Bayesian Data Analysis
Modelling a Gamma-Poisson Hierarchical Model

What is the goal of the problem?

You have 10 power plants pumps. For these 10 pumps, you are given the number of failures (x_i) and the number of hours in thousands that the pumps have run for (t_i).

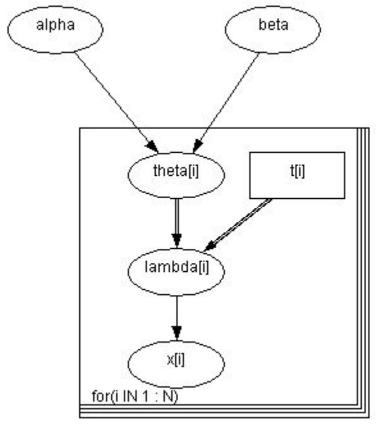
We seek to model the relationship between x_i and t_i using a Gamma-Poisson Hierarchical model as done by George *et al* in their paper *Conjugate Likelihood Distributions* in 1993.

What is the goal of the assignment?

- We will be solving this problem using Metropolis-Hastings algorithm - a popular variant of the Monte-Carlo Markov Chain (MCMC) algorithm, which will be implemented natively using "standard" libraries.
- Our implementation will then be compared against an implementation in PyMC3, a popular probabilistic programming framework.
- The sampling algorithm will be compared against an variational inference algorithm which is optimization based. This has been implemented using PyMC3 as well.

Rigorous Problem Description:

Below is a PLATE model of the aforementioned problem.



The relationship between the random variables is as follows:

- `x[i]` is Poisson-distributed with mean `lambda[i]` for every `i` = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

- `lambda[i]` is a linearly transformed random variable whose parent random variable is `theta[i]`. Mathematically, `lambda[i]` = `theta[i] * t[i]` for every `i` = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.
- `theta[i]` is Gamma-distributed with parameters `alpha` and `beta`.
- `alpha` is Exponential-distributed with mean `1`.
- `beta` is Gamma-distributed with parameters `0.1` and `1`.

Why use MCMC sampling?

- The posterior distribution for `theta[i]` for every `i` = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 is a well-defined Gamma-distribution. In fact:

```
p(theta[i] | x[i], t[i], alpha, beta) =
    Gamma(x[i] + alpha, t[i] + beta)
```

- Similarly, we can show that the posterior distribution for `beta` is also a well-defined Gamma-distribution,
as shown below:

```
p(beta | D, {theta[i]}, alpha) =
Gamma(10 * alpha + 0.1, sum({theta[i]} + 1))
```

- However, the posterior distribution for alpha is not a well-recognized distribution, and hence we would like to simulate the posterior densities.

What is the proposal distribution?

An important question that arises in the Metropolis-Hastings algorithm is about the choice of the proposal distribution. We proceed to choose the proposal distribution to be a Standard Normal Distribution i.e., Normal distribution with mean 0 and variance 1. Technically, our proposal distribution is a Normal distribution centered at the previous iterate, but due to the property of the translation of Normal distribution, we consider a Standard Normal distribution.

A brief description of the algorithm

We use the proposal distribution to give us new samples. The acceptance probability is sampled at random from a Uniform distribution with limits 0 and 1. If the ratio between the posterior density of the new parameter and the posterior density of the old parameter is greater than the acceptance probability, then we accept the new parameter sampled. We keep doing this for a preset number of iterations.

For brevity, we state the unnormalized log posterior densities below:

```
                    p(alpha | {theta[i]}, beta, D) ∝
          10 * (alpha * log(beta) - lgamma(alpha) + alpha *
                       sum({log(theta[i])}) - alpha

                    p(beta | {theta[i]}, alpha, D) ∝
 10 * alpha * log(beta) - beta * sum({theta[i]}) - 0.9 * log(beta) -
                                 beta

                     p(theta[i] | alpha, beta, D) ∝
   (x[i] + alpha - 1) * log(theta[i]) - theta[idx] * (t[i] + beta)
```

Results from this implementation

We will first show statistics of the accepted samples. After 20000 iterations (5000 iterations for burn-in and 15000 iterations for sampling), we have:

```
            mean       std
alpha   0.766384  0.304020
beta    1.076379  0.585128
theta0  0.064024  0.031248
theta1  0.125794  0.096030
theta2  0.095867  0.042908
theta3  0.119648  0.035979
theta4  0.658483  0.355683
theta5  0.623211  0.158055
theta6  1.016109  0.758077
theta7  1.004133  0.724206
theta8  1.664692  0.803794
theta9  2.001735  0.449913
```

The predictions are as follows:
```
6.0, 2.0, 6.0, 15.0, 3.0, 20.0, 1.0, 1.0, 3.0, 21.0
```

The ground-truth values are:
```
5.0, 1.0, 5.0, 14.0, 3.0, 19.0, 1.0, 1.0, 4.0, 22.0
```

The number of iterations per second (on average): 4055 iterations per second, thereby taking 5 seconds to complete.

Comparison with PyMC3 MCMC implementation

We will show the statistics for the accepted samples. After 15000 iterations, we have:

```
            mean       std
alpha      0.707511  0.262961
beta       0.944160  0.526700
theta0     0.059822  0.024620
theta1     0.101594  0.077252
theta2     0.089694  0.037962
theta3     0.115618  0.030451
theta4     0.601133  0.316245
theta5     0.606061  0.136341
theta6     0.877168  0.695554
theta7     0.903454  0.737731
theta8     1.567180  0.752644
theta9     1.983739  0.425361
```

The statistics are close as compared to the native implementation. The predictions are as follows:

```
5.0, 1.0, 6.0, 15.0, 3.0, 19.0, 1.0, 1.0, 3.0, 20.0
```

The ground-truth values are:

```
5.0, 1.0, 5.0, 14.0, 3.0, 19.0, 1.0, 1.0, 4.0, 22.0
```

The number of iterations per second (on average): 705 draws per second, thereby taking 20 seconds to complete, which tells us that our implementation is **at least 5 times faster**.

Comparison with the PyMC3 Variational Inference implementation

Variational Inference is essentially a minimization algorithm. In PyMC3, Variational Inference is done by considering the mean field approximation of the parameters whose posterior is required to be found.

After 20000 iterations, the predictions are as follows:

```
6.0, 2.0, 6.0, 15.0, 3.0, 19.0, 1.0, 1.0, 4.0, 21.0
```

Epilogue

The code for the experiments are available in the following files:

1. `numpy_mcmc.py`: Performs MCMC using NumPy - this is the native implementation
2. `pymc3_mcmc.py`: Performs MCMC using PyMC3
3. `pymc3_vi.py`: Performs Variational Inference using PyMC3