Probabilistic Programming

Marius Popescu

popescunmarius@gmail.com

2019 - 2020

Bayesian Modeling

Bayes' Theorem

$$P(A,B) = P(A|B)P(B) = P(B|A)P(A)$$

$$\Downarrow$$

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

1702 - 1761

Bayesian Modeling

$D =$ observed data

$\theta =$ model parameters

Likelihood

Prior Distribution

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

$$P(\theta|D) \propto P(D|\theta)P(\theta)$$

Posterior Distribution

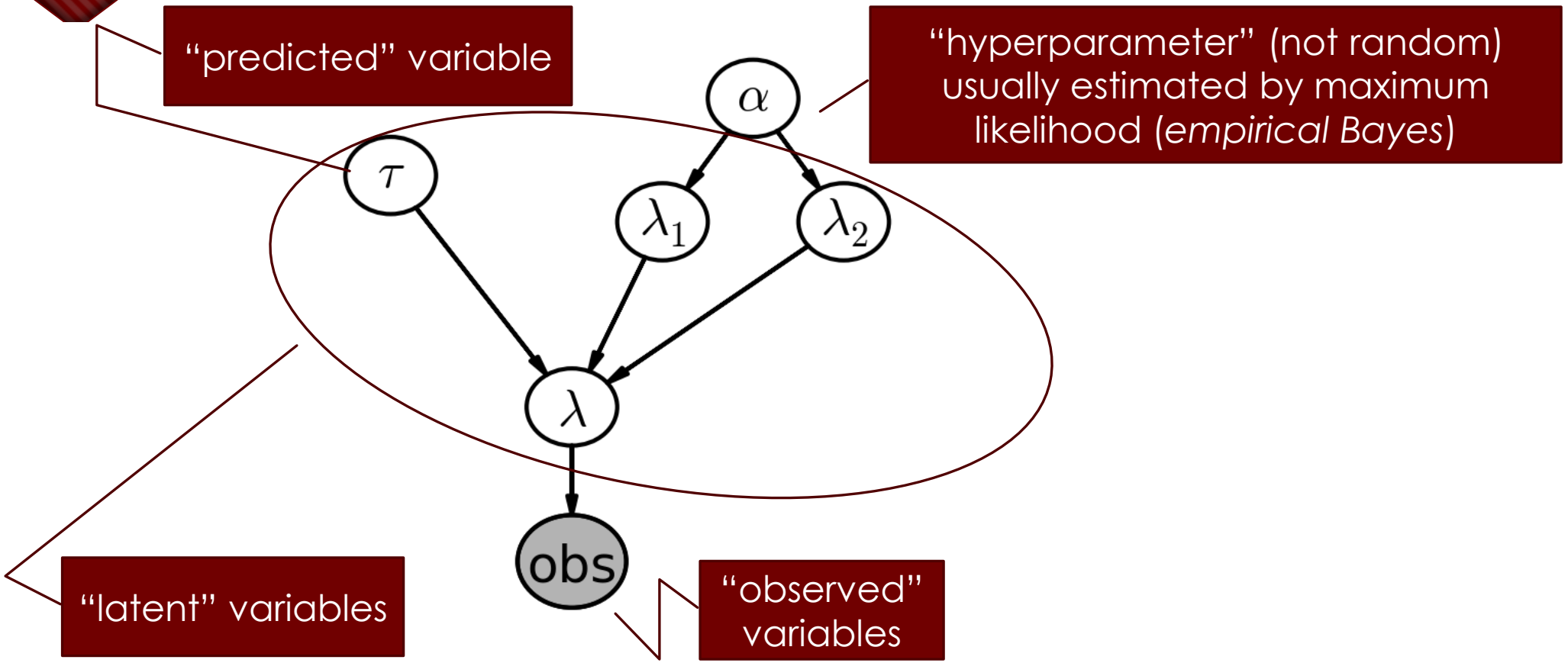
Generative Models

Generative models are statistical models that describe a joint distribution over three types of random variables:

- The "observed" random variables (often the "input space"), which are the random variables we have data for.
- The "latent" random variables, which are the random variables that play a role in the statistical model, but which are never observed (in the Bayesian setting, these are usually, at the very least, the parameters of the model).
- The "predicted" random variables, which are the random variables that represent the target predictions.

This categorization for the random variables in the joint distribution is not mutually exclusive (though observed random variables are never latent). For example, the predicted random variables can be also latent, such as in the unsupervised setting

The Switch Point Model



Learning from Data Scenarios

Learning Setting	Learning Input Data	Learning Output
Supervised (inductive)	$(x^{(1)}, z^{(1)}), \ldots, (x^{(n)}, z^{(n)})$	Mechanism to predict z values on arbitrary input instances
Supervised (transductive)	$(x_0^{(1)}, z^{(1)}), \ldots, (x_0^{(n)}, z_n)$ and $x_1^{(1)}, \ldots, x_1^{(m)}$	Predictions of z values on $x_1^{(i)}, i \in \{1, \ldots, m\}$
Semi-supervised	$(x_0^{(1)}, z_1), \ldots, (x_0^{(n)}, z_n)$ and $x_1^{(1)}, \ldots, x_1^{(m)}$	Mechanism to predict z values on arbitrary input instances
Unsupervised (instance-general)	$x^{(1)}, \ldots, x^{(n)}$	Mechanism to predict z values on arbitrary input instances
Unsupervised (instance-specific)	$x^{(1)}, \ldots, x^{(n)}$	Predictions of z on $x^{(1)}, \ldots, x^{(n)}$

Common procedures for learning from data. Observed data comes from the X random variable distribution, target predictions from the Z random variable distribution (indexed as appropriately for different instances).

Learning from Data Scenarios

An important concept common to all of these learning settings is that of marginal likelihood. The marginal likelihood is a quantity that denotes the likelihood of the observed data according to the model. In the Bayesian setting, marginalization is done over the parameters (taking into account the prior) and the latent variables.

Supervised Learning

$$\mathcal{L}\left(x^{(1)},\ldots,x^{(n)},z^{(1)},\ldots,z^{(n)}\right)=\int_{\theta}\left(\prod_{i=1}^{n}p\left(z^{(i)}|\theta\right)\,p\left(x^{(i)}|z^{(i)},\theta\right)\right)p(\theta)d\theta.$$

Unsupervised Learning

$$\mathcal{L}\left(x^{(1)},\ldots,x^{(n)}\right)=\int_{\theta}\left(\prod_{i=1}^{n}\sum_{z^{(i)}}p\left(z^{(i)}|\theta\right)p\left(x^{(i)}|z^{(i)},\theta\right)\right)p(\theta)d\theta.$$

Transductive Learning

$$\mathcal{L}\left(x^{(1)},\ldots,x^{(n)},z^{(1)},\ldots,z^{(n)},x'^{(1)},\ldots,x'^{(m)}\right)$$
$$=\int_{\theta}\left(\prod_{i=1}^{n}p\left(z^{(i)}|\theta\right)p\left(x^{(i)}|z^{(i)},\theta\right)p(\theta)\right)$$
$$\times\left(\prod_{i=1}^{m}\sum_{z'^{(i)}}p\left(z'^{(i)}|\theta\right)p\left(x'^{(i)}|z'^{(i)},\theta\right)\right)p(\theta)d\theta.$$

Bayesian Modeling in PyMC

German Tank Problem

By 1941-42, the allies knew that US and even British tanks had been technically superior to German Panzer tanks in combat, but they were worried about the capabilities of the new marks IV and V. More troubling, they had really very little idea of how many tanks the enemy was capable of producing in a year.

The statisticians had one key piece of information, which was the serial numbers on captured mark V tanks. The statisticians believed that the Germans, being Germans, had logically numbered their tanks in the order in which they were produced. And this deduction turned out to be right. It was enough to enable them to make an estimate of the total number of tanks that had been produced up to any given moment.

