Modern Bayesian methods: principles and practice

Vinayak Rao, Purdue University

Jan 5, 2021

Material at `github.com/varao/ds3-bayesian`

Goals

Broadly: To learn some (hopefully new) things and to have some fun

Specific topics: Basics of

- Some foundations of Bayesian thinking
- Stan
- Hierarchical Bayesian modeling
- Bayesian computation
- Bayesian nonparametrics

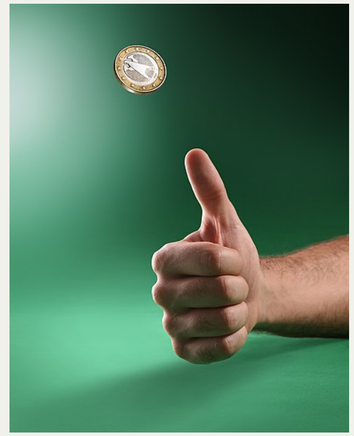
Rather than go in too much depth, the hope to provide you with enough background to learn more on your own

All figures from Wikipedia

What does it mean to be "a Bayesian"?

I'll focus on "subjective Bayesians" to make my point:

- What does someone mean when they say: "The probability of heads on flipping a coin is 0.5"?

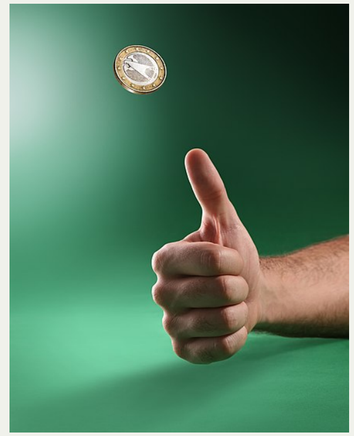


What does it mean to be "a Bayesian"?

I'll focus on "subjective Bayesians" to make my point:

- What does someone mean when they say: "The probability of heads on flipping a coin is 0.5"?

Is coin-flipping really *random*?

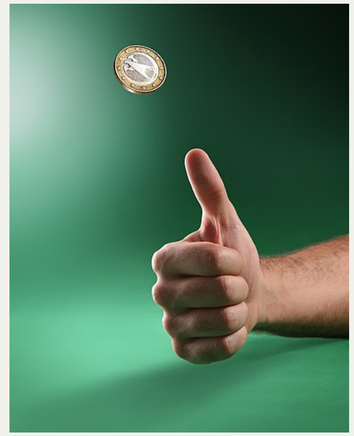


What does it mean to be "a Bayesian"?

I'll focus on "subjective Bayesians" to make my point:

- What does someone mean when they say: "The probability of heads on flipping a coin is 0.5"?

Is coin-flipping really *random*?
Is the frequency of heads on flipping a coin really 0.5?



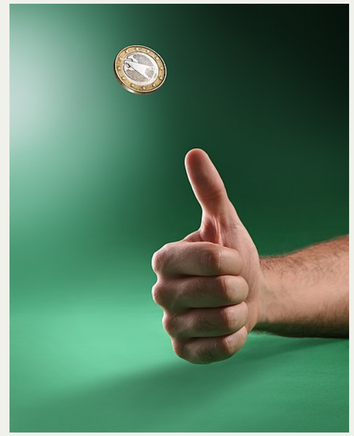
What does it mean to be "a Bayesian"?

I'll focus on "subjective Bayesians" to make my point:

- What does someone mean when they say: "The probability of heads on flipping a coin is 0.5"?

Is coin-flipping really *random*?
Is the frequency of heads on flipping a coin really 0.5?
How does this change as we have more information? E.g.

- What if we know the state of the coin before flipping
- What if we know the exact angular momentum of the coin?



What does it mean to be "a Bayesian"?

I'll focus on "subjective Bayesians" to make my point:

- What does someone mean when they say: "The probability of heads on flipping a coin is 0.5"?

Is coin-flipping really *random*?
Is the frequency of heads on flipping a coin really 0.5?
How does this change as we have more information? E.g.

- What if we know the state of the coin before flipping
- What if we know the exact angular momentum of the coin?

Bottomline: To a Bayesian,

- Probability quantifies beliefs over events
- Probabilities should be updated with new information

The two central points of Bayesian statistics

Why is probability the ‘right’ way to represent+manipulate beliefs?

Lots of justifications: Dutch book theorems, Decision-theoretic arguments, De Finetti’s theorem for exchangeability etc.

Why is probability the ‘right’ way to represent+manipulate beliefs?

Lots of justifications: Dutch book theorems, Decision-theoretic arguments, De Finetti’s theorem for exchangeability etc.

Unlike a frequentist, a Bayesian has no conceptual problems with statements like:

- ”The 50th digit of π is 7 with probability 0.1”?
- ”A 69 percent chance that Biden will win the 2020 election”?
- "The average age of a ML PhD student follows a Gaussian distribution with mean 27"
- "The unknown R_0 for the new strain of COVID follows a Gaussian with mean 1.5"

A cartoon view of Bayesian inference

Let θ represent the state of the world/the quantity of interest

- $p(\theta)$ Prior beliefs about the world
- $p(x|\theta)$ Likelihood: belief about data generation process

A cartoon view of Bayesian inference

Let θ represent the state of the world/the quantity of interest

- $p(\theta)$ Prior beliefs about the world
- $p(x|\theta)$ Likelihood: belief about data generation process

$$\underbrace{p(\theta|x)}_{\text{Posterior}} \propto \underbrace{p(\theta)}_{\text{Prior}} \underbrace{p(x|\theta)}_{\text{Likelihood}} \qquad \text{(Bayes' Rule)}$$

(Bayes' rule is just basic probability: $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$)

A cartoon view of Bayesian inference

Let θ represent the state of the world/the quantity of interest

- $p(\theta)$ Prior beliefs about the world
- $p(x|\theta)$ Likelihood: belief about data generation process

$$\underbrace{p(\theta|x)}_{\text{Posterior}} \propto \underbrace{p(\theta)}_{\text{Prior}} \underbrace{p(x|\theta)}_{\text{Likelihood}} \qquad \text{(Bayes' Rule)}$$

(Bayes' rule is just basic probability: $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$)

The posterior summarizes all prior knowledge together with the data

A cartoon view of Bayesian inference

Let θ represent the state of the world/the quantity of interest

- $p(\theta)$ Prior beliefs about the world
- $p(x|\theta)$ Likelihood: belief about data generation process

$$\underbrace{p(\theta|x)}_{\text{Posterior}} \propto \underbrace{p(\theta)}_{\text{Prior}} \underbrace{p(x|\theta)}_{\text{Likelihood}} \quad \text{(Bayes' Rule)}$$

(Bayes' rule is just basic probability: $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$)

The posterior summarizes all prior knowledge together with the data

The end

Why bother introducing a prior over $p(\theta)$?

Why bother introducing a prior over $p(\theta)$?

Pros:

- Can bring prior knowledge to the problem
- The posterior allows you to quantify uncertainty
- Can propagate uncertainty (sequential learning, data fusion etc)

‘Cons’:

- Are subjective (frequentist methods are more objective/involve less assumptions)

Cons:

- Modeling and inference require more careful thought
- Working with a full distribution is computationally more expensive

Bayesianism vs Frequentism

- Frequentist techniques analyze estimators/algorithms over multiple datasets
- Bayesian methods condition on a particular datasets

There is no real conflict: frequentists methods can be used to analyze Bayesian methods

The three steps of Bayesian analysis (Gelman et al, 2013):

Modeling: define a full probability model of all observable and unobservable quantities of the problem, incorporating current knowledge to the best extent

Estimation: given data, estimate the posterior distribution of latent variables

Evaluation: given the posterior distribution, evaluate the fit the model to the data, or the prediction of new data. If it is insufficient, go back to step one.

In [ ]:

```
import pandas as pd
import numpy as np
import scipy as sp
import pystan
import arviz as az
from matplotlib import pyplot as plt
#import numdifftools as nd
```

In [ ]:
```
# Data from https://github.com/ImperialCollegeLondon/covid19model
#covid_agg = pd.read_csv('covid_agg.csv')
covid_agg.head()
```

In [ ]:
```
# Data from https://github.com/ImperialCollegeLondon/covid19model
#covid_agg = pd.read_csv('covid_agg.csv')
covid_agg.head()
```

In [ ]:
```
covid_agg['rate'] = covid_agg.cases/covid_agg.popData2018
covid_agg.rate.hist()
```

Modeling the infection rate

Question: why would we want to model the infection rate?

Modeling the infection rate

Question: why would we want to model the infection rate?

What is a reasonable model (likelihood) for the infection rate?

Modeling the infection rate

Question: why would we want to model the infection rate?

