Projectile ML problem

Martin C.

To get a sense of what's going on, I plot 50 of the 100 trajectories in Figure 1.

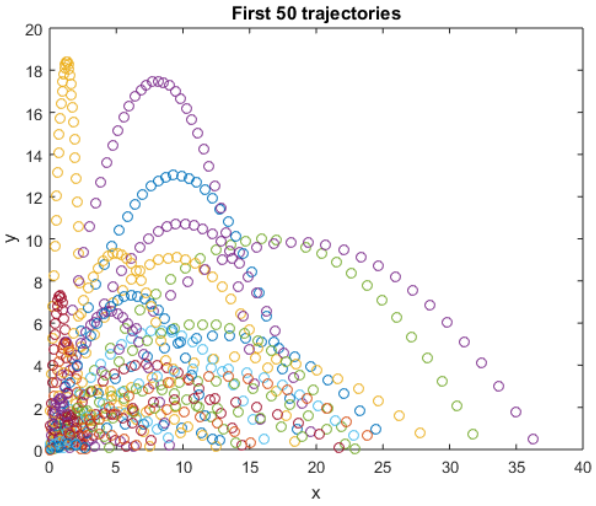

Figure 1

The physics:

$$x(t) = \dot{x}(0)\ t$$

$$y(t) = \left(\dot{y}(0) - \frac{gt}{2}\right)\ t$$

where $g = 9.8m/s^2$.

But let's suppose I don't know this. I want to come up with a statistical model that predicts the trajectory based on the position (x,y) at the first nonzero timestep (100ms). So the input to my model should be two real numbers, and the output, t,x,y vectors.

To "train" my model I use all 100 given trajectories (since I know the answer from physics I can test with any initial condition, there's no need to train with just a subset of the data). To reduce complexity, the first step is to fit a curve to the trajectories, and use the parameters of the fit as the "target" of the model. I know these are parabolas but I use 4th order polynomials to be more general. This gives 5 coefficients for each trajectory in the data. Now the idea is to look at how these parameters vary as a function of the initial condition (x,y at 100ms), and fit polynomials to *that* data. Then we end up with a 4th order polynomial that describes the trajectory, whose coefficients are functions of the initial condition.

This is done with x and y independently. The x and y equations above are decoupled, so it works in this case. Predicting x and y separately won't work in general obviously.

Running the code with initial (x,y) = (0.707106781187, 0.658106781187) which corresponds to an initial velocity of 10m/s at 45 degrees gives the results shown in Figure 2. The agreement is good, *too* good probably. This is hardly "machine learning," there is a simple (polynomial) relationship between the initial condition and the fit parameters of the trajectory.

I looked into some of the built-in MATLAB machine learning functions. In "projectiles-matlab.m" I use "fitlm" from the machine learning toolbox. The results are indistinguishable from Firgure 2. The advantage of "fitlm" is that it doesn't assume that x and y are independent. So I wanted to test that...

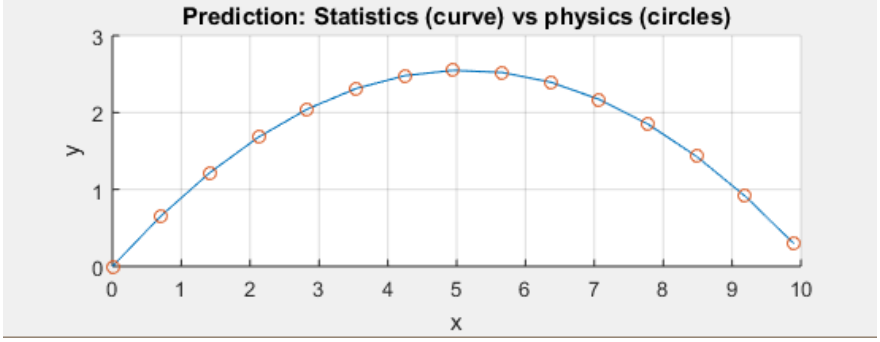

Figure 2

From a physics course I had a script that solves these (coupled nonlinear) equations of motion:

$$(x-1)\ddot{x} = \dot{y}$$

$$(x-1)\ddot{y} = -\dot{x}$$

These model a particle in a magnetic field if I recall correctly. A typical trajectory looks something like in Figure 3. The initial position is at (-1,0). Here there are oscillations which a polynomial can't model well. And for example, changing the initial y velocity (without changing the initial x velocity) changes how x evolves (x,y not independent) so this is trickier. I generate a bunch of data similar to the projectiles data and run my model. Figure 4 shows that the model fails here...

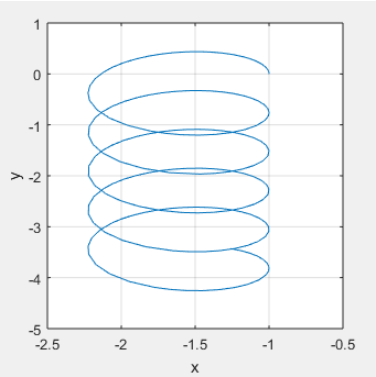

Figure 3

—What made you choose this specific model?
It seemed simple and effective for this particular data set. It may not be generalizable

—Is your model well suited to the learning objective? How would you evaluate your model?
Too well suited I would say (for the projectile problem). It probably won't work with more complex data, but here, it works very well.

—Can your model predict projectiles launched at arbitrary angles and velocities equally well (or badly)?
Equally well.

—What assumptions does your model make?
Discussed above.

—Will your approach/model change if we hadn't told you that the data was from a projectile?

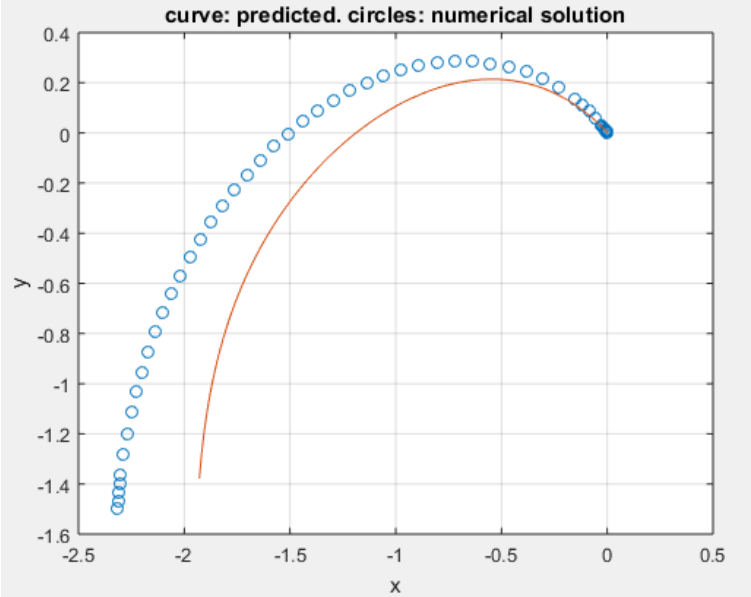

Figure 4

That helped me understand initially what was going on, but of course the source of the data is irrelevant. One can perform a regression with raw numerical data.

—Did you refer to any relevant literature while solving this problem?

Mostly just the MATLAB machine learning toolbox documentation.

—If you were given enough time, how would you improve the model?

I would try to make it more "general." Maybe I could use piecewise polynomial fitting for oscillating phenomena for example.