After the war, the allies captured German production records, showing that the true number of tanks produced in those three years was almost exactly what the statisticians had calculated, and less than one fifth of what standard intelligence had thought likely.

German Tank Problem

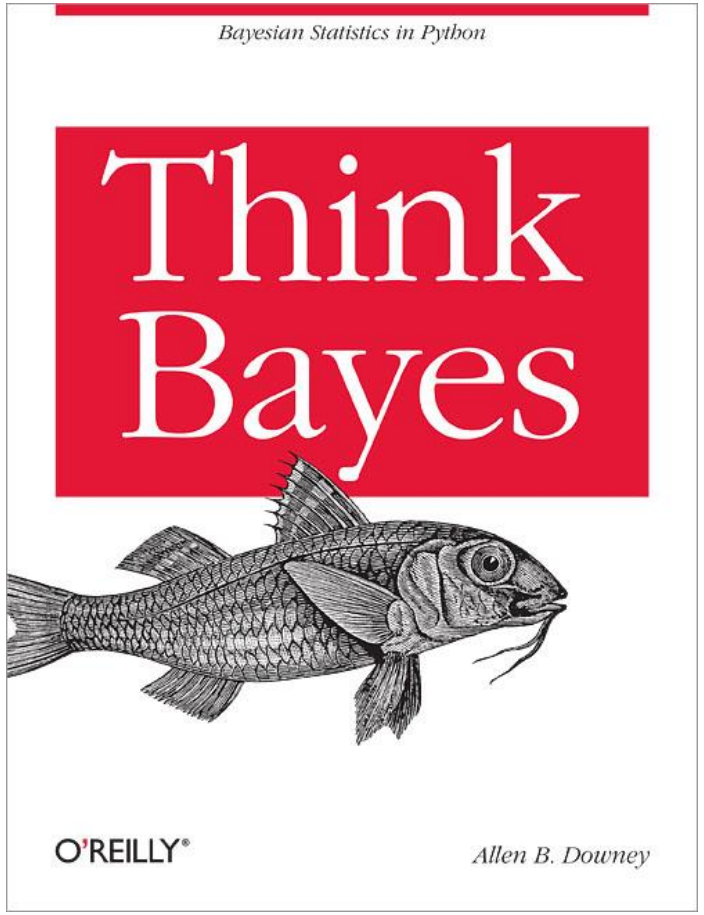
It become a classic problem in statistics: estimating the total number of tanks from a small sample

Suppose four tanks are captured with the serial numbers 10, 256, 202, and 97. Assuming that each tank is numbered in sequence as they are built, how many tanks are there in total?

The problem consists in estimating the maximum of a discrete uniform distribution from sampling without replacement. In simple terms, suppose we have an unknown number of items which are sequentially numbered from 1 to N. We take a random sample of these items and observe their sequence numbers; the problem is to estimate N from these observed numbers.

The problem can be approached using either frequentist inference or Bayesian inference

https://en.wikipedia.org/wiki/German_tank_problem



German Tank Problem: The Model

- Since we are Bayesianists, we don't want a singular estimate, we want a probability distribution for the total number of tanks. Therefore, we need to calculate the distribution of total tanks N, given the serial numbers D:

$$PN|D) \propto P(D|N)P(N)$$

- To decide how to model the likelihood, we can think about how we would create our data. Simply, we just have some number of tanks, with serial numbers 1,2,3,...,N, and we uniformly draw four tanks from the group. Therefore, we should use a discrete uniform distribution:

$$P(D_i|N) \sim DiscreteUniform(0, N)$$

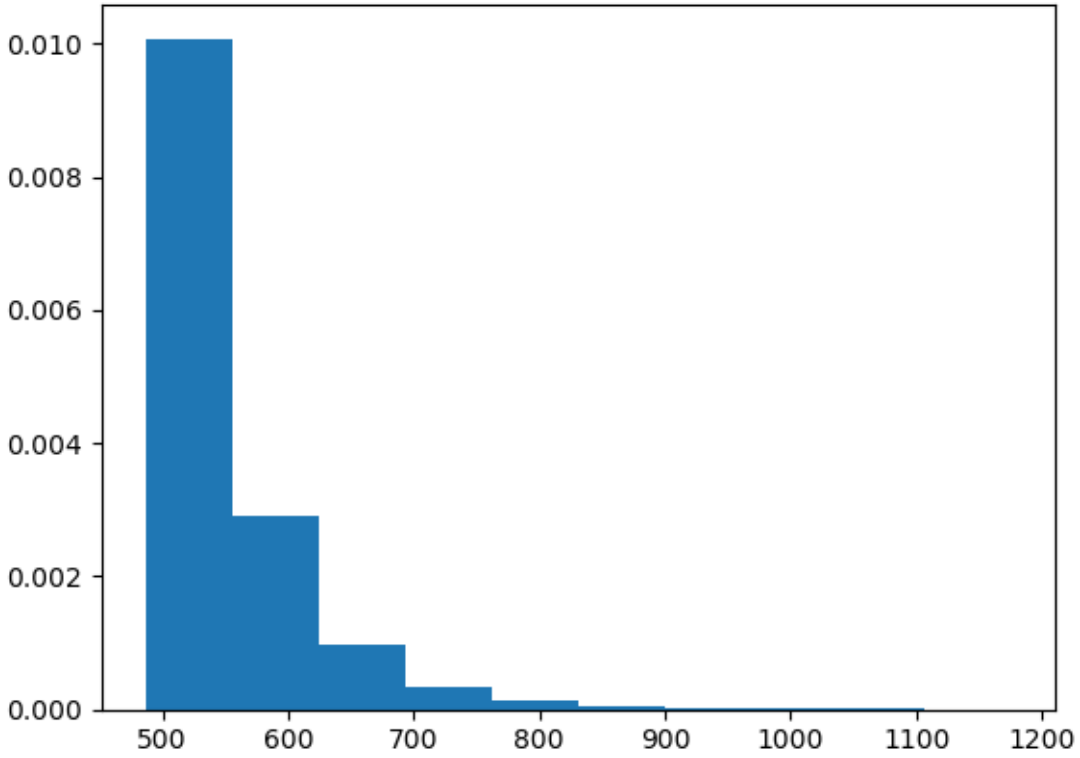
- Consider the prior information about N, $P(N)$. We know that it has to be at least equal to the largest serial number, m. As for an upper bound, we can guess that it isn't into the millions, since every serial number we saw is less than 300. set an upper bound at 10000:

$$P(N) \sim DiscreteUniform(m, 10000)$$

German Tank Problem: PyMC

```
true_N = 500
D = pm.rdiscrete_uniform(1, true_N, size = 10)

N = pm.DiscreteUniform("N", lower=D.max(), upper=10000)

observation = pm.DiscreteUniform("obs", lower=0, upper=N, value=D, observed=True)

model = pm.Model([observation, N])

mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000, 1)

N_samples = mcmc.trace('N')[:]

plt.hist(N_samples, normed = True)
plt.show()
```

German Tank Problem: Results



Graphical Models

Bayesian Networks

The Problem

Bayesian modeling:

$$D = \text{observed data}$$
$$\theta = \text{model parameters}$$

$$P(\theta|D) \propto P(D|\vartheta)P(\theta)$$
$$\theta = (\theta_1, \theta_2, \dots, \theta_n)$$

How can we estimate the $P(\theta_1, \theta_2, \dots, \theta_n|D)$?

Generally, how can we estimate a joint distribution $P(x_1, x_2, \dots, x_n)$?

