Week 5: Bayesian neural networks

Introduction

This week, we will review Bayesian inference and Bayesian neural networks taking into account MCMC methods and probability distributions. We will cover Bayesian logistic regression and Bayesian neural networks where we will use MCMC methods. We note that your textbook does not feature the lesson for this week and hence we have to rely on other materials for further information.

Additional Reading material

1. Andrieu, C., De Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine learning*, *50*(1-2), 5-43. https://www.cs.ubc.ca/~arnaud/andrieu_defreitas_doucet_jordan_intromontecarlomachinelearning.pdf
2. http://www.columbia.edu/~mh2078/MachineLearningORFE/MCMC_Bayes.pdf
3. Welling, M., & Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 681-688). http://people.ee.duke.edu/~lcarin/398_icmlpaper.pdf
4. Neal, R. M. (2011). MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, *2*(11), 2. https://arxiv.org/pdf/1206.1901.pdf%20http://arxiv.org/abs/1206.1901.pdf

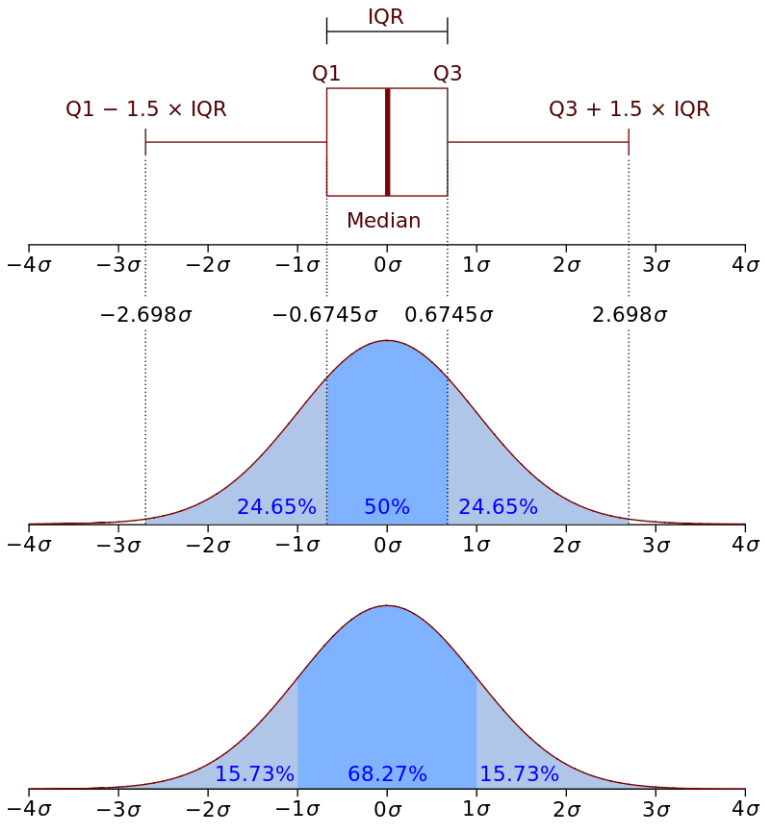
Programming support

1. https://www.r-tutor.com/elementary-statistics/probability-distributions
2. https://web.stanford.edu/class/archive/cs/cs109/cs109.1198/handouts/pythonForProbability.html

Probability distribution

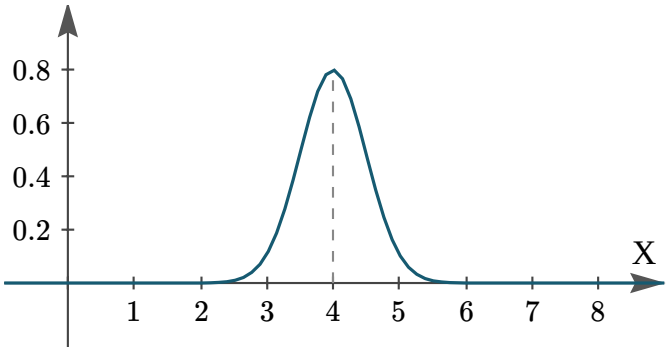
Gaussian (Normal) Distribution

A normal probability density or distribution can be visualised as follows where Q1, Q2 and Q3 refer to respective quartiles and IQR refers to the inter-quantile range.

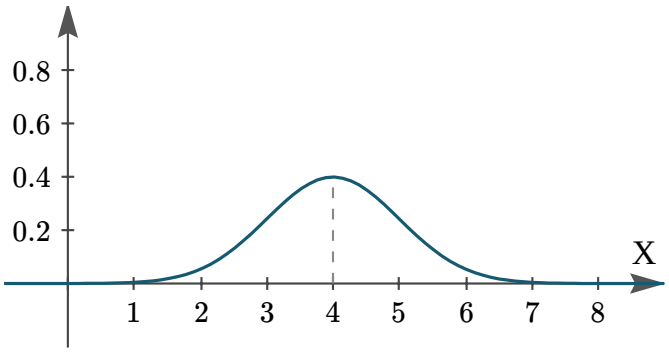


Source: https://en.wikipedia.org/wiki/Probability_density_function

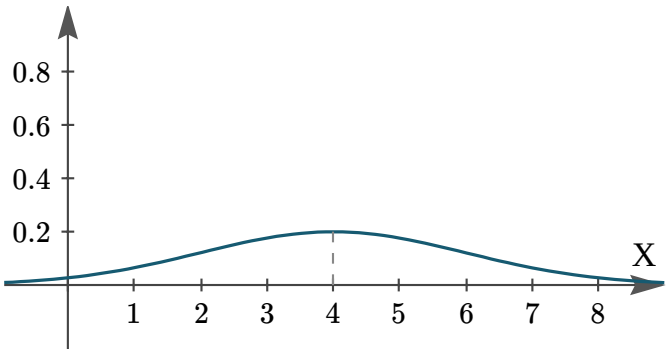
Let us visualise what happens when standard deviation (std) changes and mean remains the same for the a distribution.



mean = 4, std = 0.5



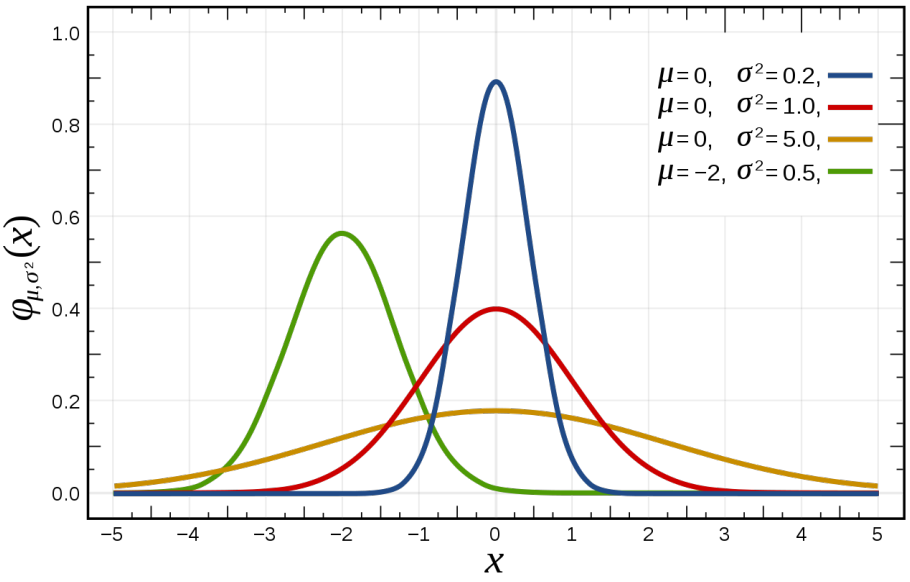
mean = 4, std = 1



mean = 4, std = 2

Source:https://www.intmath.com/counting-probability/11-probability-distributions-concepts.php

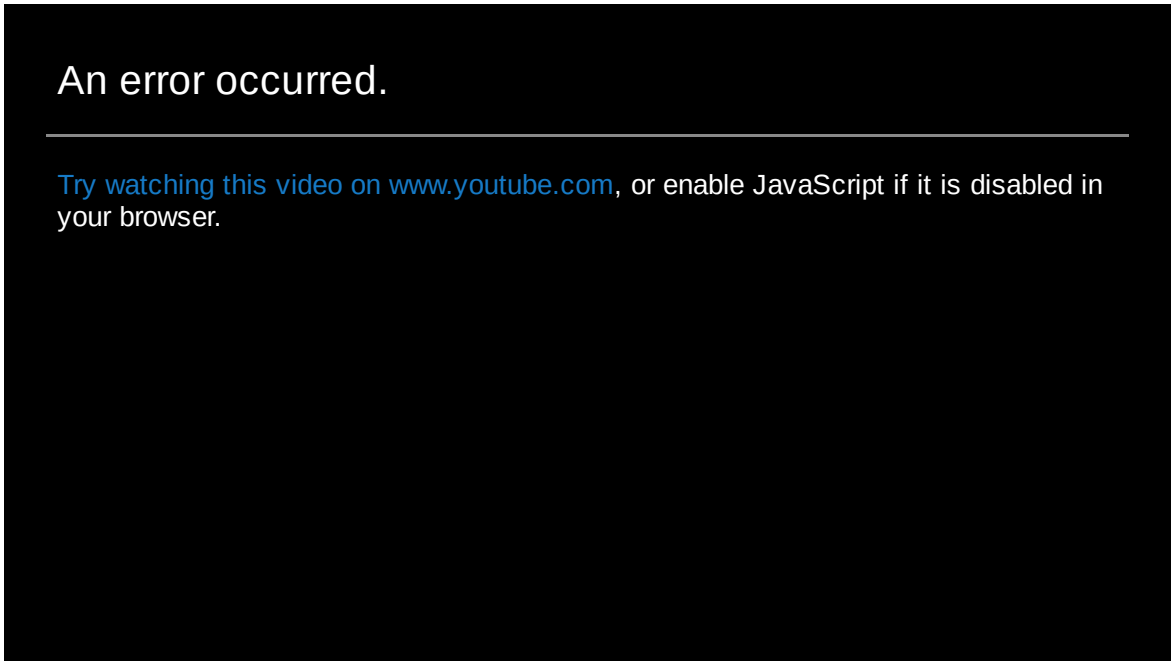
We see some more examples where changes to the mean and standard deviation gives us different shapes of the probability density function (PDF).



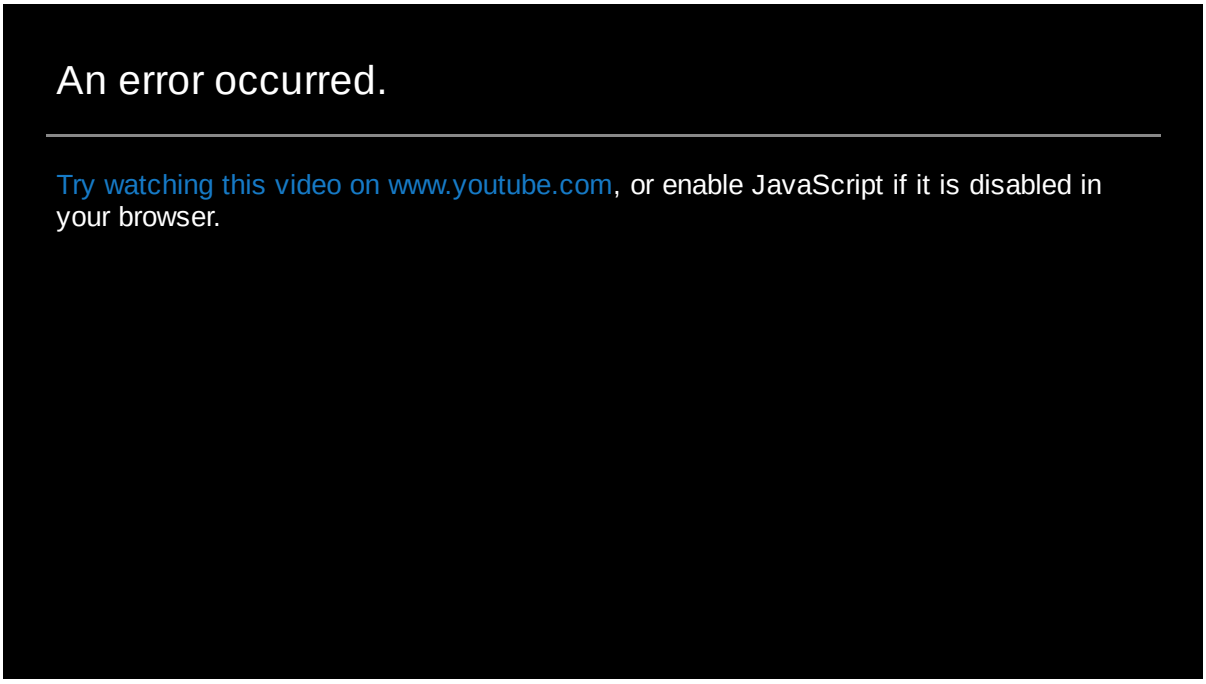
The equation for Gaussian of Normal PDF taking mean μ and standard deviation σ is given below:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

We note that σ^2 is the variance. Source: https://en.wikipedia.org/wiki/Normal_distribution



Note that probability and likelihood are not the same in the field of statistics, while in everyday language they are used as if they are the same. See the video below:



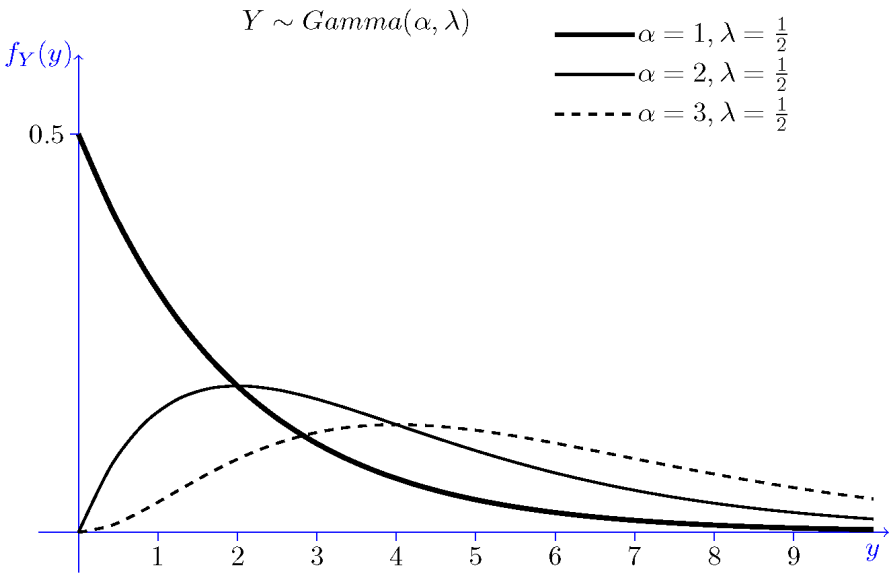
Gamma distribution

A continuous random variable x is said to have a *gamma* distribution with parameters α and β as shown below.

$$f(x;\alpha,\beta) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}$$

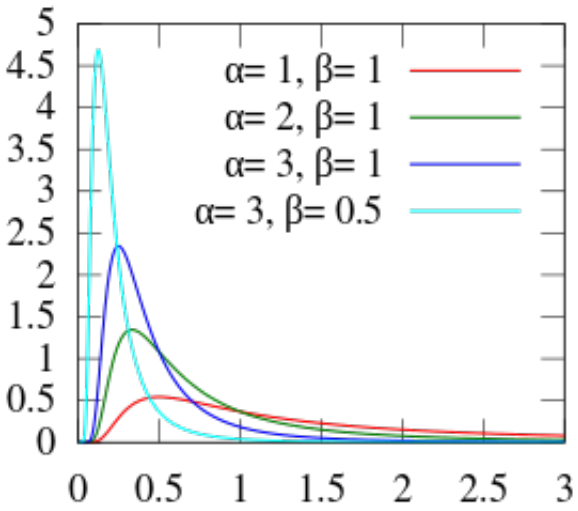
for $x > 0 \quad \alpha, \beta > 0$

where $\Gamma(n) = (n-1)!$



Note in figure above β is λ

Image source: https://en.wikipedia.org/wiki/Gamma_distribution



The **inverse-Gamma** distribution is given above: Source: https://en.wikipedia.org/wiki/Inverse-gamma_distribution

We review random number generation using Python:

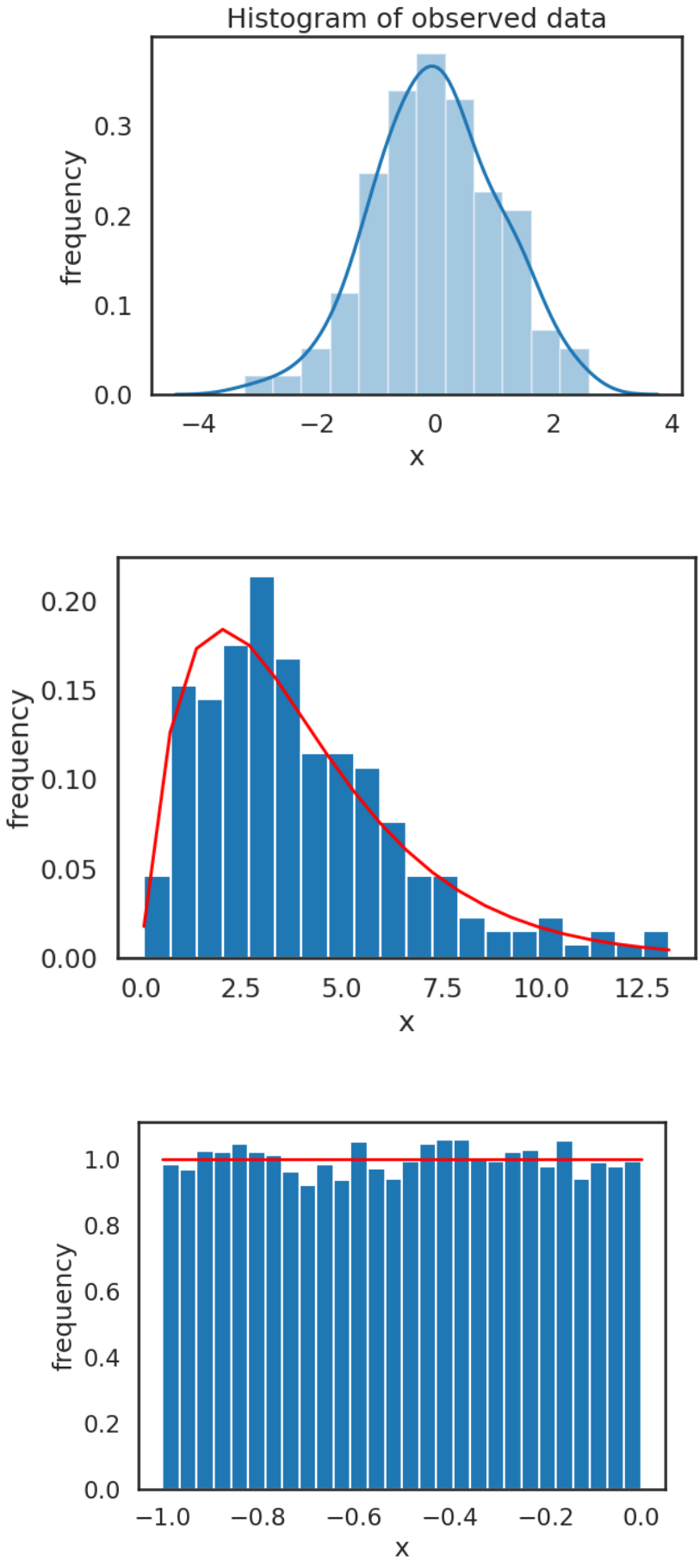
Run PYTHON

```python
from numpy import random
#https://www.datacamp.com/community/tutorials/numpy-random
#https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.rando
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.norm.h
x = random.randint(100)
print(x, ' random.randint(100) ')

x = random.rand()
print(x, ' random.rand() ')

x=random.randint(10, size=(4))
print(x, ' x=random.randint(10, size=(4))')

x = random.randint(100, size=(3, 5))
```

Next, we use Seaborn and Matplotlib python library:

Run PYTHON

```python
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.stats import norm

sns.set_style('white')
sns.set_context('talk')

np.random.seed(123)

data = np.random.randn(200)
```



The above figure shows output given by Normal (top), Gamma(middle), and Uniform (bottom) distribution.

