Final Project:

Stan Modeling of 2D GPS Model for Noisy Phyre Simulation

CS146: Computational Methods for Bayesian Statistics
Minerva University

Nikita Koloskov
April 2022

Final Project

i. Introduction

For this final project, I am creating a 2-dimensional extension of the GPS model that we worked through in class and applying it to the simulated ball movement in an intuitive physics benchmark simulator named Phyre. This assignment can be split in 3 parts:

- Generating the noisy ball movement observations;
- Defining and implementing the Stan GPS model for these observations;
- Interpreting the results.

ii. Data preparation

The first step I took is generating the noisy ball movement observations. My goal is to simulate a free-fall movement of the ball with some non-vertical initial velocity. Due to the design of the simulator, I am unable to specify the initial velocity for the object by myself. The path that I chose was to create a simulation of the ball movement, and then take away the first few frames in which the ball is not yet in the free fall. In this way, the first of the remaining frames will have the free-falling ball with some velocity that I can consider “initial”.

To implement this, I chose a Phyre template on which the ball is initially at rest on a platform. The initial scene is shown in Figure 1 below.

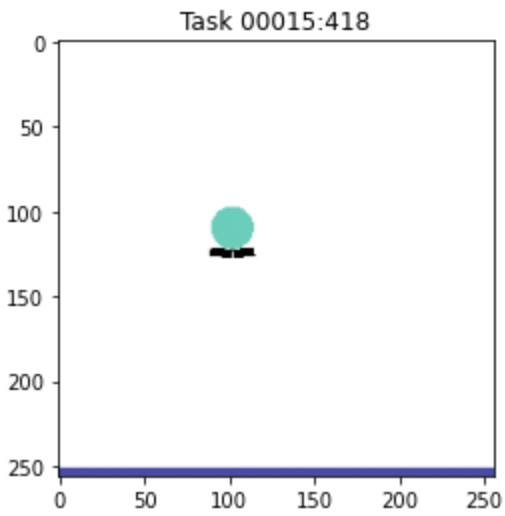


Figure 1. The initial state of the chosen simulation scene.

I then randomly sample 100 actions that will make this green ball fall from the platform. By action here I mean putting another ball (red) on a scene in the correct location and of the correct size to make a green ball move through a collision. I choose the first action that achieves this goal and then save the results of the simulation. The results of the chosen simulation, frame by frame, are shown in Appendix A. I set up the simulation in such a way that it has 48 simulation frames, to assure that I have enough data points for my model.

Finally, to make the model meaningful, I need to add some noise to the simulator output. When I will be doing further research on this topic, such noise can come from some machine learning location estimators. However, right now I just randomly added the noise to each observation. Figure 2 below shows the true and noisy trajectories of the ball. Please do not be concerned by the numbers - the output of the simulator provides coordinates normalized by the width and height of the scene, so true numbers are bound between 0 and 1.

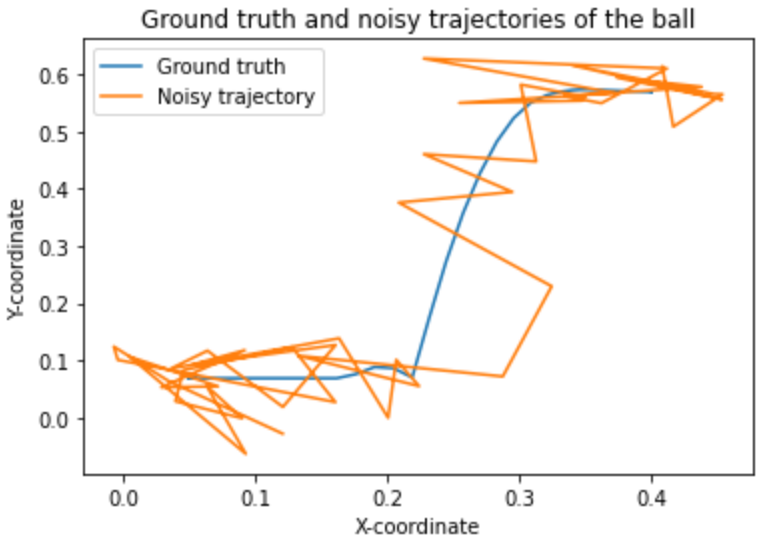


Figure 2. Ground truth and noisy trajectories of the green ball.

