

# Posterior inference

Joachim Vandekerckhove

# Posterior inference

- ▶ The posterior distribution gives a lot of information

# Posterior inference

- ▶ The posterior distribution gives a lot of information
- ▶ Ideally, we could just show a graph of it and leave the interpretation to the reader

# Posterior inference

- ▶ The posterior distribution gives a lot of information
- ▶ Ideally, we could just show a graph of it and leave the interpretation to the reader
  - ▶ But often the posterior will have many dimensions

# Posterior inference

- ▶ The posterior distribution gives a lot of information
- ▶ Ideally, we could just show a graph of it and leave the interpretation to the reader
  - ▶ But often the posterior will have many dimensions
  - ▶ And also that seems lazy

# Posterior inference

- ▶ The posterior distribution gives a lot of information
- ▶ Ideally, we could just show a graph of it and leave the interpretation to the reader
  - ▶ But often the posterior will have many dimensions
  - ▶ And also that seems lazy
- ▶ We need a way to describe the posterior distribution

# Posterior inference

- ▶ The posterior distribution gives a lot of information
- ▶ Ideally, we could just show a graph of it and leave the interpretation to the reader
  - ▶ But often the posterior will have many dimensions
  - ▶ And also that seems lazy
- ▶ We need a way to describe the posterior distribution
  - ▶ Mean? SD? Skew? Kurtosis? Mass at or around a certain value?  $p(.8 \leq P_R \leq .9 | \#R, \#W)$ ?

## Posterior inference

Determining these summary statistics analytically can be daunting and may in fact not be possible in general.



## Posterior inference

Determining these summary statistics analytically can be daunting and may in fact not be possible in general.

Here, with the discrete domain of  $P_R$ :

$$\begin{aligned} p(.8 \leq P_R \leq .9 | \#R, \#W) = \\ p(P_R = .80 | \#R, \#W) \\ + p(P_R = .85 | \#R, \#W) \\ + p(P_R = .90 | \#R, \#W) \end{aligned}$$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$

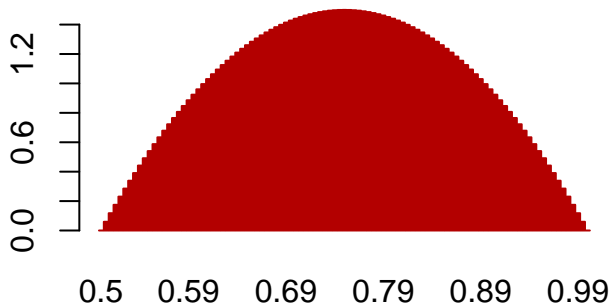
## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$

**wine**



## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$



## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$
  - ▶ Likelihood is the same:  $p(\#R, \#W|P_R) = CP_R^{\#R}(1-P_R)^{\#W}$

## Posterior inference

- ▶ A more continuous case on  $x = \frac{P_R+1}{2}$ 
  - ▶  $p(x) = Kx^{\alpha-1}(1-x)^{\beta-1} = B(x|\alpha, \beta)$
  - ▶ Beta distribution, say  $\alpha = \beta = 2$
  - ▶ Likelihood is the same:  $p(\#R, \#W|P_R) = CP_R^{\#R}(1-P_R)^{\#W}$
  - ▶ So the posterior must be:  
$$p(P_R|\#R, \#W) \propto \left(\frac{P_R+1}{2}\right)^{\alpha-1} \left(1 - \frac{P_R+1}{2}\right)^{\beta-1}$$

# Functional programming

Sometimes it is useful in R to turn a function into a variable to change it quickly

You can make a function “on the fly” inside a function or script file like this:

```
funcname <- function(n, x) { rep(x, n) }
```

So that prior and likelihood can be written like:

```
prior <- function(p) { dbeta(2 * (p-.5), 2, 2) }  
likelihood <- function(p) { dbinom(5, 6, p) }
```

# Functional programming

Prior and likelihood:

```
prior <- function(p) { dbeta(2 * (p-.5), 2, 2) }  
likelihood <- function(p) { dbinom(5, 6, p) }
```