Chain Rule

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \dots P(x_n|x_1, x_2, \dots, x_{n-1})$$

- Become intractably large as the number of variables grows
- We would need an awful lot of data to learn so many parameters

Conditional Independence

- The key to efficiently representing and estimating large joint distributions is to make some assumptions about conditional independence
- x and y are conditionally independent given z, denoted $x \perp y \mid z$, if and only if the conditional joint can be written as a product of conditional marginals:

$$x \perp y \mid z \Leftrightarrow P(x, y|Z) = P(z|z)P(y|z)$$

- An extreme case: Naïve Bayes, all the variable are independent

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2) \dots P(x_n)$$

Graphical Models

- A graphical model is a way to represent a joint distribution by making conditional independence assumptions. In particular, the nodes in the graph represent random variables, and the (lack of) edges represent conditional independence assumptions
- There are several kinds of graphical model, depending on whether the graph is directed, undirected, or some combination of directed and undirected
- Directed acyclic graph → Bayesian Networks

Bayesian Networks

A Bayesian network is a directed graph in which:

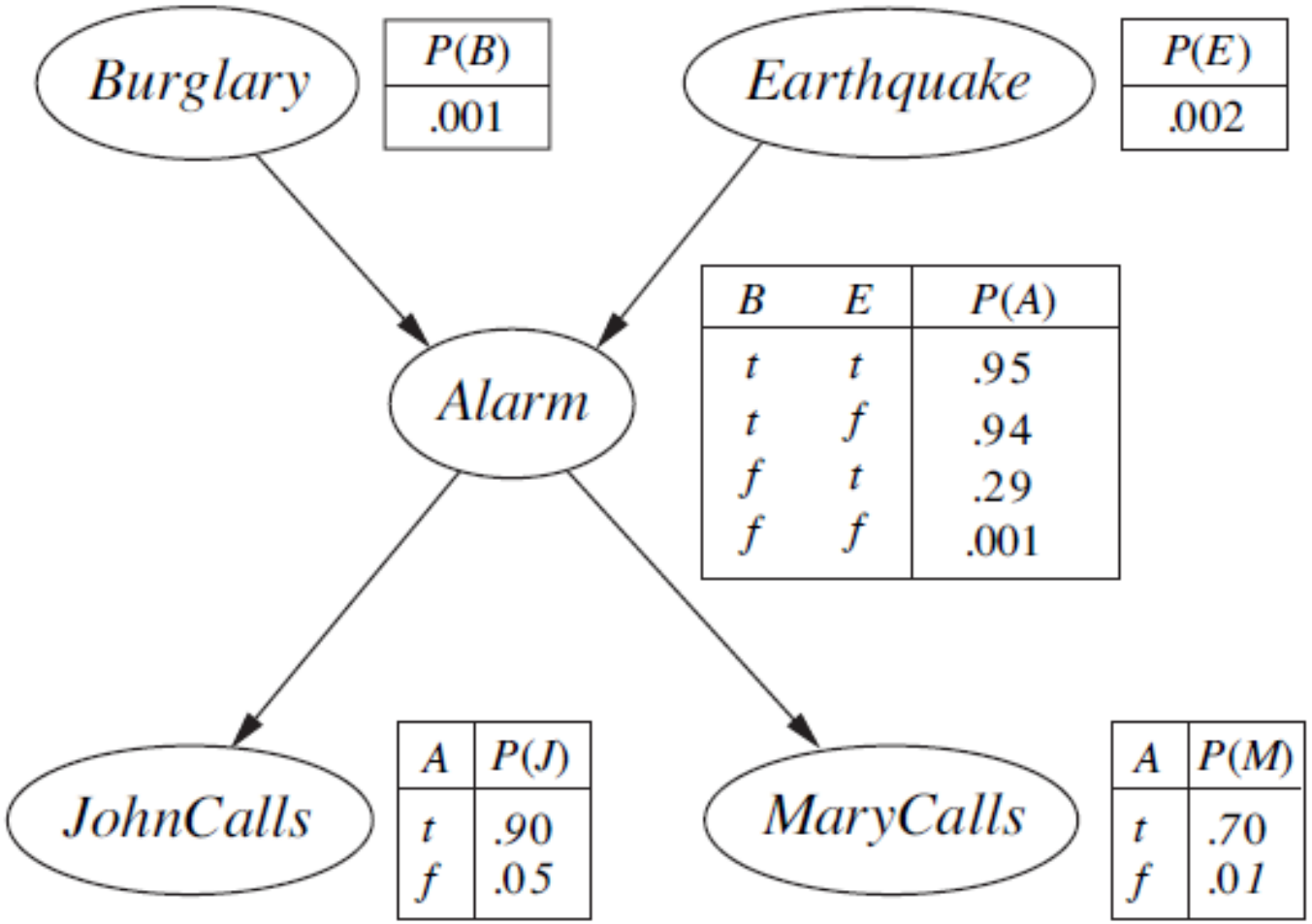
- Each node corresponds to a random variable, which may be discrete or continuous
- A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be a *parent* of Y. The graph has no directed cycles (and hence is a directed acyclic graph, or DAG)
- Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i)$ that quantifies the effect of the parents on the node

The topology of the network — the set of nodes and links — specifies the conditional independence relationships that hold in the domain. The combination of the topology and the conditional distributions suffices to specify (implicitly) the full joint distribution for all the variables.

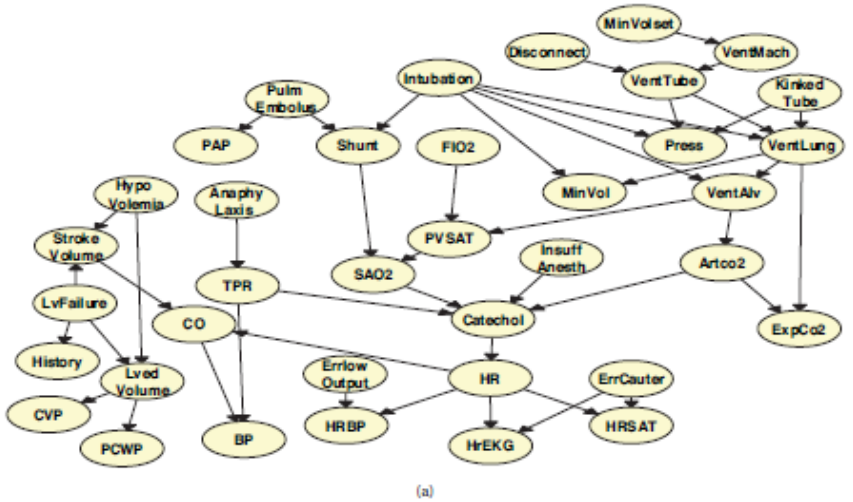
A Typical Bayesian Network: Burglar System

You have a new burglar alarm installed at home. It is fairly reliable at detecting a burglary, but also responds on occasion to minor earthquakes. (This example is due to Judea Pearl, a resident of Los Angeles—hence the acute interest in earthquakes.) You also have two neighbors, John and Mary, who have promised to call you at work when they hear the alarm. John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and often misses the alarm altogether. Given the evidence of who has or has not called, we would like to estimate the probability of a burglary.