I then use this noisy data in the 2D GPS model to predict the true trajectory.

iii. Modeling

To predict the true trajectory of the ball from the noisy measurements, I created a 2D generalization of the GPS model that we studied in class. In this model, I assumed that the Physics behind the Phyre simulation follows classical mechanics. Thus, the movement along the x-axis is happening with a constant velocity, while the movement along the y-axis is happening with a constant free-fall acceleration. Figure 3 below contains all the equations that I used in my model definition.

As an initial frame of the model I will take the fourth frame of the simulation, assuming that on the fourth frame the ball is already free-falling.

Notation:

- $\tilde{x}_t$ - x-coordinate at time t
- $\tilde{y}_t$ - y-coordinate at time t
- x_t - measurement of x-coordinate at time t
- y_t - measurement of y-coordinate at time t
- v_{xt} - x-component of velocity at time t
- v_{yt} - y-component of velocity at time t
- g - free-fall acceleration

Initial state:

- x-coordinate: $x_0 \sim N(\mu_{x0}, \sigma_{x0}^2)$
- y-coordinate: $y_0 \sim N(\mu_{x0}, \sigma_{x0}^2)$
- x-component of velocity: $v_{x0} \sim N(\mu_{v_{x0}}, \sigma_{v_{x0}}^2)$
- y-component of velocity: $v_{y0} \sim N(\mu_{v_{y0}}, \sigma_{v_{y0}}^2)$

Parameter values:

- $\mu_{v_{x0}} = x_0 - x{-}1$ (where -1 refers to the preceeding simulation frame)
- $\mu_{v_{y0}} = y_0 - y_{-1}$
- $g = v_{y1} - vy0$ - free fall acceleration

State update equations:

- x-coordinate: $\tilde{x}_{t+1} \sim N(\tilde{x}_t + v_{xt}, \sigma_{\tilde{x}}^2)$
- y-coordinate: $\tilde{y}_{t+1} \sim N(\tilde{y}_t + v_{yt}, \sigma_{\tilde{y}}^2)$
- x-component of velocity: $v_{xt+1} \sim N(v_{xt}, \sigma_{v_x}^2)$
- y-component of velocity: $v_{yt+1} \sim N(v_{yt} + g, \sigma_{v_y}^2)$

Measurement equations:

- $x_t \sim N(\tilde{x}_t, \sigma_x^2)$
- $y_t \sim N(\tilde{y}_t, \sigma_y^2)$

Figure 3. Definition of the model.

To support the understanding of the model, I created visualizations for directed and factor graphs that correspond to the model. You can see them in Figure 4 below. Bigger versions of these figures are also attached as my secondary submission.