What is a reasonable model (likelihood) for the infection rate?

$x_i \sim \text{Beta}(a, b)$

Modeling the infection rate

Question: why would we want to model the infection rate?

What is a reasonable model (likelihood) for the infection rate?

$x_i \sim \text{Beta}(a, b)$

What is a prior on the parameters a and b?

Modeling the infection rate

Question: why would we want to model the infection rate?

What is a reasonable model (likelihood) for the infection rate?

$$x_i \sim \text{Beta}(a, b)$$

What is a prior on the parameters a and b?

$$a \sim \exp(\lambda_a), \quad b \sim \exp(\lambda_b)$$

Modeling the infection rate

Question: why would we want to model the infection rate?

What is a reasonable model (likelihood) for the infection rate?

$$x_i \sim \text{Beta}(a, b)$$

What is a prior on the parameters a and b?

$$a \sim \exp(\lambda_a), \quad b \sim \exp(\lambda_b)$$

Given our data $x_1, \ldots, x_N$, what is the posterior over (a, b)?

Conjugate priors and intractable posteriors

In general, the posterior distribution is *intractable*, even if the prior and likelihood are nice

Intractable: can be evaluated only up a multiplicative factor

- $p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$
- The marginal probability of the data $p(X) = \int p(X|\theta)p(\theta)d\theta$ involves an *intractable* integral

Conjugate priors and intractable posteriors

In general, the posterior distribution is *intractable*, even if the prior and likelihood are nice

Intractable: can be evaluated only up a multiplicative factor

- $p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$
- The marginal probability of the data $p(X) = \int p(X|\theta)p(\theta)d\theta$ involves an *intractable* integral

Conjugate priors, when the posterior belongs to the same family as the prior, form an exception

Examples:

$x \sim N(\mu, \sigma^2), \quad \mu \sim N(\mu_0, \sigma_0^2)$

$x \sim N(\mu, \sigma^2), \quad (\mu, \sigma^2) \sim \text{Normal-Inverse Wishart}$

Other examples, Beta-Bernoulli, Dirichlet-Multinomial, Gamma-Poisson

How to deal with intractable posteriors?

Questions: what aspects of the posterior do we care about?

Linear functionals including:

- the mean, the mean-square etc: $\int \theta p(\theta|X)d\theta$
- the predictive distribution: $p(x_{new}|X) = \int p(x_{new}|\theta)p(\theta|X)d\theta$

These take the form of expectations $\int h(\theta)p(\theta|X)d\theta$

Bayesian computation is about computing posterior expectations

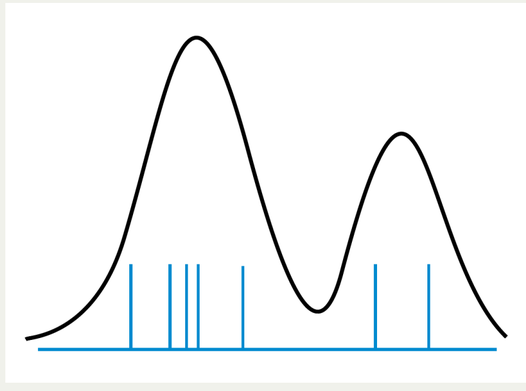
- Usually, these integrals are intractable

Monte Carlo simulation

A standard approach to Bayesian computation

- Draw a bunch of samples $\theta_1, \ldots, \theta_n$ from the distribution of interest
- Approximate the expectation with the sample average:

  $\int h(\theta) p(\theta|X) d\theta \approx \frac{1}{n} \sum_{i=1}^{n} h(\theta_i)$



Advantages:

- Unbiased
- Consistent (the law of large numbers tells us that as the number of samples increase, the approximation gets better).

A brief intro to Stan (from the Stan website)

Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation.

Users specify log density functions in Stan's probabilistic programming language and get:

- full Bayesian statistical inference with MCMC sampling (NUTS, HMC)
- approximate Bayesian inference with variational inference (ADVI)
- penalized maximum likelihood estimation with optimization (L-BFGS)

Stanislaw Ulam

Stan interfaces with the most popular data analysis languages (R, Python, shell, MATLAB, Julia, Stata)

Stan User guide: https://mc-stan.org/docs/2_25/stan-users-guide/index.html

Stan reference manual: https://mc-stan.org/docs/2_25/reference-manual/index.html

PyStan: https://pystan.readthedocs.io/en/latest/api.html

In [ ]:

```
model1_code = """
data {
    int<lower=0> N; // number of countries
    vector[N] x; // proportion of cases
}

parameters {
    real<lower=0> a;
    real<lower=0> b;

}

model {
    a ~ exponential(.0001);
    b ~ exponential(.0001);
    for(i in 1:N) {
      x[i] ~ beta(a,b);
    }
}

"""
sm1 = pystan.StanModel(model_code=model1_code)
```

In [ ]:
```
model1_data = {'N': len(covid_agg.rate), 'x': covid_agg.rate}
#fit = sm1.sampling(data=model1_data, iter=5000, chains=1)
az.plot_density(fit);

print(fit);   #pd.DataFrame(fit.extract())
#pd.DataFrame(fit.extract('a')).plot()
```

Exercise

- Is this a good fit to the data? How would you tell?
- What is the distribution of first observation?

Instead of the infection rate, let us model the log-odds

For a probability of success p, the log odds are $\log(\frac{p}{1-p})$

Instead of the infection rate, let us model the log-odds

For a probability of success p, the log odds are $\log(\frac{p}{1-p})$

In [ ]:
```
covid_agg['logodds'] = np.log(covid_agg.rate/(1-covid_agg.rate))
covid_agg.logodds.hist(bins=20)

# Model x=log-odds as i.i.d. from N(mu, 11^2), m~N(0,10^2)
covid_agg.logodds.min(), covid_agg.logodds.max(), covid_agg.logodds.std()
```

In [ ]:

```
model2_code = """
data {
    int<lower=0> N; // number of countries
    real<lower=0> sigma;
    vector[N] x; // proportion of cases
}

parameters {
    real mu;
}

model {
    mu ~ normal(0, 10);
    for(i in 1:N) {
      x[i] ~ normal(mu,sigma);
    }
}

generated quantities {
    real mn;
    real mx;
    real std;
    {
      vector[N] x_rep;
      for(j in 1:N) {
        x_rep[j] <- normal_rng(mu, sigma);
      }
      std = sd(x_rep);
      mn = min(x_rep);
      mx = max(x_rep);
    }
}
"""
sm2 = pystan.StanModel(model_code=model2_code)
```

In [ ]:
```
model2_data = {'N': len(covid_agg.logodds), 'sigma':11, 'x': covid_agg.logodds}
fit = sm2.sampling(data=model2_data, iter=5000, chains=1)

az.plot_density(fit);

print(fit);
```

Model checking

Is the previous model a good fit of the data?

How can you quantify this?

- Cross-validation
- Posterior predictive checks

Posterior predictive checks (Gelman et al, 2013):

If the model fits, then replicated data generated under the model should look similar to observed data. To put it another way, the observed data should look plausible under the posterior predictive distribution. ... Our basic technique for checking the fit of a model to data is to draw simulated values from the joint posterior predictive distribution of replicated data and compare these samples to the observed data. Any systematic differences between the simulations and the data indicate potential failings of the model.

In [ ]:

```
model3_code = """
data {
    int<lower=0> N; // number of countries
    vector[N] x; // proportion of cases
}

parameters {
    real mu;
    real<lower=0> sigma;

}

model {
    mu ~ normal(0, 10);
    sigma ~ gamma(.01,.01);
    for(i in 1:N) {
      x[i] ~ normal(mu,sigma);
    }
}
"""
sm3 = pystan.StanModel(model_code=model3_code)
```

In [ ]:
```
model3_data = {'N': len(covid_agg.logodds), 'x': covid_agg.logodds}

fit = sm3.sampling(data=model3_data, iter=5000, warmup=100, chains=1)
az.plot_density(fit)
```

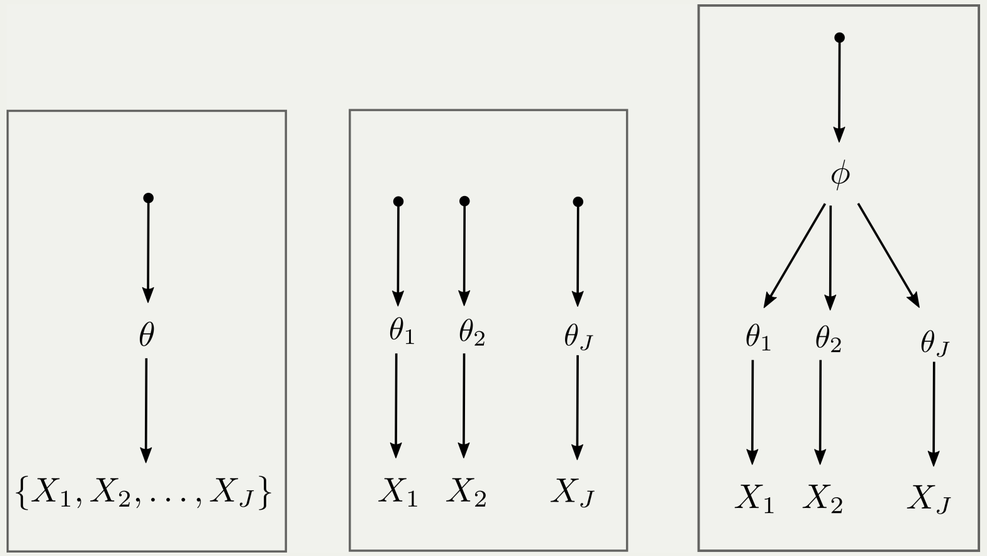
Exercises

- Can we place a prior on the prior mean?
- What properties of the data might the previous model fail to capture?
- Modify the code to quantify this

Hierarchical Bayes

Suppose we have observations grouped into J groups.