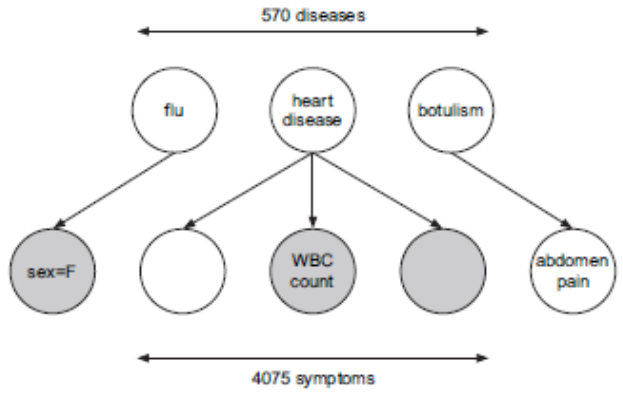
A Typical Bayesian Network: Burglar System



The quick medical reference or QMR network models infectious diseases



(a)



Inference in Bayesian Networks

The task of inference in Bayesian Networks is to compute the posterior probability distribution for a set of *query variables*, given some observed event—that is, some assignment of values to a set of *evidence variables*

- There are exact inference algorithms like: *variable elimination and clique tree* but *the general case is intractable*
- There are also *approximate inference* like: *Markov chain Monte Carlo*, *variational methods* and *message passing algorithms*

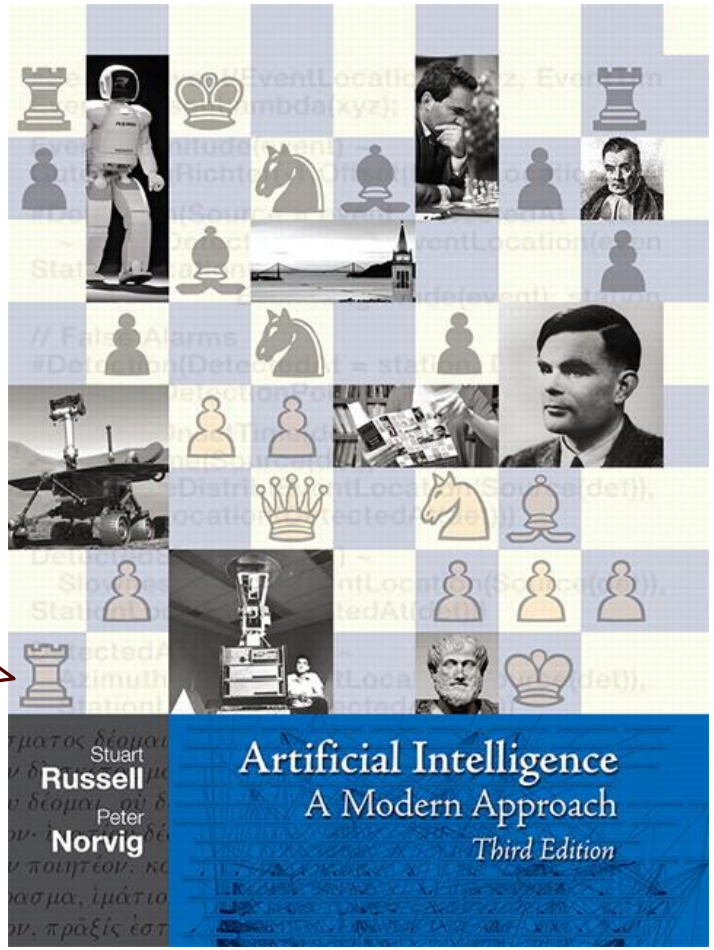
Inference in Burglar System

In the burglary network, we might observe the event in which *JohnCalls* = *true* and *MaryCalls* = *true*. We could then ask for, say, the probability that a burglary has occurred. This mean the posterior distribution of $P(B|J = t, M = t)$.

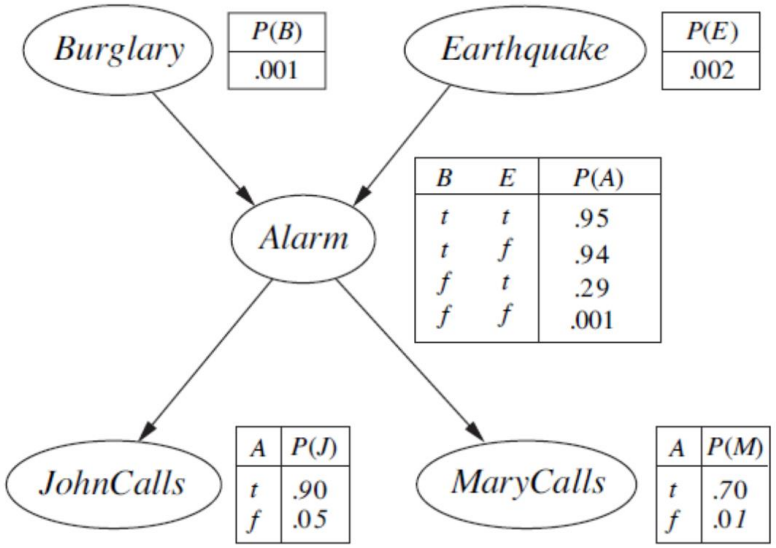
An exact inference will give us the distribution:

$$\begin{pmatrix} t & f \\ 0.284 & 0.716 \end{pmatrix}$$

See chapter 14



Burglar System in PyMC




```
B = pm.Bernoulli("B", 0.001)
E = pm.Bernoulli("E", 0.002)

p_A = pm.Lambda("p_A", lambda B = B, E = E: np.where(B, np.where(E, .95, .94), np.where(E, .29, 0.001)))
A = pm.Bernoulli("A", p_A)

p_J = pm.Lambda("p_J", lambda A = A: np.where(A, .9, .05))
J = pm.Bernoulli('J', p_J, value = [1], observed = True)
p_M = pm.Lambda("p_M", lambda A = A: np.where(A, .7, .01))
M = pm.Bernoulli('M', p_M, value = [1], observed = True)

model = pm.Model([B, E, A, J, M])
mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000, 1)
B_samples = mcmc.trace('B')[:]

print("Burglary probability:", B_samples.mean())   # Burglary probability:0.283566
```

3

Graph-Based Inference

Jan-Willem van de Meent
College of Computer and Information Science
Northeastern University
j.vandemeent@northeastern.edu

Brooks Paige
Alan Turing Institute
University of Cambridge
bpaige@turing.ac.uk

Hongseok Yang
School of Computing
KAIST
hongseok.yang@kaist.ac.kr

Frank Wood
Department of Computer Science
University of British Columbia
fwood@cs.ubc.ca

3.1 Compilation to a Graphical Model

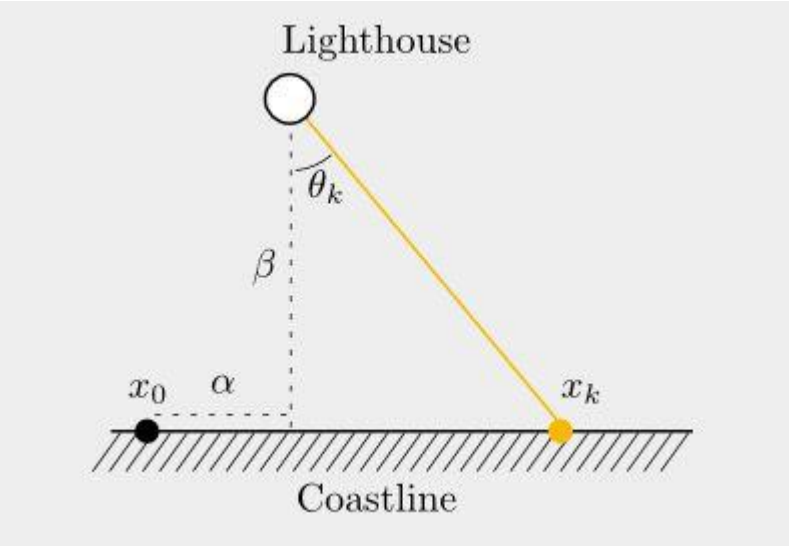
Programs written in the FOPPL specify probabilistic models over finitely many random variables. In this section, we will make this aspect clear by presenting the translation of these programs into finite graphical models. In the subsequent sections, we will show how this translation can be exploited to adapt inference algorithms for graphical models to probabilistic programs.

Data Likelihood

@observed

The Lighthouse Problem

There exists a lighthouse α miles along a straight coastline (relative to some position $x = 0$) and β miles offshore. As it rotates, it briefly flashes highly collimated beams of light at random intervals, each time the lighthouse flashes the angle of the beam is sampled uniformly between $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$. We have light detectors along the coastline, and so for the kth time the light flashes (not counting the times the light is facing away from the coast) we record a position x_k. Given a sequence of observations, infer the position coordinates α and β.