Now some examples in R

Run R

```r
#Source: https://www.cyclismo.org/tutorial/R/probability.html
dnorm(0)
dnorm(0,mean=4)
dnorm(0,mean=4,sd=10)

v <- c(0,1,2)
dnorm(v)

x <- seq(-2,2,by=.1)
print(x)
y <- dnorm(x)
plot(x,y)
```

Multivariate Normal distribution

The multivariate normal distribution or joint normal distribution generalises univariate normal distribution to more variables or higher dimensions.

$$f_{\mathbf{X}}(x_1,\ldots,x_k) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k|\boldsymbol{\Sigma}|}}$$

where x is a real "k"-dimensional column vector and |Σ| is the determinant of symmetric covariance matrix Σ which is positive definite. Multivariate normal distribution reducves to univariate normal distribution if Σ is a single real number.

Bivariate case

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}}\exp\left(-\frac{1}{2(1-\rho^2)}\left[\frac{(x-\mu_X)^2}{\sigma_X^2}+\frac{(y-\mu_Y)^2}{\sigma_Y^2}-\frac{2\rho(x-\mu_X)(y-\mu_Y)}{\sigma_X\sigma_Y}\right.\right.$$

where

ρ is the correlation between X and Y, given $\sigma_X > 0$ and $\sigma_Y > 0$.

$$\boldsymbol{\mu} = \begin{pmatrix}\mu_X\\ \mu_Y\end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix}\sigma_X^2 & \rho\sigma_X\sigma_Y\\ \rho\sigma_X\sigma_Y & \sigma_Y^2\end{pmatrix}.$$

Source: https://en.wikipedia.org/wiki/Multivariate_normal_distribution

Note that the covariance matrix gives the covariance between each pair of elements, where the diagonal represents the variance, which gives the covariance of each element with itself.

PYTHON

```python
#Implementation: https://peterroelants.github.io/posts/multivariate-norm
def multivariate_normal(x, d, mean, covariance):
    """pdf of the multivariate normal distribution."""
    x_m = x - mean
    return (1. / (np.sqrt((2 * np.pi)**d * np.linalg.det(covariance))) *
            np.exp(-(np.linalg.solve(covariance, x_m).T.dot(x_m)) / 2))
```

▸ Run

PYTHON

```python
#https://stackoverflow.com/questions/11615664/multivariate-normal-densit
from numpy import *
import math
# covariance matrix
sigma = matrix([[2.3, 0, 0, 0],
           [0, 1.5, 0, 0],
           [0, 0, 1.7, 0],
           [0, 0,   0, 2]
          ])
# mean vector
mu = array([2,3,8,10])

# input
x = array([2.1,3.5,8, 9.5])
```

```r

install.packages("MASS")                                  # Install MASS packa
library("MASS")                                           # Load MASS package
#https://statisticsglobe.com/bivariate-multivariate-normal-distribution-


my_n2 <- 1000                                             # Specify sample siz
my_mu2 <- c(5, 2, 8)                                      # Specify the means
my_Sigma2 <- matrix(c(10, 5, 2, 3, 7, 1, 1, 8, 3),        # Specify the covari
                    ncol = 3)
mvrnorm(n = my_n2, mu = my_mu2, Sigma = my_Sigma2)        # Random sample from


```

Bernoulli distribution

Bernoulli distribution is a discrete probability distribution which takes the value 1 with probability p and the value 0 with probability $q = 1 - p$ which can be used for modelling binary classification problems. Hence, the probability mass function for this distribution over k possible outcomes is given by

$$f(k;p) = p^k(1-p)^{1-k} \quad \text{for } k \in 0, 1$$

Binomial distribution

The probability of getting exactly "k" successes in "n" independent Bernoulli trials is given by

$$f(k,n,p) = \Pr(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for $k = 0, 1, 2, ..., n$, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Run PYTHON

```python
from scipy.stats import binom
# setting the values
# of n and p
n = 6
p = 0.6
# defining the list of r values
r_values = list(range(n + 1))
print(r_values, ' r_values')
# obtaining the mean and variance
mean, var = binom.stats(n, p)
# list of pmf values
dist = [binom.pmf(r, n, p) for r in r_values ]
# printing the table
print("r\tp(r)")
```

Multinomial distribution

We consider an experiment of extracting *n* balls of *k* different colours from a bag and replacing the extracted ball after each draw. The balls of the same colour are equivalent. The number of extracted balls of colour *i* (*i* = 1, ..., *k*) as X_i, and denote as p_i the probability that a given extraction will be in color i.

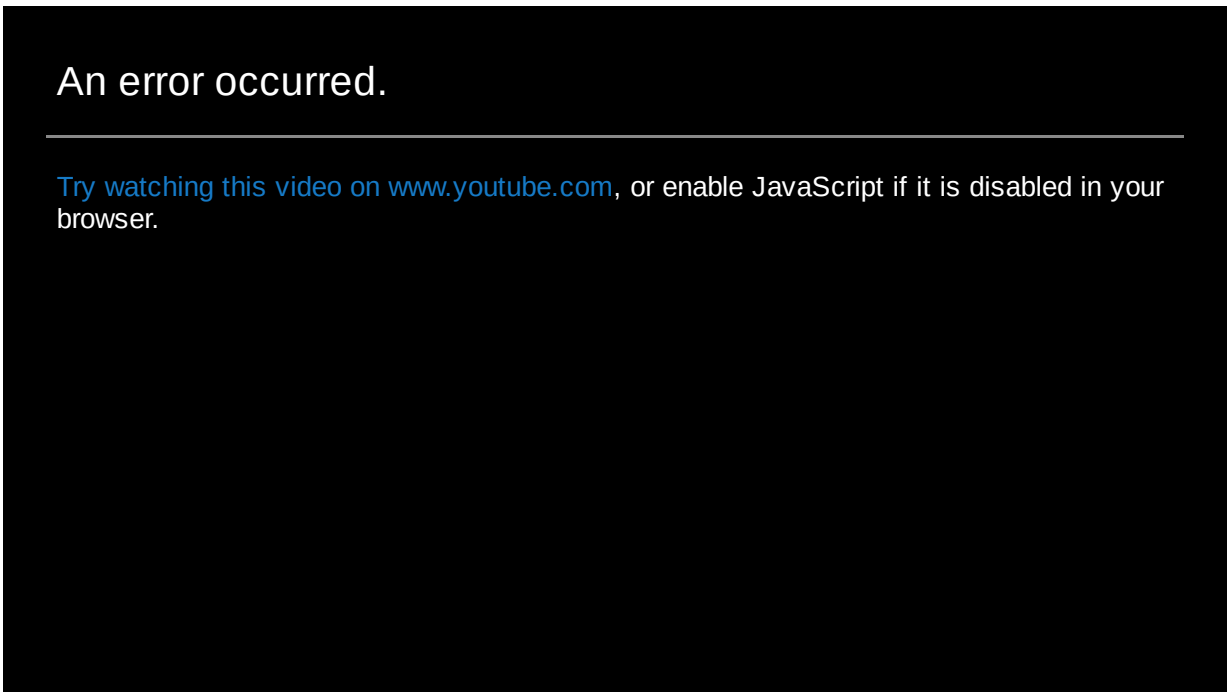
$$f(x_1,\ldots,x_k;n,p_1,\ldots,p_k) = \Pr(X_1 = x_1 \text{ and } \ldots \text{ and } X_k = x_k)$$
$$= \begin{cases} \dfrac{n!}{x_1!\cdots x_k!}p_1^{x_1}\times\cdots\times p_k^{x_k}, & \text{when } \sum_{i=1}^{k} x_i = n \\ 0 & \text{otherwise,} \end{cases}$$

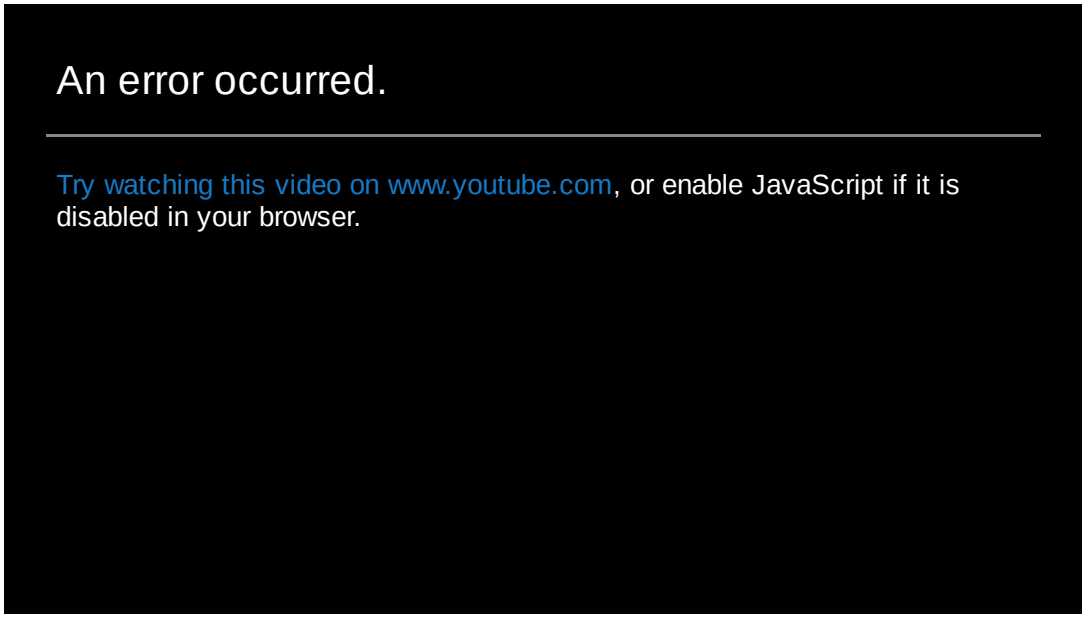
Source: https://en.wikipedia.org/wiki/Multinomial_distribution

More info: https://stattrek.com/probability-distributions/multinomial.aspx

Run PYTHON

```python
#https://numpy.org/doc/stable/reference/random/generated/numpy.random.mu

import numpy as np

draw = np.random.multinomial(20, [1/6.]*6, size=1)
print(draw, ' first draw')
#array([[4, 1, 7, 5, 2, 1]]) # random
#It landed 4 times on 1, once on 2, etc.

#Now, throw the dice 20 times, and 20 times again:
draw = np.random.multinomial(20, [1/6.]*6, size=2)
print(draw, ' second  draw')
#array([[3, 4, 3, 3, 4, 3], # random
     #  [2, 4, 3, 4, 0, 7]])
```





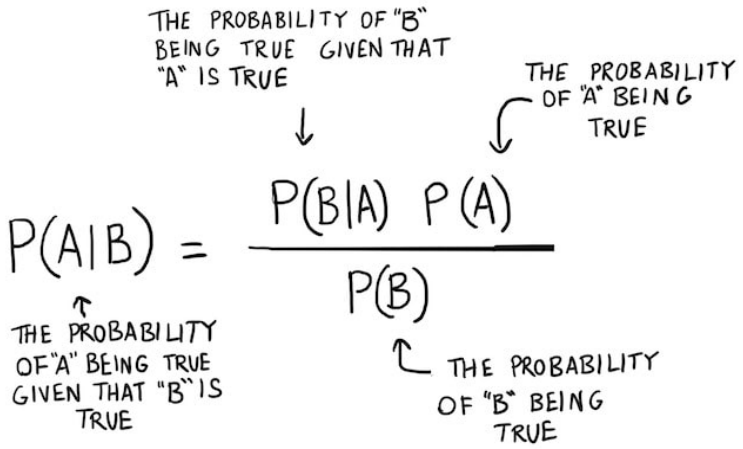
More info: http://users.umiacs.umd.edu/~jbg/teaching/INST_414/04c.pdf

Bayesian inference

Bayesian methods account for the uncertainty in prediction and decision making via the posterior distribution. Note that the posterior is the conditional probability determined after taking into account the prior distribution and the relevant evidence or data via sampling methods.

Bayesian methods can account for the uncertainty in parameters (weights) and topology by marginalisation over the predictive posterior distribution.

Hence, as opposed to conventional neural networks, Bayesian neural learning use probability distributions to represent the weights



Thomas Bayes is the guy behind Bayes' theorem which is the foundation of Bayesian inference.

Image Source: https://en.wikipedia.org/wiki/Thomas_Bayes

Lets review it again

Conditional Probability

- The probability of A given B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \text{ if } P(B) > 0$$

- Multiplication rule:

$$P(A \cap B) = P(A)P(B|A)$$

- Bayes law:

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

Key probability rules are given here: http://www.milefoot.com/math/stat/prob-rules.htm

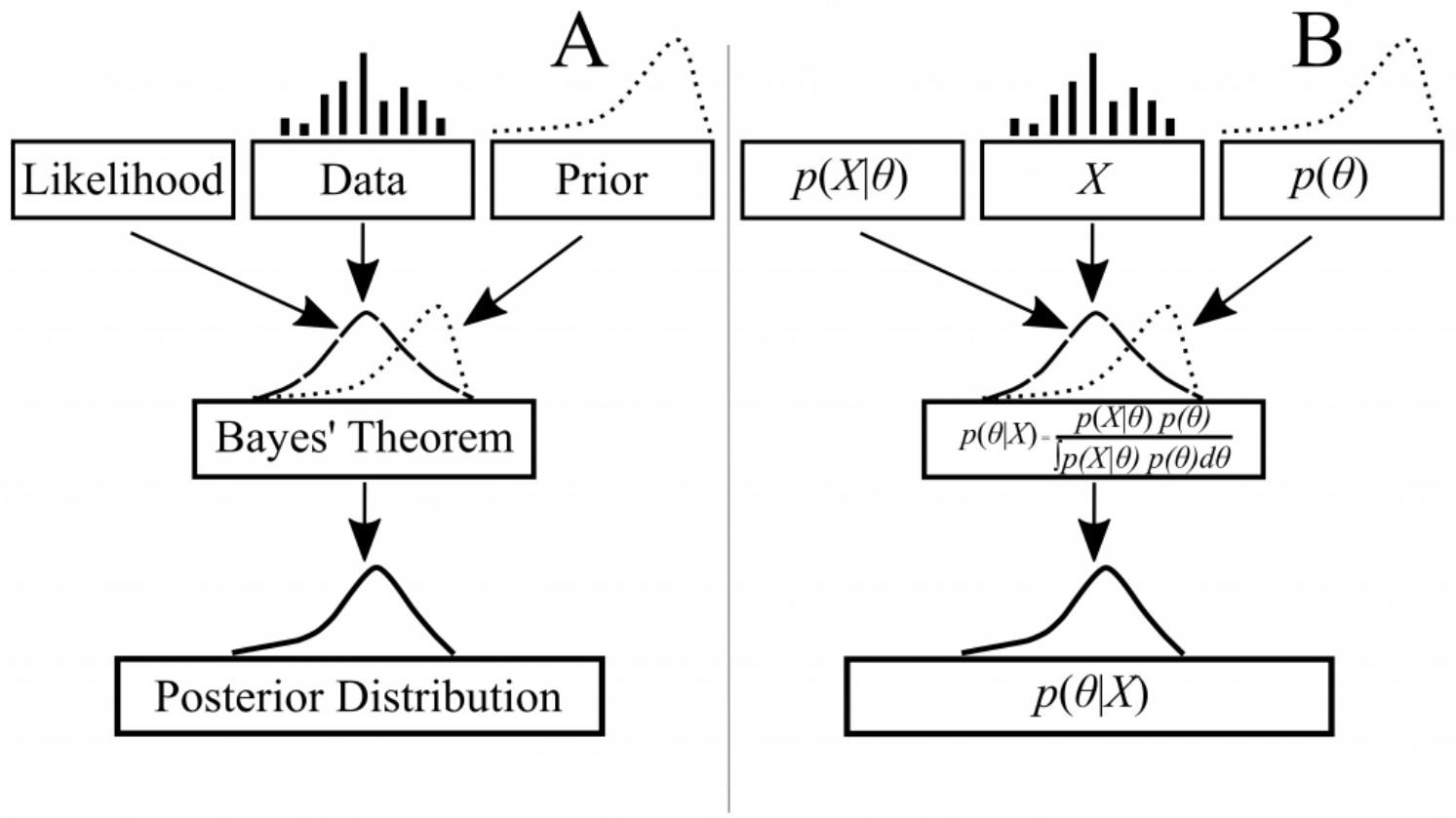


Figure Source: TBA

The figure above gives an overview of the Bayesian inference framework that uses data with prior and likelihood to construct or sample from the posterior distribution. This is the building block of the rest of the lessons that will feature Bayesian logistic regression and Bayesian neural networks.