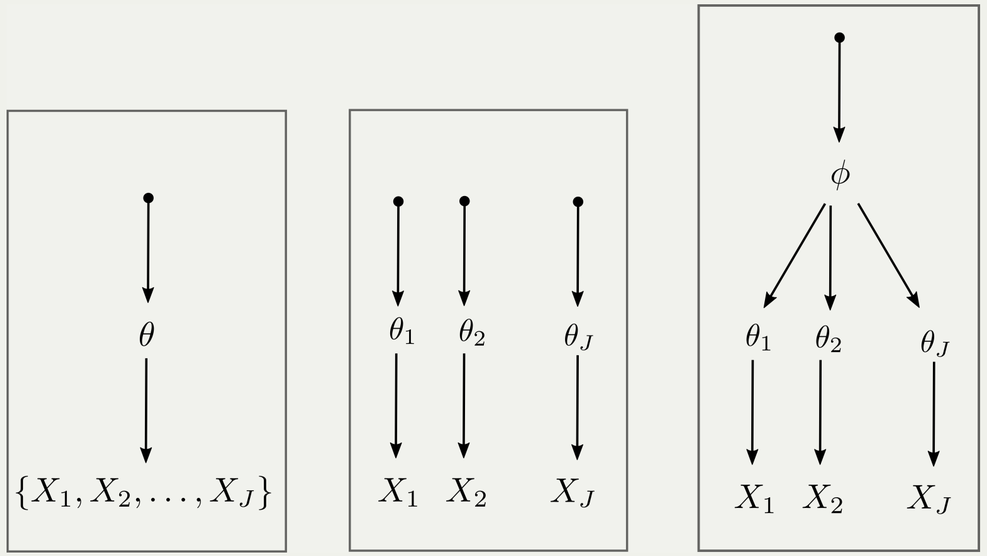
- infection rates in different continents
- rainfall in J states
- topics in different documents



Hierarchical Bayes

Suppose we have observations grouped into J groups.

- infection rates in different continents
- rainfall in J states
- topics in different documents



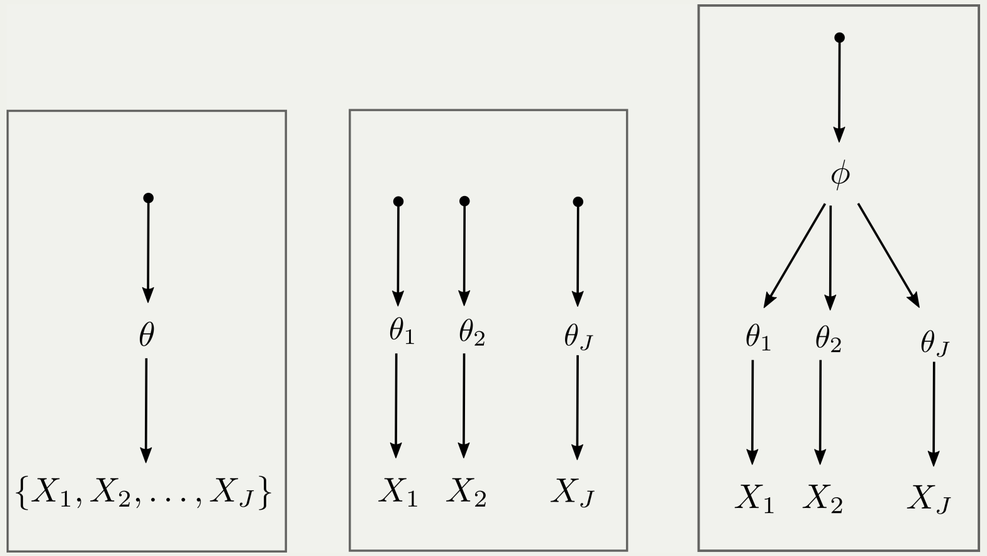
How do we want to model this?

- Collect all observations into a single group (left plot)?
- Model each group separately (middle plot)?

Hierarchical Bayes

Suppose we have observations grouped into J groups.

- infection rates in different continents
- rainfall in J states
- topics in different documents



How do we want to model this?

- Collect all observations into a single group (left plot)?
- Model each group separately (middle plot)?

Bayesian hierarchical modeling (right) allows statistical sharing without losing heterogeneity

Bayesian methods and Big Data

Do we need Bayesian methods in this era of Big Data?

- “With more observations, prior washes out, all uncertainty vanishes and we can use standard approaches.” Assumes a fixed simple model
- More realistically, Big Data = { Large Set of little data sets }

Really need models in which:

- we allow heterogeneity but also statistical sharing among different groups
- the number of parameters grows with the size of the data set (c.f. nonparametrics)

We need to guard from overfitting/represent uncertainty

- a coherent way to do this is to use probabilistic Bayesian models

In [ ]:

```
model_hier_code = """
data {
    int<lower=0> N; // number of countries
    int<lower = 1> C; // number of continents

    vector[N] x; // log-odds of proportion of cases
    int<lower=1, upper=C> cont[N];  // continent
}

parameters {
    real mu0;
    real<lower=0> sigma0;

    vector[C] mu_cont;
    real<lower=0> sigma;
}

model {
    mu0 ~ normal(0, 10);
    sigma0 ~ gamma(.01,.01);
    sigma ~ gamma(.01,.01);

    for(i in 1:C) {
      mu_cont[i] ~ normal(m0,sigma0);
    }
    x ~ normal(mu_cont[cont],sigma);
    // for(i in 1:N) {     x[i] ~ normal(mu_cont[cont[i]],sigma); }
}
"""
sm_hier = pystan.StanModel(model_code=model_hier_code)
```

In [ ]:
```
model_hier_data = {'N': len(covid_agg.logodds), 'C': max(covid_agg.cont), 'x': covid_agg.logodds,
                   'cont': covid_agg.cont}
fit = sm_hier.sampling(data=model_hier_data, iter=5000, warmup=1000, chains=1)
#az.plot_density(fit)
print(fit)
```

Exercise

- What are different ways you can extend the hierarchical model above to allow more flexibility?
- How do you expect the parameter estimates to differ from the situation where each group is modeled independently?
- Write Stan code for the latter case

Markov chain Monte Carlo (MCMC)

Monte Carlo methods translate the problem of posterior computation to posterior simulation

- can still be challenging

Recall we typically know the posterior only up to a multiplicative constant:

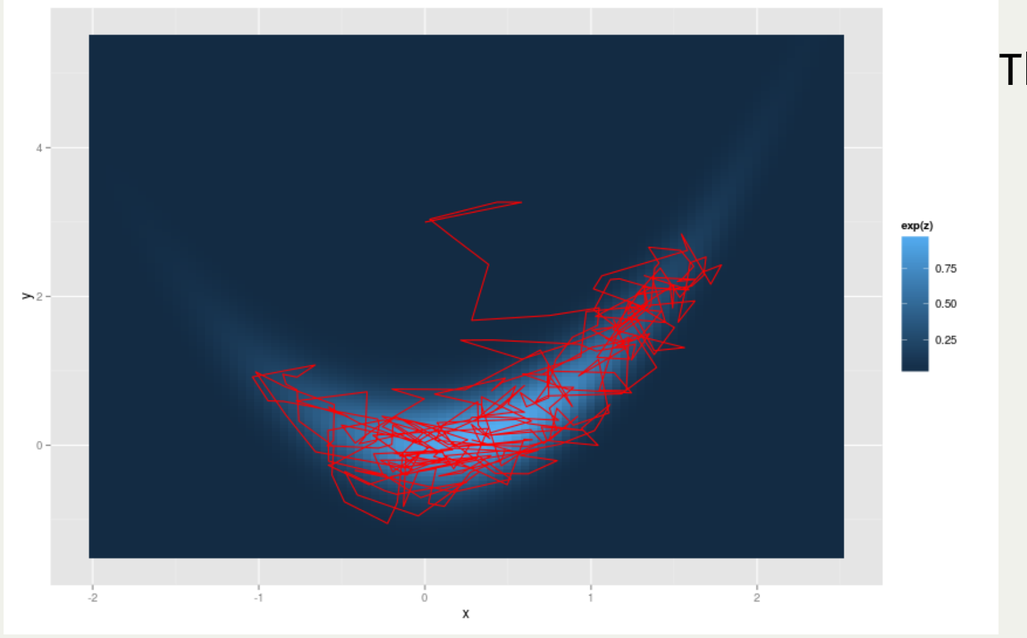
$$p(\theta|X) \propto p(X|\theta)p(\theta)$$

