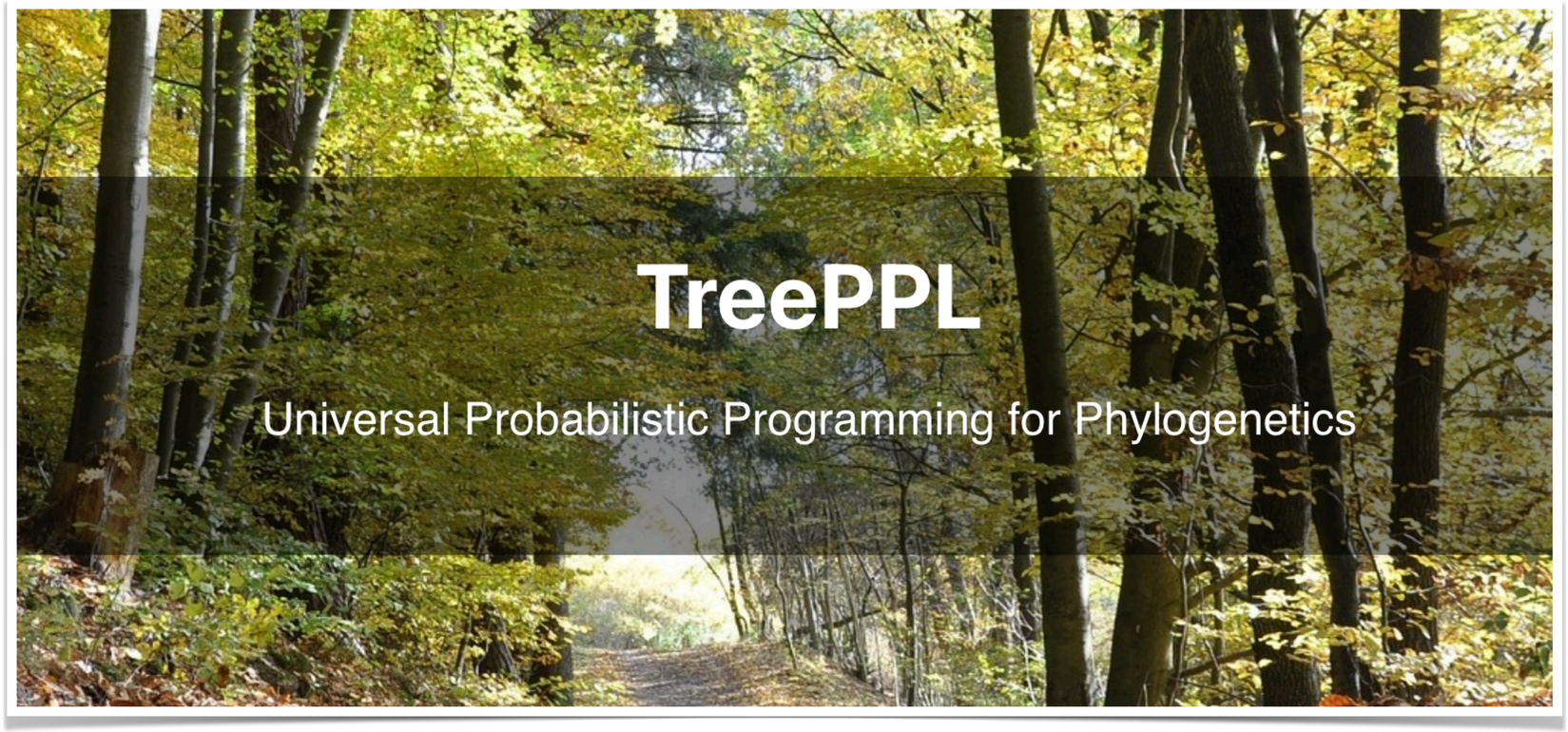
A Basic Tutorial to Probabilistic Programming in TreePPL



David Broman

Sampling - Hello flips in TreePPL

In Jupyter notebook.
Tells it to use TreePPL

Name of the resulting compiler object

Uses by default sequential Monte Carlo (SMC). Number of samples (particles)

PPL model function. Returns flips of type Bool

Assumes a random variable with Bernoulli distribution. Samples when executed.

