Introduction to pyMC

Probabilistic Programming in Python

Sachin Shanbhag

Department of Scientific Computing
Florida State University,
Tallahassee, FL 32306.

Contents

- Resources
- Probabilistic Programming
- pyMC Basics
- Examples (follow accompanying notebook)
  - Bayesian Inference to Estimate Cheating Levels
  - Parameter Estimation and Uncertainty Quantification
  - Sampling Custom Distributions

pyMC is an active project. Please use a “newer” version (v5)

You can either install pyMC locally or use Google Colab

Resources

- Probabilistic Programming in Python with pyMC

  YouTube video by Chris Fonnesbeck $\sim$ 90 minutes

- Introductory Overview

  Based on PeerJ publication, adapted for pyMC v5

- pyMC Quickstart Tutorial

  If you want to dig into code directly

- Visualize Different MCMC Samplers

  Demo

Probabilistic Programming

It is a tool for statistical inference.

It is not about writing software that behaves probabilistically.

It is a programming framework in which probabilistic models are specified and inference is performed automatically.

pyMC is one of several probabilistic programming tools

Others include:

Stan	low level, R flavor, market leader
BUGS	standalone, GUI, classic
pyro	Meta, big data, python
Tensorflow probability	Google, python
Edward	python, ML + MCMC
Turing.jl	Julia, ML + MCMC

Bayesian Inference

Main goal of probabilistic programming is Bayesian inference

$$\pi(\theta|\text{data}) = \frac{\pi(\text{data}|\theta) \times \pi(\theta)}{\pi(\text{data})} \tag{1}$$

If the prior and likelihood are conjugate the posterior distribution $\pi(\theta|\text{data})$ can be computed analytically

For complex problems, we use MCMC to sample posterior

In probabilistic programming we specify

- the prior and likelihood as probability distributions
- the MCMC method to sample the posterior

Framework provides sampling, analysis, and visualization tools.

pyMC

- Bayesian inference using MCMC, variational inference etc.
- large suite of statistical distributions
- easy to specify custom distributions that may not be available by default
- large suite of MCMC algorithms
  - Metropolis
  - Gibbs
  - Hamiltonian Monte Carlo
  - No U-Turn Sampler
  - Slice
- uses `ArviZ` for analysis and visualization of the posterior distribution

Sampling Algorithms

Metropolis MCMC is quite good for small dimensional problems.

But since it is essentially a random walk, it can take a long time to sample large spaces or complex models

I will outline the intuition behind other three popular methods, which shall not otherwise cover in class

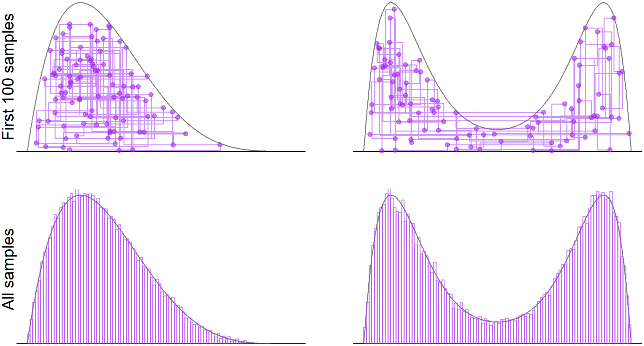
- Slice
- Hamiltonian Monte Carlo
- No U-Turn Sampler

Visualization of Samplers

Slice Sampling

Wikipedia has a helpful entry

Sophisticated extension of accept/reject



select next sample by uniformly sampling the domain corresponding to a "horizontal slice"

HMC/NUTS

Consider the energy landscape $U(\boldsymbol{x})$ corresponding to the target probability distribution $\pi(\boldsymbol{x})$

$$\pi(\boldsymbol{x}) \sim \exp\left(-U(\boldsymbol{x})\right) \tag{2}$$

Hamiltonian Monte Carlo (HMC) essentially simulates the motion of a frictionless "puck" or particle in this landscape

At each step:

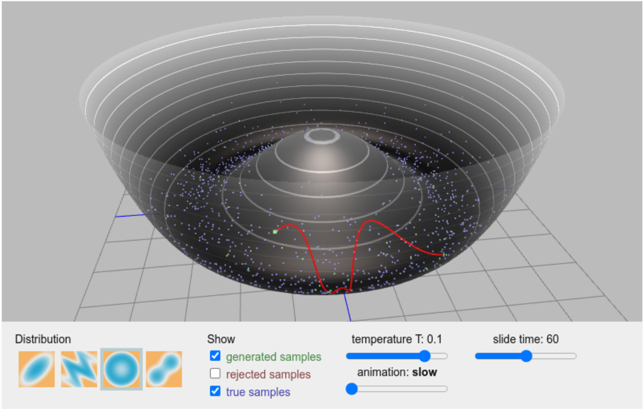
- sample velocity from normal distribution (kick)
- find where puck ends up after a certain time interval
- this location is the new sample

Solving the "equations of motion" using the leap-frog method requires computation of derivatives

pyMC builds computational graphs to compute derivatives

HMC

Samples are less correlated than Metropolis MCMC



Animation

Acceptance rates are generally high (of order 0.8)

NUTS

NUTS (No U-Turn Sampler) is the most common sampling method for continuous variables

It is the default algorithm in pyMC

It is an auto-tuning version of HMC that avoids U-Turns.

What are U-Turns, and why do we want to avoid them?

If you begin climbing a hill from a valley due to a kick, there is a tendency to slip back to the valley during the next turn.

NUTS attempts to avoid this.

Divergences

Divergences occur when the simulated HMC/NUTS trajectory departs from the true trajectory as measured by total energy

This often occurs when the target distribution has high curvature

Leap-frog takes small steps to simulate particle trajectory.

Small step sizes are inefficient. Large step sizes cause divergences.

pyMC provides warnings about divergences. If there are too many divergences *relative* to the total number of draws you should

- increase target acceptance rate (longer trajectories)
- reparametrize the model

Tutorial

It is best to demonstrate pyMC hands-on

Please see the accompanying Jupyter notebook