Monte Carlo simulation typically still requires absolute rather than relative probabilities

Solution: MCMC

- Rather than independent samples, produce a sequence of dependent samples
- Each new sample depends on the current sample according to some *transition kernel* $T(\theta_{new}|\theta_{old})$

$$\theta_{i+1} \sim T(\cdot|\theta_i)$$



The Rosenbrock (aka banana) density:

$$\pi(\theta^{(1)}, \theta^{(2)}) \propto \exp -(a - \theta^{(1)})^2 - b(\theta^{(2)} - (\theta^{(1)})^2)^2$$

(here a = .3, b = 3)

MCMC to sample from $\pi(\theta)$ (at a high level):

- Initialize θ_0 from some arbitrary distribution $\pi_0(\theta)$.
- Run your Markov chain for (B + N) iterations, at each step simulating $\theta_{i+1} \sim T(\cdot|\theta_i)$
- Discard the first B 'burn-in' samples.

Calculate average using the remaining N samples: $\mathbb{E}_\pi[h] \approx \frac{1}{N} \sum_{i=B+1}^{B+N} h(\theta_i)$

We need a transition kernel $T(\theta_{new}|\theta_{old})$ that is:

- **Correct**: For any function h, as $N \to \infty$, $\frac{1}{N}\sum_{i=1}^{N} h(\theta_i) \to \mathbb{E}_\pi[h]$ (Ergodicity)
- **Efficient**: Roughly, for any function h, and any finite N, the MCMC average has similar mean and variance as an average using independent samples from π
    - N dependent samples usually has smaller *effective sample size*

Metropolis-Hastings

A random walk algorithm

Choose a proposal distrib. $q(\theta_{new}|\theta_{old})$. E.g. $\theta_{new} \sim N(\theta_{old}, \sigma^2 I)$

Initialize chain at some starting point θ_0 .

Repeat:

- Propose a new point θ^* according to $q(\theta^*|\theta_n)$.
- Define $\alpha = \min\left(1, \frac{\pi(\theta^*)q(\theta_n|\theta^*)}{\pi(\theta_n)q(\theta^*|\theta_n)}\right)$
- Set $\theta_{n+1} = \theta^*$ with probability α, else $\theta_{n+1} = \theta_n$.

Comments:

- Accept/reject steps ensure this has the correct distribution.
- For posterior sampling, $\frac{\pi(\theta^*)}{\pi(\theta)} = \frac{p(\theta^*, X)}{p(\theta, X)}$.
    - Do not need to calculate the normalization constant $p(X)$ which cancels out.
- Keep all samples (i.e. don't discard repetitions)

Comments:

- Accept/reject steps ensure this has the correct distribution.
- For posterior sampling, $\frac{\pi(\theta^*)}{\pi(\theta)} = \frac{p(\theta^*, X)}{p(\theta, X)}$.
    - Do not need to calculate the normalization constant $p(X)$ which cancels out.
- Keep all samples (i.e. don't discard repetitions)

For a symmetric proposal ($q(\theta^*|\theta_n) = q(\theta_n|\theta^*)$):

- $\alpha = \min\left(1, \frac{\pi(\theta^*)}{\pi(\theta)}\right)$

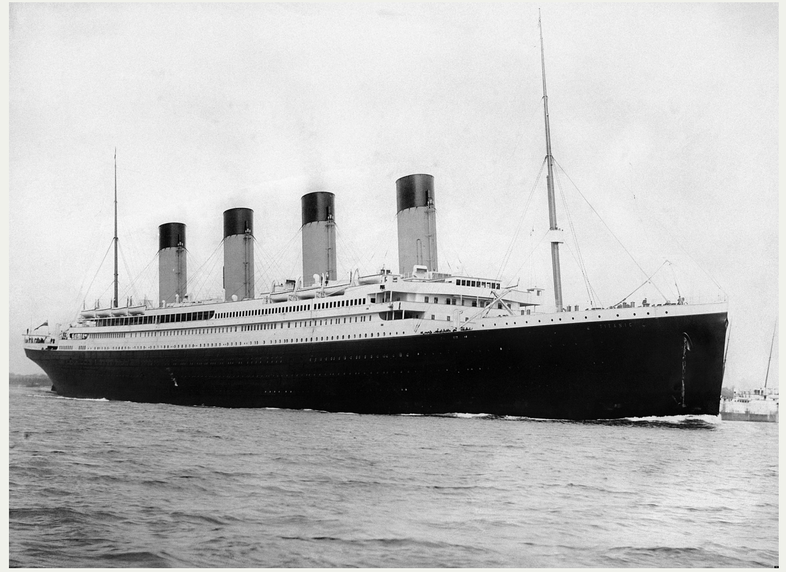
The most common setting is $q(\theta^*|\theta_n) = N(\theta_n, \Sigma)$, where Σ is a parameter of the algorithm.

- How might you set Σ?

The Titanic dataset

Around 800 measurements including:

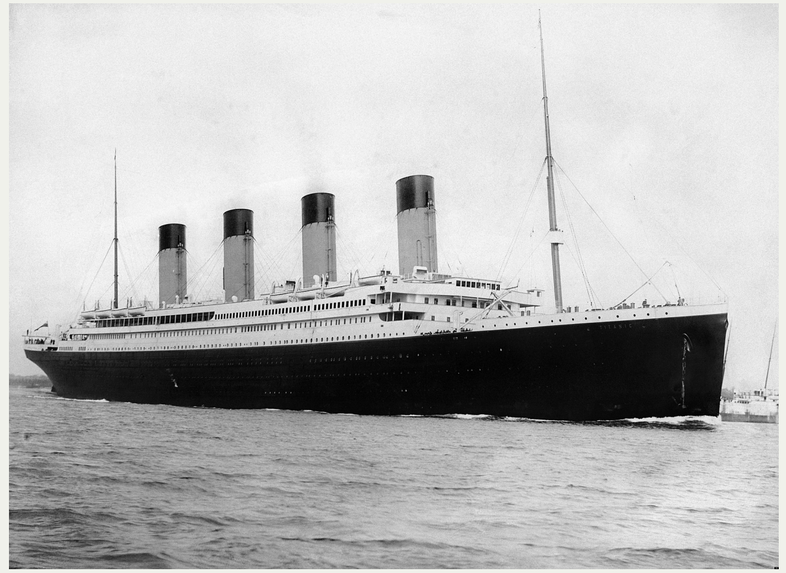
- survival (y): Survival (0 = No; 1 = Yes)
- pclass (x_1): Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- sex (x_2): Sex (1 = female, 0 = male)
- age (x_3): Age
- ticket (x_3): Ticket Number
- fare (x_4): Passenger Fare



The Titanic dataset

Around 800 measurements including:

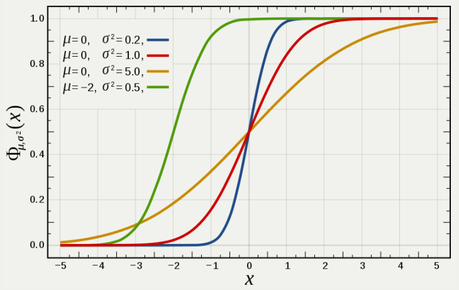
- survival (y): Survival (0 = No; 1 = Yes)
- pclass (x_1): Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- sex (x_2): Sex (1 = female, 0 = male)
- age (x_3): Age
- ticket (x_3): Ticket Number
- fare (x_4): Passenger Fare



In [ ]:
```
titanic = pd.read_csv('titanic.csv')
titanic.head()
```

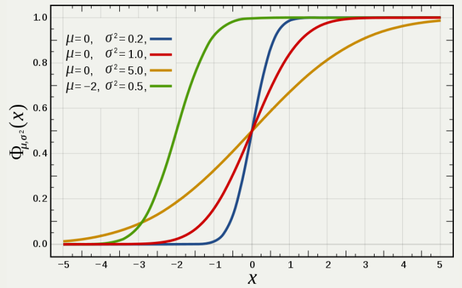
We use probit regression to model y: $p(y = 1|x) = \Phi(\theta^T x)$