Returns the sample

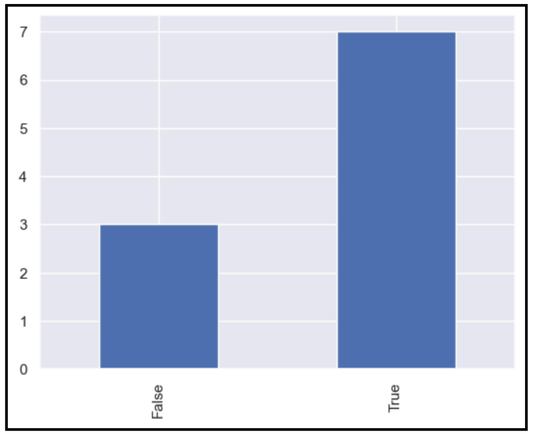
```
%%treeppl flip samples=10

model function flip() => Bool {
  assume p ~ Bernoulli(0.7);
  return p;
}
```

Run and print Samples

```
res = flip()
print(res.samples)
```

```
[True, False, False, True, False, True, True, True, True, True]
```

```
[True, True, False, True, True, True, True, True, False, False]
```



Sampling - Beta

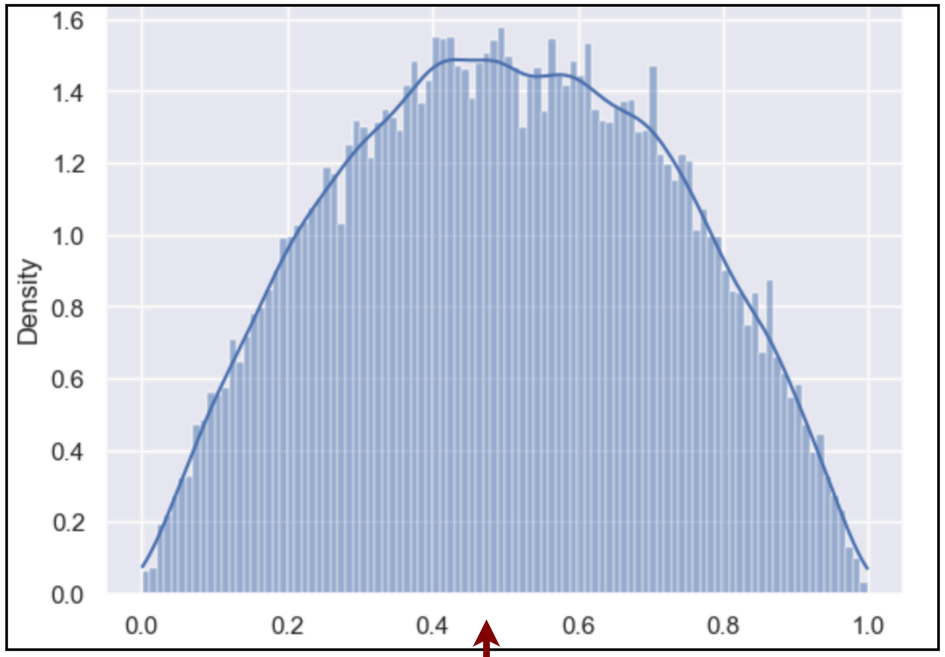
Sample from the Beta distribution

```
[66]: %%treeppl coin1 samples=10
      model function coin1() => Real {
          assume p ~ Beta(2.0,2.0);
          return p;
      }
```

```
[69]: res = coin1()
      res.samples
```

```
[69]: [0.169945521325,
       0.480157660145,
       0.236170470036,
       0.158170175226,
       0.699396620939,
       0.487037283544,
       0.922739621178,
       0.317941856273,
       0.425529510374,
       0.306744291676]
```

Real numbers (floating point approximations)

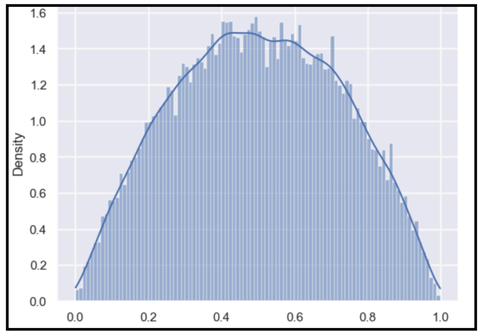


Plotting using 30000 samples

Observations and Inference

Previous model

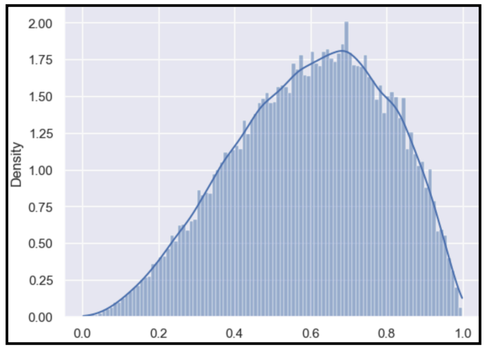
```
model function coin1() => Real {
    assume p ~ Beta(2.0,2.0);
    return p;
}
```