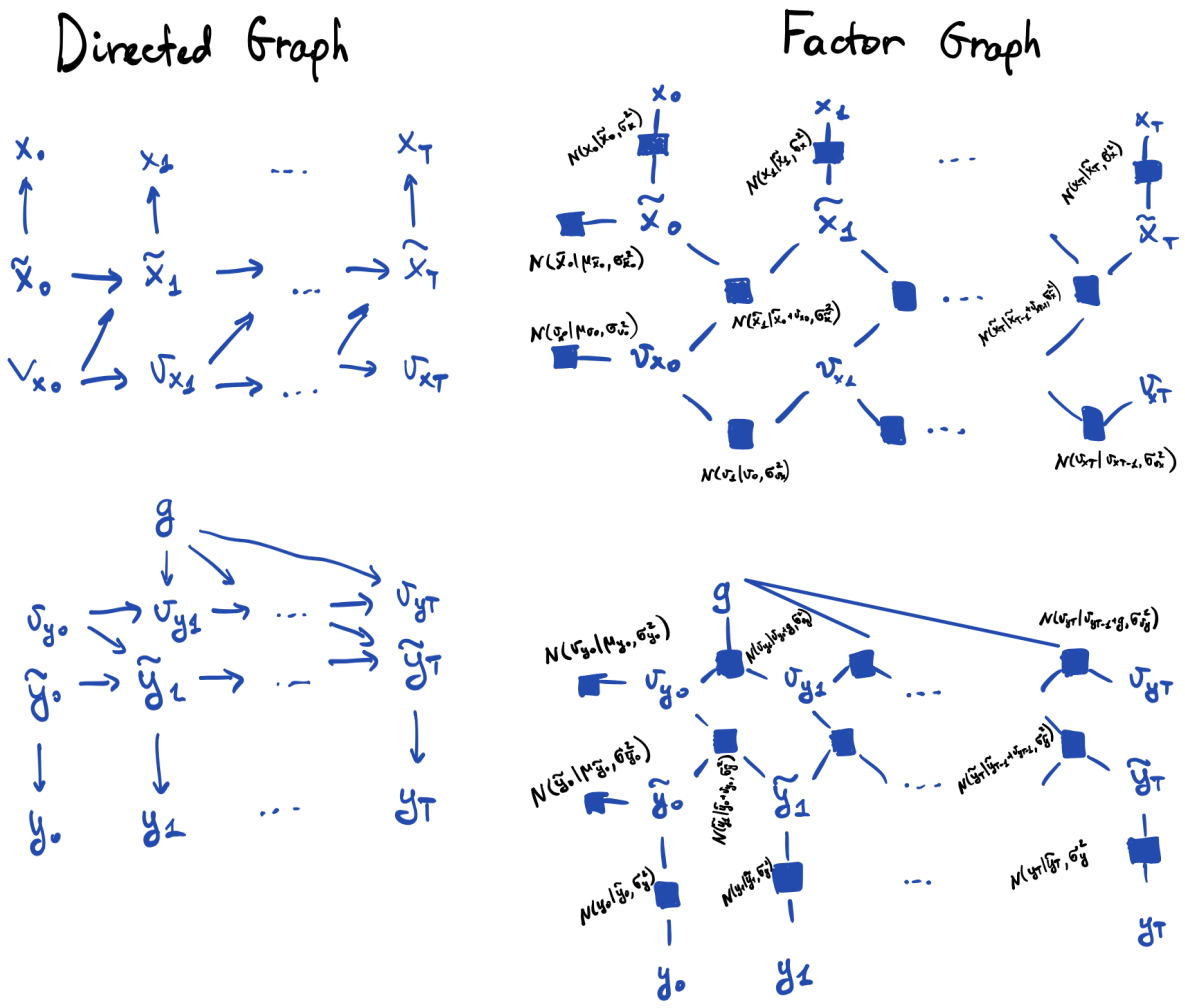


Figure 4. Directed and factor graphs for my GPS model.

From the simulation attached in Appendix A, you can see that on the first 15 frames the green ball tries to “escape” the platform. I consider frame 16 (labeled as Timestamp 15) as the first frame where the ball is in free-fall (and, thus, this is my time step 0 for the model).

The simulator also only gives us the positions of the ball, without any ground truth for velocities and acceleration. I calculate the velocity as a difference in positions between two subsequent time steps, and the acceleration - difference between velocities on two subsequent frames somewhere in the middle of the simulation and set it as constant.

Finally, I used Stan to model the true trajectory of the ball.

iv. Results and Conclusion

The output of the Stan model is 4,000 simulated true trajectories. Let's take a look at one of them and see how well it performs by looking at one of the simulated trajectories. Figure 5 below shows the simulated trajectory of the ball.

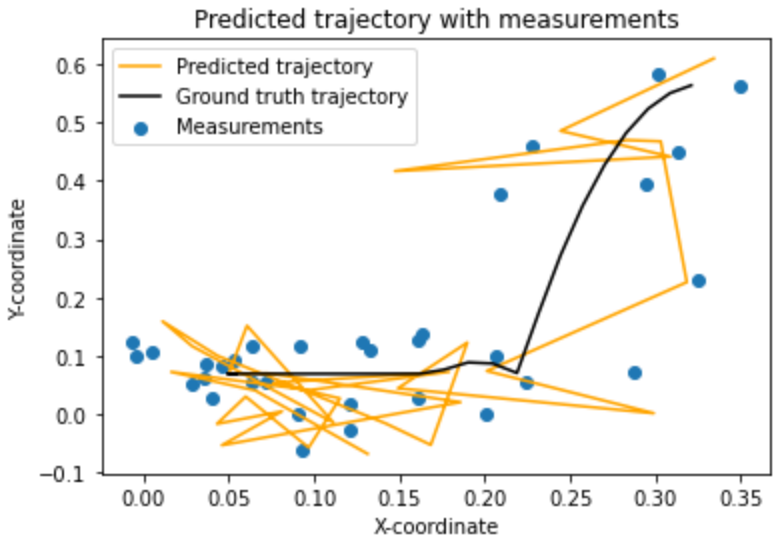


Figure 5. One of the simulated trajectories by Stan model.

