MCMC and Stan

Sytems Biology for Scientific Computing: week two

Introduction

Recap

Bayesian inference: Statistical inference resulting in probabilities

Why in general? Probability is expressive, old and decomposes nicely

Why in biology? Hierarchical regression models with ODEs

The big challenge Integrating high dimensional probability functions

Plan for today

1. MCMC
2. Stan

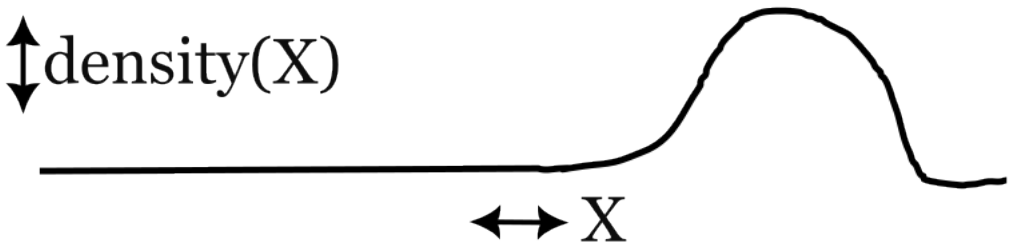
MCMC

MCMC: the big picture

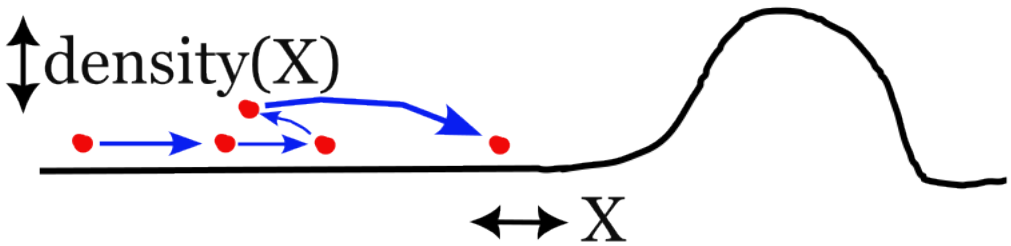
In A rule for evaluating the target function and maybe its gradients

Out: A **M**arkov **C**hain of numbers that you can do **M**onte **C**arlo integtation with

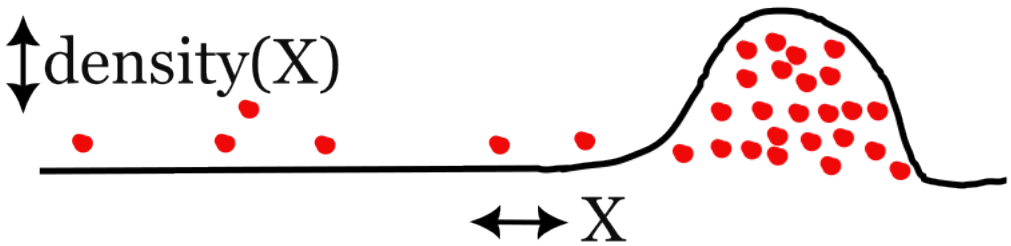
MCMC: simple case



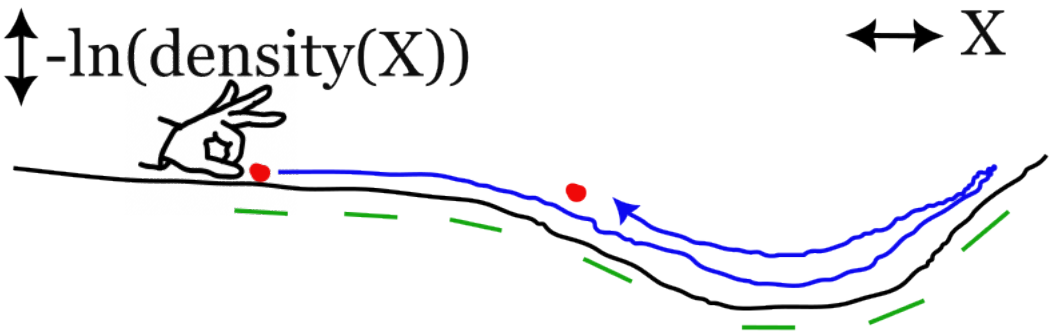
MCMC: simple case



MCMC: what we want to happen



Hamiltonian Monte Carlo



Hamiltonian Monte Carlo: reading

A Conceptual Introduction to Hamiltonian Monte Carlo

The Markov Chain Monte Carlo interactive gallery

Stan

Stan: what?

A language for specifying probability density functions

A compiler that turns Stan programs into instructions for inference engines

A library of inference engines implementing adaptive HMC (among others)

Some interfaces for popular computer tools

Stan: why?

Big, active and knowledgeable community (most important reason)

Featureful (complex numbers, Sundials, good diagnostics)

Fast (for CPU-bound, general purpose adaptive HMC)

Stan: installation

Install cmdstanpy (interface between python and cmdstan)

```
pip install cmdstanpy
```

Cmdstanpy comes with a command for installing cmdstan, the command line interface for Stan.

```
install_cmdstan
```

I like to store Stan outputs using the library arviz. It also does nice plots.

```
pip install arviz
```

How to write a Stan program

A Stan program consists of function definitions, variable declarations and statements, organised into `{...}` delimited blocks, e.g.

```
data {
  real y;  # a variable declaration
}
model {
  y ~ normal(0, 1.4);  # a statement
}
```

How to write a Stan program

The purpose of a Stan program is to define the probability density of any combination of data and parameters.

```
transformed data {
  real y = 2;  # this is both a statement and a declaration!
}
model {
  y ~ normal(0, 1.4);  # total density is N(2 | 0, 1.4) = 0.103
}
```

How to write a Stan program

The purpose of a Stan program is to define the probability density of any combination of data and parameters.

```
parameters {
  real y;
}
model {
  y ~ normal(0, 1.4);  # whatever the value of y, we know the density
}
```

Cmdstanpy workflow: 1

Use standard Python tools to make a dictionary mapping data variables to inputs e.g.

```
my_stan_input = {"y": 2}
```

(Optional) Save the input as a json file:

```
import json

with open("my_stan_input.json", "w") as f:

    json.dump(my_stan_input, f)
```

Cmdstanpy workflow: 2

Instantiate a `CmdstanModel`

```
from cmdstanpy import CmdStanModel

my_model = CmdStanModel(stan_file="my_stan_program.stan")
```

Cmdstanpy (via cmdstan) will use Stan's compiler to create `.hpp` and exectuatable files.

Cmdstanpy workflow: 3

Use the method `CmdStanModel.sample` to trigger adaptive HMC.

```
my_mcmc_results = my_model.sample(data=my_stan_input)
```

The results are (optionally) saved as files is a `CmdStanMCMC` object.

Cmdstanpy workflow: 4

Use the methods `CmdStanMCMC.diagnose` and `CmdStanMCMC.summary` for quick diagnostics.

```
summary = my_mcmc_results.summary()

diagnostics = my_mcmc_results.diagnose()
```

Cmdstanpy workflow: 5

Convert to arviz `InferenceData` and save

```
import arviz

my_idata = arviz.from_cmdstanpy(my_mcmc_results)

my_idata.to_json("my_arviz_idata.json")
```

Stan references

Cmdstanpy docs

Stan reference manual

Stan functions reference

Stan User's guide

stan-dev github organisation