Still assumes p (beta), which is the prior

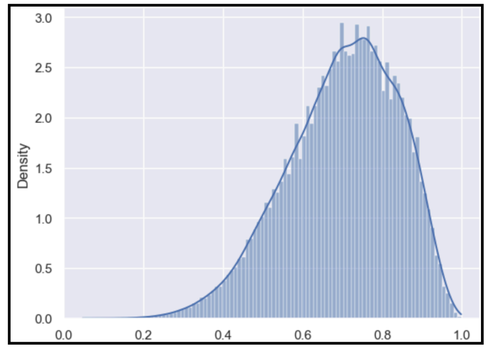
We make one observation

```
model function coin2() => Real {
    assume p ~ Beta(2.0,2.0);
    observe true ~ Bernoulli(p);
    return p;
}
```



What if several observations?

```
%%treeppl coin samples=30000
model function coin3() => Real {
    assume p ~ Beta(2.0,2.0);
    observe true ~ Bernoulli(p);
    observe true ~ Bernoulli(p);
    observe true ~ Bernoulli(p);
    observe false ~ Bernoulli(p);
    observe true ~ Bernoulli(p);
    observe true ~ Bernoulli(p);
    return p;
}
```



What is the result?

Bayes' rule and probabilistic programming languages

Goal: Estimate the parameters θ given the observation of data x.

Likelihood of generating data x given the parameters θ

In PPLs, the likelihood is given by all **observe** constructs.

Prior, our beliefs of the parameters right now.

In PPLs, the prior is stated using **assume** or **sample** constructs.

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)}$$

Posterior, the inferred parameters θ given the observed data.

In PPLs, the posterior is the returned output value, which can be the latent variable.

Normalizing constant. Determined by prior and likelihood.

In PPLs, some inference algorithms can approximate the normalization constant (e.g. importance sampling and SMC).

External data

What if we want to separate the data from the model code?

Give the data as input to the model

For loop, observe each data point

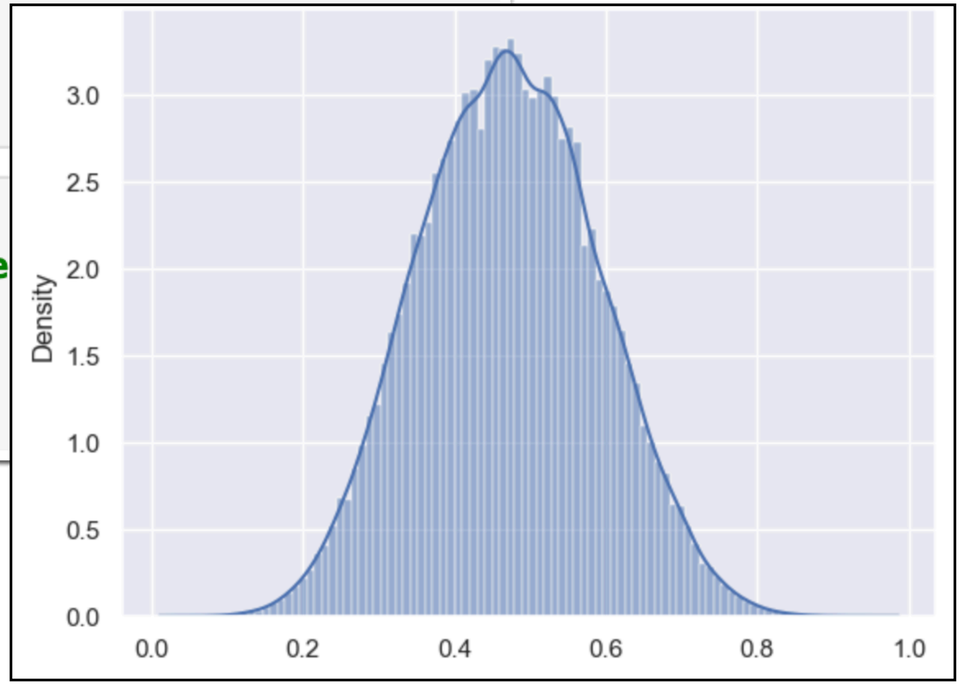
In this example: a less informative prior