Φ is the cumulative distribution function of a standard Gaussian



We use probit regression to model y: $p(y = 1|x) = \Phi(\theta^T x)$

Φ is the cumulative distribution function of a standard Gaussian

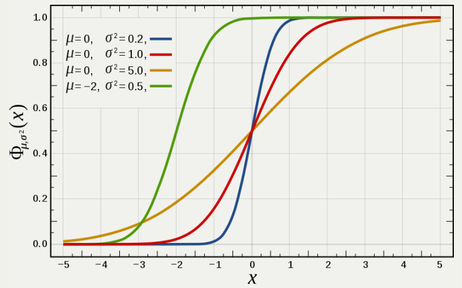


Give a dataset (X, Y), and a prior $p(\theta)$, the posterior distribution is:

$$p(\theta|X, Y) \propto p(\theta)p(Y|\theta, X) = p(\theta) \prod_{i=1}^{N} \Phi(\theta^T x_i)^{y_i} (1 - \Phi(\theta^T x_i))^{1-y_i}$$

We use probit regression to model y: $p(y = 1|x) = \Phi(\theta^T x)$

Φ is the cumulative distribution function of a standard Gaussian



Give a dataset (X, Y), and a prior $p(\theta)$, the posterior distribution is:

$$p(\theta|X, Y) \propto p(\theta)p(Y|\theta, X) = p(\theta) \prod_{i=1}^{N} \Phi(\theta^T x_i)^{y_i} (1 - \Phi(\theta^T x_i))^{1-y_i}$$

Weakly informative prior on $p(\theta)$: Gaussian with mean 0 and standard deviation 100

First standardize inputs:

- Shift inputs to have mean zero and stardard deviation 0.5
- Place independent mean 0, standard deviation 100 Gaussian priors on the coefficients

In [ ]:

```
model_code = """

data {
  int<lower=0> N; // number of obs
  int<lower=0> P; // number of predictors
  int<lower=0,upper=1> y[N]; // outcomes
  matrix[N, P] x; // predictor variables
}
parameters {
  vector[P] theta; // theta coefficients
}
model {
  vector[N] mu;
  theta ~ normal(0, 100);   // cauchy(0,10) is what Gelman et al recommend
  mu <- x*theta;
  for (n in 1:N) mu[n] <- Phi(mu[n]);
  y ~ bernoulli(mu);
}
"""
sm_probit = pystan.StanModel(model_code=model_code)
```

In [ ]:
```
titanic_data = {'N': len(titanic.survived), 'P': titanic.shape[1]-1,
                'x': titanic.drop('survived',axis=1), 'y': titanic.survived}
fit = sm_probit.sampling(data=titanic_data, iter=5000, warmup=1000, chains=1)
```

In [ ]:
```
titanic_data = {'N': len(titanic.survived), 'P': titanic.shape[1]-1,
                'x': titanic.drop('survived',axis=1), 'y': titanic.survived}
fit = sm_probit.sampling(data=titanic_data, iter=5000, warmup=1000, chains=1)
```

In [ ]:
```
az.plot_density(fit)
print(fit)
```

In [ ]:

```
def probit_loglik(theta, x,y):
    mu = x @ theta

    a = y*sp.stats.norm(0,1).logcdf(mu)
    b =  (1-y)*np.log(1-sp.stats.norm(0,1).cdf(mu))
    return sum(a+b)-(theta.T @ theta)/(2*100)

def probit_mh(niter, M, data):
    N, P, x, y = data['N'], data['P'], data['x'], data['y']
    y = y[:,np.newaxis]

    theta = np.zeros([P, niter])

    for i in np.arange(1, niter):
        prop = np.random.multivariate_normal(theta[:,i-1], M)
        prop = prop[:,np.newaxis]
        #print(theta[i,].T)

        #print(probit_loglik(theta[i,].T,x,y))

        if np.log(np.random.uniform()) < (probit_loglik(prop,x,y) - probit_loglik(theta[:,[i-i]],x,y)):
            theta[:,[i]] = prop
        else:
            theta[:,[i]] = theta[:,[i-1]]
    return theta.T

M = np.identity(titanic_data['P'])
#M = np.linalg.inv(iM)
rslt = probit_mh(1000,M/5,titanic_data)
```

In [ ]:
```
rslt.mean(0), rslt.std(0)
#jnk = rslt[1,]
#jnk[:,np.newaxis]
#plt.plot(rslt[:,0])
```

In [ ]:
```
rslt.mean(0), rslt.std(0)
#jnk = rslt[1,]
#jnk[:,np.newaxis]
#plt.plot(rslt[:,0])
```

In [ ]:
```
[az.ess(rslt[:,[i]].T) for i in range(6) ]
```

Exercise

- Play around with the M above. What gives you higher effective sample size (ESS)? How does it compare to Stan's results?
- Use the MCMC code to simulate from a simple distribution (e.g. a Gaussian). How would you ensure your sampler is working correctly?
- Does a high ESS mean a better sampler?
- Does a high acceptance rate mean a better sampler?
- Can the proposal variance also depend on the current location?

Gibbs sampling

Another kind of MCMC algorithm, that makes local moves in a different way

Useful when dealing with multivariate posterior distributions $p(\theta_1, \ldots, \theta_d | X)$

Proceeds by grouping elements of $\theta_1, \ldots, \theta_d$ into groups of smaller components

- Sequentially updates each group of components, keeping the rest fixed.

These conditional distributions are lower-dimensional and often easier to sample from

In the most common setting, each group contains a single component

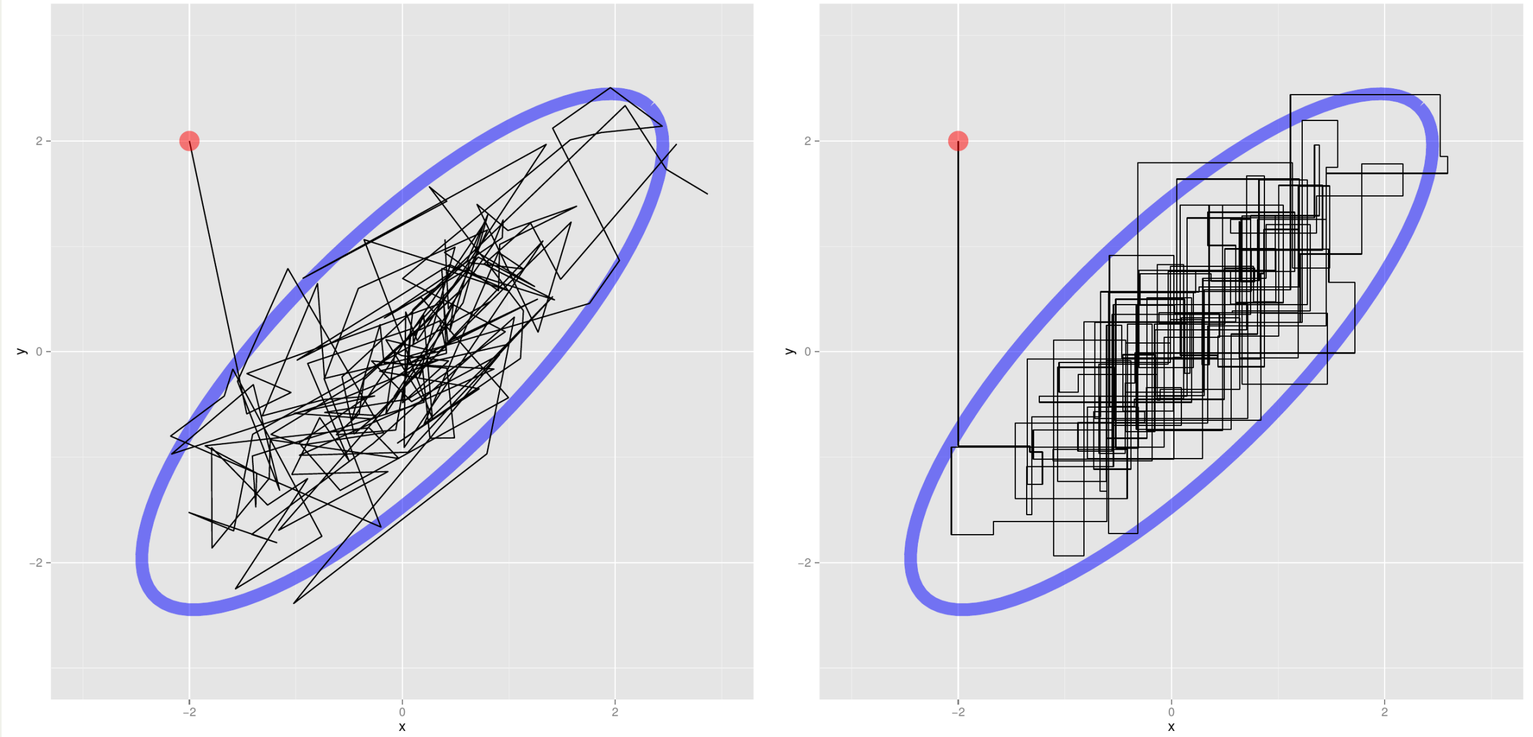
Performance deteriorates with strong coupling between variables.