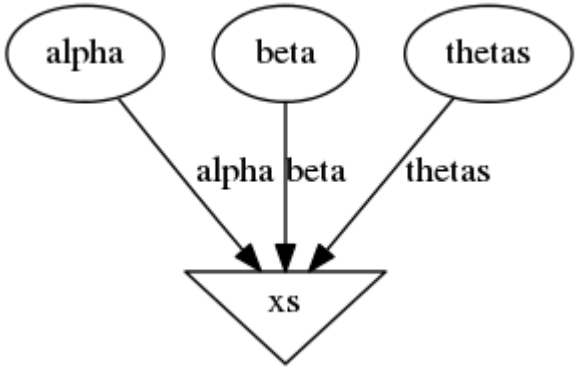
Gull, Stephen F. "Bayesian inductive inference and maximum entropy." Maximum-entropy and Bayesian methods in science and engineering. Springer, Dordrecht, 1988. 53-74.

The Lighthouse Problem: Creating the Data

```
# The parameters to be inferred. We only know them here because we are synthesising the data.
true_alpha = 10
true_beta = 50

num_flashes = 5000

# Generate the angles
true_thetas = np.random.uniform(low=-0.5*np.pi, high=0.5*np.pi, size=num_flashes)

# Generate the x coordinates of the flashes along the coastline
data = true_alpha + true_beta * np.tan(true_thetas)
```

The Lighthouse Problem: First Modelling Attempt

```
# Our prior distribution on alpha and beta.
# Alpha is normally distributed with a standard deviation of 50 miles.
# Beta is exponentially distributed with a mean value of 100 miles.
alpha = pm.Normal("alpha", 0, 1.0/50**2)
beta = pm.Exponential("beta", 1.0/100)

# We have a prior distribution for the angle of the lighthouse for every time we observed a flash, uniform over [-pi/2, pi/2]
thetas = pm.Uniform("thetas", lower=-0.5*np.pi, upper=0.5*np.pi, size=num_flashes)

@pm.deterministic
def xs(alpha=alpha, beta=beta, thetas=thetas):
  return alpha + beta * np.tan(thetas)
```



A node in PyMC cannot be simultaneously deterministic and observed

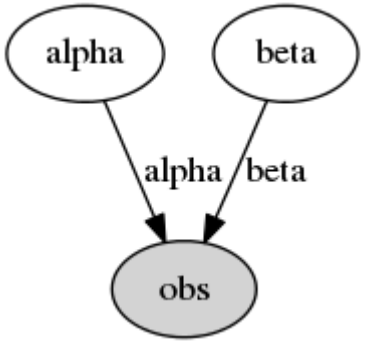
Why a node in PyMC cannot be simultaneously deterministic and observed?

In MCMC the values of all stochastic, non-observed nodes are initialized randomly. Then, a new position in the space of possible values for each of these nodes is considered. Whether we move to the new location depends on the value of the posterior distribution at the old location and at the new location, and this process is repeated.

Because xs is a deterministic function of its parents, the likelihood function for a given value of x, $p(x|\alpha, \beta, \theta)$ is zero for almost all values of α, β, and θ. As a result, the posterior distribution is also zero almost everywhere. Since almost everywhere the MCMC process looks, it sees a posterior probability of zero, it has no way to explore the space effectively.

But we are interested only in the posterior distribution of α and β

$$P(\alpha, \beta|D) \propto P(D|\alpha, \beta)P(\alpha, \beta)$$



The likelihood $P(D|\alpha, \beta)$ depends on the probability of θ, but if we have a probability density function $f_X(x)$ for the random variable X, then probability density function for the random variable $g(X)$ is $f_{g(X)}(y) = f_X(g^{-1}(y))\frac{\partial(g^{-1}(y))}{\partial(y)}$

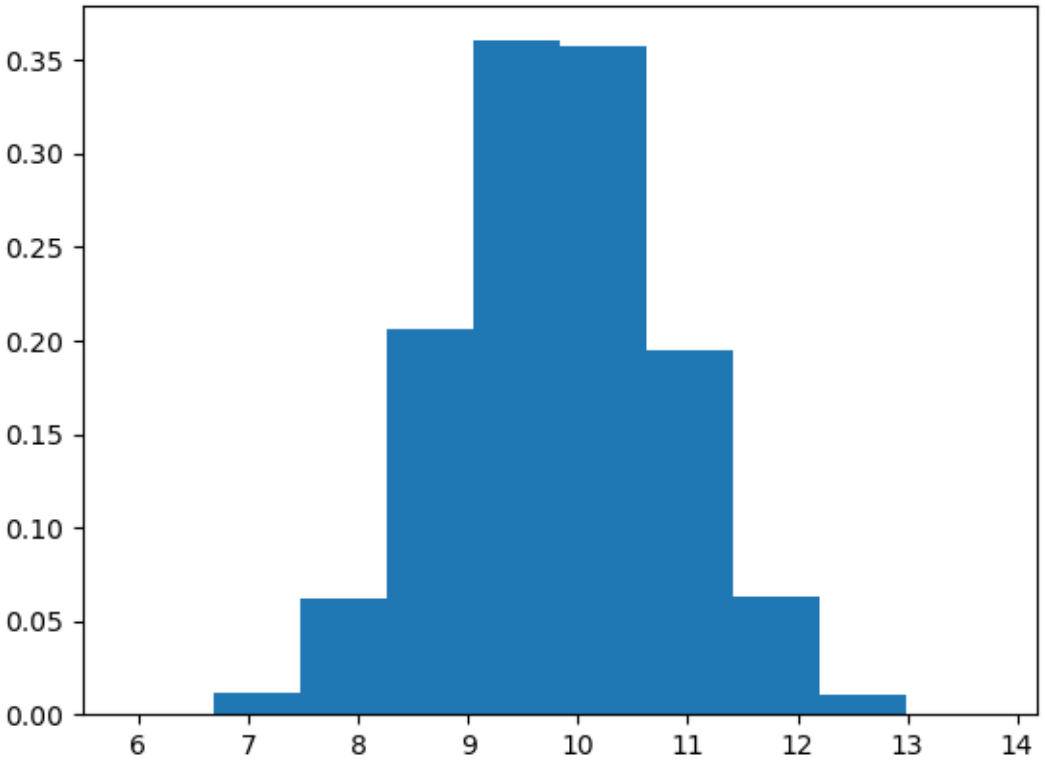
$$f_\theta(\theta) = \frac{1}{\pi} \Rightarrow f_X(x) = \frac{\beta}{\pi(\beta^2 + (\alpha - x)^2)}$$

The log probability is then: $\log(\beta) - \log(\pi) - \log(\beta^2 + (\alpha - x)^2)$