Data

Give the data as a parameter when running

```
[137]: %%treeppl coin samples=50000

model function coin(outcomes: Bool[]) => Real {
  assume p ~ Uniform(0.0, 1.0);
  for i in 1 to (length(outcomes)) {
    observe outcomes[i] ~ Bernoulli(p);
  }
  return p;
}
```

```
[138]: data =[
    False, True, True, False, True, False, False
    True, False, True, False, False
]
myplot(coin(outcomes=data))
```



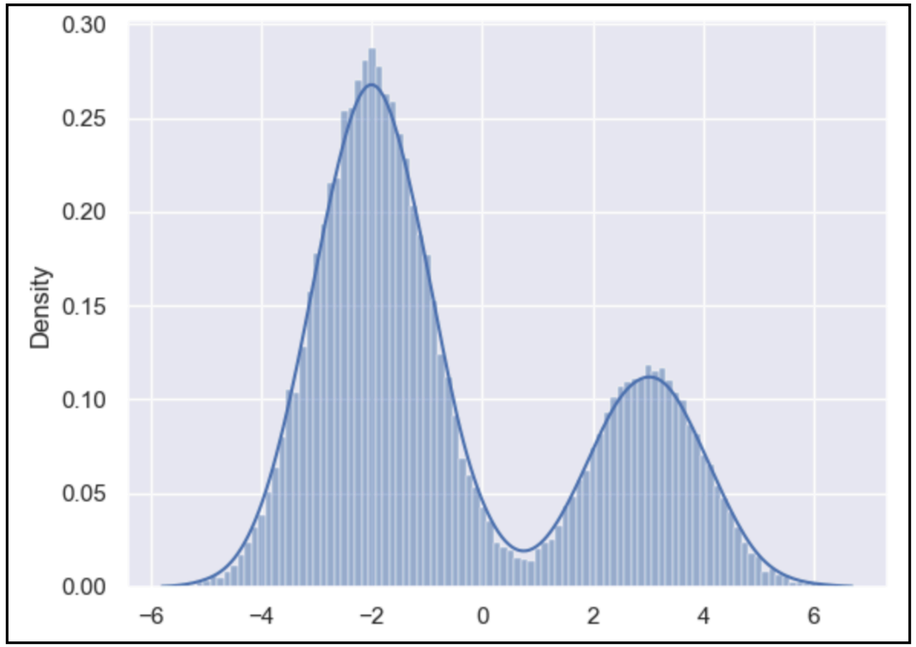
Stochastic branching

What does the resulting distribution look like?

Stochastic branching. Dependent on a random variable.

Sample a choice

```
model function foo() => Real {
  assume choice ~ Bernoulli(0.7);
  if(choice){
    assume p ~ Gaussian(-2.0, 1.0);
    return p;
  }
  else{
    assume p ~ Gaussian(3.0, 1.0);
    return p;
  }
}
```