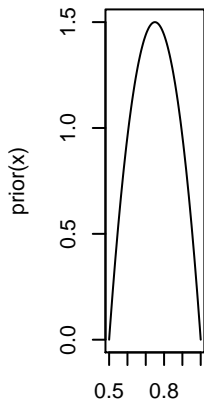
Given those, building the posterior is trivial:

```
post.usc <- function(p) { prior(p) * likelihood(p) }
```

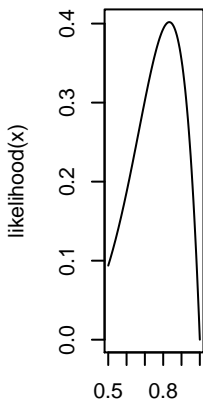
Exercise: implement this, plot the three functions

# Functional programming

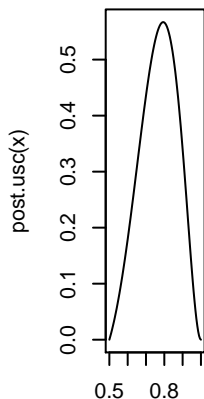
```
par(mfrow=c(1,3))  
curve(prior, 0.5, 1)  
curve(likelihood, 0.5, 1)  
curve(post.usc, 0.5, 1)
```



x



x



x

## Posterior inference

- ▶ These graphs don't tell us the mean of the posterior (or any other useful statistic)

# Posterior inference

- ▶ These graphs don't tell us the mean of the posterior (or any other useful statistic)
- ▶ How do we determine the mean of an arbitrary, somewhat complicated function?



# Posterior inference

- ▶ These graphs don't tell us the mean of the posterior (or any other useful statistic)
- ▶ How do we determine the mean of an arbitrary, somewhat complicated function?
- ▶ As it turns out, drawing random samples from a distribution is an efficient way to do that

# Posterior inference

- ▶ These graphs don't tell us the mean of the posterior (or any other useful statistic)
- ▶ How do we determine the mean of an arbitrary, somewhat complicated function?
- ▶ As it turns out, drawing random samples from a distribution is an efficient way to do that
  - ▶ Methods for doing this are called Monte Carlo methods

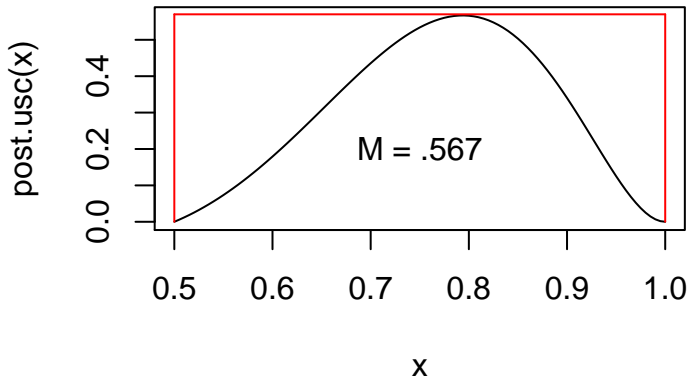
# Posterior inference

- ▶ These graphs don't tell us the mean of the posterior (or any other useful statistic)
- ▶ How do we determine the mean of an arbitrary, somewhat complicated function?
- ▶ As it turns out, drawing random samples from a distribution is an efficient way to do that
  - ▶ Methods for doing this are called Monte Carlo methods
  - ▶ Math win: Monte Carlo methods don't need those hard-to-compute  $K$  and  $C$  scaling constants

## Posterior sampling

One Monte Carlo method is the rejection sampler:

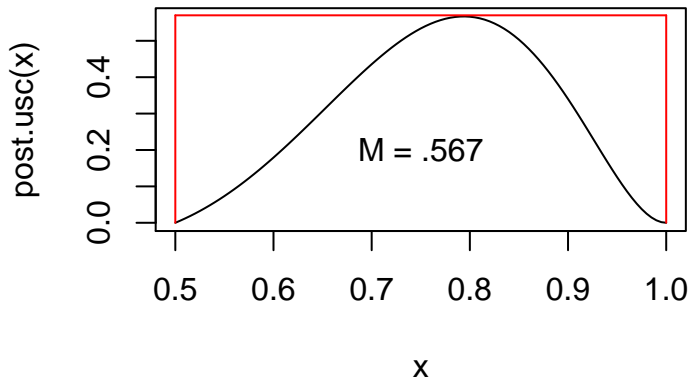
- 1) Draw a sample from some basic distribution  $S(x|...)$