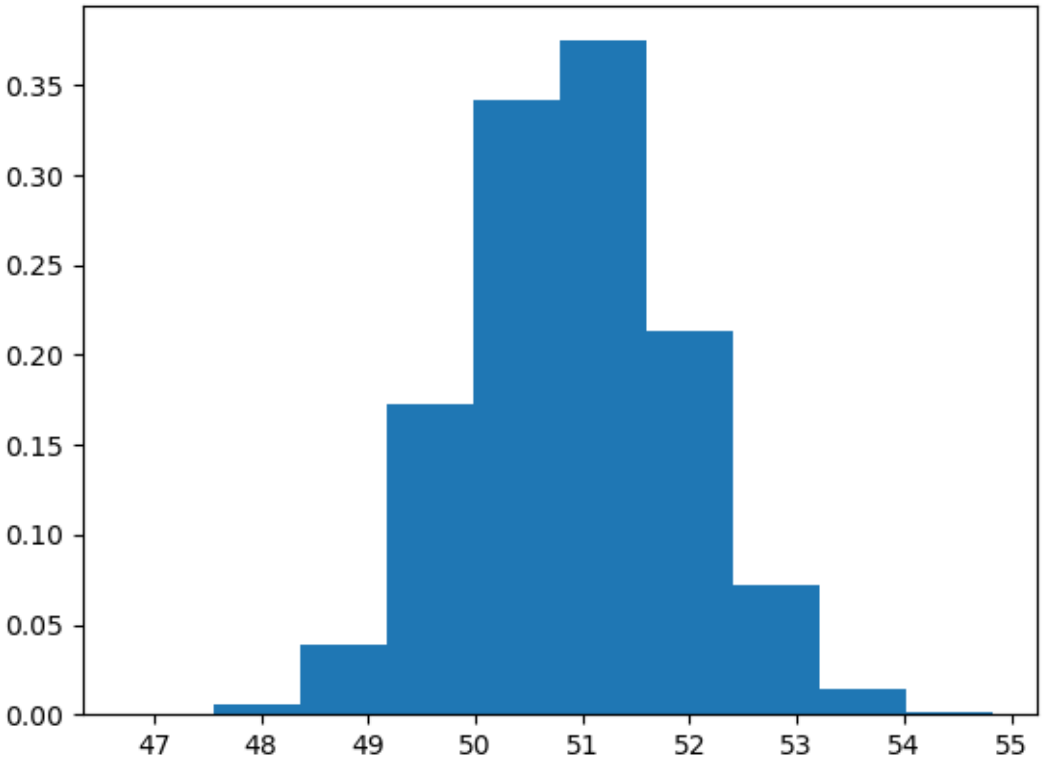
The Lighthouse Problem: The Proper Model

```
alpha = pm.Normal("alpha", 0, 1.0/50**2)
beta = pm.Exponential("beta", 1.0/100)

# @pm.stochastic(name="obs", dtype=np.float64, observed=True)
# def obs(value=data, alpha=alpha, beta=beta):
#     return np.sum(np.log(beta) - np.log(np.pi) - np.log(beta**2 + (alpha-value)**2))

@pm.observed(name="obs", dtype=np.float64)
def obs(value=data, alpha=alpha, beta=beta):
    return np.sum(np.log(beta) - np.log(np.pi) - np.log(beta**2 + (alpha-value)**2))

model = pm.Model([alpha, beta, obs])mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000, 1)
```

The Lighthouse Problem: Results



α posterior distribution



β posterior distribution

Factor Potentials

@potential

Factor Potentials

- Bayesian modeling: $P(\theta|D) \propto P(D|\theta)P(\theta)$
- In some cases, it's nice to be able to modify the joint density by incorporating terms that don't correspond to probabilities of variables conditional on parents

$$P(\theta_1, \theta_2, \ldots, \theta_n|D) \propto P(D|\theta_1, \theta_2, \ldots, \theta_n)P(\theta_1, \theta_2, \ldots, \theta_n)I(|\theta_3 - \theta_4| < 1)$$

- Arbitrary factors are implemented by objects of class `Potential` (decorator `@potential`)
- Potentials have one important attribute, `logp`, the log of their current probability or probability density value given the values of their parents
- Potentials have no methods. They have no trace attribute, because they are not variables. They cannot serve as parents of variables (for the same reason)

Survival Analysis

Suppose you are interested in the median lifetime of users of a service. It is not correct to exclude the current users: your estimate will overemphasize the presence of users who sign up and then leave almost immediately (very short lifespans), and completely underemphasize the surviving users (who generally have long lifetimes - the mere fact they are alive is evidence of this). The other possibility is to use the alive user's current lifetimes along with all other lifetimes. This is not advisable as the implicit assumption you've made is that no user can have a lifetime, even theoretically, larger than your own service's existence.

We need some new tools to deal with this situation

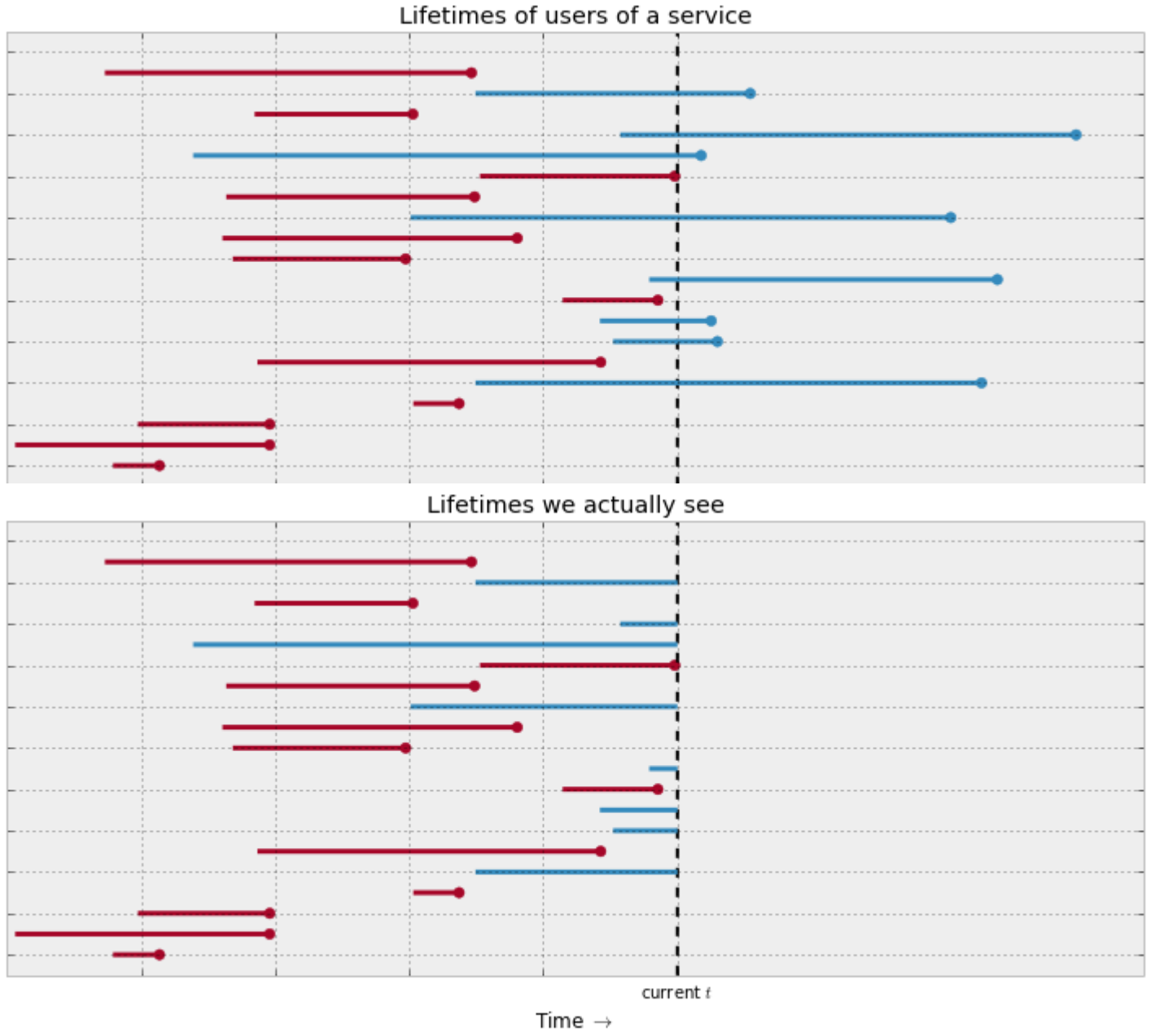
Survival Analysis

In the service model - when a customer joins your service, we consider that a birth, and when a customer leaves your service, that is a death. We have a record of all births, but at points in time we do not necessarily see all the deaths. Consider the graphic below. Each line represents a user moving through time. For each user, we of course observe their birth (joins service). At current time t, we only observe some deaths, denoted by red points, whereas the still active users have not suffered this death event yet. We call deaths we have not yet seen *right-censored events*.

Our objective is to calculate the median lifetime of all users, regardless of dead or alive.