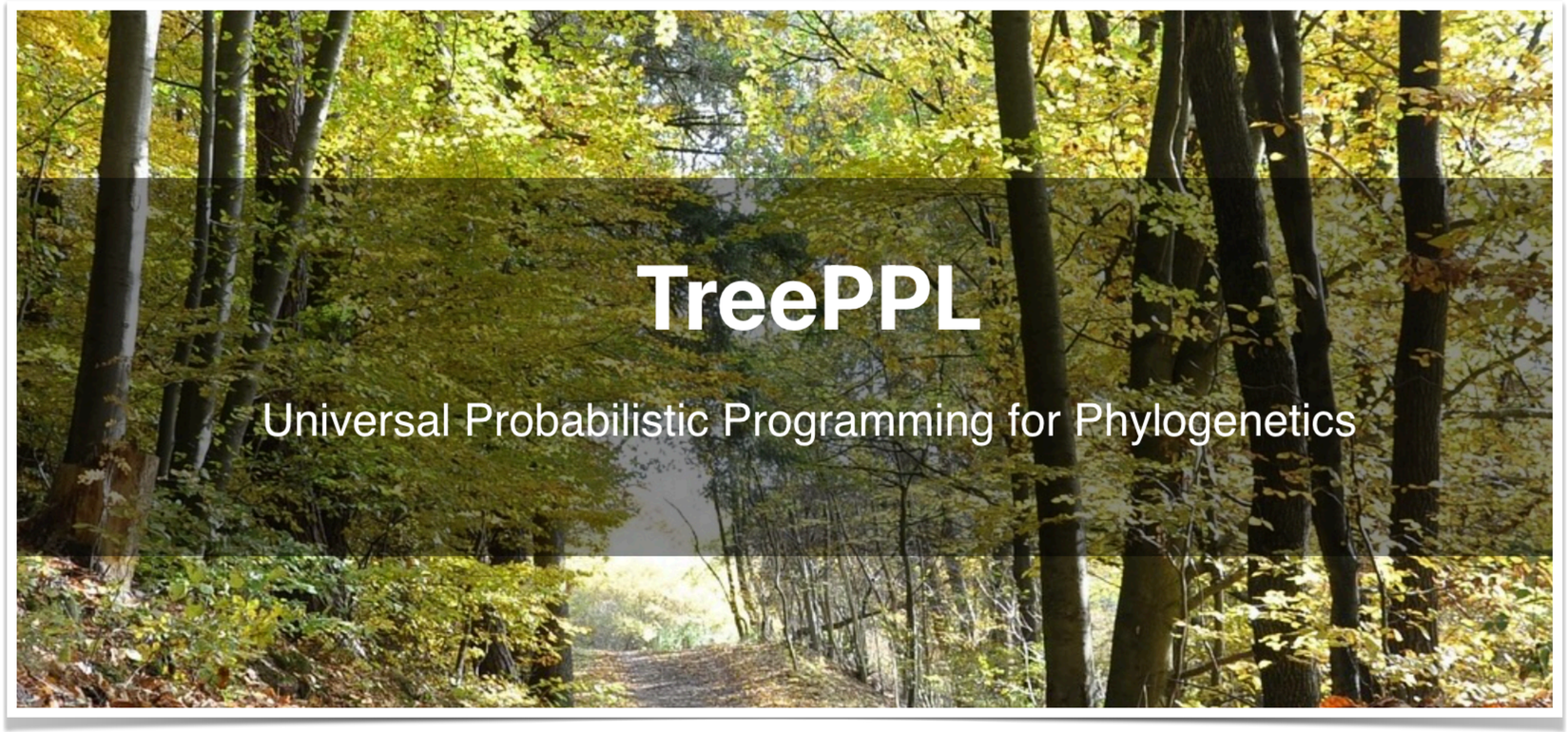
Returns samples from different distributions dependent on the choice.

Basically defines a mixture model!

TreePPL exercises (non phylogenetic)

1. **Download and install TreePPL** https://treeppl.org/

2. **Download the Jupyter notebook exercise file**
   https://treeppl.org/courses/basic-tutorial.zip



Exercise 1: recursively defined models

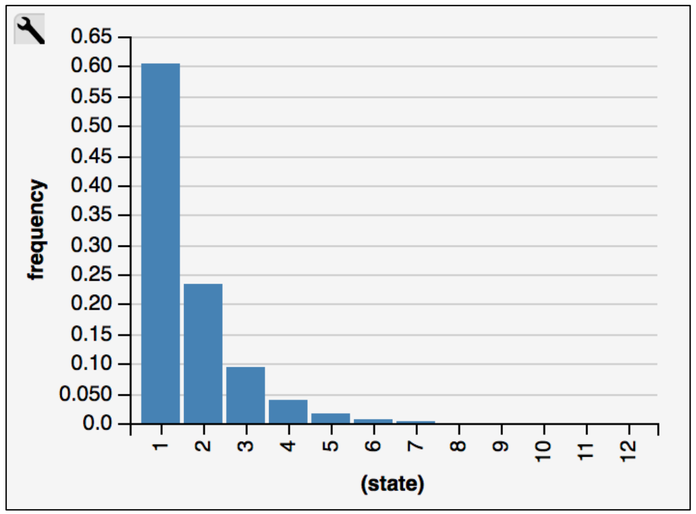
Exercise 1:

Create a model describing the geometric distribution, using the Bernoulli distribution.

