



PyQB

Monga

Theano

Monte-Carlo

# Programming in Python<sup>1</sup>

Mattia Monga

Dip. di Informatica  
Università degli Studi di Milano, Italia  
mattia.monga@unimi.it

Academic year 2021/22, II semester

<sup>1</sup> © 2022 M. Monga. Creative Commons Attribution — Condividi allo stesso modo 4.0 Internazionale. <http://creativecommons.org/licenses/by-sa/4.0/deed.it>



PyQB

Monga

Theano

Monte-Carlo

# Lecture XXII: Probabilistic programming



PyQB

Monga

Theano

Monte-Carlo

# How science works

Describing one single “scientific method” is problematic, but a schema many will accept is:

- ① Imagine a hypothesis
- ② Design (mathematical/convenient) **models** consistent with the hypothesis
- ③ Collect experimental data
- ④ Discuss the fitness of data given the models

It is worth noting that the falsification of models is not *automatically* a rejection of hypotheses (and, more obviously, neither a validation).



PyQB

Monga

Theano

Monte-Carlo

# The role of Bayes Theorem

In this discussion, a useful relationship between data and models is Bayes Theorem.

$$P(M, D) = P(M|D) \cdot P(D) = P(D|M) \cdot P(M)$$

Therefore:

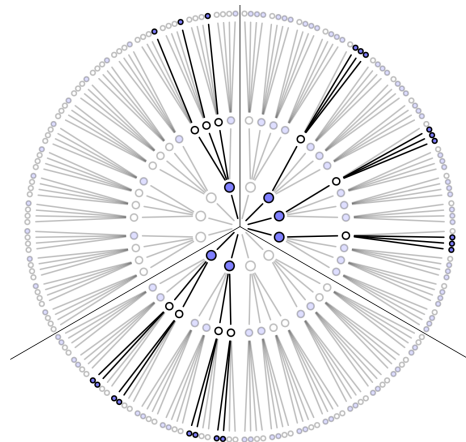
$$P(M|D) = \frac{P(D|M) \cdot P(M)}{P(D)}$$

The plausibility of the model given some observed data, is proportional to the number of ways data can be *produced* by the model and the prior plausibility of the model itself.

## Simple example



- Model: a bag with 4 balls in 2 colors B/W (but we don't know which of BBBB, BBBW, BBWW, BWWW, WWWW)
- Observed: BWB
- Which is the plausibility of BBBB, BBBW, BBWW, BWWW, WWWW?



Picture from: R. McElreath, Statistical Rethinking

Bayes Theorem is **counting**

149

PyQB

Monga

Theano

Monte-Carlo

## A computational approach



This Bayesian strategy is (conceptually) easy to transform in a computational process.

- 1 Code the models
- 2 Run the models
- 3 Compute the plausibility of the models based on observed data

150

PyQB

Monga

Theano

Monte-Carlo

## Classical binomial example



- Which is the proportion  $p$  of water covering Earth? The models are indexed by the float  $0 < p < 1$
- Given  $p$ , the probability of observing some  $W, L$  in a series of **independent random observations** is:  
$$P(W, L|p) = \frac{(W+L)!}{W!L!} p^W \cdot (1-p)^L$$
 (binomial distribution).
- Do we have an initial (prior) idea?
- Make observations, apply Bayes, update prior!

151

PyQB

Monga

Theano

Monte-Carlo

## A conventional way of expressing the model



$$W \sim \text{Binomial}(W + L, p)$$
$$p \sim \text{Uniform}(0, 1)$$

Probabilistic programming is systematic way of coding this kind of models, combining predefined statistical distributions and Monte Carlo methods for computing the posterior plausibility of parameters.

152

PyQB

Monga

Theano

Monte-Carlo

## In principle you can do it by hand



PyQB

Monga

Theano

Monte-Carlo

```
def dbinom(success: int, size: int, prob: float) -> float:
    fail = size - success
    return np.math.factorial(size)/(np.math.factorial(success)*np.math.factorial(fail))*p
    ↪ rob**success*(1-prob)**(fail)

W, L = 7, 3
p_grid = np.linspace(start=0, stop=1, num=20)
prior = np.ones(20)/20

likelihood = dbinom(W, n=W+L, p=p_grid)

unstd_posterior = likelihood * prior

posterior = unstd_posterior / unstd_posterior.sum()
```

Unfeasible with many variables!