Survival Analysis

Each line represents a user moving through time. For each user, we of course observe their birth (joins service). At current time t, we only observe some deaths, denoted by red points, whereas the still active users have not suffered this death event yet.



Weibull Distribution

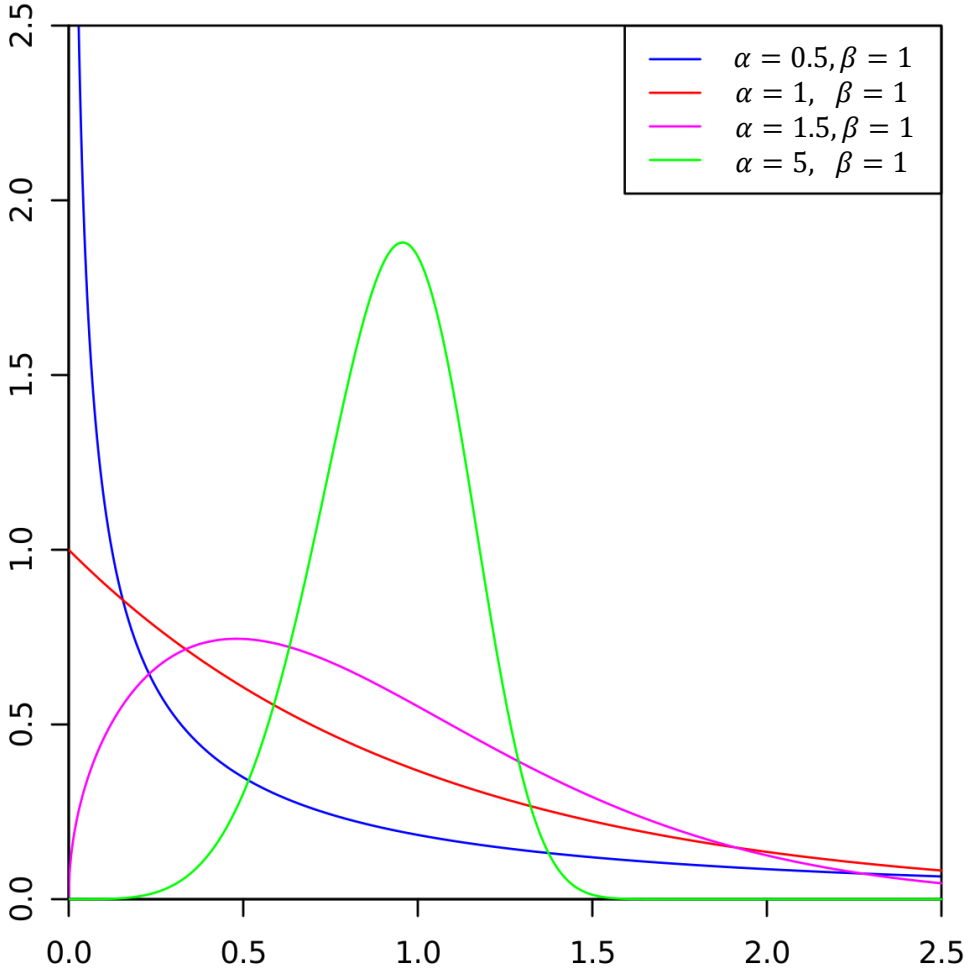
$$Z \sim \text{Weibull}(\alpha, \beta)$$

$$f_Z(z|\alpha,\beta) = \frac{\alpha z^{\alpha-1} e^{-\left(\frac{z}{\beta}\right)^{\alpha}}}{\beta^{\alpha}}, \qquad \alpha > 0, \beta > 0, z \geq 0$$

$$E(Z|\alpha,\beta) = \int_{-\infty}^{\infty} z f_Z(z|\alpha,\beta) dz = \beta\Gamma\left(1 + \frac{1}{\alpha}\right)$$

$$\text{median}(z) = \beta(\log 2)^{\frac{1}{\alpha}}$$

- α is the shape parameter and β is the scale parameter of the distribution, and the quantity Z can be viewed as a "time-to-failure“
- A value of $\alpha < 1$ indicates that the failure rate decreases over time. This happens if there is significant "infant mortality", or defective items failing early and the failure rate decreasing over time as the defective items are weeded out of the population
- A value of $\alpha = 1$ indicates that the failure rate is constant over time. This might suggest random external events are causing mortality, or failure. The Weibull distribution reduces to an exponential distribution
- A value of $\alpha > 1$ indicates that the failure rate increases with time. This happens if there is an "aging" process, or parts that are more likely to fail as time goes on



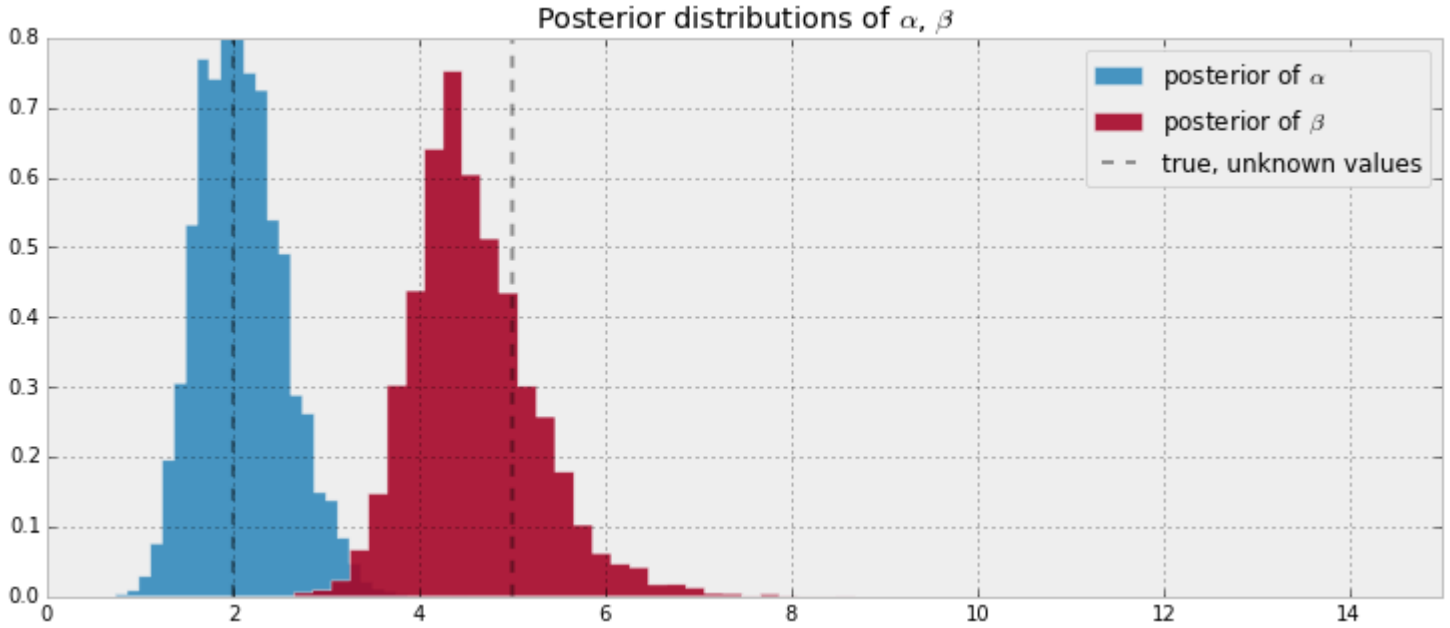
Survival Analysis: The model

```
N = 20


#create some artificial data.
lifetime = pm.rweibull( 2, 5, size = N )
birth = pm.runiform(0, 10, N)
censor = (birth + lifetime) > 10
#an individual is right-censored if this is True
lifetime_ = np.ma.masked_array( lifetime, censor ) #create the censorship event.
lifetime_.set_fill_value( 10 ) #good for computations later.


#this begins the model
alpha = pm.Uniform("alpha", 0,20)
#lets just use uninformative priors
beta = pm.Uniform("beta", 0,20)
obs = pm.Weibull( 'obs', alpha, beta, value = lifetime_, observed = True )
```