Geometric distribution: X number of Bernoulli trials needed to get one success.

Parameter: $0 < p \leq 1$ probability of success

Support: x number of trials, $x \in \{1,2,3,\ldots\}$



Exercise 2: a graphical model

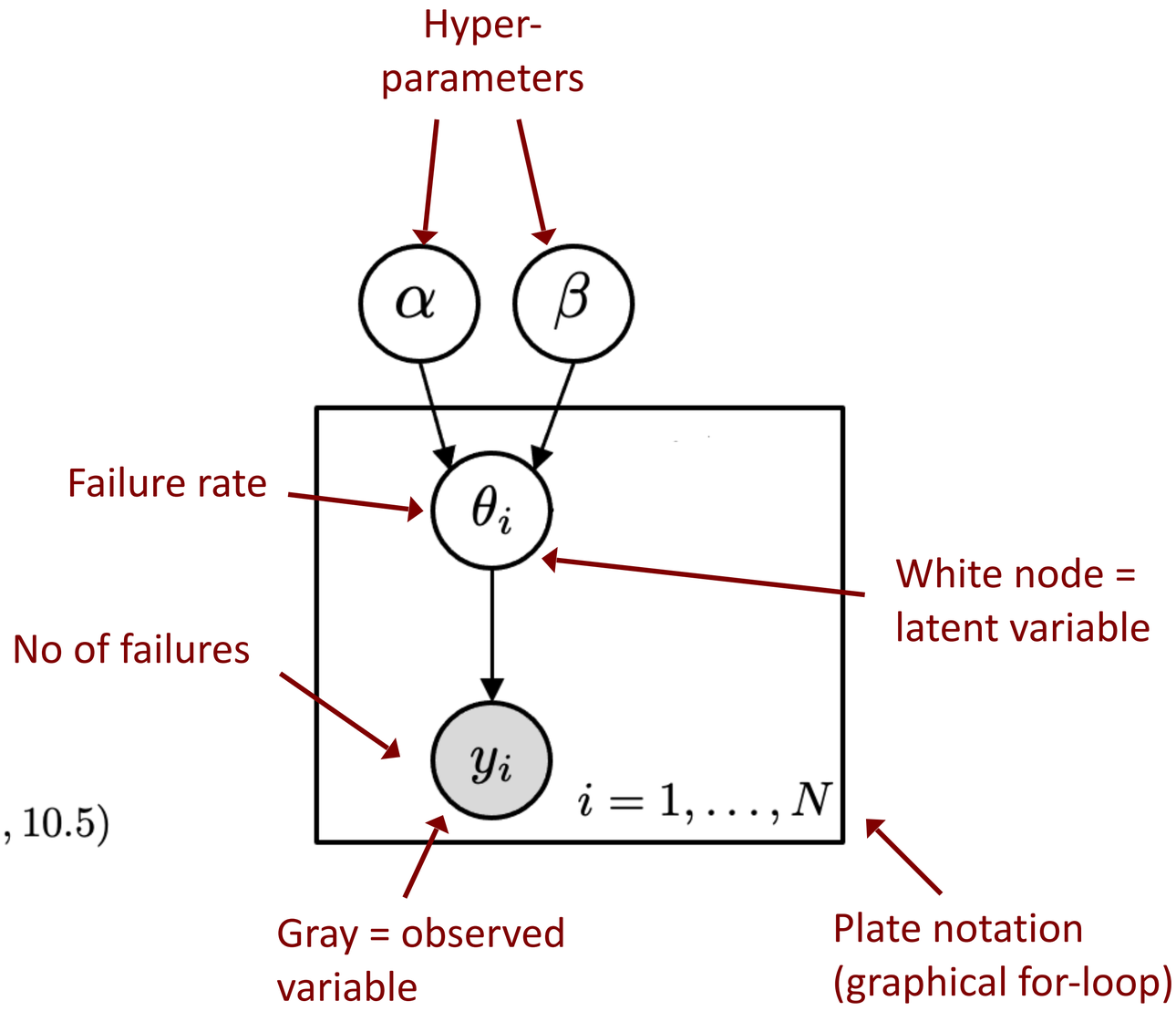
Pump example (George et al., 1993)

$$\alpha \sim \text{Exponential}(1)$$
$$\beta \sim \text{Gamma}(0.1, 1)$$
$$\theta_i \sim \text{Gamma}(\alpha, \beta) \quad i = 1, \ldots, N$$
$$y_i \sim \text{Poisson}(\theta_i t_i) \quad i = 1, \ldots, N$$

$$y = (5, 1, 5, 14, 3, 19, 1, 1, 4, 22)$$
$$t = (94.3, 15.7, 62.9, 126, 5.24, 31.4, 1.05, 1.05, 2.1, 10.5)$$
$$N = 10$$

