Probabilistic models and inference: Project

December 12, 2025

Part I: Introduction

In this first part of the project, your task is to think how to break inference procedures. By now, you have seen several different inference procedures. And you know that there is no single inference procedure that works for every model. Some, as we mentioned during the lectures, tend to work well for more problems than others, but generally you have to put thought into designing the inference procedure for a specific problem. In week 4, you will see how to do that effectively, but that will be the topic for Part 2 of the project.

Part A: problem library

Your first task is to collect a set of probabilistic models. The purpose of this collection is to stress test inference procedures and gain insight into their properties, including strengths and weaknesses. We want you to focus on a particular class of probabilistic programs, namely something known as **simulation-based inference**. In simple terms, in simulation-based inference, probabilistic programs use a (black-box) simulator to perform a part of the computation. As a simple example, look at https://turinglang.org/docs/tutorials/bayesian-differential-equations/ from the `Turing.jl` probabilistic programming language. This example presents a probabilistic program performing inference over the Lotka-Volterra model modelling the interaction between predators and prey. The differential equation solver in the example plays the role of a simulator.

How do you choose which models to select? The models you chose should allow you to test how effective an inference procedure is on a given model. Therefore, start from the inference procedures we covered in the lectures. What kind of models make it easy for an inference procedure to estimate the right posterior quickly? Which kind of model would make an inference procedure slow in estimating the posterior? Maybe there is a particular structure of a model that makes it nearly impossible to estimate the posterior? Starting from a model you expect an inference procedure to work well on, how do you have to modify it to make it more difficult?

Where do you find the models?

- Google interesting simulators. Just make sure that they execute fast; as a rule of thumb, one run of a simulator should take at

most a few seconds. Typically, you only need to understand what goes into the simulator and what the simulator will output. Think which parts should be probabilistically modelled. For instance, many physics simulators are deterministic but their initial parameters are unknown and need to be inferred from data (typically, a quantity you can observe over time that simulator will output); you can think of simulators this way.

- You will find plenty of examples in the documentation of the probabilistic programming language Pyro: https://pyro.ai/examples/
- Bayesian Method for Hackers has a nice bunch of probabilistic programs
- STAN Case Studies publishes case studies with probabilistic programs in STAN
- Modify one program into several ones! We are not asking for models to be unique. For instance, you will find several version of Lotka-Volterra models online.
- you can interpret the simulator very broadly as *any algorithm you can somehow parametrise with your probabilistic model* and estimate posterior over some quantities computed by that algorithm. So, think about any scientific simulator (plenty of them around for physics, biology, chemistry), game engines (but keep them very simple, 2D worlds), constraint solvers, planners, ... make it interesting for yourself!

What to submit? Your submission should be a PDF document containing the following:

- at least two probabilistic models/programs per group member
- a full probabilistic program for each problem
- for each program, an explanation of why you chose this program/model as a test case. For example, which inference procedures should do well on this models; which inference procedures should do badly; if you modified a model, why did you modify it in this particular way... anything that factored into your reasoning.
- Focus on the inference procedures we have seen in the class: Metropolis-Hastings, Importance Sampling, Hamiltonian Monte Carlo, and Particle filtering.

  Upload the PDF and the code on Brightspace.

Last few practicalities

- You can use any programming language of your choice. Just make sure we can run your code.
- We highly recommend using Gen.jl for this part of the project. You can technically do it in any other probabilistic programming language, but Gen will make the project significantly easier for you.
- Right now, you just have to make your probabilistic programs runnable. You don't have to make an experimental comparison of the inference procedures, just *a* theoretical one.
- The tasks are somewhat imprecise *by* design. You will have to make choices, and we are interested in your thought process behind those decisions.
- The **deadline** is the end of Week 7.

Part II: Introduction

In the second part of the project, you will pick up from where you stopped in Part I.A. You will, however, assume a different role: instead of trying to break the inference procedures, you will now develop custom inference algorithms that perform well on a given problem.

Part A

Your task is to develop custom inference procedure for the problems you have collected for Part I.A. To create the custom inference procedures, use ideas of programmable inference we have seen in the lectures, thorugh the Gen.jl probabilistic programming language. Your task is to customise an inference procedure for every problem such that it minimises the number of samples needed to reliably estimate (your desired) posterior. The important aspect here is that your procedures should be tailored to individual problems, and not widely applicable across different models.

Note: you are allowed to use other probabilistic programming languages, but Gen.jl will make this task significantly easier.

Think of the reasons why you included the models in your benchmark. How can you leverage those to develop a specialised inference procedure that works well on the problem? How can you leverage what you know about the problem itself to get good samples quickly?

How do we know that one inference procedure is better than another? Well, that is also up to you to define. The standard definition focuses on the rate of convergence – how many samples do we need to get close to our posterior? You can stick to that, or introduce a more nuanced definition that considers the properties of your problem.

Compare your inference procedure to the out-of-the-box implementations of Importance Sampling, Metropolis-Hastings, and Hamiltonian Monte Carlo from Gen.jl.

What to submit? Your submission should be a PDF document containing the following:

- description of the custom inference procedures for every problem, including your reasoning how you got to this procedure
- experimental comparison of inference procedures for each problem
- Focus on the inference procedures we have seen in the class:

Metropolis-Hastings, Importance Sampling, Hamiltonian Monte Carlo, and Particle filtering (if applicable).

- the deadline is the end of Week 10

  Upload the PDF and the code on Brightspace.

References