- Important to group together correlated variables into single conditional update if possible.

Advantages:

- Simple, with no free parameters

Metropolis-Hastings vs Gibbs



Hamiltonian Monte Carlo

A special kind of Metropolis-Hastings algorithm that underlies Stan.

Suppose we want to run MCMC on some distribution $p(\theta)$

Call $U(\theta) = -\log(p(\theta))$ the potential energy of 'position' θ.

Introduce an auxiliary variable ϕ of same dimensionality as θ:

$$\phi \sim N(0, M) \quad \text{(M is a parameter of the algorithm)}$$

Define $K(\phi) = -\log p(\phi) = \frac{1}{2}\phi^T M{-}1\phi + C$.

Call ϕ the momentum, and $K(\phi)$ the kinetic energy.

The total energy (Hamiltonian): $H(\theta, \phi) = U(\theta) + K(\phi)$

$p(\theta, \phi) \propto \exp(-H(\theta, \phi)) = \exp(-U(\theta)) \cdot \exp(-K(\phi)) \propto p(\theta)p(\phi)$

Observe than p(θ, φ) has p(θ) as its marginal.

Run a Markov chain on the augmented space (θ, ϕ).

- Sampling $\phi|\theta$ is easy (how?)
- Sampling $\theta|\phi$ is harder: must produce dependent updates of θ

We will use (deterministic) Hamiltonian dynamics (from physics) to jointly update (θ, ϕ).

- Simulate a system at position θ with momentum ϕ for time T .
- This is a deterministic step!

Hamiltonian dynamics: Physical laws governing evolution of (position,momentum) in a potential energy field

- Laws of physics are Markovian (easy to simulate)
- Energy is conserved
- Laws of physics are reversible

Let (θ, ϕ) be the current configuration.

Simulate 'Hamiltonian' dynamics for some 'time' T giving (θ^*, ϕ^*).

Let $(\theta^*, -\phi^*)$ be the new configuration.

Observe:

- $H(\theta, \phi) = H(\theta^*, -\phi^*)$
- If we started at $(\theta^*, -\phi^*)$, we would end up at (θ, ϕ), i.e. $q(\theta^*, -\phi^*)|(\theta^*, -\phi^*)) = q(\theta^*, -\phi^*)|(\theta^*, -\phi^*)) = 1$

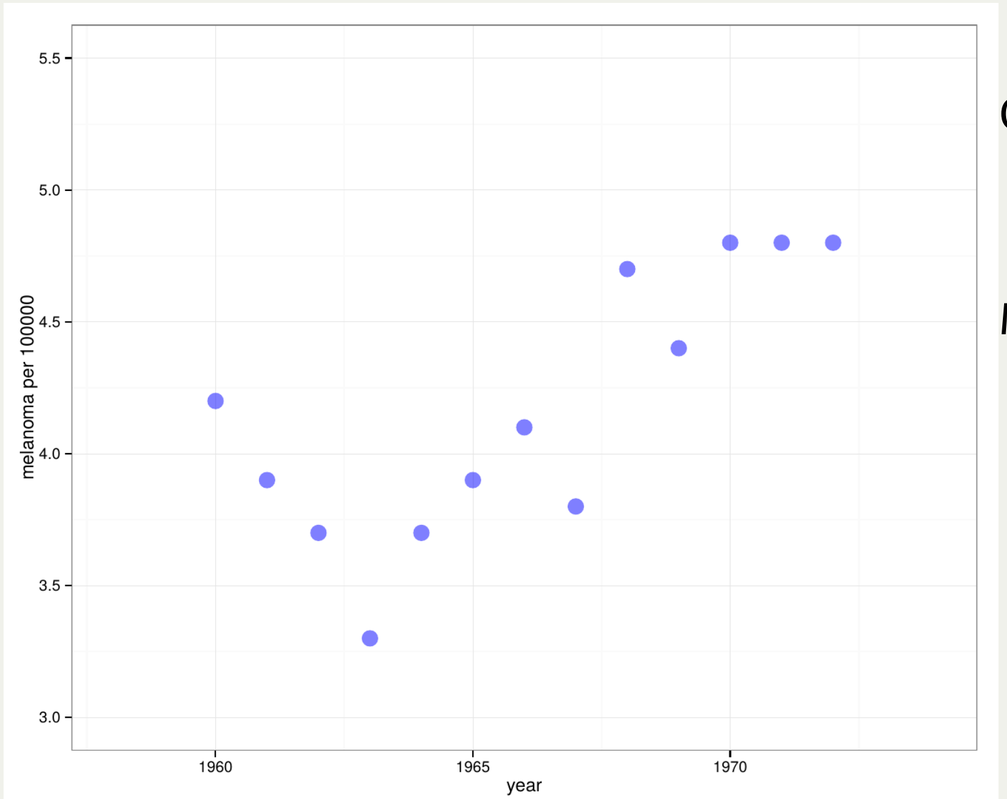
The MH acceptance probability is thus 1. (technical aside: we also need to show that Hamiltonian dynamics are volume-preserving)

Structure of an iteration of idealized HMC:

- Let the current sample be θ_n.
- Sample $\phi_n \sim N(0, M)$.
- Simulate Hamiltonian dynamics to produce (θ^*, ϕ^*).
- Set $(\theta_{n+1}, \phi_{n+1}) = (\theta^*, -\phi^*)$.
- Discard ϕ_{n+1}.

(Since we only keep θ_{n+1} , we don't need to set $\phi_{n+1} = -\phi^*$)

A brief look at Bayesian nonparametrics

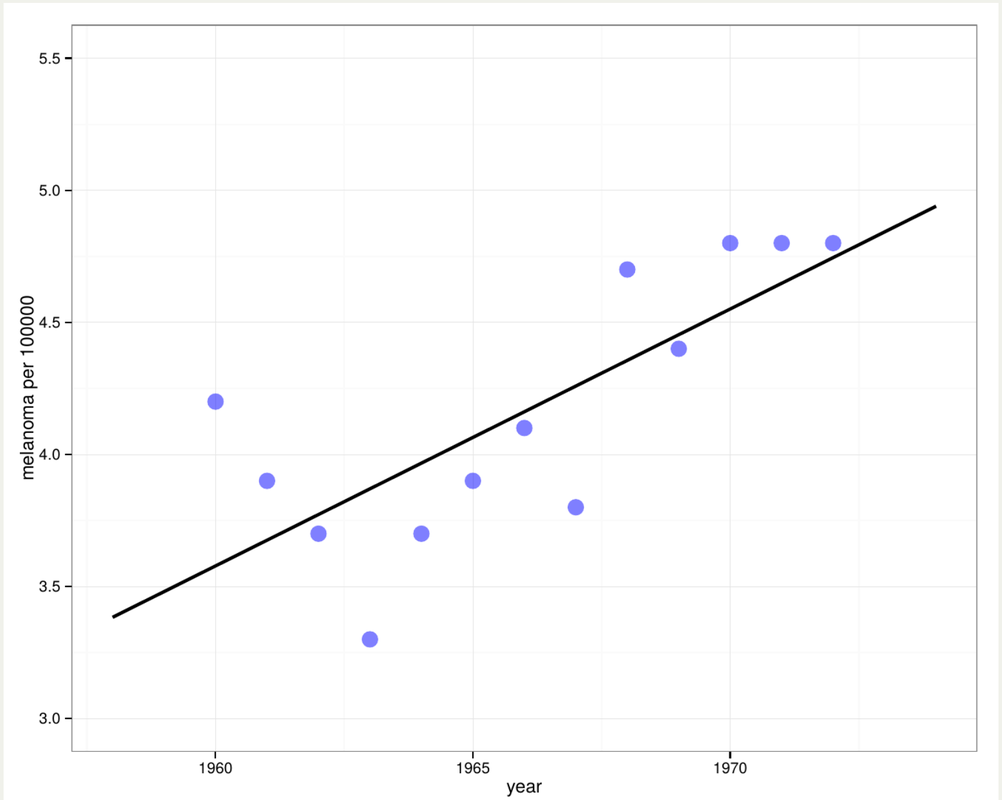


Consider a *regression problem*:

- Given predictors x, want to predict response y

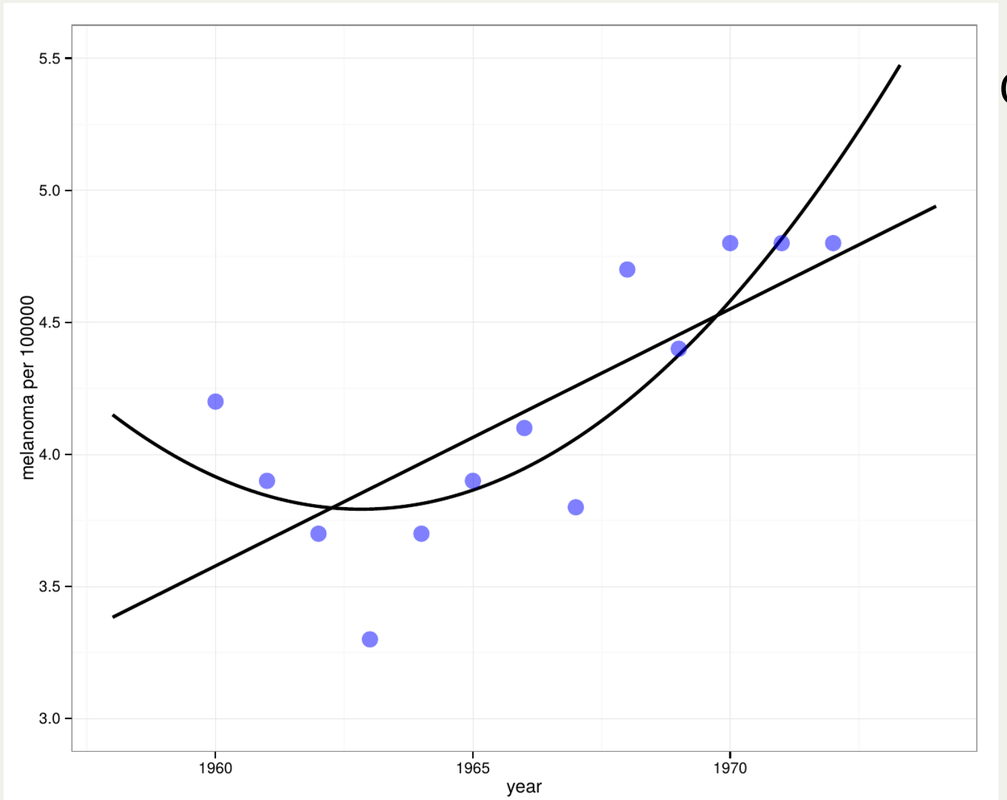
Model data as $y = f(x) + \epsilon$.

- How to place a prior over f ?