## Posterior sampling

One Monte Carlo method is the rejection sampler:

- 1) Draw a sample from some basic distribution  $S(x|...)$
- 2) Reject the sample with probability  $q = \frac{f_x}{M \times S(x)}$ , where  $M$  is chosen so that this is always  $\leq 1$  (but ideally sometimes close to 1)



# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm

# Posterior sampling

► Exercise: Implement a rejection algorithm

1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$

# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm
- 1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$
- 2. Check if  $u < p(P_R|\#R, \#W)/M$  (or, better, if  $uM < p(P_R|\#R, \#W)$ )



# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm
- 1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$
- 2. Check if  $u < p(P_R | \#R, \#W) / M$  (or, better, if  $uM < p(P_R | \#R, \#W)$ )
  - ▶ If true, accept  $x$

# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm
- 1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$
- 2. Check if  $u < p(P_R|\#R, \#W)/M$  (or, better, if  $uM < p(P_R|\#R, \#W)$ )
  - ▶ If true, accept  $x$
  - ▶ Otherwise, reject the value and draw a new sample

# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm
- 1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$
- 2. Check if  $u < p(P_R | \#R, \#W) / M$  (or, better, if  $uM < p(P_R | \#R, \#W)$ )
  - ▶ If true, accept  $x$
  - ▶ Otherwise, reject the value and draw a new sample
- 3. Repeat many times to get a few thousand samples

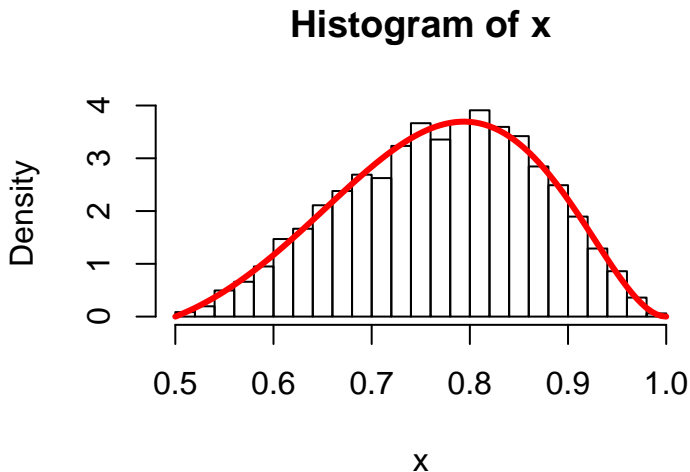
# Posterior sampling

- ▶ Exercise: Implement a rejection algorithm
- 1. Sample  $P_R$  from  $g(P_R) = U(0.5, 1.0)$  and  $u$  from  $U(0, 1)$
- 2. Check if  $u < p(P_R|\#R, \#W)/M$  (or, better, if  $uM < p(P_R|\#R, \#W)$ )
  - ▶ If true, accept  $x$
  - ▶ Otherwise, reject the value and draw a new sample
- 3. Repeat many times to get a few thousand samples
- 4. Make a histogram and compare the shapes of the distribution

## Posterior sampling

```
N <- 10000
x <- vector(,N)
c <- 1
M <- .567
while(c <= N) {
  x[c] <- runif(1, 0.5, 1)
  u <- runif(1, 0, M)
  if (u < post.usc(x[c])) c <- c + 1;
}
hist(x, breaks=25,freq=FALSE)
K <- integrate(post.usc, 0.5, 1)$value
lines(domain, post.usc(domain)/K, lwd=3, col='red')
```

## Posterior sampling



# Posterior sampling

- ▶ With a few thousand samples, the shape of the posterior is well approximated

# Posterior sampling

- ▶ With a few thousand samples, the shape of the posterior is well approximated
  - ▶ Now we can compute the mean of that sample: 0.7699475



# Posterior sampling

- ▶ With a few thousand samples, the shape of the posterior is well approximated
  - ▶ Now we can compute the mean of that sample: 0.7699475
  - ▶ ... or the proportion of samples that are  $> .85$ : 0.2279