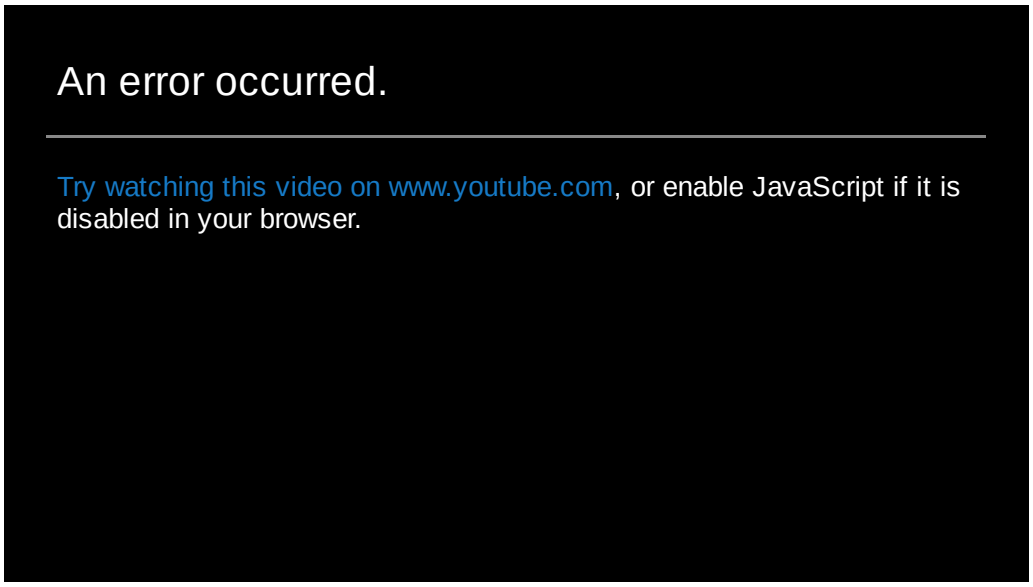
Essentially, Bayesian inference refers to a principled way of estimating unknown variables using prior information or belief about the variable. Prior information is captured in the form of a distribution. A simple example of a prior belief is a distribution that has a positive real-valued number in some range, this essentially would imply that our result or posterior distribution would likely be a distribution of positive numbers in some range which would be similar to the prior but not the same.

If the posterior and prior are both same, this is known as conjugate priors and if the prior is helpful, it is known as informative prior.

Basics summed up in the video below:

Note that in the above videos, the software stata was used but we will use Python/R in this course.

Here is a nice overview of the history of the field.



Further information

1. Bayesian vs Frequentist: https://stats.stackexchange.com/questions/22/bayesian-and-frequentist-reasoning-in-plain-english
2. Bayesian vs Frequentist video: https://www.youtube.com/watch?v=meivbbfHmK0

MCMC sampling

Bayesian inference

The need for efficient sampling methods to implement Bayesian inference has been the focus of research in computational statistics, especially for the case of multi-modal and irregular posterior distributions. Bayesian inference is typically implemented by **Markov Chain Monte Carlo (MCMC)** sampling methods which are used to update the probability for a hypothesis (proposal Θ) as more information becomes available. The hypothesis is given by a prior probability distribution that expresses one's belief about a quantity (or free parameter in a model) before some data is taken into account. MCMC methods enable to samples from a distribution iteratively using **proposal distribution**, **prior distribution** $P(\Theta)$ and a **likelihood function** to construct the **posterior distribution** $P(\Theta|data)$.

$$P(\mathbf{\Theta}|data) = \frac{P(data|\mathbf{\Theta}) \times P(\mathbf{\Theta})}{P(data)}.$$

We note that $P(data|\Theta)$ could be seen as the **likelihood distribution** in disguise. $P(data)$ *is the* marginal distribution of the data and is often seen as a normalising constant and ignored. Hence, ignoring it, we can also express the above in this way

$$P(\mathbf{\Theta}|data) \propto P(data|\mathbf{\Theta}) \times P(\mathbf{\Theta})$$

The likelihood function is a function of the parameters of a given model provided specific observed data. The likelihood function can be seen as a fitness measure of the proposals which are drawn from the proposal distribution.

The posterior distribution is constructed after taking into account the relevant evidence (data) and prior distribution with the likelihood that considers the proposal and the model. MCMC methods essentially implement Bayesian inference via a numerical approach that marginalize or integrate over the posterior distribution.

MCMC sampling

MCMC methods have seen much success in many applications, such as machine learning, astrophysics, geoscientific inversions, Earth and environmental sciences, and any application that uses some form of model over data.

We note that a Markov process is uniquely defined by its transition probabilities $P(x' \mid x)$ which defines the probability of transitioning from any given state x to other given state x'. The Markov process has a unique stationary distribution $\pi(x)$ given the following two conditions are met.

- There must be the existence of stationary distribution given by the sufficient detailed balance condition that requires that each transition $x \rightarrow x'$ is reversible. This implies that for every pair of states x, x', the probability of being in state x and moving to state x' must be equal to the probability of being in state x' and moving to state x, hence,

$$\pi(x)P(x' \mid x) = \pi(x')P(x \mid x').$$

  *More information: http://prob140.org/sp17/textbook/ch14/Detailed_Balance.html

- The stationary distribution must be unique which is guaranteed by ergodicity of the Markov process. Ergodicity is guaranteed when every state is aperiodic where the system does not return to the same state at fixed intervals, and when every state is positive recurrent where the expected number of steps for returning to the same state is finite. In other words, an ergodic system is one that mixes well, i.e. you get the same result whether you average its values over time or over space.

*More information: Grazzini, J., 2012. Analysis of the emergent properties: Stationarity and ergodicity. *Journal of Artificial Societies and Social Simulation*, *15*(2), p.7. http://jasss.soc.surrey.ac.uk/15/2/7.html

Given that $\pi(x)$ is chosen to be $P(x)$, the condition of detailed balance becomes

$$P(x' \mid x)P(x) = P(x \mid x')P(x')$$

which is re-written as

$$\frac{P(x'|x)}{P(x|x')} = \frac{P(x')}{P(x)}$$

Here is a basic MCMC algorithm that samples till max_samples is reached for training data, $\mathbf{D}$.

for $i=1 **until** max_samples

1. Propose a value $x'|x_i \sim q(x_i)$, where $q(.)$ is the proposal distribution.
2. Given x', execute model $f(x', \mathbf{D})$ and compute the predictions (output y) and the log-liklihood
3. Calculate the acceptance probability
   $\alpha = \min\left(1, \frac{P(x')}{P(x_i)}\frac{q(x_i|x')}{q(x'|x_i)}\right)$

4. Generate from a uniform distribution $u \sim U(0, 1)$

if $\alpha < u$

accept by setting $x_i = x'$

else

reject by setting $x_i = x_{i-1}$

The algorithm above proceeds by proposing new values of the parameter (Step 1) from the selected proposal distribution, which is random-walk (multivariate) normal distribution $q(.)$ with user-defined mean (generally 0) and the step-size (standard deviation) ϕ or covariance matrix Σ. Conditional on these proposed values, the model $f(x', \mathbf{D})$ computes or predicts an output using proposal x' and data $\mathbf{D}$ (Step 2).

Using the predictions, the likelihood is computed, and then the Metropolis-Hasting criterion is used for determining whether to accept or reject the proposal (Step 3). If the proposal is accepted, the chain moves to this proposed value. If rejected, the chain stays at the current value (Step 4). The process is repeated until the convergence criterion is met, which in this case is the maximum number of samples (max_samples) defined by the user. We note that the proposal distribution can use gradients if available but the acceptance criterion will slightly change.

Below is a framework that gives an overview of MCMC for a simple data-driven model such as neural network or logistic regression.

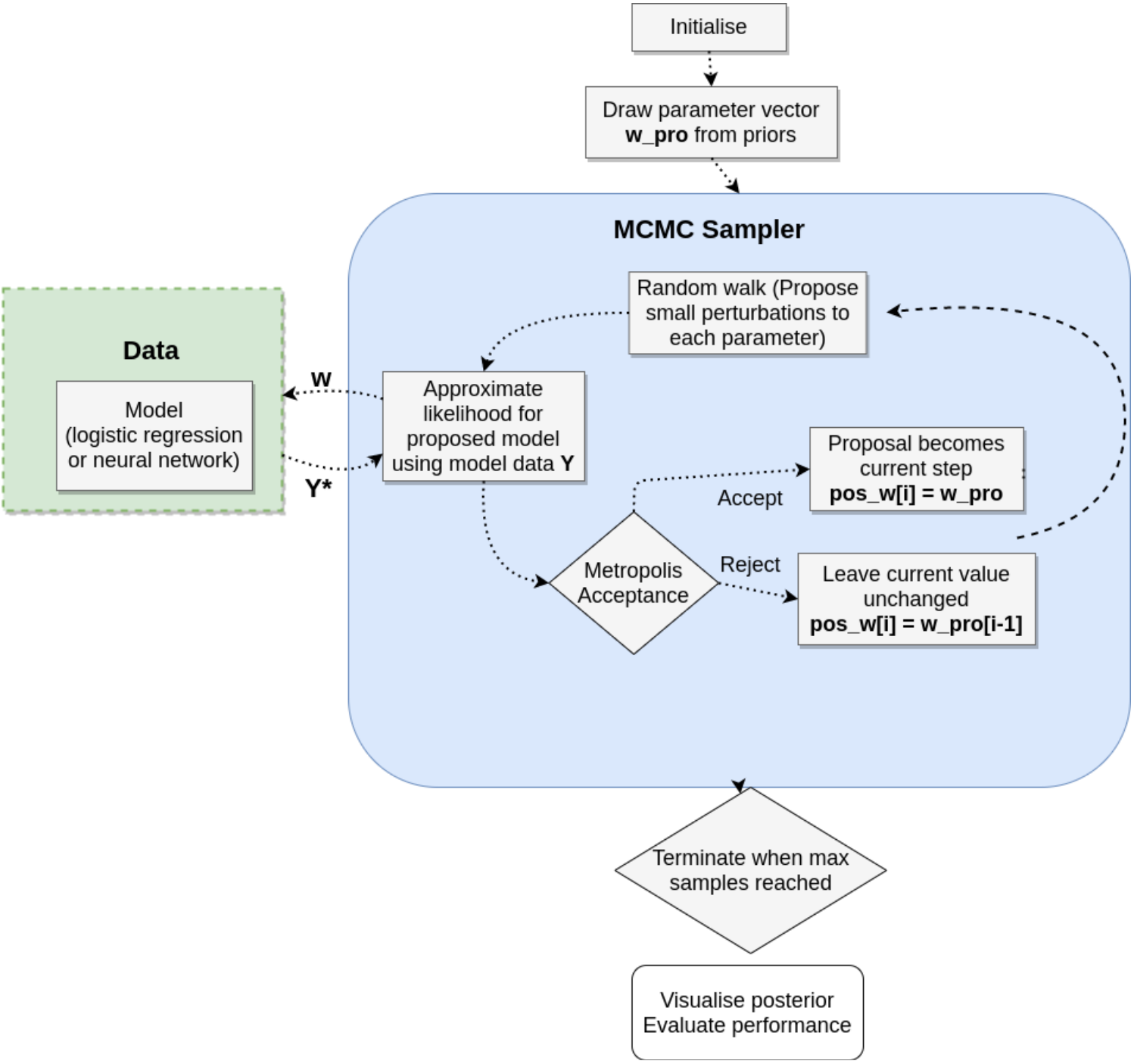
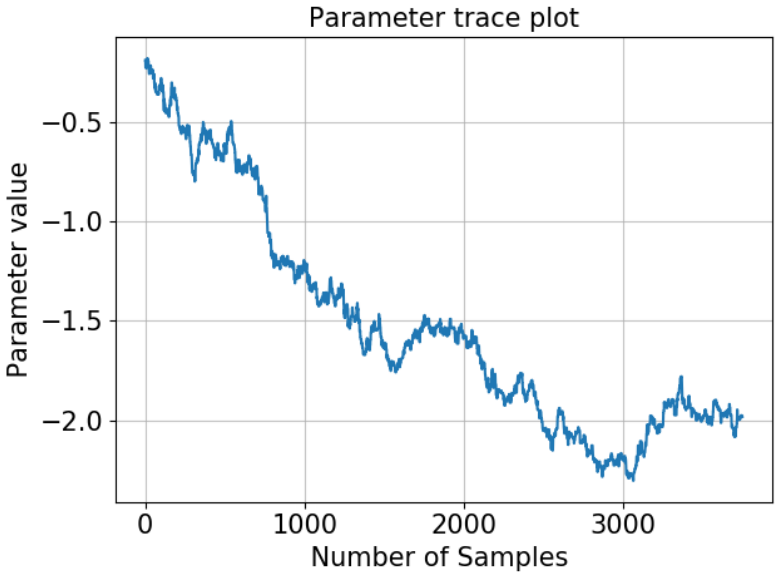
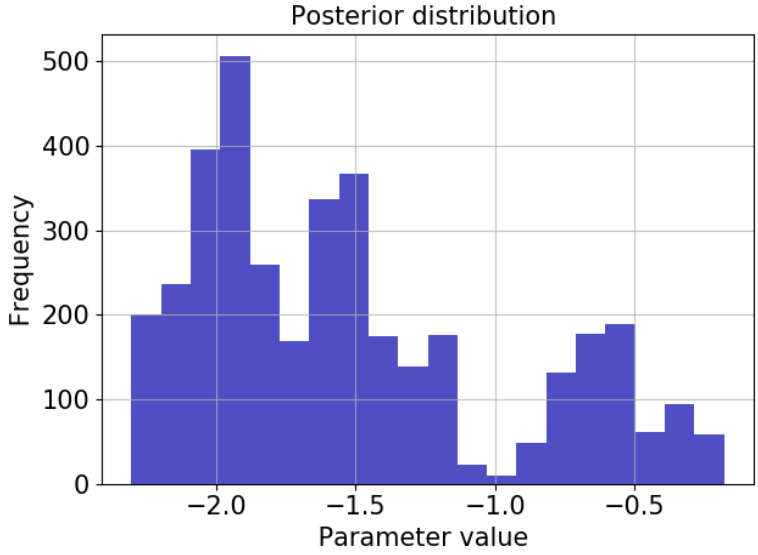
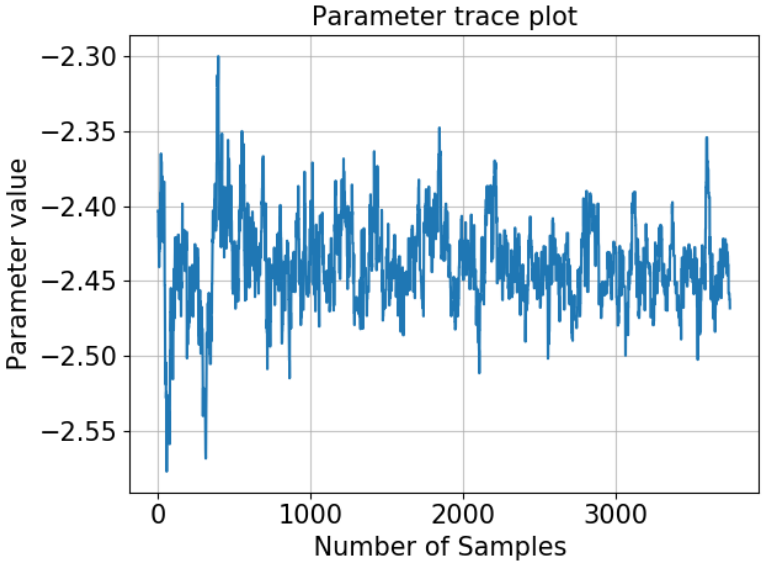


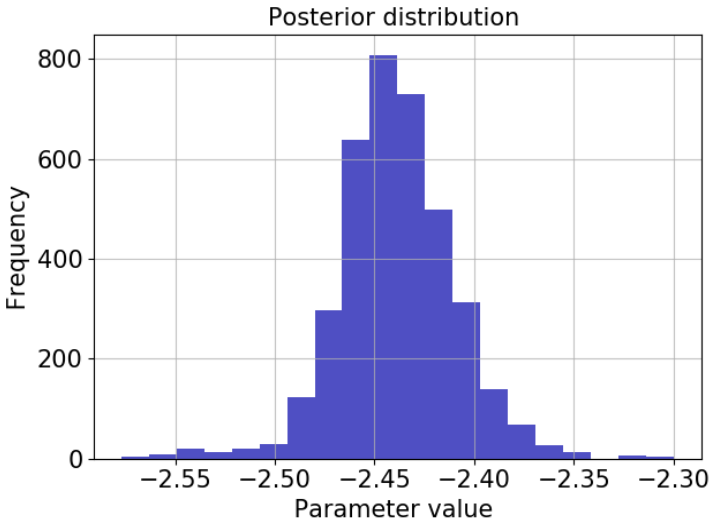
Figure Source: Edited by R. Chandra based on: Chandra R; Azam D; Müller RD; Salles T; Cripps S, 2019, 'Bayeslands: A Bayesian inference approach for parameter uncertainty quantification in Badlands', *Computers and Geosciences*, vol. 131, pp. 89 - 101, http://dx.doi.org/10.1016/j.cageo.2019.06.012

Below is an example of trace-plot and posterior for two selected variables in Bayesian logistic regression with MCMC.


Parameter trace plot
Parameter value
−0.5
−1.0
−1.5
−2.0
0
1000
2000
3000
Number of Samples


Posterior distribution
Frequency
500
400
300
200
100
0
−2.0
−1.5
−1.0
−0.5
Parameter value


Parameter trace plot
Parameter value
−2.30
−2.35
−2.40
−2.45
−2.50
−2.55
0
1000
2000
3000
Number of Samples



Below are some videos that can shed more light on MCMC sampling. Note they may not show results using R or Python, but in the coming lessons, we will get into more details with them.

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

An error occurred.

Try watching this video on www.youtube.com, or enable JavaScript if it is disabled in your browser.

Additional notes about MCMC is here:

1. http://phylo.bio.ku.edu/slides/BayesianMCMC-2013.pdf
2. http://www.southampton.ac.uk/~sks/utrecht/mcmc.pdf
3. https://jellis18.github.io/post/2018-01-02-mcmc-part1/

Additional resources:

Video on Erodicity: https://www.youtube.com/watch?v=1Vxe3LBykRI

Video on Detailed balance: https://www.youtube.com/watch?v=Bg7gaJzzPN0

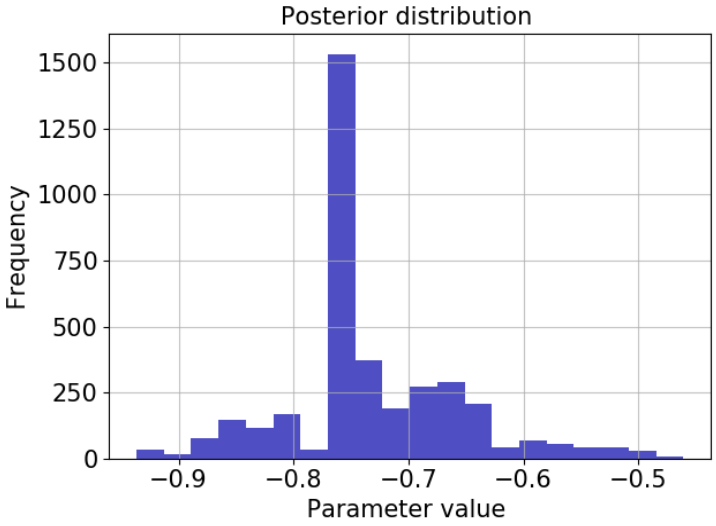
Bayesian Logistic Regression

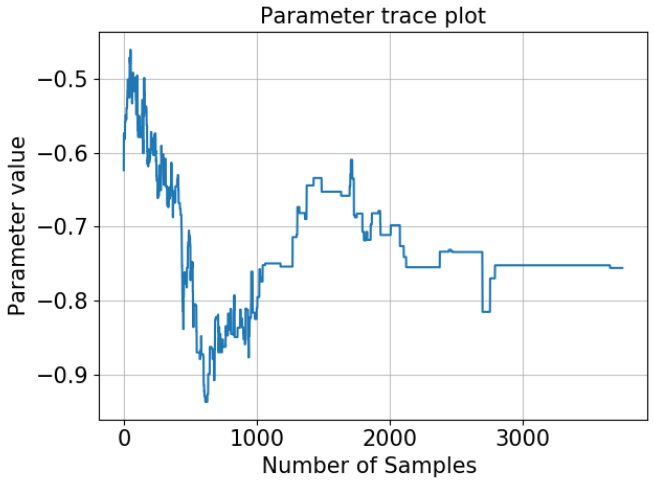
Bayesian logistic regression for a single step ahead (eg. Sunspot time series). After running the code, you can see trace-plot and histogram of the posterior distribution.