153

## PyMC



PyQB

Monga

Theano

Monte-Carlo

```
import pymc as pm

W, L = 7, 3
earth = pm.Model()
with earth:
    p = pm.Uniform("p", 0, 1) # uniform prior
    w = pm.Binomial("w", n=W+L, p=p, observed=W)
    posterior = pm.sample(2000)

posterior['p']
```

154

## Behind pymc3



PyQB

Monga

Theano

Monte-Carlo

The probabilistic programming approach of pymc3 is built on two “technologies”:

- 1 A library that mixes numerical and symbolic computations (Theano, soon becoming Aesara)
- 2 Markov Chain Monte-Carlo (MCMC) algorithms to estimate posterior densities

155

## Theano



PyQB

Monga

Theano

Monte-Carlo

It bounds numerical computations to its symbolic structure (“graph”)

```
import theano
from theano import tensor

a = tensor.dscalar('alpha')
b = tensor.dscalar('beta')

c = a + b**2

f = theano.function([a,b], c)

assert f(1.5, 2) == 5.5
```

156

## Symbolic manipulations



PyQB

Monga

Theano

Monte-Carlo

Variables can be used to compute values, but also symbolic manipulations.

```
d = tensor.grad(c, b)
```

```
f_prime = theano.function([a, b], d)
```

```
assert f_prime(1.5, 2) == 4.
```

Note you still need to give an a because the symbolic structure needs it.

157

## Markov Chain Monte-Carlo



PyQB

Monga

Theano

Monte-Carlo

It's way of estimating (relative) populations of "contiguous" states.

- It needs the capacity of evaluate the population/magnitude of any two close states (but a global knowledge of all the states *at the same time*)
- It's useful to estimate *posterior* distribution *without explicitly computing*  $P(D)$ :  $P(M|D) = \frac{P(D|M) \cdot P(M)}{P(D)}$

158

## Metropolis



PyQB

Monga

Theano

Monte-Carlo

The easiest MCMC approach is the so-called Metropolis algorithm (in fact appeared as Metropolis, N., **Rosenbluth, A., Rosenbluth, M.**, Teller, A., and Teller, E., 1953)

```
steps = 100000
positions = np.zeros(steps)
populations = [1,2,3,4,5,6,7,8,9,10]
current = 3
```

```
for i in range(steps):
    positions[i] = current
    proposal = (current + np.random.choice([-1,1])) %
    ↪ len(populations)
    prob_move = populations[proposal] /
    ↪ populations[current]
    if np.random.uniform(0, 1) < prob_move:
        current = proposal
```

159

## Convergence

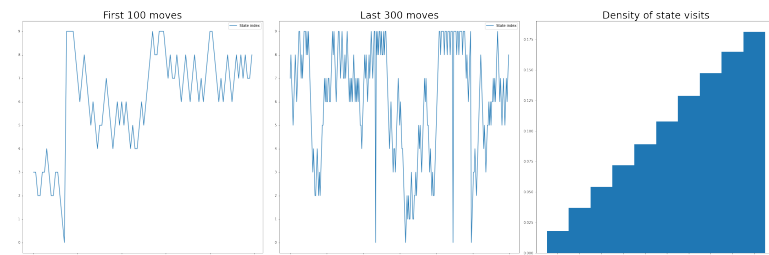


PyQB

Monga

Theano

Monte-Carlo



**Eventual** convergence is guaranteed, but it can be painful slow (and you don't know if you are there...). Many algorithms try to improve: Gibbs, Hamiltonian-MC, NUTS...

160

## Putting them together



PyQB

Monga

Theano

Monte-Carlo

```
import pymc3 as pm

linear_regression = pm.Model()

with linear_regression:
    # Theano variables
    sigma = pm.Uniform('sigma_h', 0, 50)
    alpha = pm.Normal('alpha', 178, 20)
    beta = pm.Normal('beta', 0, 10)
    mu = alpha + beta*(adult_males['weight'] -
    ↪ adult_males['weight'].mean())
    # Observed!
    h = pm.Normal('height', mu, sigma,
    ↪ observed=adult_males['height'])

trace = pm.sample() # MCMC sampling
```