs when we simulate it with those parameters* best matches the data!

1. Re-run your spatial simulation from Problem Set 4 for a bigger system and make sure there is an epidemic outbreak for the parameters you use.

   Save the data to a CSV file from that notebook, using the following code. You may need to install the `Tables` package.

   ```
   using CSV, Tables

   CSV.write("mydata.csv", Tables.table(M))
   ```

   where `M` is a matrix storing t, S, I and R as a function of time t in separate columns.

   If you have vectors, you can make them into a matrix using `hcat(ts, Ss, Is, Rs)`. If you prefer, you can make a dataframe instead.

2. Load your data into the current notebook. Make sure to include this data in a zip file with your notebook when you submit the pset.

   Extract the data into vectors `ts`, `Ss`, `Is` and `Rs`.

3. Make a loss function $\mathcal{L}(\beta, \gamma)$. This will compare *the solution of the SIR equations* with parameters β and γ with your data.

   To do this, once we have generated the solution of the SIR equations with parameters β and γ, we must evaluate that solution at times t from the vector `ts`, so that we can compare with the data at that time.

   Normally we would do this by some kind of **interpolation** (fitting a function locally through nearby data points). As a cheap alternative, we will just use the results corresponding to the closest value t' with $t' < t$ that does occur in the solution of the ODEs. Write a function to calculate this value.

   (You should be able to do it *without* searching through the whole vector. Hint: Use the fact that the times are equally spaced.)

   The loss function should take the form

   $$\mathcal{L} = \mathcal{L}_S + 2\mathcal{L}_I + \mathcal{L}_R$$

   where e.g. $\mathcal{L}_S$ is the loss function for the S data, defined as in [4.4]. The factor 2 weights I more heavily, since the behaviour of I is what we particularly care about, so we want to fit that better. You should experiment with different weights to see what effect it has.

4. Minimize the loss function and make an interactive visualization of how the solution of the SIR model compares to the data, as the parameters change during the gradient descent procedure.

   If you made it this far, congratulations – you have just taken your first step into the exciting world of scientific machine learning!