Bayesian logistic regression - multiple outputs

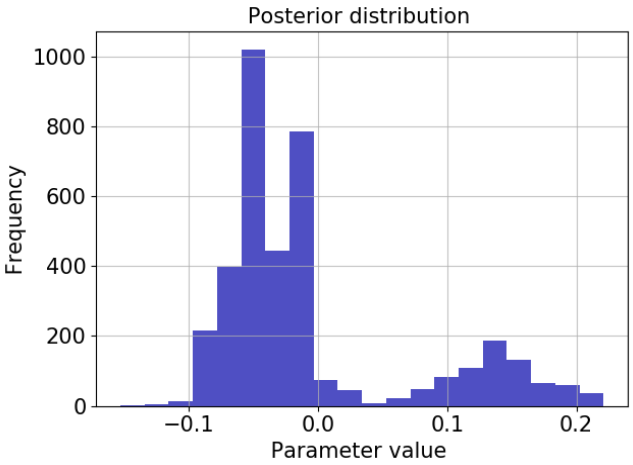
Bayesian logistic regression for a single step ahead (Sunspot time series) and multi-step ahead time series prediction (MMM stock market). Below you can see trace-plot and histogram of the posterior distribution.

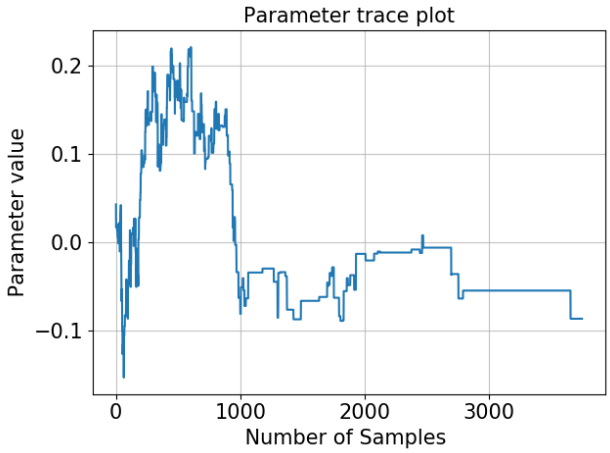
We note that in multi-step time series prediction, 5 step-ahead would mean 5 output neurons. We use sigmoid units in the output layer. Note that gradients are not used and you can compare the results with SGD which is present in the code.





Another selected parameter from the model shown below.





Other posterior visuals: https://github.com/rohitash-chandra/Bayesianlogisticreg_multioutputs/tree/master/posterior

You can execute the code and uncomment some of the print statements to understand.

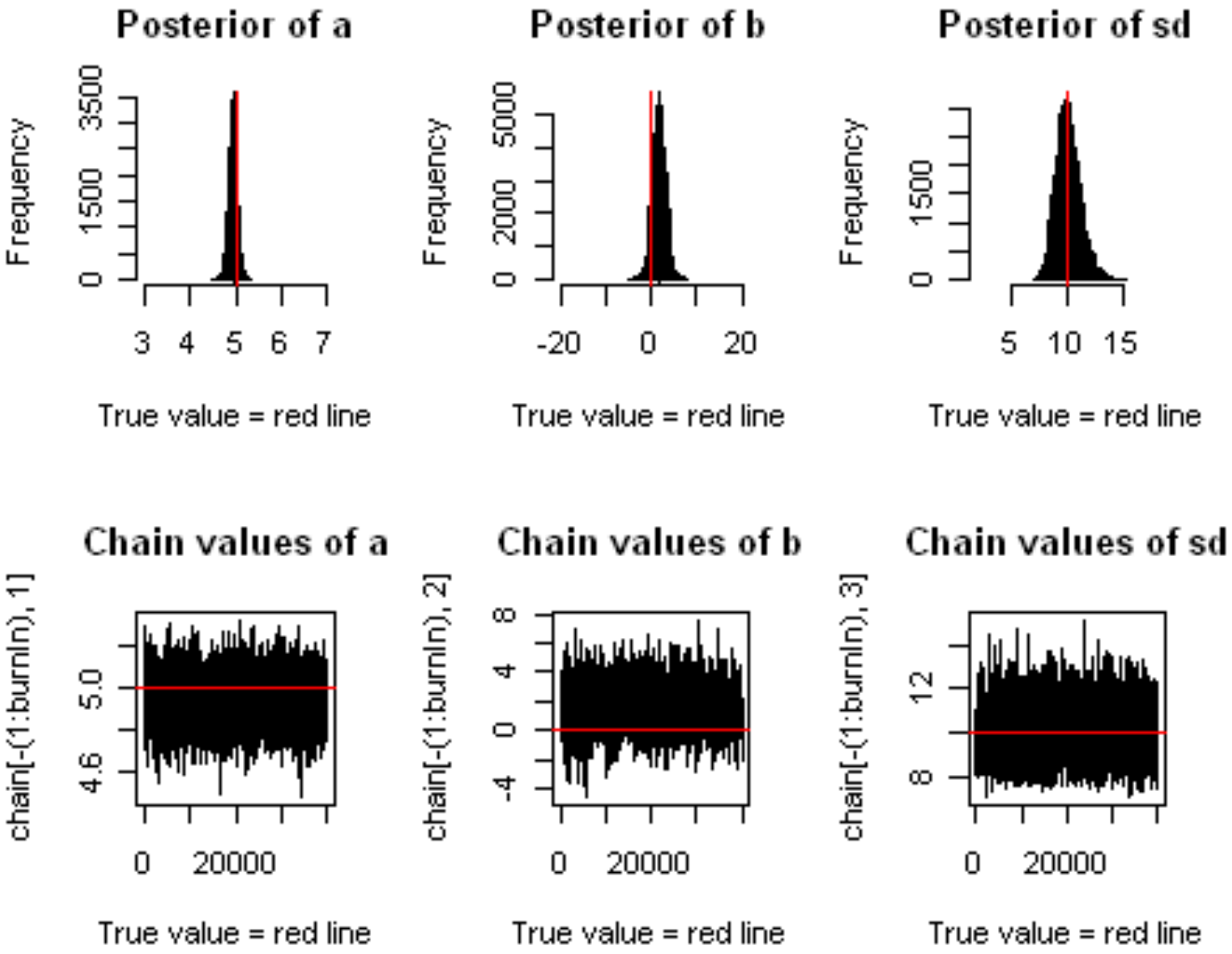
Bayesian Logistic Regression in R

Here is an example of Bayesian Logistic Regression with MCMC in R. Note that this follows the same approach as before, with some minor changes. The way the Metropolis-Hasting acceptance criterion is computed is slightly different in syntax but essentially the same.

Run R

```r
#source: https://theoreticalecology.wordpress.com/2010/09/17/metropolis

#Creating test data: we create some test data that will be used to fit
#Let's assume a linear relationship between the predictor and the respo
#so we take a linear model and add some noise.

trueA <- 5
trueB <- 0
trueSd <- 10
sampleSize <- 31

# create independent x-values
x <- (-(sampleSize-1)/2):((sampleSize-1)/2)
# create dependent values according to ax + b + N(0,sd)
```

Note the posterior plots below.



Further notes on easy visualisations with some related libraries:
https://theoreticalecology.wordpress.com/2011/12/09/mcmc-chain-analysis-and-convergence-diagnostics-with-coda-in-r/

Exercise 1

R Challenge

- Try extending the R Logistic Regression code for multi-step time series prediction
- Look for regression problems from UCI machine learning repository and test single and multi-output Bayesian logistic regression.
- Apply to COVID-19 prediction - single and multistep prediction for USA/India.

Python Challenge

- Look for regression problems from UCI machine learning repository and test single and multi-output Bayesian logistic regression.
- Apply to COVID-19 prediction - single and multistep prediction for USA/India.

Intro to Bayesian Neural Networks

A Bayesian neural network is essentially a probabilistic implementation of a standard neural network with the key difference being that the weights and biases are represented via the posterior probability distributions rather than single point values as shown in the figure below.

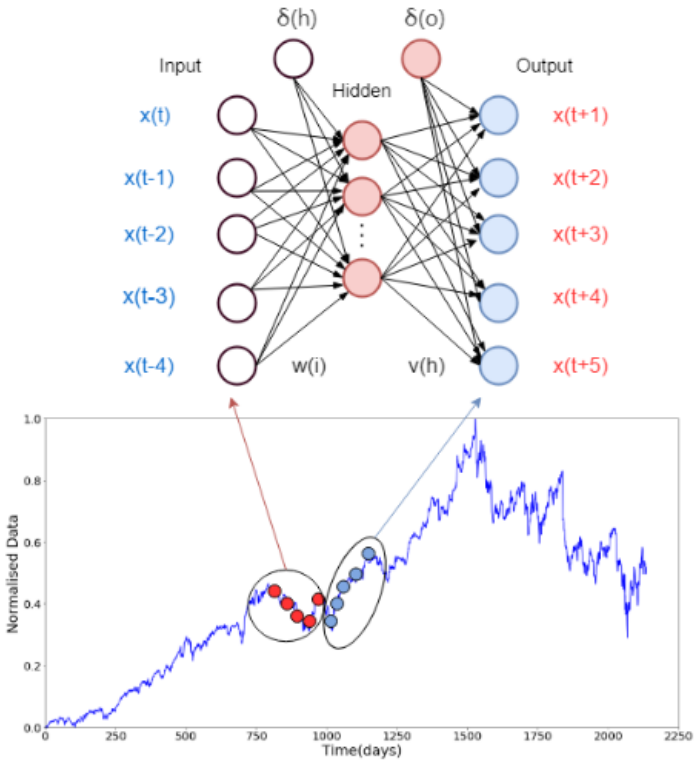
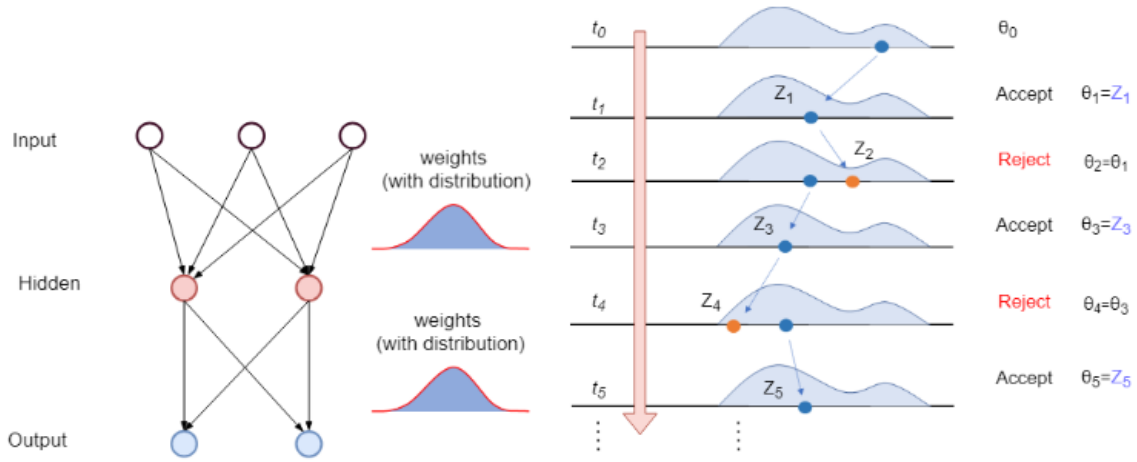
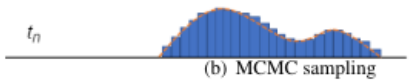


Figure 1: The time series (shown in red circles) is used as input for the neural network which predicts 5 steps-ahead in time (shown by blue circles). A sliding window approach is used to reconstruct the dataset in this way using Taken's theorem.





(a) Bayesian neural network (b) MCMC sampling

Figure 2: Bayesian neural network and MCMC sampling. Note that the posterior distribution is shown that represents weights in Panel (a)

Source: R. Chandra and Y. Xu, "Bayesian neural networks for stock market prediction before and during COVID-19", Under Review, 2020.

Similarly to standard neural networks, Bayesian neural networks also have universal continuous function approximation capabilities. On the other hand, the probabilistic model directly specifies the model through the interaction between known parameters to generate data. The probabilistic neural network model employs the posterior distribution to provides uncertainty quantification on the predictions.

The challenge of Bayesian inference is to learn a posterior distribution of neural network weights and biases to represent the data. We begin inference with prior distributions over the weights and biases of the network with a sampling scheme and a likelihood function given training data.

Since non-linear activation functions exist in the network, the conjugacy of prior and posterior is lost and inference sample scheme is used to construct the posterior distribution using the prior distribution and the data.

Likelihood function

We use the same idea from Bayesian logistic regression that uses Metropolis-Hastings MCMC with Random-Walk proposal distribution. Consider the equations used initially for logistic regression to be used for the neural network model in the next lesson.

The likelihood is given by

$$p(\mathbf{y_S}|\boldsymbol{\theta}) = -\frac{1}{(2\pi\tau^2)^{S/2}} \times \exp\left(-\frac{1}{2\tau^2}\sum_{t\in S}\left(\mathbf{y_t} - f(\bar{\mathbf{x}}_\mathbf{t})\right)^2\right)$$

The prior is given by

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{-\frac{1}{2\sigma^2}\left(\sum_{i=1}^{M}\theta\right)\right\} \times \tau^{2(1+\nu_1)}\exp\left(\frac{-\nu_2}{\tau^2}\right)$$

L represents the number of weights and biases which will increase in case of neural networks. The

next lesson demonstrates it further.

References:

1. Vehtari, A., Sarkka, S., & Lampinen, J. (2000, July). On MCMC sampling in Bayesian MLP neural networks. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* (Vol. 1, pp. 317-322). IEEE. https://ieeexplore.ieee.org/abstract/document/857855 http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.6539&rep=rep1&type=pdf
2. Andrieu, C., De Freitas, N., Doucet, A., & Jordan, M. I. (2003). An introduction to MCMC for machine learning. *Machine learning*, *50*(1-2), 5-43. https://www.cs.ubc.ca/~arnaud/andrieu_defreitas_doucet_jordan_intromontecarlomachinelearning.pdf
3. Neal, R. M. (2011). MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, *2*(11), 2. https://arxiv.org/pdf/1206.1901.pdf%20http://arxiv.org/abs/1206.1901.pdf
4. Song, J., Zhao, S., & Ermon, S. (2017). A-nice-mc: Adversarial training for mcmc. In *Advances in Neural Information Processing Systems* (pp. 5140-5150). http://papers.nips.cc/paper/7099-a-nice-mc-adversarial-training-for-mcmc.pdf
5. Sharaf, T., Williams, T., Chehade, A., & Pokhrel, K. (2020). BLNN: An R package for training neural networks using Bayesian inference. *SoftwareX*, *11*, 100432. https://www.sciencedirect.com/science/article/pii/S235271101930322X

MCMC Neural Network

Random-walk MCMC for Bayesian neural network for time series prediction problem. Note that no Langevin-gradients are used in this version.

Langevin-gradient Bayesian neural networks

We demonstrate the idea of using Langevin gradients using one-step ahead time series prediction. Note that the following equations have been taken from (Chandra et. al, 2017)

Let y_t denote a univariate time series modelled by:

$$y_t = f(\mathbf{x}_t) + \epsilon_t, \text{ for } t = 1, 2, \ldots, n \tag{1}$$

where $f(\mathbf{x}_t) = E(y_t|\mathbf{x}_t)$, is an unknown function, $\mathbf{x}_t = (y_{t-1}, \ldots, y_{t-D})$ is a vector of lagged values of y_t, and ϵ_t is the noise with $\epsilon_t \sim \mathcal{N}(0, \tau^2)$ $\forall t$.

In order to use neural networks for time series prediction, the original dataset is constructed into a state-space vector through Taken's theorem [15] which is governed by the embedding dimension (D) and time-lag (T).
Define

$$\mathcal{A}_{D,T} = \{t; t > D, \quad \text{mod } (t - (D + 1), T) = 0\} \tag{2}$$

Next, we look at the definition of a feedforward neural network with a single hidden layer.

Let $\mathbf{y}_{\mathcal{A}_{\mathcal{D},\mathcal{T}}}$ to be the collection of y_t's for which $t \in \mathcal{A}_{\mathcal{D},\mathcal{T}}$, then, $\forall t \in A_{D,T}$, we compute the $f(\mathbf{x}_t)$ by a feedforward neural network with one hidden layer defined by the function

$$f(\mathbf{x}_t) = g\left(\delta_o + \sum_{h=1}^{H} v_j g\left(\delta_h + \sum_{d=1}^{D} w_{dh} y_{t-d}\right)\right) \tag{3}$$

where δ_o and δ_h are the bias weights for the output o and hidden h layer, respectively. V_j is the weight which maps the hidden layer h to the output layer. w_{dh} is the weight which maps y_{t-d} to the hidden layer h and g is the activation function, which we assume to be a sigmoid function for the hidden and output layer units.

Let $\boldsymbol{\theta} = (\tilde{\mathbf{w}}, \mathbf{v}, \boldsymbol{\delta}, \tau^2)$, with $\boldsymbol{\delta} = (\delta_o, \delta_h)$, denote $L = (DH + (2 * H) + O + 1)$ vector of parameters that includes weights and biases, with O number of neurons in output layer. H is the number of hidden neurons required to evaluate the likelihood for the model given by (1), with $\tilde{\mathbf{w}} = (\mathbf{w_1}.', \ldots, \mathbf{w}'_{D.})'$, and $\mathbf{w_d}. = (w_{d1}, \ldots, w_{dH})'$, for $d = 1, \ldots, D$.