# Posterior sampling

- ▶ With a few thousand samples, the shape of the posterior is well approximated
  - ▶ Now we can compute the mean of that sample: 0.7699475
  - ▶ ... or the proportion of samples that are  $> .85$ : 0.2279
  - ▶ ... or indeed any quality we fancy

# Generative model representation

- ▶ Basic unit of a Bayesian model is a distribution function

# Generative model representation

- ▶ Basic unit of a Bayesian model is a distribution function
- ▶ The posterior is defined through a *generative model representation*

# Generative model representation

- ▶ Basic unit of a Bayesian model is a distribution function
- ▶ The posterior is defined through a *generative model representation*
  - ▶ ... which is basically a sequence of distributional assumptions

# Generative model representation

- ▶ Basic unit of a Bayesian model is a distribution function
- ▶ The posterior is defined through a *generative model representation*
  - ▶ ... which is basically a sequence of distributional assumptions
  - ▶ In Bayesian statistics, a “model” is just a special kind of distribution function over parameters and data (and so with possibly very many dimensions)

## Generative model representation

Let's define a really trivial model  $\mathcal{M}_t$  in which we estimate the parameters  $\mu$  and  $\tau$  ( $= 1/\sigma^2$ ) of a normal distribution, applied to some data points  $d_j$ :

$$\mathcal{M}_t : \begin{cases} \forall j \in (1, \dots, J) : d_j \sim N(\mu, \tau) \\ \mu \sim N(0, 0.1) \\ \tau \sim \Gamma(4, 0.01) \end{cases}$$

## Generative model representation

Let's define a really trivial model  $\mathcal{M}_t$  in which we estimate the parameters  $\mu$  and  $\tau$  ( $= 1/\sigma^2$ ) of a normal distribution, applied to some data points  $d_j$ :

$$\mathcal{M}_t : \begin{cases} \forall j \in (1, \dots, J) : d_j \sim N(\mu, \tau) \\ \mu \sim N(0, 0.1) \\ \tau \sim \Gamma(4, 0.01) \end{cases}$$

Notice how every statement is a distributional assumption! (Either priors on parameters or likelihoods on data.)



## Generative model representation

Let's define a really trivial model  $\mathcal{M}_t$  in which we estimate the parameters  $\mu$  and  $\tau$  ( $= 1/\sigma^2$ ) of a normal distribution, applied to some data points  $d_j$ :

$$\mathcal{M}_t : \begin{cases} \forall j \in (1, \dots, J) : d_j \sim N(\mu, \tau) \\ \mu \sim N(0, 0.1) \\ \tau \sim \Gamma(4, 0.01) \end{cases}$$

Notice how every statement is a distributional assumption! (Either priors on parameters or likelihoods on data.)

$$\begin{aligned} p(d_1, \dots, d_J \mid \mathcal{M}_t) &= \left( \prod_{j=1}^J N(d_j \mid \mu, \tau) \right) \\ &\times N(\mu \mid 0, 0.1) \\ &\times \Gamma(\tau \mid 4, 0.01) \end{aligned}$$

## JAGS code is (almost) perfect

$$\mathcal{M}_s : \begin{cases} \forall j \in (1, \dots, J) : d_j \sim N(\mu, \tau) \\ \mu \sim N(0, 0.1) \\ \tau \sim \Gamma(4, 0.01) \end{cases}$$

The program needs to know the specifics of the model:

```
model {  
  for (j in 1:J) {  
    d[j] ~ dnorm(mu, tau)  
  }  
  mu ~ dnorm(0,0.1)  
  tau ~ dgamma(4,0.01)  
}
```

## A psychological model: Signal detection theory

$$\mathcal{M}_{sdt} : \begin{cases} \delta \sim N(1, 1) & \beta \sim N(0, 1) \\ \phi_h = \Phi(\delta/2 - \beta) & \phi_f = \Phi(-\delta/2 - \beta) \\ h \sim B(\phi_h, n_s) & f \sim B(\phi_f, n_n) \end{cases}$$