It looks quite ugly, but we can still gain some insights from it. We can see the model is quite noisy at the beginning of the path (which starts in the top right corner) but gets a bit closer to the path with time. However, once the ball hits the ground, we can see a complete mess. This is explainable - once the ball is on the ground, it stops falling down, so the modeling with the normal distribution is perhaps not really efficient. Same for the wall - once the ball hits the wall of the simulated environment, the model starts breaking. Let's look at the time evolution of the predicted coordinates to see if this claim is true.

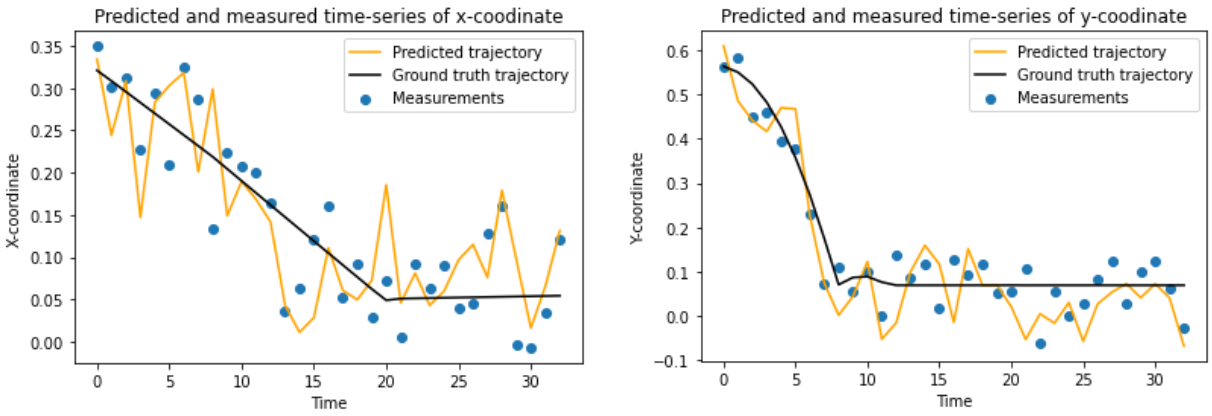
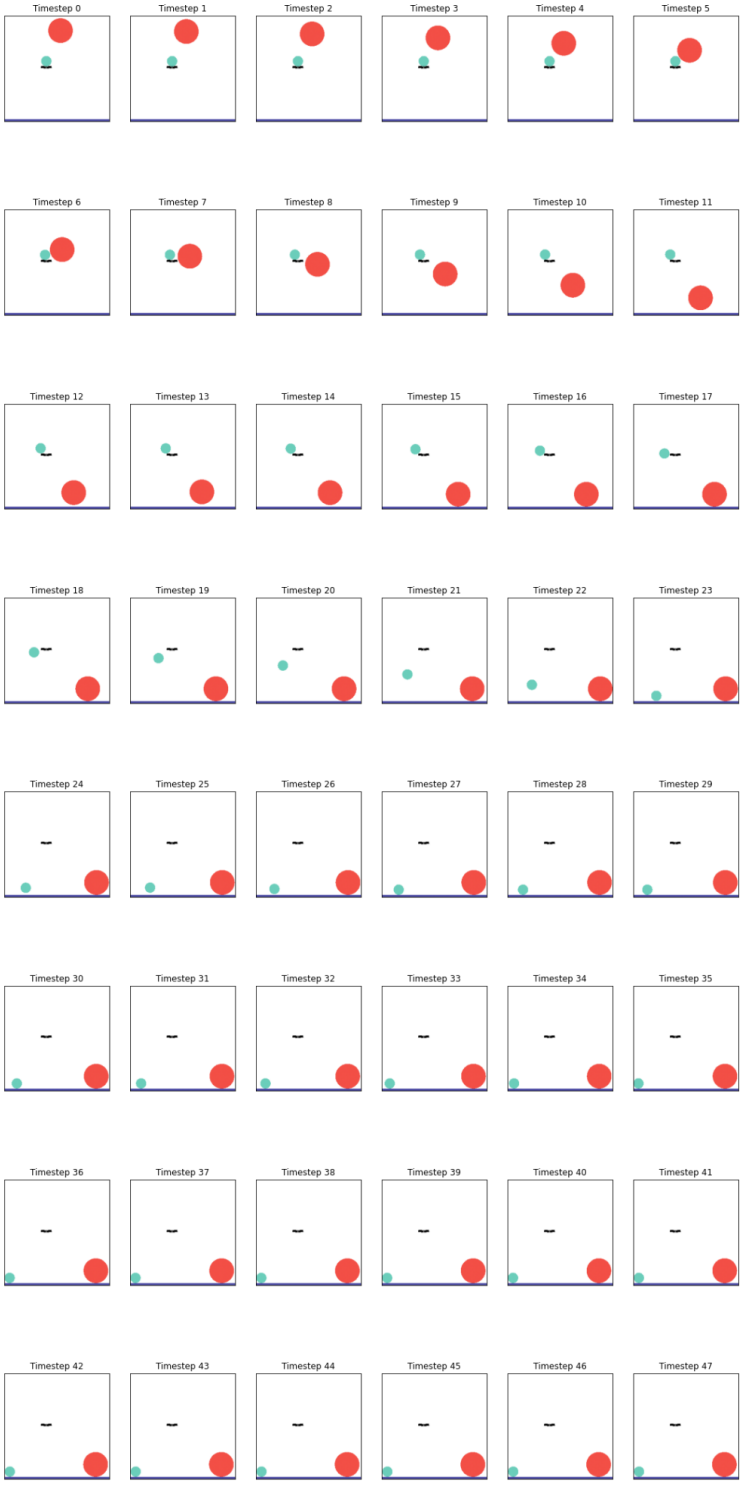