To conduct a Bayesian analysis, we need to specify prior distributions for the elements of θ which we choose to be

$$\begin{aligned}
v_h &\sim \mathcal{N}(0, \sigma^2) \text{ for } h = 1, \ldots, H, \\
\delta_0 &\sim N(0, \sigma^2) \\
\delta_h &\sim N(0, \sigma^2) \\
w_{dj} &\sim \mathcal{N}(0, \sigma^2) \text{ for } h = 1, \ldots, H \text{ and } d = 1, \ldots, D, \\
\tau^2 &\sim \mathcal{IG}(\nu_1, \nu_2)
\end{aligned} \tag{4}$$

where H is the number of hidden neurons. In general the log posterior is

$$\log\left(p(\boldsymbol{\theta}|\mathbf{y})\right) = \log\left(p(\boldsymbol{\theta})\right) + \log\left(p(\mathbf{y}|\boldsymbol{\theta})\right)$$

In our particular model the log likelihood is

$$\log\left(p(\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}|\boldsymbol{\theta})\right) = -\frac{n-1}{2}\log(\tau^2) - \frac{1}{2\tau^2}\sum_{t \in \mathcal{A}_{\mathcal{D},\mathcal{T}}} \left(y_t - E(y_t|\mathbf{x}_t)\right)^2 \tag{5}$$

where $E(y_t|\mathbf{x}_t)$ is given by (3). We further assume that the elements of θ are independent *apriori* so that the log of the prior distributions is

$$\begin{aligned}
\log\left(p(\boldsymbol{\theta})\right) = &-\frac{HD + H + 2}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\left(\sum_{h=1}^{H}\sum_{d=1}^{D} w_{dh}^2 + \sum_{h=1}^{H}(\delta_h^2 + v_h^2) + \delta_o^2\right) \\
&-(1+\nu_1)\log(\tau^2) - \frac{\nu_2}{\tau^2}
\end{aligned} \tag{6}$$

Now we look at code that implements the above equations. Note that in Equation 6, the number of weights and biases are explicitly shown which summaries to the following.

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{ -\frac{1}{2\sigma^2}\left(\sum_{i=1}^{M}\theta^2\right)\right\} \times \tau^{-2(1+\nu_1)} \exp\left(\frac{-\nu_2}{\tau^2}\right)$$

Note the code below does not run. We will run them all together in the next session with data.

Run PYTHON

```
#https://github.com/rohitash-chandra/LDMCMC_timeseries/blob/master/lange



    def likelihood_func(self, neuralnet, data, w, tausq):
        y = data[:, self.topology[0]]
        fx = neuralnet.evaluate_proposal(data, w)
        rmse = self.rmse(fx, y)
        loss = -0.5 * np.log(2 * math.pi * tausq) - 0.5 * np.square(y -
        return [np.sum(loss), fx, rmse]

    def prior_likelihood(self, sigma_squared, nu_1, nu_2, w, tausq):
        h = self.topology[1]  # number hidden neurons
        d = self.topology[0]  # number input neurons
```

Below code implements the functions of the neural network. The important function here is evaluate_proposal( ) which encodes the weights in neural networks. In order to implement Langevin gradients, we need to compute gradients with the backward-pass.

PYTHON

```
#https://github.com/rohitash-chandra/LDMCMC_timeseries/blob/master/lange

    def decode(self, w):
        w_layer1size = self.Top[0] * self.Top[1]
        w_layer2size = self.Top[1] * self.Top[2]

        w_layer1 = w[0:w_layer1size]
        self.W1 = np.reshape(w_layer1, (self.Top[0], self.Top[1]))

        w_layer2 = w[w_layer1size:w_layer1size + w_layer2size]
        self.W2 = np.reshape(w_layer2, (self.Top[1], self.Top[2]))
        self.B1 = w[w_layer1size + w_layer2size:w_layer1size + w_layer2s
        self.B2 = w[w_layer1size + w_layer2size + self.Top[1]:w_layer1si

```

Next, we look at the main algorithm that shows how the samples are accepted taking Langevin gradeints into account.

Alg. 1 Langevin Dynamics for neural networks

Data: Univariate time series $\mathbf{y}$
Result: Posterior of weights and biases $p(\boldsymbol{\theta}|\mathbf{y})$
Step 1: State-space reconstruction $\mathbf{y}_{\mathcal{A}_{D,T}}$ by Equation 2
Step 2: Define feedforward network as given in Equation 3
Step 3: Define $\boldsymbol{\theta}$ as the set of all weights and biases
Step 4: Set parameters σ^2 ,ν_1, ν_2 for prior given in Equation 6

for *each k until max-samples* **do**
    1.Compute gradient $\Delta\boldsymbol{\theta}^{[k]}$given by Equation 8
    2. Draw $\boldsymbol{\eta}$ from $\mathcal{N}(0, \Sigma_\eta)$
    3. Propose $\boldsymbol{\theta}^* = \theta^{[k]} + \Delta\boldsymbol{\theta}^{[k]} + \boldsymbol{\eta}$
    4. Draw from uniform distribution $u \sim \mathcal{U}[0, 1]$
    5. Obtain acceptance probability α given by Equation 9
    if $u < \alpha$ **then**
        $\boldsymbol{\theta}^{[k+1]} = \boldsymbol{\theta}^*$
    end
    else
        $\boldsymbol{\theta}^{[k+1]} = \boldsymbol{\theta}^{[k]}$
    end
end

posterior distribution $p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}})$. The proposed MCMC algorithm consists of a single Metropolis Hasting step with proposals formed using Langevin dynamics that employs gradients. In particular, we propose a new value of $\boldsymbol{\theta}$ from

$$\boldsymbol{\theta}^p \sim \mathcal{N}(\bar{\boldsymbol{\theta}}^{[k]}, \Sigma_\theta), \text{ where} \tag{7}$$

$$\bar{\boldsymbol{\theta}}^{[k]} = \boldsymbol{\theta}^{[k]} + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}], \tag{8}$$

$$E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}] = \sum_{t \in \mathcal{A}_{D,T}} (y_t - f(\mathbf{x}_t)^{[k]})^2,$$

$$\nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}] = \left(\frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_L}\right)$$

r is the learning rate, $\Sigma_\theta = \sigma_\theta^2 I_L$ and I_L is the $L \times L$ identity matrix. So that the newly proposed value of $\boldsymbol{\theta}^p$, consists of 2 parts:

1. An gradient descent based weight update given by Equation (8)
2. Add an amount of noise, from $\mathcal{N}(0, \Sigma_\theta)$.

Hereafter, we refer to the proposed Langevin dynamics for neural networks as LD-MCMC. This combined update is used as a proposal in a Metropolis-Hastings step, which accepts the proposed value of $\boldsymbol{\theta}^p$ with the usual probability α, where

$$\alpha = \min\left\{1, \frac{p(\boldsymbol{\theta}^p|\mathbf{y}_{\mathcal{A}_{D,T}})q(\boldsymbol{\theta}^{[k]}|\boldsymbol{\theta}^p)}{p(\boldsymbol{\theta}^{[k]}|\mathbf{y}_{\mathcal{A}_{D,T}})q(\boldsymbol{\theta}^p|\boldsymbol{\theta}^{[k]})}\right\} \tag{9}$$

where $p(\boldsymbol{\theta}^p|\mathbf{y}_{\mathcal{A}_{D,T}})$ and $p(\boldsymbol{\theta}^{[k]}|\mathbf{y}_{\mathcal{A}_{D,T}})$ can be computed using Equation (5) and Equation (6). $q(\boldsymbol{\theta}^p|\boldsymbol{\theta}^{[k]})$, is given by Equation (7) and $q(\boldsymbol{\theta}^{[k]}|\boldsymbol{\theta}^p) \sim N(\bar{\boldsymbol{\theta}}^p, \Sigma_{\boldsymbol{\theta}})$, with $\bar{\boldsymbol{\theta}}^p = \boldsymbol{\theta}^p + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^p]$, thus ensuring that the detailed balance condition holds and the sequence $\boldsymbol{\theta}^{[k]}$ converges to draws from the posterior $p(\boldsymbol{\theta}|\mathbf{y})$.

For testing, given a test input $\tilde{\mathbf{x}}$ (with missing label $\tilde{y}$), the uncertainty learned in training is transferred to prediction, yielding the following posterior distribution:

$$p(\tilde{y}|\tilde{x}, \mathbf{y}_{\mathcal{A}_{D,T}}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}})}[p(\tilde{y}|\tilde{x}, \theta)] = \int_\theta p(\tilde{y}|\tilde{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}}) d\boldsymbol{\theta} \tag{10}$$

The predicted distribution of $\tilde{y}$ can be viewed in terms of model averaging across parameters, based on the learned $p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}})$; this is contrasted with learning a single point estimate of $\boldsymbol{\theta}$ based on $\mathbf{y}_{\mathcal{A}_{D,T}}$.

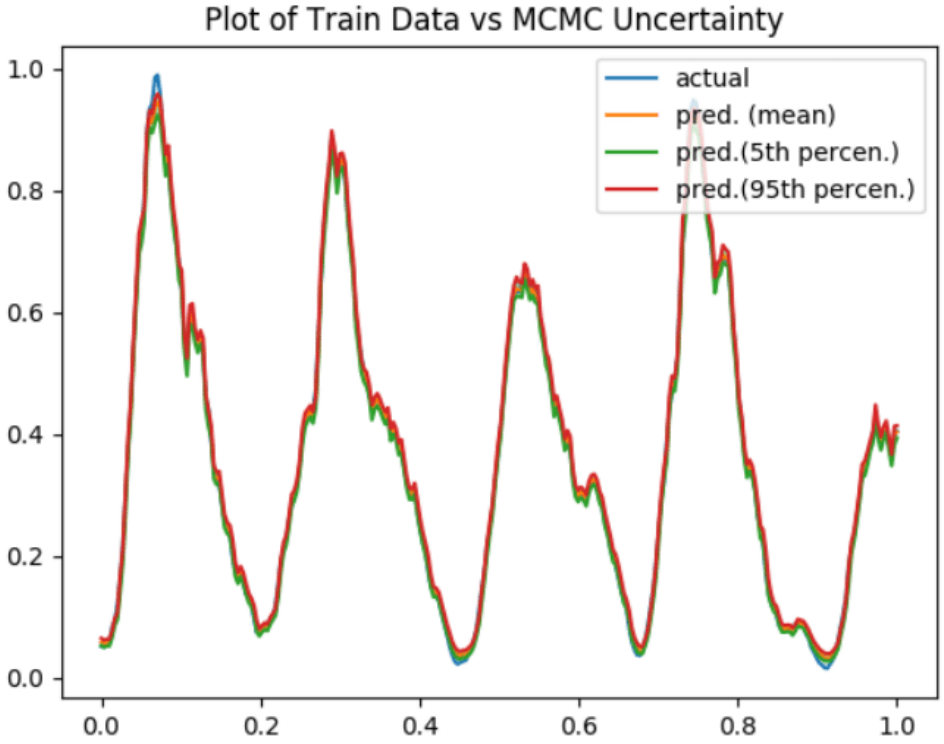
```python
#Source: https://github.com/rohitash-chandra/LDMCMC_timeseries/blob/mast

for i in range(samples - 1):

    lx = np.random.uniform(0,1,1)

    if (self.use_langevin_gradients is True) and (lx< self.l_prob):
        w_gd = neuralnet.langevin_gradient(self.traindata, w.copy(), sel
        w_proposal = np.random.normal(w_gd, step_w, w_size) # Eq 7
        w_prop_gd = neuralnet.langevin_gradient(self.traindata, w_propos
        #first = np.log(multivariate_normal.pdf(w , w_prop_gd , sigma_di
        #second = np.log(multivariate_normal.pdf(w_proposal , w_gd , sig

        wc_delta = (w- w_prop_gd)
```

The results summary of benchmark problems is shown below.

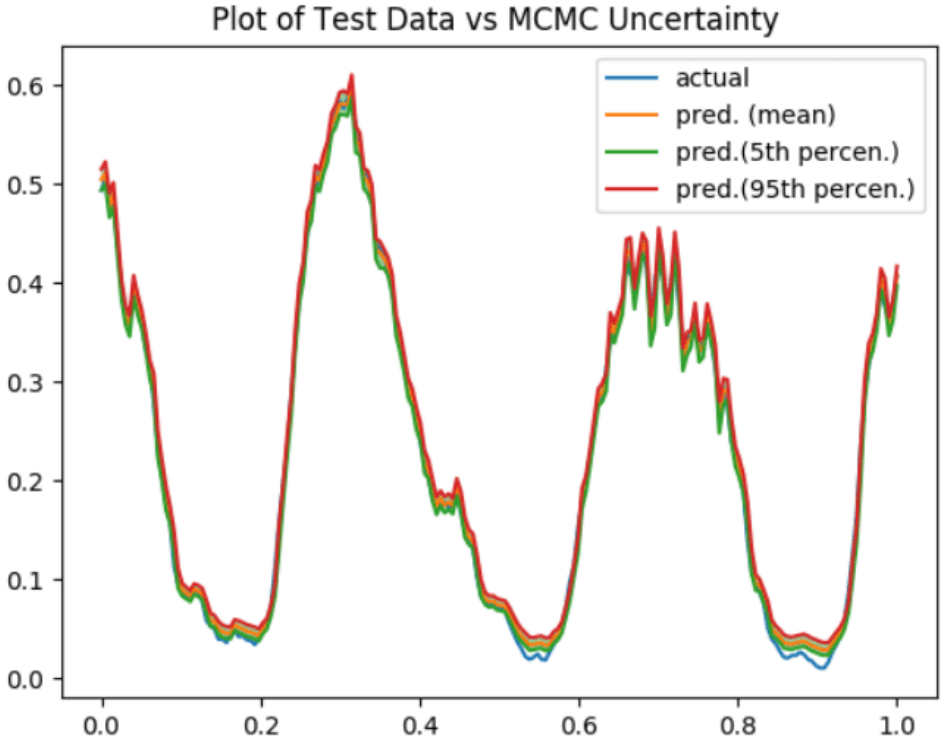
Table 1: Results for the respective methods

Problem	Method	Train (mean)	Train (std)	Test (mean)	Test(std)	% Accepted
Lazer	GD*	0.02191	0.00129	0.02675	0.00085	-
	GD	0.01035	0.00162	0.01732	0.00142	-
	MCMC	0.02549	0.00837	0.02643	0.00718	9.70
	LD-MCMC	0.01658	0.00211	0.02280	0.00351	57.86
Sunspot	GD*	0.02117	0.00160	0.02359	0.00209	-
	GD	0.00775	0.00026	0.00975	0.00031	-
	MCMC	0.01466	0.00174	0.01402	0.00178	4.55
	LD-MCMC	0.01155	0.00167	0.01090	0.00146	45.89
Mackey	GD*	0.00590	0.00026	0.00669	0.00029	-
	GD	0.00286	0.00025	0.0033	0.00026	-
	MCMC	0.00511	0.00058	0.00520	0.00058	1.74
	LD-MCMC	0.00615	0.00091	0.00627	0.00091	30.35
Lorenz	GD*	0.01570	0.00056	0.01608	0.00052	-
	GD	0.00400	0.00024	0.00460	0.00026	-
	MCMC	0.00813	0.00174	0.00713	0.00150	2.74
	LD-MCMC	0.00890	0.00211	0.00821	0.00207	26.95
Rossler	GD*	0.01570	0.00056	0.01608	0.00052	-
	GD	0.00281	0.00029	0.00462	0.00045	-
	MCMC	0.01371	0.00291	0.01355	0.00297	3.69
	LD-MCMC	0.00722	0.00121	0.00692	0.00120	35.53
Henon	GD*	0.01366	0.00033	0.01778	0.00025	-
	GD	0.00555	0.00029	0.00604	0.00025	-
	MCMC	0.03256	0.03920	0.03127	0.03850	8.71
	LD-MCMC	0.00948	0.00112	0.00912	0.00114	27.17

A typical experimental run with actual and predicted values along with uncertainty is shown next.



(a) Prediction and uncertainty quantification over training data



(b) Prediction and uncertainty quantification over test data

Fig. 1: Results for Sunspot time series using LD-MCMC algorithm. Note that the x-axis represents the time in year while y-axis gives the Sunspot index.

The next lesson shows the code that combined all the functions and data - run and see.

References:

1. Welling, M., & Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 681-688). http://people.ee.duke.edu/~lcarin/398_icmlpaper.pdf
2. Chandra R; Azizi L; Cripps S, 2017, 'Bayesian neural learning via Langevin dynamics for chaotic time series prediction', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 564 - 573, http://dx.doi.org/10.1007/978-3-319-70139-4_57 https://github.com/rohitash-chandra/research/blob/master/2017/LangevinNeuralnet_ICONIP2017.pdf Code: https://github.com/rohitash-chandra/LDMCMC_timeseries
3. Chandra R; Jain K; Deo RV; Cripps S, 2019, 'Langevin-gradient parallel tempering for Bayesian neural learning', *Neurocomputing*, vol. 359, pp. 315 - 326, http://dx.doi.org/10.1016/j.neucom.2019.05.082 https://github.com/rohitash-chandra/research/blob/master/2019/Chandra_LangevinNeurocom2019.pdf

Langevin-gradient Bayesian neural networks

Convergence diagnosis

Gelman-Rubin diagnostic

The Gelman-Rubin diagnostic evaluates MCMC convergence by analyzing the behaviour of multiple Markov chains. Given multiple chains from different experimental runs, assessment is done by comparing the estimated between-chains and within-chain variances for each parameter, where large differences between the variances indicate non-convergence.

We calculate the potential scale reduction factor (PSRF) which gives the ratio of the current variance in the posterior variance for each parameter compared to that being sampled. The values for the PSRF near 1 indicates convergence.