## A psychological model: Signal detection theory

$$\mathcal{M}_{sdt} : \begin{cases} \delta \sim N(1, 1) & \beta \sim N(0, 1) \\ \phi_h = \Phi(\delta/2 - \beta) & \phi_f = \Phi(-\delta/2 - \beta) \\ h \sim B(\phi_h, n_s) & f \sim B(\phi_f, n_n) \end{cases}$$

```
model {  
  d ~ dnorm(1, 1)  
  b ~ dnorm(0, 1)  
  
  phih <- phi( d / 2 - b)  
  phif <- phi(-d / 2 - b)  
  
  h ~ dbin(phih, sigtrials)  
  f ~ dbin(phif, noistrials)  
}
```

## Signal detection theory ~ implementation

```
library(rjags)

data <- list( h = 32 ,  sigtrials  = 39 ,
              f = 12 ,  noistrials = 60 )

modelString = "
  model {
    d ~ dnorm(1, 1)
    b ~ dnorm(0, 1)

    phih <- phi( d / 2 - b)
    phif <- phi(-d / 2 - b)

    h ~ dbin(phih, sigtrials)
    f ~ dbin(phif, noistrials)
  }
"
```

## Signal detection theory ~ implementation

```
writelnLines( modelString , con = "sdt.txt" )

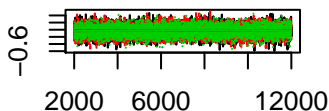
jagsModel = jags.model( file      = "sdt.txt" ,
                        data      =      data ,
                        n.chains  =      3 ,
                        n.adapt   =    1000 )

set.seed(0)
update( jagsModel , n.iter = 1000 ) # burn-in

samples = coda.samples( jagsModel ,
                        variable.names = c("d", "b") ,
                        n.iter        =    10000 )
```

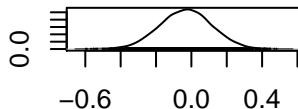
## Signal detection theory ~ results

**Trace of  $b$**



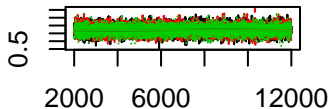
Iterations

**Density of  $b$**



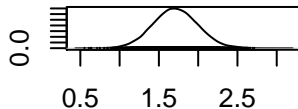
$N = 10000$  Bandwidth = 0.0198

**Trace of  $d$**



Iterations

**Density of  $d$**



$N = 10000$  Bandwidth = 0.0384

## Signal detection theory ~ summary statistics

```
summary(samples)$statistics
```

	Mean	SD	Naive SE	Time-series SE
## b	-0.03151051	0.1470531	0.0008490114	0.001122493
## d	1.71214501	0.2853355	0.0016473855	0.002259586

```
summary(samples)$quantiles
```

	2.5%	25%	50%	75%	97.5%
## b	-0.3202094	-0.1300609	-0.03018976	0.06750984	0.2559907
## d	1.1627554	1.5196783	1.70826988	1.90130744	2.2812061



## Signal detection theory ~ convergence

```
effectiveSize(samples)
```

```
##           b           d  
## 17190.34 16014.55
```

```
gelman.diag(samples)
```

```
## Potential scale reduction factors:
```

```
##
```

```
## Point est. Upper C.I.
```

```
## b           1           1
```

```
## d           1           1
```

```
##
```

```
## Multivariate psrf
```

```
##
```

```
## 1
```

## Summary and conclusion

The outcome of a Bayesian analysis is a probability distribution over parameters of interest

## Summary and conclusion

The outcome of a Bayesian analysis is a probability distribution over parameters of interest

To communicate results, we needed a method to summarize arbitrary distributions

## Summary and conclusion

The outcome of a Bayesian analysis is a probability distribution over parameters of interest

To communicate results, we needed a method to summarize arbitrary distributions

Monte Carlo methods provide such a method

## Summary and conclusion

The outcome of a Bayesian analysis is a probability distribution over parameters of interest

To communicate results, we needed a method to summarize arbitrary distributions

Monte Carlo methods provide such a method

JAGS (and Stan) have a straightforward modeling language that lets users specify a model

## Summary and conclusion

The outcome of a Bayesian analysis is a probability distribution over parameters of interest

To communicate results, we needed a method to summarize arbitrary distributions

Monte Carlo methods provide such a method

JAGS (and Stan) have a straightforward modeling language that lets users specify a model

They then draw samples from the posterior distributions, so we can calculate any summaries we like