Figure 6. Predicted time series of coordinates with measurements and ground truth.

As we can see from Figure 6, the model indeed got closer to estimating how the coordinates changed while the ball is free-falling, but becomes noisier when it stops moving. The potential fix to this would be to model the movement not just using the normal distribution, but rather as a combination of distributions weighted by some indicator variable that changes depending on whether the ball hit the boundary of the scene or not.

It is also important to note the output of the Stan model. For every simulated parameter, the number of effective samples n_eff exceeds 4,000, with a mean (on eye) somewhere around 6-7 thousand. I don't have an explanation how this could happen given the 4,000 samples overall, so I cannot tell if it is a good sign. The Rhat value, however, is 1.0 for every single parameter that is modeled, which indicates that every chain in the Stan model converges well.

Appendix A. Frame by frame simulation of the scene.



LOs

#RightDistribution: For the modeling done in this assignment, I used normal distribution to draw the coordinates and velocities in the following time steps given the values on the previous time steps. The choice of the normal distribution is justified by the nature of noise in the problem - while it was artificially generated, in practice normality of noise would come from imperfect estimators that would center the estimations around the true values of objects on the scene.

#InterpretingProbabilities: The parameters of the normal distributions were chosen in such a way that the measurements have relatively large variance, while the true values have a variance much smaller. This comes from the fact that the trajectory of the ball is quite deterministic, and the small variance allows one to model them using quite narrow normal distributions. The measurements, however, are not guaranteed to have a small variance due to the errors in machine learning algorithms that would generate estimations.

#PythonImplementation: For this assignment, I first used Phyre to generate the measurements from the simulator, followed by manually adding the noise to those measurements to have a more realistic scenario. This step allowed me to generate a dataset. Then I used Stan library to create a model that allowed me to make the inference about the true trajectory of the ball using the estimations simulated before.

References

Facebook AI Research. (n.d.). *PHYRE - a benchmark for physical reasoning*. PHYRE. Retrieved April 21, 2022, from https://phyre.ai/