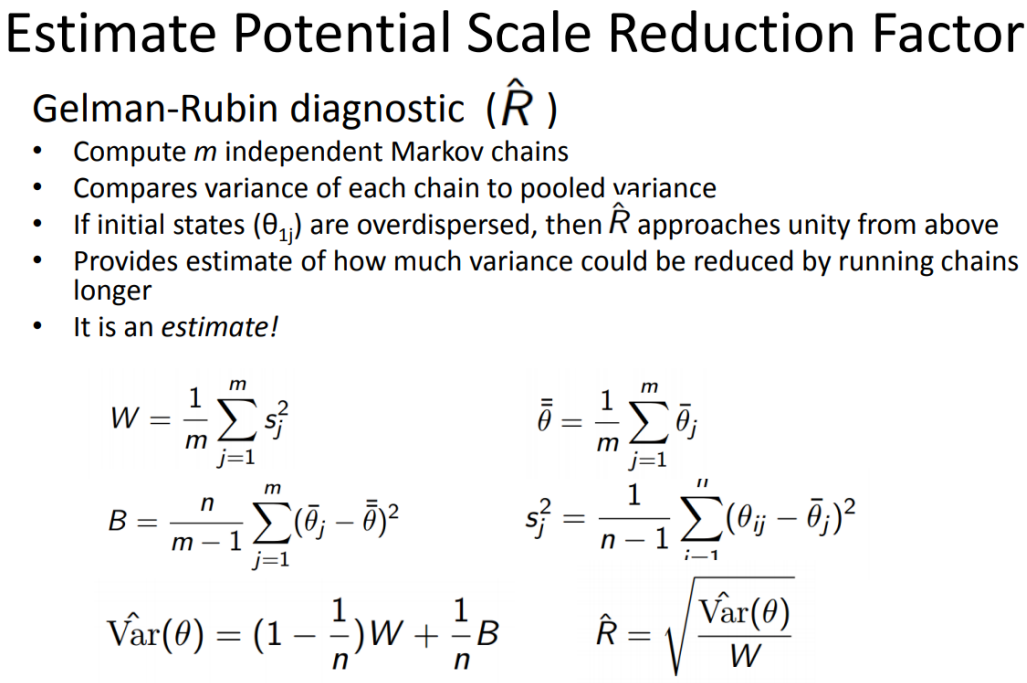


Figure source: https://astrostatistics.psu.edu/RLectures/diagnosticsMCMC.pdf

PYTHON

```python
#Source: https://github.com/intelligentEarth/pt-Bayeslands/blob/master/c
#Authors: R Scalzo and R Chandra
import numpy  as np

def gelman_rubin(data):
    """
    Apply Gelman-Rubin convergence diagnostic to a bunch of chains.
    :param data: np.array of shape (Nchains, Nsamples, Npars)
    """
    Nchains, Nsamples, Npars = data.shape
    B_on_n = data.mean(axis=1).var(axis=0)      # variance of in-chain m
    W = data.var(axis=1).mean(axis=0)           # mean of in-chain varia

    #print(B_on_n, ' B_on_n mean')
```

Autocorrelation time

Autocorrelation refers to the correlation of a time series with a delayed copy of itself over successive time intervals which give the degree of similarity. It measures the relationship between a variable's current value and its past values. A positive 1 autocorrelation represents a perfect positive correlation, while an autocorrelation of negative 1 represents a perfect negative correlation. More information: https://online.stat.psu.edu/stat462/node/188/

Autocorrelation time is used as a convergence diagnosis for MCMC sampling algorithms. Implementation in Emcee library: https://emcee.readthedocs.io/en/stable/tutorials/autocorr/

References

1. Brooks, S. P., and A. Gelman. 1998. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics* 7: 434–455. http://www2.stat.duke.edu/~scs/Courses/Stat376/Papers/ConvergeDiagnostics/BrooksGelman.pdf
2. Gelman, A., and D. B. Rubin. 1992. Inference from iterative simulation using multiple sequences. *Statistical Science* 7: 457–472. https://projecteuclid.org/download/pdf_1/euclid.ss/1177011136

Implementation

1. Stata: https://www.stata.com/new-in-stata/gelman-rubin-convergence-diagnostic/
2. Emcee: http://greg-ashton.physics.monash.edu/the-gelman-rubin-statistic-and-emcee.html
3. Autocorrelation: https://emcee.readthedocs.io/en/stable/tutorials/autocorr/

Exercise 2

R challenge

1. Extend MCMC logistic regression code in scratch for neural networks
2. Apply to multi-step time series prediction (Stock market MMM dataset)
3. Apply convergence diagnosis

Python challenge

1. Extend MCMC and Langevin neural networks for multiple outputs
2. Apply to multi-step time series prediction (Stock market MMM dataset)
3. Apply convergence diagnosis

Further challenge

Explore MCMC libraries such as Stan (R and Python) and PyMC3 and compare your results with the above.

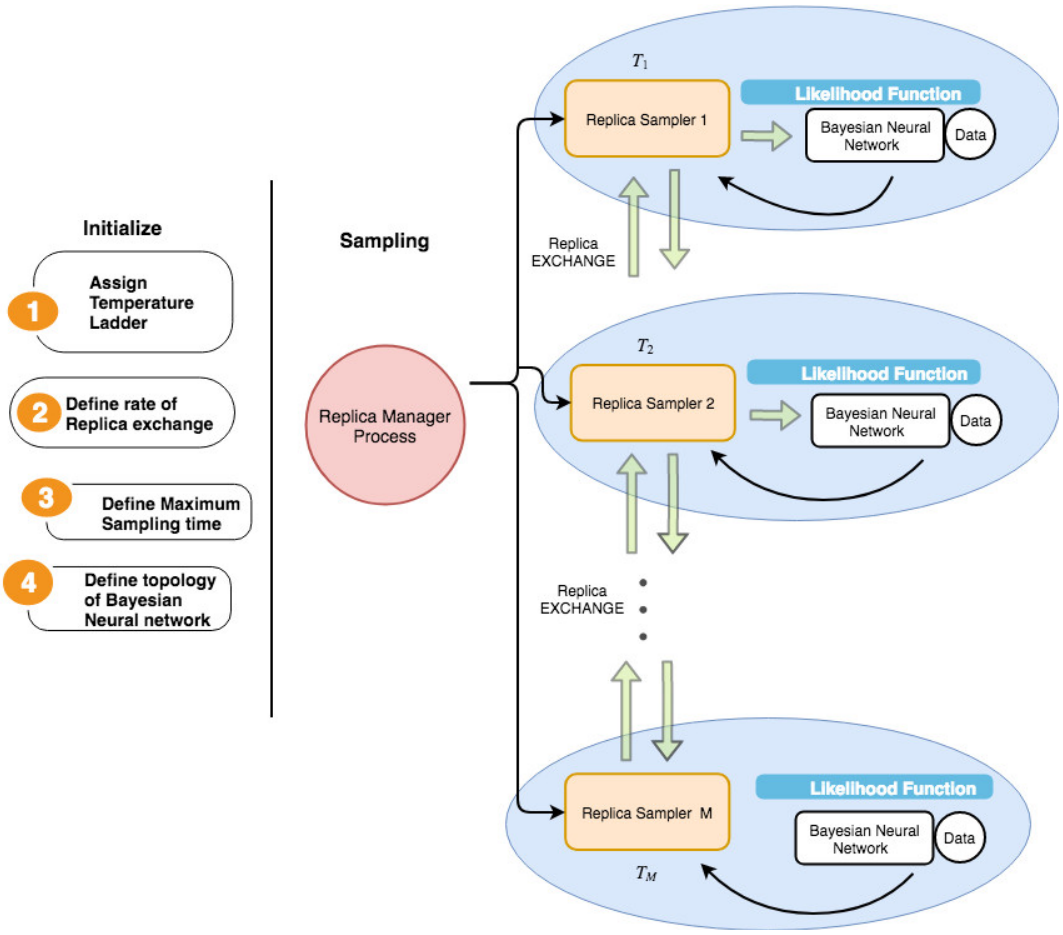
Advances in Bayesian neural networks

There are a number of MCMC variants: Metropolis-Hastings sampling, Gibbs Sampling, ensemble sampling, parallel tempering MCMC, adaptive MCMC, Hamiltonian Monte-Carlo, Langevin MCMC, Reversible Jump MCMC; shower we will only focus on **single-chain MCMC random-walk sampler** and **Langevin-gradient MCMC sampler** in this course. The other variates have special strengths and weakness suitable for different types of models. Note that Hamiltonian and Langevin MCMC require gradients and cannot be used in models where gradients are not present.

Langevin-gradient parallel tempering for Bayesian neural learning

Abstract: Bayesian inference provides a rigorous approach for neural learning with knowledge representation via the posterior distribution that accounts for uncertainty quantification. Markov Chain Monte Carlo (MCMC) methods typically implement Bayesian inference by sampling from the posterior distribution. This not only provides point estimates of the weights, but the ability to propagate and quantify uncertainty in decision making. However, these techniques face challenges in convergence and scalability, particularly in settings with large datasets and neural network architectures. This paper addresses these challenges in two ways. First, parallel tempering MCMC sampling method is used to explore multiple modes of the posterior distribution and implemented in multi-core computing architecture. Second, we make within-chain sampling scheme more efficient by using Langevin gradient information for creating Metropolis–Hastings proposal distributions. We demonstrate the techniques using time series prediction and pattern classification applications. The results show that the method not only improves the computational time, but provides better decision making capabilities when compared to related methods.

Chandra R; Jain K; Deo RV; Cripps S, 2019, 'Langevin-gradient parallel tempering for Bayesian neural learning', *Neurocomputing*, vol. 359, pp. 315 - 326, http://dx.doi.org/10.1016/j.neucom.2019.05.082
https://github.com/rohitash-chandra/research/blob/master/2019/Chandra_LangevinNeurocom2019.pdf



Multinomial likelihood function

Given a discrete dataset such as the outcomes or class labels of a classification problem, it is inappropriate to model the data as Gaussian. Hence for discrete data with K possible classes, we assume that the data $\mathbf{y} = (y_1, \ldots, y_n)$ are generated from a multinomial distribution with parameter vector $\pi = (\pi_1, \ldots, \pi_K)$ where $\sum_{k=1}^{K} \pi_k = 1$. Note that this property is given by the ***softmax*** activation function. Further information:

1. Chandra R; Jain K; Deo RV; Cripps S, 2019, 'Langevin-gradient parallel tempering for Bayesian neural learning', *Neurocomputing*, vol. 359, pp. 315 - 326, http://dx.doi.org/10.1016/j.neucom.2019.05.082 https://github.com/rohitash-chandra/research/blob/master/2019/Chandra_LangevinNeurocom2019.pdf
2. Chandra R; Kapoor A, 2020, 'Bayesian neural multi-source transfer learning', *Neurocomputing*, vol. 378, pp. 54 - 64, http://dx.doi.org/10.1016/j.neucom.2019.10.042 https://github.com/rohitash-chandra/research/blob/master/2020/Chandra_NC2020.pdf

Future directions

Bayesian Optimisation: Bayesian optimisation and surrogate-assisted optimisation employs machine learning models to estimate the objective function using a surrogate model or acquisition function during optimisation which handy for expensive problems. The major advantage of Bayesian optimisation has been in reducing computational load by approximating the actual model with an acquisition function that is computationally cheaper. More information:

1. Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*. https://arxiv.org/pdf/1012.2599.pdf
2. Chandra R, Jain K, Kapoor A, Aman A, 'Surrogate-assisted parallel tempering for Bayesian neural learning'. Eng. Appl. Artif. Intell. 94: 103700 (2020) https://github.com/rohitash-chandra/research/blob/master/2020/Chandra_EngAppAI2020.pdf

Variational inference: Provides an analytical approximation to the posterior probability. Rather than sampling directly from the posterior, variational inference methods approximate it, making them applicable to problems where a large number of variables are present and where MCMC sampling becomes too computationally expensive. More information:

1. https://en.wikipedia.org/wiki/Variational_Bayesian_methods
2. http://www.robots.ox.ac.uk/~sjrob/Pubs/vbTutorialFinal.pdf

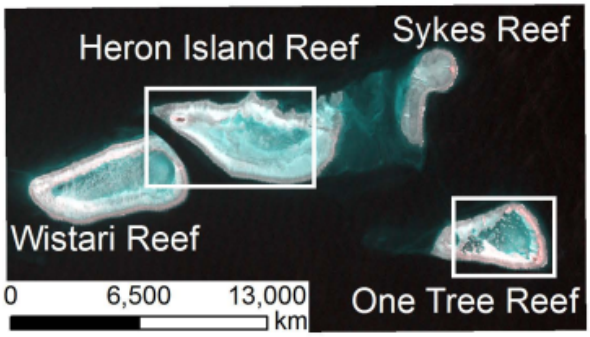
Applications of MCMC

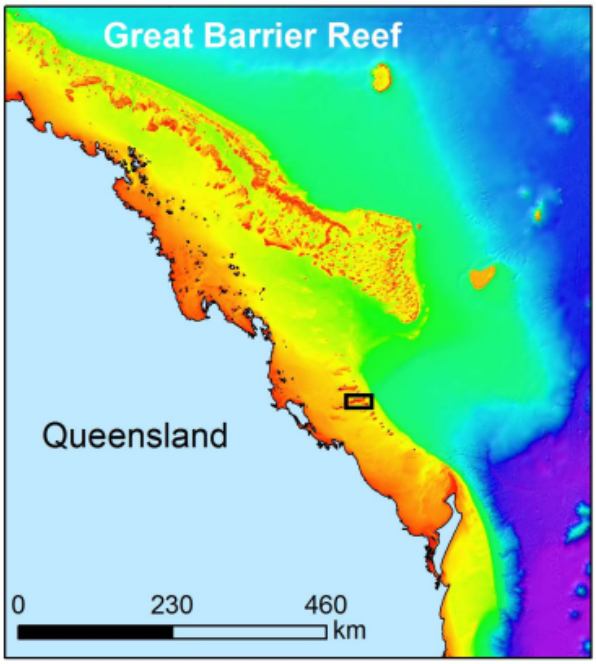
There are models out there where gradients are not present, such as those in Geosciences and environmental science. I spent my postdoc fellowship at the University of Sydney looking at them and these are some publications that look at models where no gradients are available using single-chain MCMC, Adaptive parallel tempering MCMC.

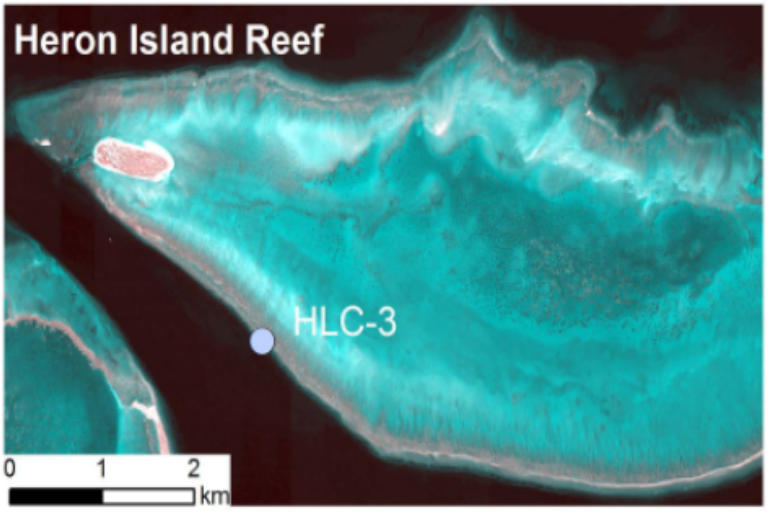
Abstract: Bayesreef: A Bayesian inference framework for modelling reef growth in response to environmental change and biological dynamics

Estimating the impact of environmental processes on vertical reef development in geological time is a very challenging task. *pyReef-Core* is a deterministic carbonate stratigraphic forward model designed to simulate the key biological and environmental processes that determine vertical reef accretion and assemblage changes in fossil reef drill cores. We present a Bayesian framework called *Bayesreef* for the estimation and uncertainty quantification of parameters in pyReef-Core that represent environmental conditions affecting the growth of coral assemblages in geological timescales. Weencounter multimodal posterior distributions and investigate the challenges of sampling using Markov chain Monte-Carlo (MCMC) methods, which includes parallel tempering MCMC. We use a synthetic reef-core to investigate fundamental issues and then apply the methodology to a selected reef-core from the *Great Barrier Reef* in Australia. The results show that Bayesreef accurately estimates and provides uncertainty quantification of the selected parameters that represent environment and ecological conditions in *pyReef-Core*. Bayesreef provides insights into the complex posterior distributions of the parameters in *pyReef-Core*, which provides the groundwork for future research in this area.

Pall J; Chandra R; Azam D; Salles T; Webster JM; Scalzo R; Cripps S, 2020, 'Bayesreef: A Bayesian inference framework for modelling reef growth in response to environmental change and biological dynamics', *Environmental Modelling and Software*, vol. 125, pp. 104610 - 104610, http://dx.doi.org/10.1016/j.envsoft.2019.104610



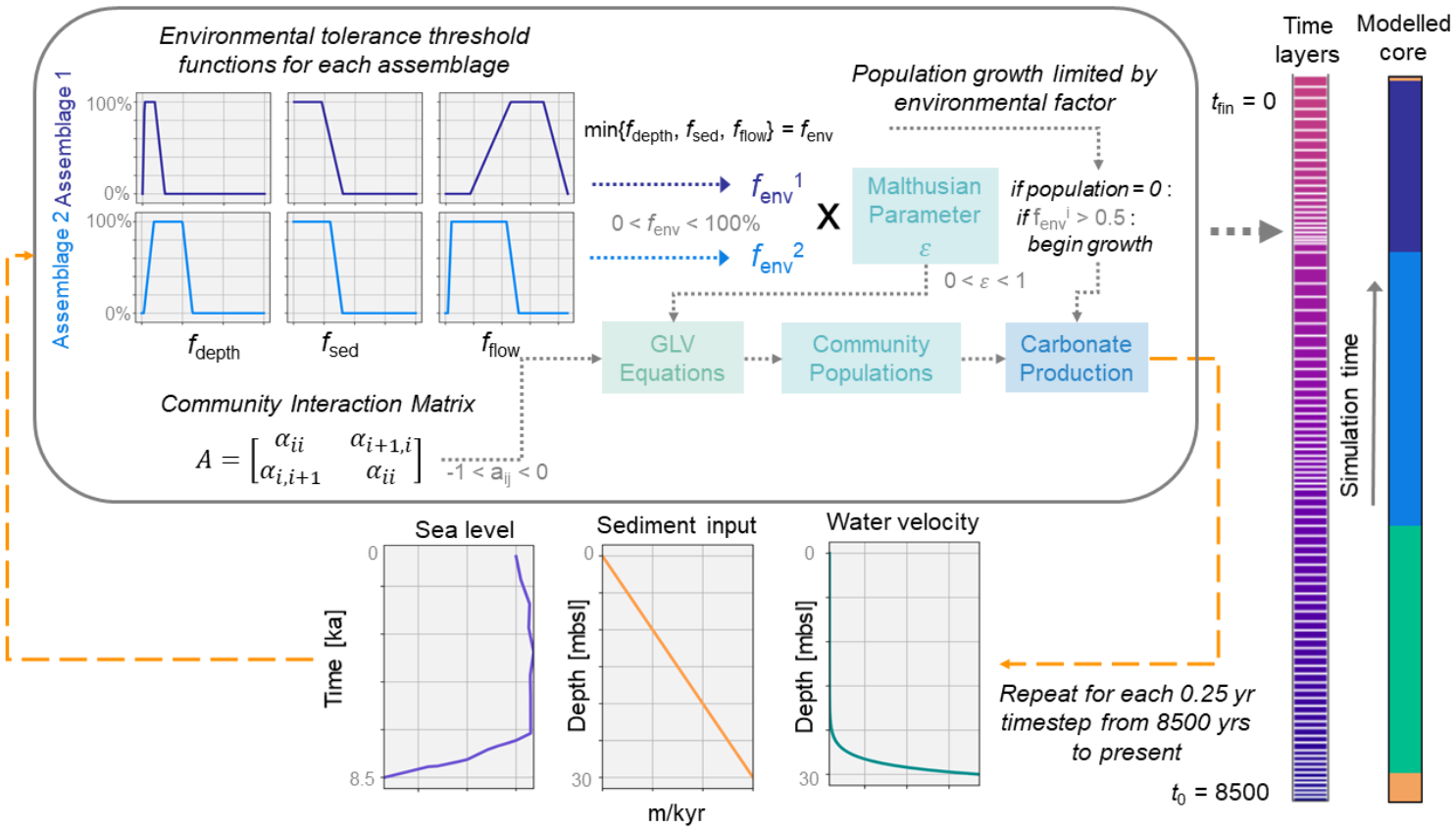




Salas-Saavedra et al.