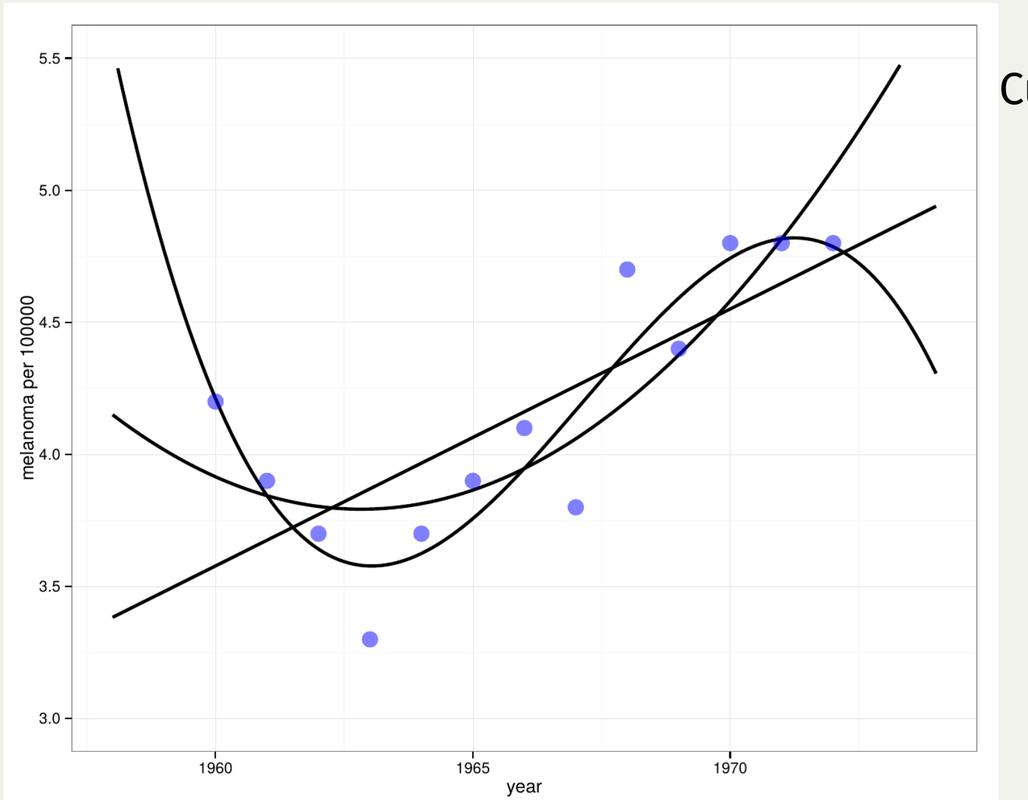


Linear regression: $y = w_1 x + w_0 + \epsilon$

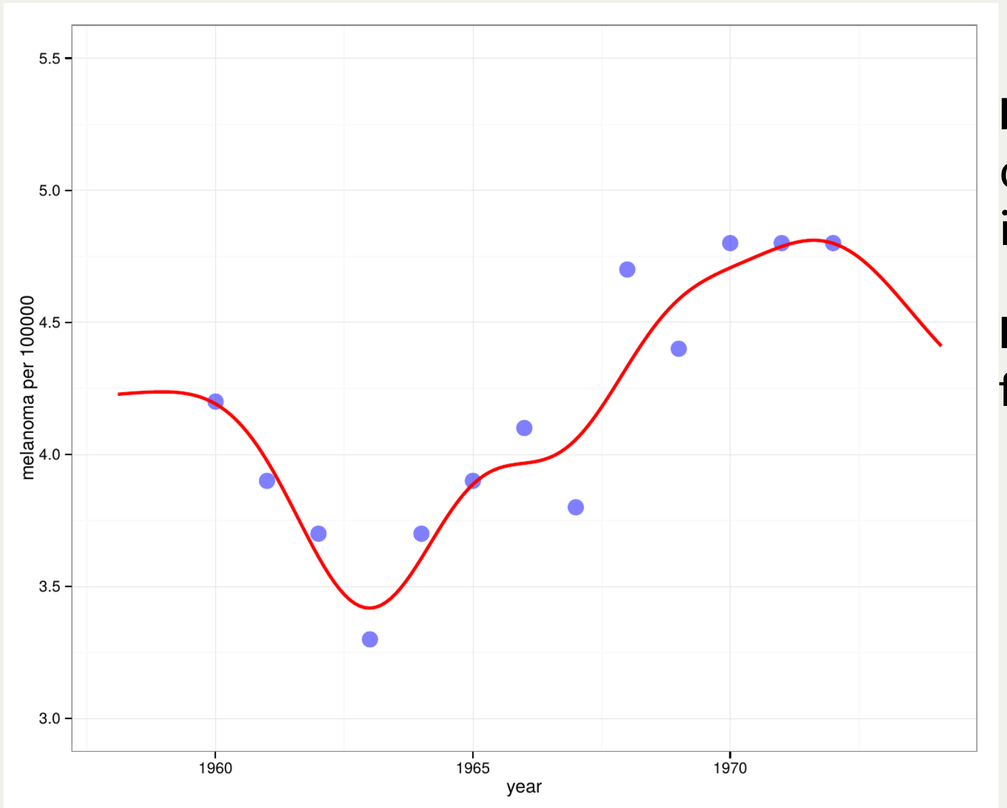
- Place a prior over w_1, w_0



Quadratic regression: $y = w_2 x^2 + w_1 x + w_0 + \epsilon$

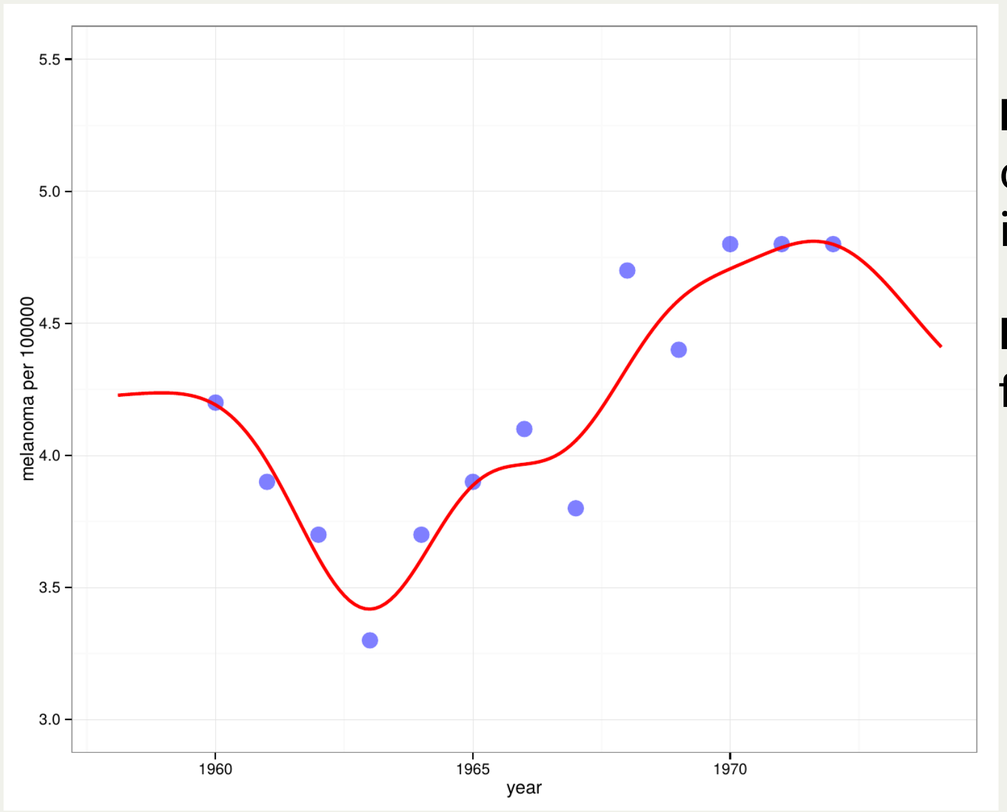


Cubic regression



Nonparametric regression : rather than restrict ourself to some narrow *parametric family* of curves indexed by θ, directly fit a smooth curve f to the data

Nonparametrics Bayes: directly place a prior over functions f



Nonparametric regression : rather than restrict ourself to some narrow *parametric family* of curves indexed by θ, directly fit a smooth curve f to the data

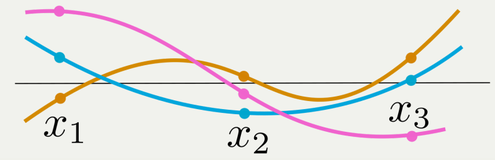
Nonparametrics Bayes: directly place a prior over functions f

How do we interpret/elicit such a prior? Through properties like mean, smoothness, differentiability.

A **Gaussian process** is such a nonparametric prior $p(f)$ over functions $f : \mathbb{X} \to \Re$

- A GP is specified by a mean function $\mu(\cdot)$ and a covariance kernel $K(\cdot, \cdot)$

Definition: For any finite set $X = (x_1, \ldots, x_N), x_i \in \mathbb{X}$, the vector $f_X := (f(x_1), \ldots, f(x_N))$ is distributed as an N-dim Gaussian:



$$f_X \sim N(\mu_X, \Sigma_X)$$

μ is an N-dim vector $(\mu(x_1), \ldots, \mu(x_N))$

Σ_X is an $N \times N$ matrix whose element (i, j) is $K(x_i, x_j)$

It turns out this is a valid stochastic process for any kernel where Σ_X is positive definite for any X

- $\mu(\cdot)$ captures prior knowledge about the average trend in f
- $K(\cdot, \cdot)$ captures smoothness properties

Consider a GP with $\mu(x) = 0$ and $K(x, y) = s \exp(-\frac{1}{2l}(x - y)^p)$

- The code below simulates it on the grid $(0, .01, .02, \ldots, 9.99, 10)$

In [ ]:
```
def my_kernel(X,s,l):
    n = len(X)
    p = 1.5
    D = np.zeos([n,n])

    for i in range(n):
        D[i,] = np.abs((X[i] - X)) ** p

    return s*np.exp(-0.5*D/l) + 1e-5*np.eye(n)  # s * exp((x_i-x_j)/l^ p)

X = np.arange(0,10,step=.01)
cv = my_kernel(X,3,1)

plt.plot(X, np.random.multivariate_normal(X*0,cv))
```

Consider a GP with $\mu(x) = 0$ and $K(x, y) = s \exp(-\frac{1}{2l}(x - y)^p)$

- The code below simulates it on the grid $(0, .01, .02, \dots, 9.99, 10)$

In [ ]:
```
def my_kernel(X,s,l):
    n = len(X)
    p = 1.5
    D = np.zeos([n,n])

    for i in range(n):
        D[i,] = np.abs((X[i] - X)) ** p

    return s*np.exp(-0.5*D/l) + 1e-5*np.eye(n)  # s * exp((x_i-x_j)/l^ p)

X = np.arange(0,10,step=.01)
cv = my_kernel(X,3,1)

plt.plot(X, np.random.multivariate_normal(X*0,cv))
```

Exercise: play around with the parameters s,l,p of the the kernel. Note down your conclusions

Effectively a GP is an infinite-dimensional multivariate normal define on the entire input space

- define an infinitely fine mesh on the input space
- the component of the normal associated with meshpoint x is $f(x)$

Despite the sample realizations being infinitely complex, computation can be carried out in finite time

Follows from the tractability of the Gaussian distribution:

- suppose (f_A, f_B) is a sample from a normal

We can first sample f_A, and then $f_B|f_A$

For GPs, f_A is the function values on some finite e.g. training data A

f_B are the function values on some other finite set of points B (e.g. test data)

If we have posterior samples of f_A, we can easily simulate $f_B|f_A$ for any B

In [ ]:

```
model_code = """
data {
  int<lower=1> N;
  real x[N]; vector[N] y;
}
parameters {
  real<lower=0> rho; real<lower=0> alpha; real<lower=0> sigma;
  vector[N] eta;
}
transformed parameters {
  vector[N] f;
  {
    matrix[N, N] L_K;
    matrix[N, N] K = cov_exp_quad(x, alpha, rho);

    for (n in 1:N)
      K[n, n] = K[n, n] + 1e-4;
    L_K = cholesky_decompose(K);
    f = L_K * eta;
  }
}
model {
  real sq_sigma = square(sigma);
  eta ~ std_normal();
  rho ~ inv_gamma(5, 5);
  alpha ~ std_normal();
  sigma ~ std_normal();
  y ~ normal(f, sq_sigma);
}
"""

sm_gp = pystan.StanModel(model_code=model_code)
```

In [ ]:
```
model_gp_data = {'N': len(covid_agg.rate), 'y': np.log(covid_agg.cases), 'x': np.log(covid_agg.popData2018),
                 'cont': covid_agg.cont}
fit = sm_gp.sampling(data=model_gp_data, iter=1000,  chains=1)
#az.plot_density(fit)
```

In [ ]:
```
tmp = fit.extract('f')['f']
mn = tmp.mean(axis=0)
sd = tmp.std(axis=0)

rslt = pd.DataFrame({'x': model_gp_data['x'], 'mn':mn, 'uci': mn+3*sd, 'lci': mn-3*sd})
rslt =  rslt.sort_values(by='x')


plt.plot(rslt.x,rslt.mn)
plt.plot(rslt.x,rslt.uci)
plt.plot(rslt.x,rslt.lci)
plt.scatter(rslt.x, model_gp_data['y'])
```

Conclusions and further directions

This was a whirlwind tour of some of the principles and practice of modern Bayesian statistics

Hopefully you have learned enough to come up with and play around with your own Bayesian models

- With Stan, you no longer have to derive and code up a new MCMC sampler for each model

Currently, Stan's engine is HMC (really NUTS)

- Does not work with discrete latent variables

Another probabilistic language in PyMc3

- Written specifically for Python (good and bad)
- Handles discrete variables
- Scales better to large datasets (?)

Finally, there is a lot of theory and methodology I have not covered

- A big one is variational Bayes (an alternative to MCMC)

Some good reference are:

- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). Bayesian data analysis. CRC press.
- Robert, C., & Casella, G. (2013). Monte Carlo statistical methods. Springer Science & Business Media.
- Williams, C. K., & Rasmussen, C. E. (2006). Gaussian processes for machine learning (Vol. 2, No. 3, p. 4). Cambridge, MA: MIT press
- The internet!