Exercise 2:

Implement this graphical model and infer alpha



Exercise 3: Bayesian linear regression

Exercise 3: Suppose you have the following data: `length = [55, 57, 52, 64, 53, 64]` and `age = [4, 10, 2, 17, 6, 20]`, where the tuple `(length[i], age[i])` represents the length in cm and age in weeks for a baby. Create a TreePPL script that infers a posterior distribution over the length of six months old babies by using Bayesian linear regression of the form $l_i \sim N(\alpha + \beta a_i, \sigma)$, where N is the normal distribution, l_i the length, a_i the age, and α, β, and σ are random variables.

3a. Visualize the result and compute the expected value. What is the uncertainty of the estimation? Discuss the results.

3b. Discuss what reasonable priors for α, β, and σ can be. Test and explain how different priors affect the result.

3c. Suppose you extend the data set with one more data point, where the length is 100 cm, and the age is 5 weeks. How do the mean value and the uncertainty in the estimation change?

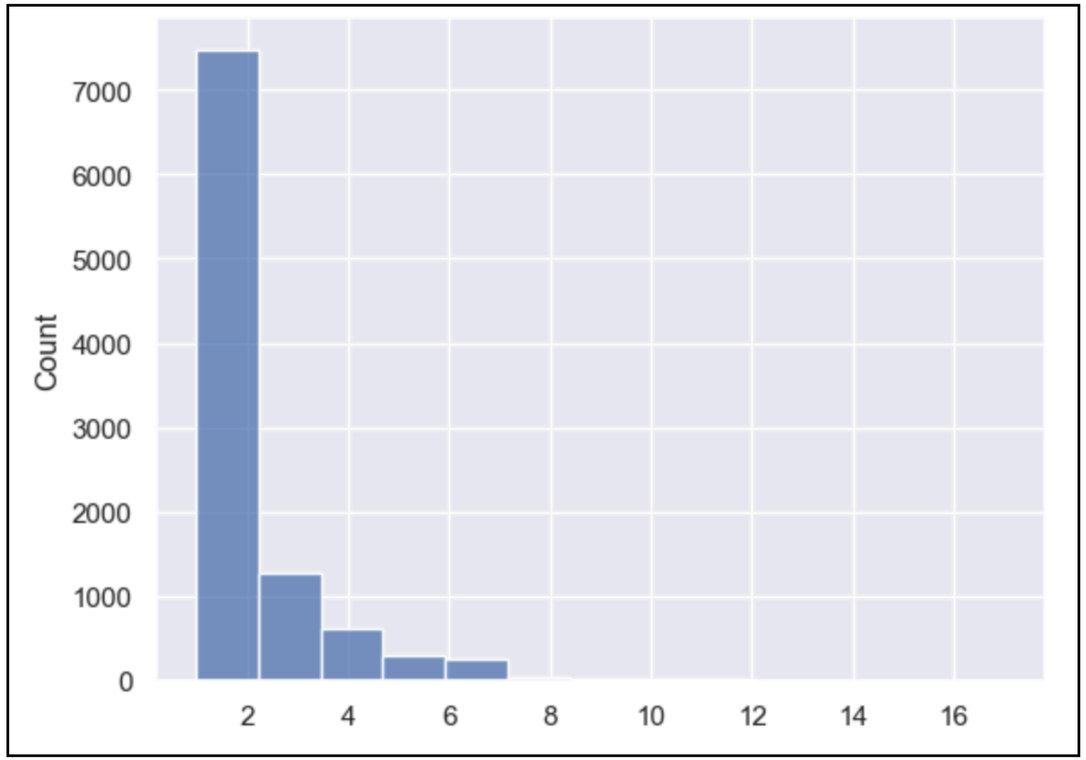
Solutions

Download the Jupyter notebook solution file here
https://treeppl.org/courses/basic-tutorial-solutions.zip