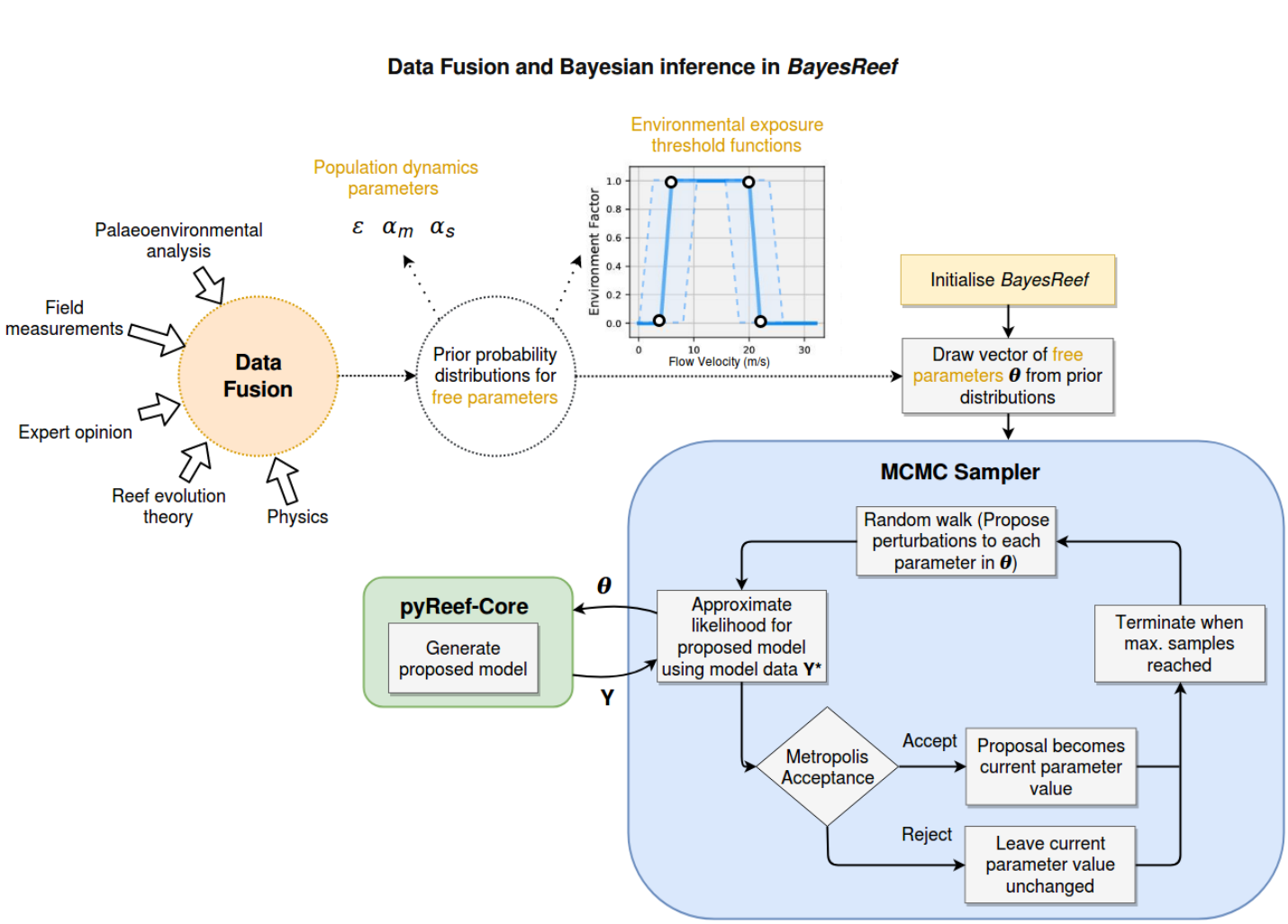
Coordinate System: GDA 1994
Data Sources:
GBRMPA:GBR islands, reefs and cays.
USyd: Geocoastal Research Group, unpublished.
Beaman, R.J. (2010). 3DGBR: A high-resolution depth model for the Great Barrier Reef and Coral Sea. MTSRF Project 2.5i.1a:12.

Above figure gives the local for the study area in the great barrier reef.



Above figure gives the schematic for py-Reef-Core model that simulates vertical reef development over thousands of years.

Below we refer to an MCMC framework for estimating parameters in a geological reef-core model (py-Reef-Core):

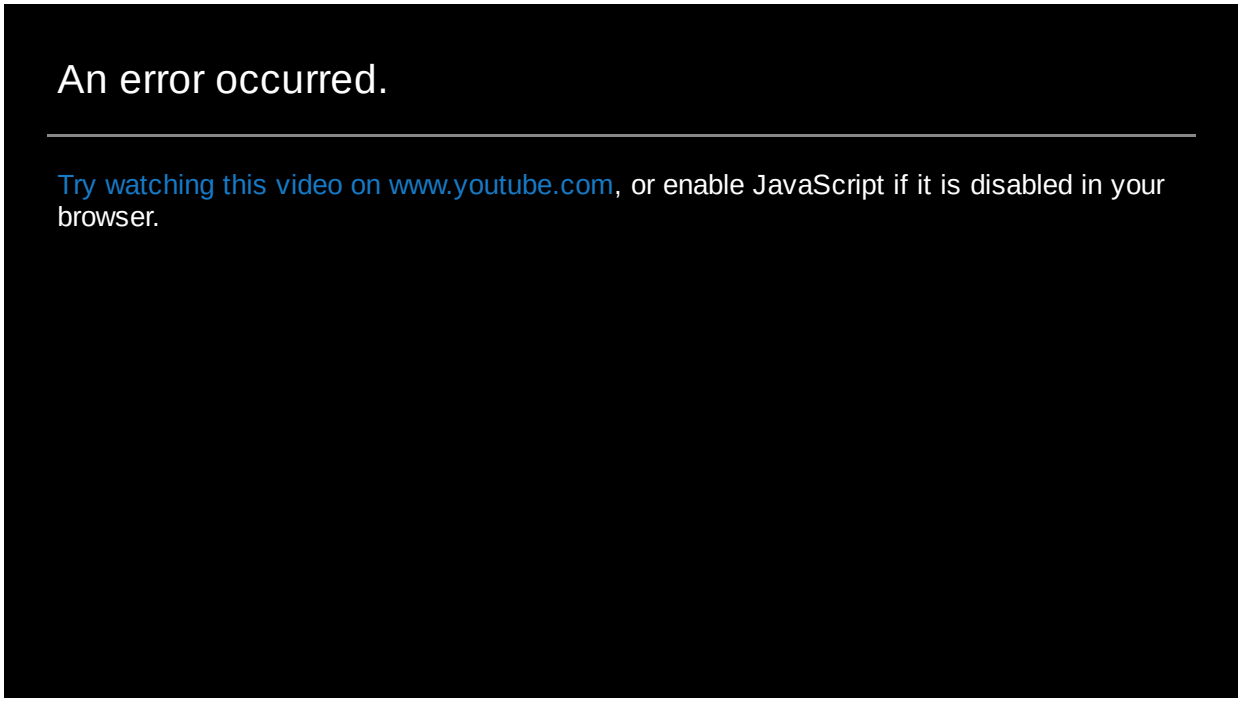


Bayeslands: A Bayesian inference approach for parameter uncertainty quantification in Badlands

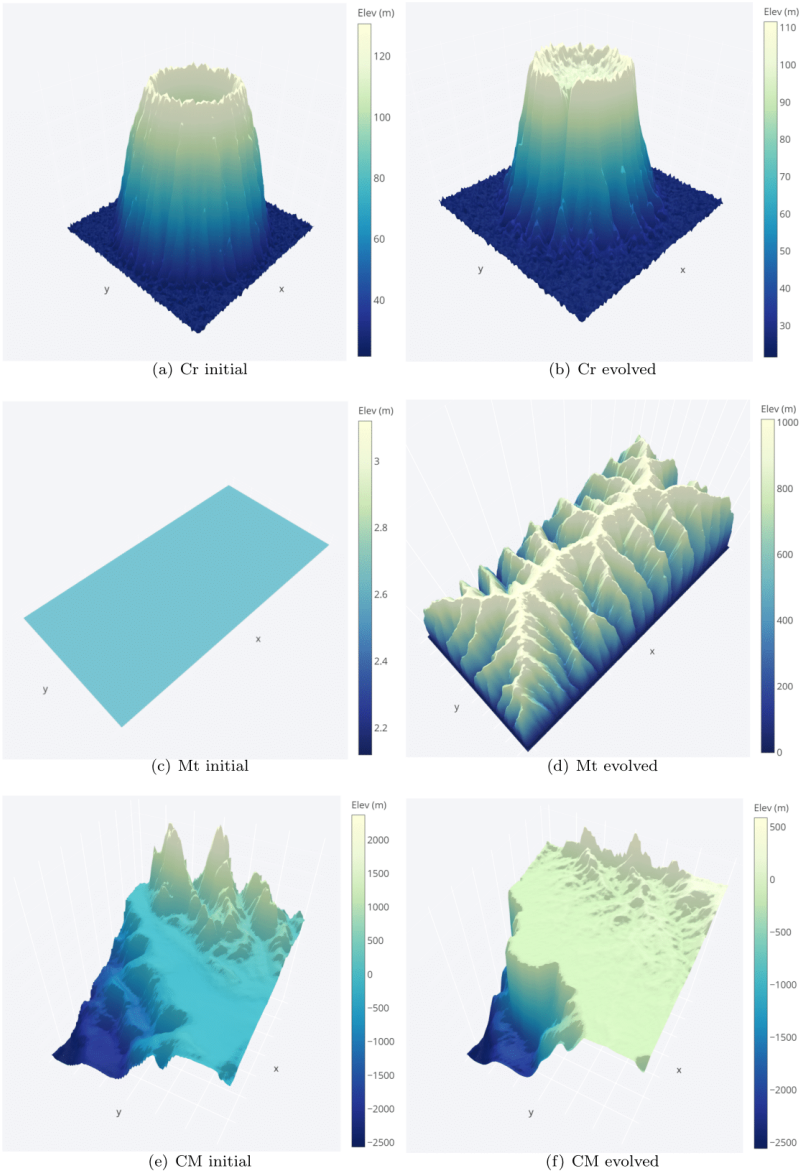
Abstract: Bayesian inference provides a rigorous methodology for estimation and uncertainty quantification of unknown parameters in geophysical forward models. Badlands is a landscape evolution model that simulates topography development at various space and time scales. Badlands consists of a number of geophysical parameters that needs estimation with appropriate uncertainty quantification; given the observed present-day ground truth such as surface topography and the stratigraphy of sediment deposition through time. The inference of the unknown parameters is challenging due to the scarcity of data, sensitivity of the parameter setting, and complexity of the model. In this paper, we take a Bayesian approach to provide inference using Markov chain Monte Carlo sampling (MCMC). We present *Bayeslands*; a Bayesian framework for Badlands that fuses information obtained from complex forward models with observational data and prior knowledge. As a proof-of-concept, we consider a synthetic and real-world topography with two parameters for Bayeslands; namely, precipitation and erodibility. We demonstrate the challenge in sampling irregular and multi-modal posterior distributions using a likelihood surface that has a range of sub-optimal modes. The results of the experiments show that Bayeslands yields a promising distribution of the selected Badlands parameters.

Chandra R; Azam D; Müller RD; Salles T; Cripps S, 2019, 'Bayeslands: A Bayesian inference approach for parameter uncertainty quantification in Badlands', *Computers and Geosciences*, vol. 131, pp. 89 - 101, http://dx.doi.org/10.1016/j.cageo.2019.06.012 https://github.com/rohitash-chandra/research/blob/master/2019/Chandra_Bayeslands_Computers-and-Geoscience.pdf

Here is an example of simulation from a landscape evolution model:



Below are examples of some test problems.



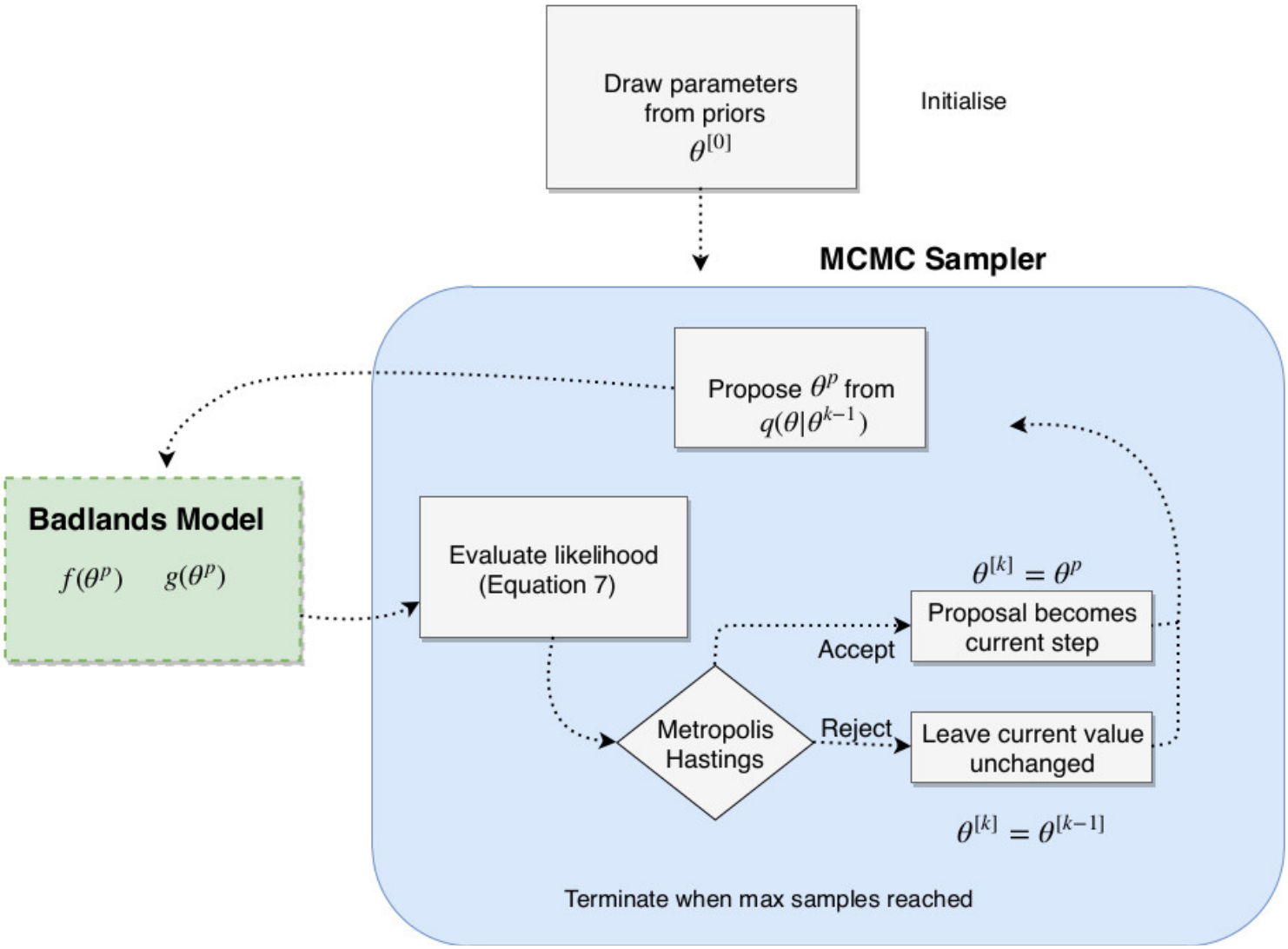
(a) Cr initial (b) Cr evolved

(c) Mt initial (d) Mt evolved

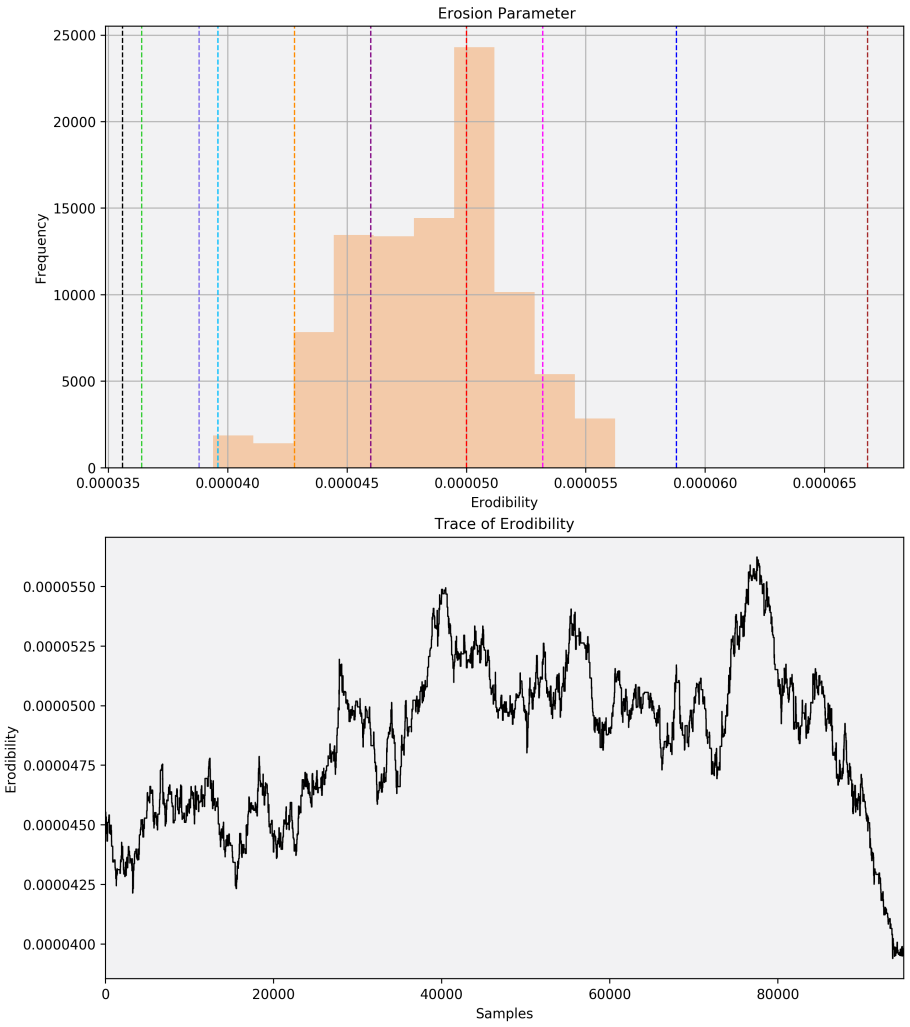
(e) CM initial (f) CM evolved

Figure above shows three different test problems, with initial and final topographies.

Next we see an example of MCMC for landscape evolution model (Badlands)



Source: Chandra R; Azam D; Müller RD; Salles T; Cripps S, 2019, 'Bayeslands: A Bayesian inference approach for parameter uncertainty quantification in Badlands', *Computers and Geosciences*, vol. 131, pp. 89 - 101, http://dx.doi.org/10.1016/j.cageo.2019.06.012



Posterior and trace plot of a parameter called erodibility in the Badlands model by Bayeslands framework.