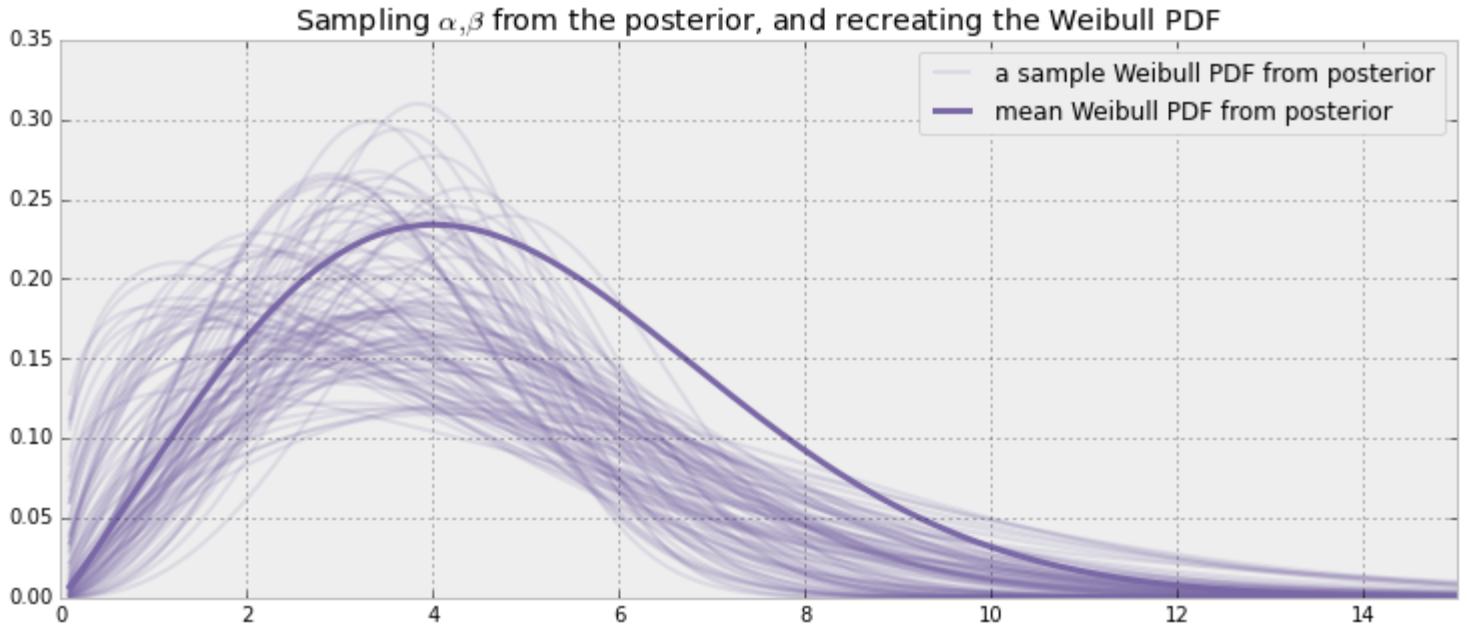
Survival Analysis: Censor Factor

```
# if any guess don't follow the hard limit that some samples are right-censored,
# than we discard that guess and try again

@pm.potential
def censor_factor(obs=obs):
    if np.any((obs + birth < 10)[lifetime_.mask] ):
        return -np.inf
    else:
        return 0

mcmc = pm.MCMC([alpha, beta, obs, censor_factor ] )
mcmc.sample(50000, 30000)
```

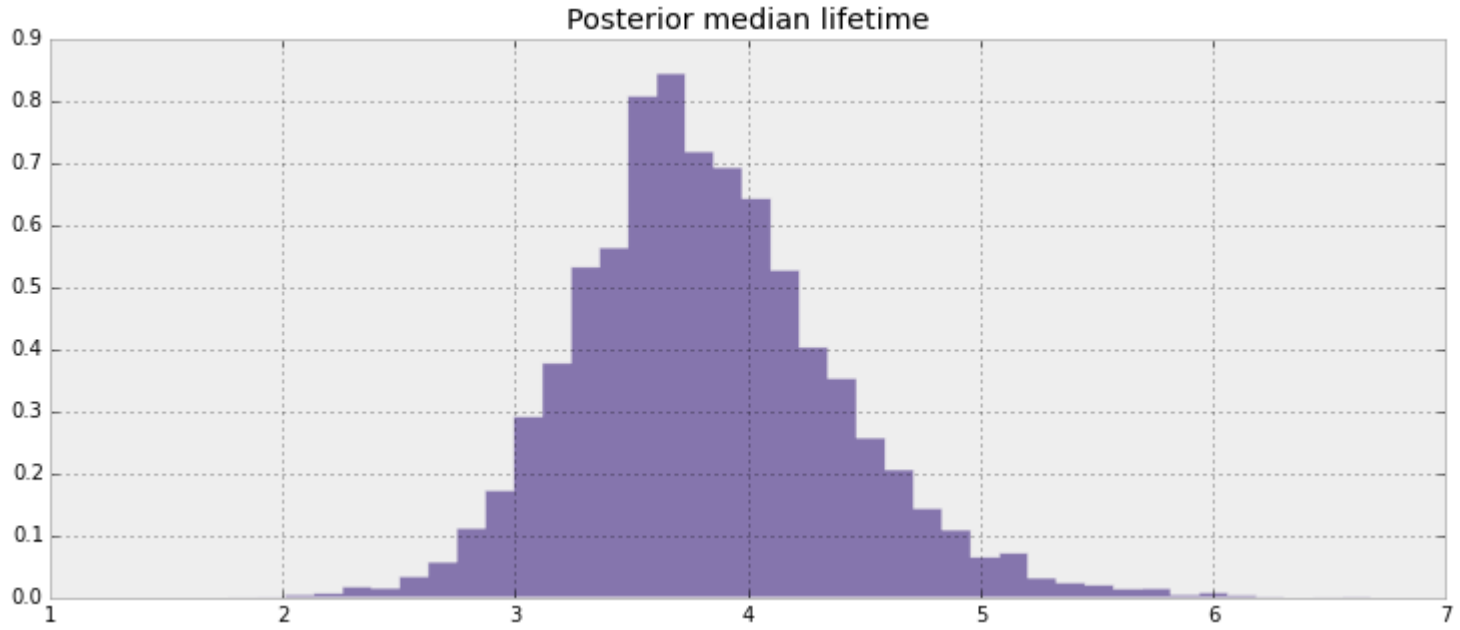
Survival Analysis: Results



Survival Analysis: Results



Survival Analysis: Results




```
alpha_samples = mcmc.trace('alpha')[:]
beta_samples = mcmc.trace('beta')[:]


medianlifetime_samples = beta_samples * (np.log(2)**(1 / alpha_samples))
```

Using the log-likelihood for the Weibull distribution

```
lifetime = pm.rweibull( 2, 5, size = N )
birth = pm.runiform(0, 10, N)
censor = (birth + lifetime) > 10 #an individual is right-censored if this is True
lifetime_ = lifetime.copy()
lifetime_[censor] = 10 - birth[censor] #we only see this part of their lives.

#this begins the model
alpha = pm.Uniform("alpha", 0,20)
#lets just use uninformative priors
beta = pm.Uniform("beta", 0,20)

@pm.observed
def survival(value=lifetime_, alpha = alpha, beta = beta ):
    return np.sum( (1-censor) * (np.log( alpha/beta) + (alpha-1)*np.log(value/beta) - (value/beta)**(alpha)))

mcmc = pm.MCMC([alpha, beta, survival ] )
mcmc.sample(50000, 30000)
```