Exercise 1: recursively defined models - a solution

```
%%treeppl geometric samples=10000

model function geometric(p : Real) => Int {
  assume cont ~ Bernoulli(p);
  if cont {
    return 1;
  }
  return 1 + geometric(p);
}
```

```
res = geometric(p = 0.5)
sns.histplot(
    x=res.samples,
    binwidth=1.2
)
```



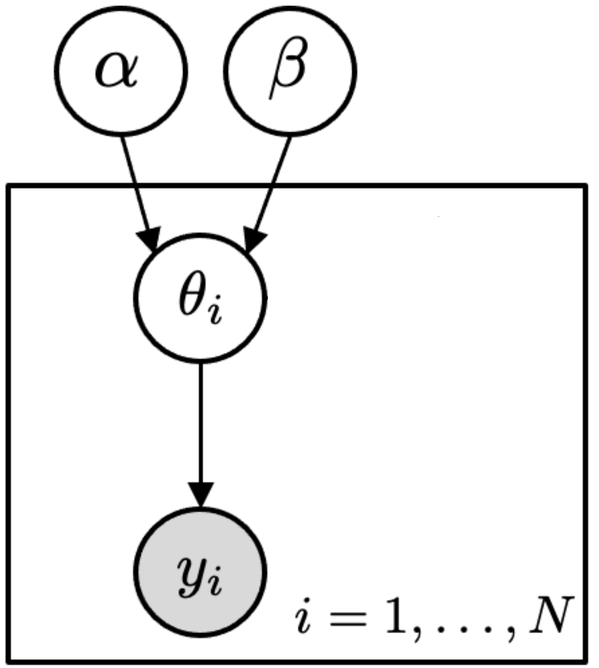
Exercise 2: a graphical model - a solution

```
%%treeppl pump samples=1000000
model function pump(y : Int[], t : Real[]) => Real {
  assume alpha ~ Exponential(1.0);
  assume beta ~ Gamma(0.1, 1.0);
  for i in 1 to (length(y)) {
    assume theta ~ Gamma(alpha, beta);
    observe y[i] ~ Poisson(theta * t[i]);
  }
  return alpha;
}
```

```
res = pump(
    y = [5, 1, 14, 3, 19, 1, 1, 4, 22],
    t = [94.3, 15.7, 62.9, 126, 5.24, 31.4, 1.05, 2.1, 10.5],
)
sns.histplot(
    x=res.samples, stat='density', bins=100, weights=res.nweights
)
```

Pump example (George et al., 1993)

$$\alpha \sim \text{Exponential}(1)$$
$$\beta \sim \text{Gamma}(0.1, 1)$$
$$\theta_i \sim \text{Gamma}(\alpha, \beta)$$
$$y_i \sim \text{Poisson}(\theta_i t_i)$$



Exercise 3: Bayesian linear regression - a solution

Exercise 3: Suppose you have the following data: `length = [55, 57, 52, 64, 53, 64]` and `age = [4, 10, 2, 17, 6, 20]`, where the tuple `(length[i], age[i])` represents the length in cm and age in weeks for a baby. Create a TreePPL script that infers a posterior distribution over the length of six months old babies by using Bayesian linear regression of the form $l_i \sim N(\alpha + \beta a_i, \sigma)$, where N is the normal distribution, l_i the length, a_i the age, and α, β, and σ are random variables.

```
%%treeppl thing samples=400000

type LinRes =
  | LinRes {alpha : Real, beta : Real, gamma : Real}

model function thing(y : Real[], x : Real[]) => LinRes {
  assume alpha ~ Uniform(0.0, 200.0);
  assume beta ~ Uniform(0.0, 10.0);
  assume gamma ~ Uniform(0.0, 100.0);
  for i in 1 to (length(y)) {
    observe y[i] ~ Gaussian(alpha + beta * x[i], gamma);
  }
  return LinRes {alpha = alpha, beta = beta, gamma = gamma};
}
```

```
res = thing(
    y = [55, 57, 52, 64, 53, 64],
    x = [4, 10, 2, 17, 6, 20],
)
sns.histplot(
    x=list(map(lambda x: x.beta, res.samples)), bins=40,
    stat='density', weights=res.nweights
)
```