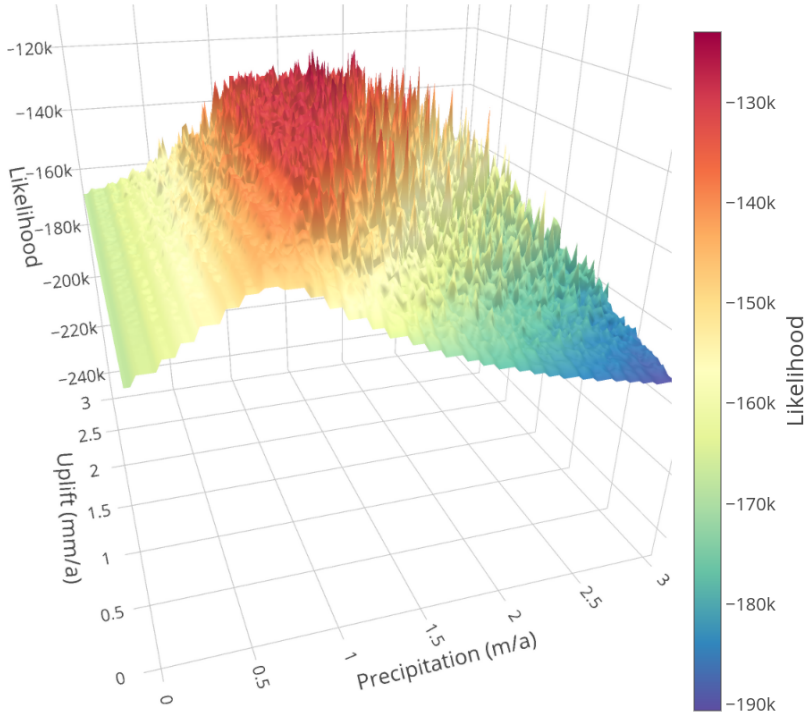
Multicore Parallel Tempering Bayeslands for Basin and Landscape Evolution

Abstract: The Bayesian paradigm is becoming an increasingly popular framework for estimation and uncertainty quantification of unknown parameters in geophysical inversion problems. Badlands is a *landscape evolution model* for simulating topography evolution at a broad range of spatial and temporal scales. Our previous work presented Bayeslands that used the Bayesian inference for estimating unknown parameters in the Badlands model using Markov chain Monte Carlo sampling. Bayeslands faced challenges in terms of computational issues and convergence due to multimodal posterior distributions. Parallel tempering is an advanced Markov chain Monte Carlo method suited for irregular and multimodal posterior distributions. In this paper, we extend Bayeslands using parallel tempering with high-performance computing to address previous limitations in Bayeslands. Our results show that parallel tempering Bayeslands not only reduces the computation time, but also provides an improvement in sampling multimodal posterior distributions, which motivates future application to continental scale landscape evolution models.

Chandra R; Müller RD; Azam D; Deo R; Butterworth N; Salles T; Cripps S, 2019, 'Multicore Parallel

Tempering Bayeslands for Basin and Landscape Evolution', *Geochemistry, Geophysics, Geosystems*, vol. 20, pp. 5082 - 5104, http://dx.doi.org/10.1029/2019GC008465 https://github.com/rohitash-chandra/research/blob/master/2019/Bayeslands_GCubed2019.pdf

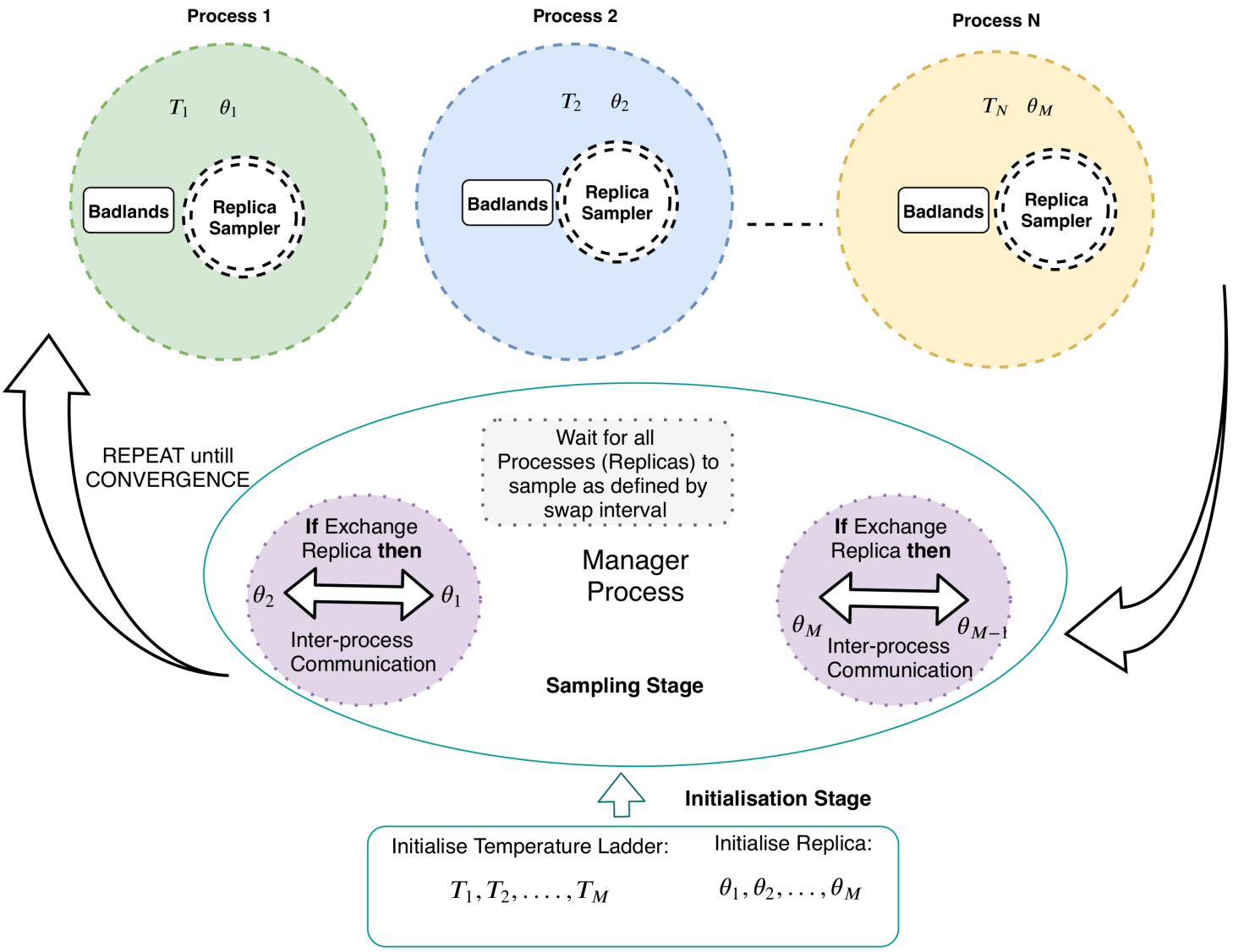
The implementation of Bayesian inference faces challenges as the number of parameters in models became larger. Complex and multimodal posterior as shown below gives further challenges.



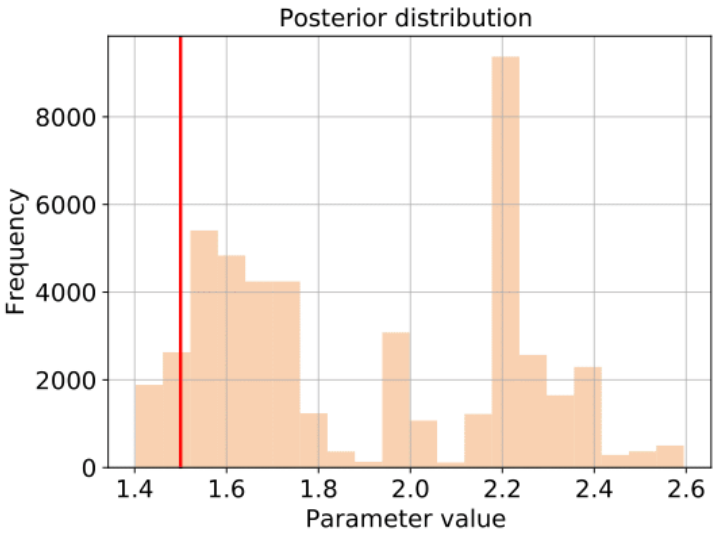
The figure above shows complex posterior (likelihood surface) for two selected parameters in Badlands model.

Furthermore, the problem becomes more challenging when the model evaluations take considerable computational time; in such case, it could take takes weeks or months for drawing thousands of samples on single-core processing units. Hence, it is important to employ sampling methods that can utilize parallel computing.

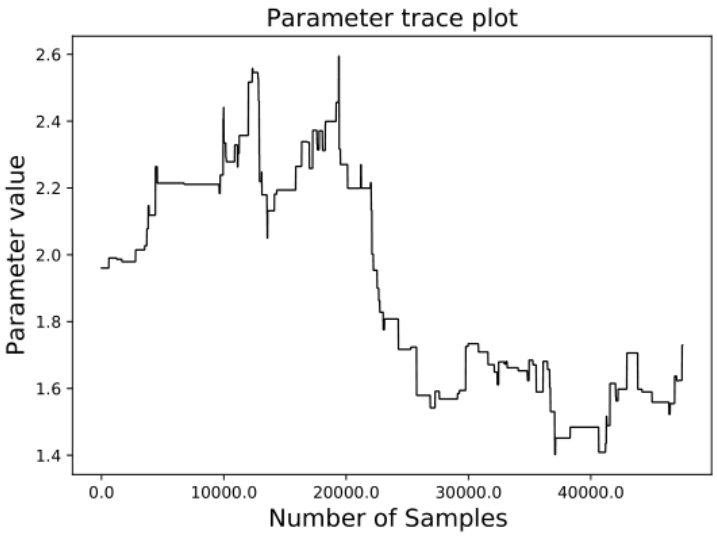
Below is an example where parallel computing is used with parallel tempering MCMC for Bayeslands.



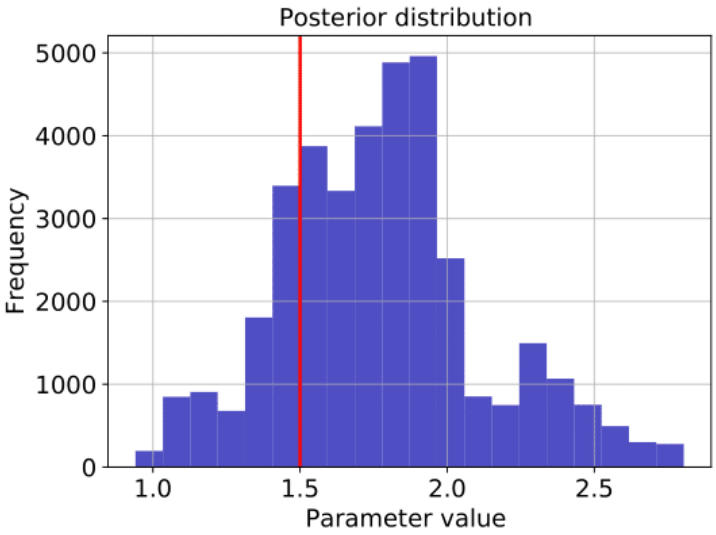
The figure above shows how different replicas are executed in parallel cores.



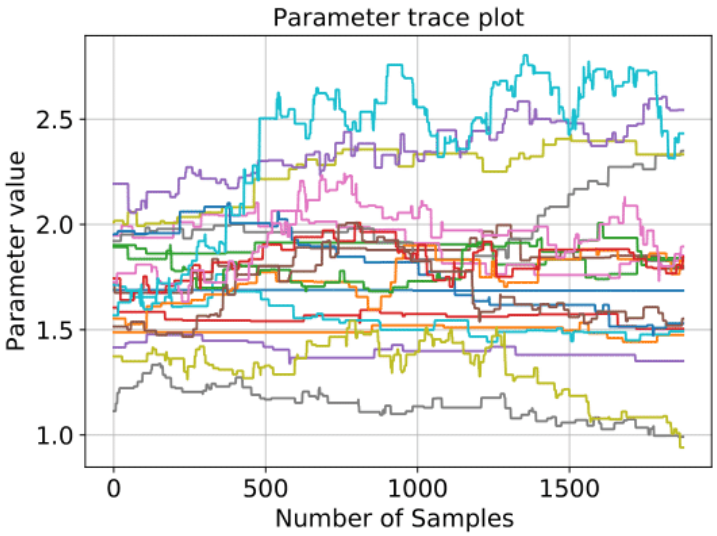
(a) Posterior (SC- Bayeslands)



(b) Trace-plot(SC- Bayeslands)

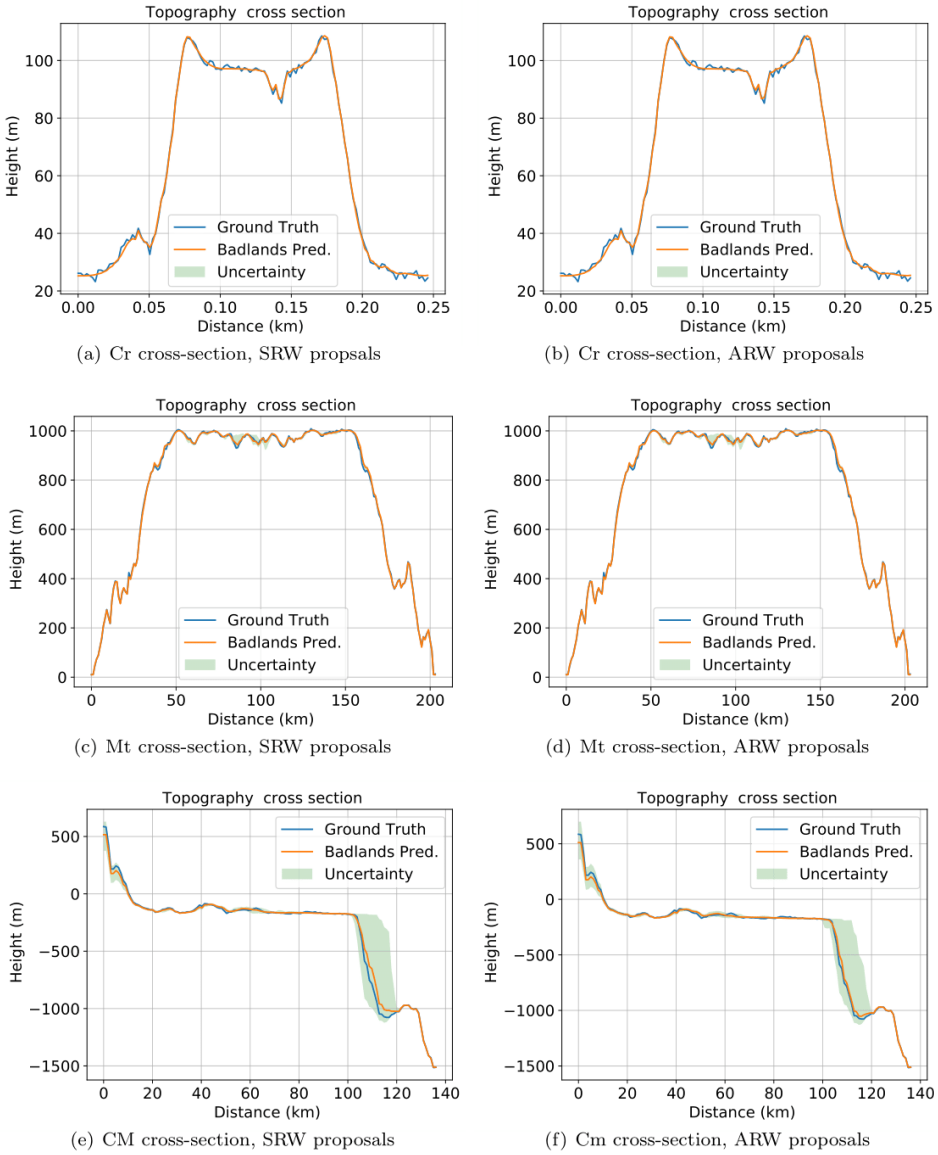


(c) Posterior (PT- Bayeslands)



(d) Trace-plot (PT- Bayeslands)

The figure above shows results that compare single-chain MCMC with parallel tempering MCMC Bayeslands.



(a) Cr cross-section, SRW propsals

(b) Cr cross-section, ARW proposals

(c) Mt cross-section, SRW proposals

(d) Mt cross-section, ARW proposals

(e) CM cross-section, SRW proposals

(f) Cm cross-section, ARW proposals

The figure above shows topography prediction results from Bayeslands.

Other applications

1. Olierook HKH; Scalzo R; Kohn D; Chandra R; Farahbakhsh E; Clark C; Reddy SM; Müller RD, 2020, 'Bayesian geological and geophysical data fusion for the construction and uncertainty quantification of 3D geological models', *Geoscience Frontiers*, http://dx.doi.org/10.1016/j.gsf.2020.04.015
2. Scalzo R; Kohn D; Olierook H; Houseman G; Chandra R; Girolami M; Cripps S, 2019, 'Efficiency and robustness in Monte Carlo sampling for 3-D geophysical inversions with Obsidian v0.1.2: Setting up for success', *Geoscientific Model Development*, vol. 12, pp. 2941 - 2960, http://dx.doi.org/10.5194/gmd-12-2941-2019

MCMC libraries

1. **PyMC3**: A comprehensive python-based statistics and machine learning library featuring MCMC methods, probability distributions, Gaussian process, variational inferences and machine learning libraries via Theano https://docs.pymc.io/ basic tutorial: https://docs.pymc.io/notebooks/api_quickstart.html
2. **Stan:** Computational statistics library available in Python and R: https://cran.r-project.org/web/packages/rstan/vignettes/rstan.html https://mc-stan.org/users/interfaces/rstan
3. **emcee:** emcee is a Python implementation of Goodman & Weare's Affine Invariant Markov chain Monte Carlo (MCMC) Ensemble sampler. It features convergence diagnosis such as autocorrelation. More information: https://emcee.readthedocs.io/en/stable/

References

1. Foreman-Mackey, D., Hogg, D. W., Lang, D., & Goodman, J. (2013). emcee: the MCMC hammer. *Publications of the Astronomical Society of the Pacific*, *125*(925), 306. https://iopscience.iop.org/article/10.1086/670067/pdf
2. Goodman, J., & Weare, J. (2010). Ensemble samplers with affine invariance. *Communications in applied mathematics and computational science*, *5*(1), 65-80. https://msp.org/camcos/2010/5-1/camcos-v5-n1-p04-s.pdf
3. Shi, J., Chen, J., Zhu, J., Sun, S., Luo, Y., Gu, Y., & Zhou, Y. (2017). Zhusuan: A library for bayesian deep learning. *arXiv preprint arXiv:1709.05870*. https://arxiv.org/abs/1709.05870