# BDA HM3

Piero Birello

January 2025

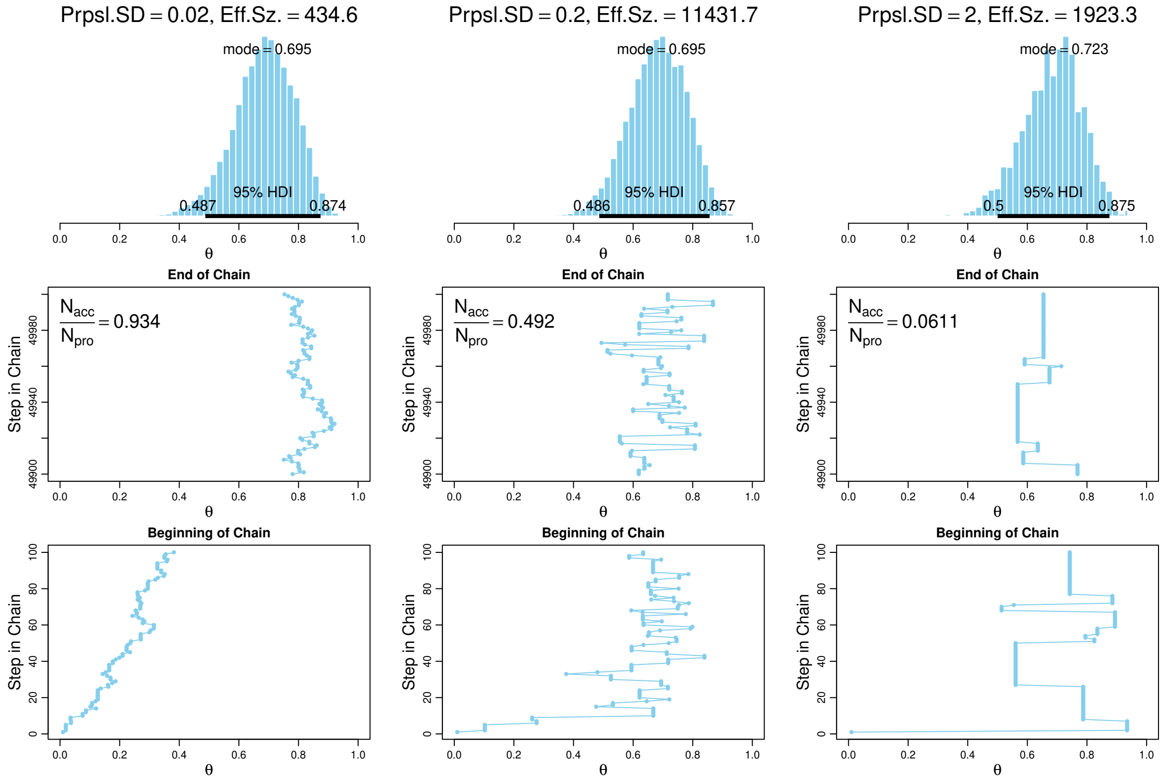
## Code: BernMetropolis.R

```
graphics.off()
rm(list=ls(all=TRUE))
fileNameRoot="Figures/BernMetrop" # for output filenames
source("DBDA2E-utilities.R")

# Specify the data, to be used in the likelihood function.
myData = c(0,1,1)
#myData = c(rep(0,6),rep(1,14))

# Define the Bernoulli likelihood function, p(D|theta).
# The argument theta could be a vector, not just a scalar.
likelihood = function( theta , data ) {
  z = sum( data )
  N = length( data )
  pDataGivenTheta = theta^z * (1-theta)^(N-z)
  # The theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # The likelihood for theta > 1 or for theta < 0 is zero:
  pDataGivenTheta[ theta > 1 | theta < 0 ] = 0
  return( pDataGivenTheta )
}

# Define the prior density function.
prior = function( theta ) {
  #pTheta = dbeta( theta , 1 , 1 )
  pTheta = (cos(4*pi*theta)+1) ** 2/1.5
  # The theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # The prior for theta > 1 or for theta < 0 is zero:
  pTheta[ theta > 1 | theta < 0 ] = 0
  return( pTheta )
}

# Define the relative probability of the target distribution,
# as a function of vector theta. For our application, this
# target distribution is the unnormalized posterior distribution.
targetRelProb = function( theta , data ) {
  targetRelProb =  likelihood( theta , data ) * prior( theta )
  return( targetRelProb )
}

# Specify the length of the trajectory, i.e., the number of jumps to try:
trajLength = 50000 # arbitrary large number
# Initialize the vector that will store the results:
trajectory = rep( 0 , trajLength )
```

```
# Specify where to start the trajectory:
trajectory[1] = 0.99 #0.01 # arbitrary value
# Specify the burn-in period:
burnIn = ceiling( 0.0 * trajLength ) # arbitrary number, less than trajLength
# Initialize accepted, rejected counters, just to monitor performance:
nAccepted = 0
nRejected = 0

# Now generate the random walk. The 't' index is time or trial in the walk.
# Specify seed to reproduce same random walk:
set.seed(47406)
# Specify standard deviation of proposal distribution:
proposalSD = c(0.02,0.2,2.0)[1]
for ( t in 1:(trajLength-1) ) {
	currentPosition = trajectory[t]
	# Use the proposal distribution to generate a proposed jump.
	proposedJump = rnorm( 1 , mean=0 , sd=proposalSD )
	# Compute the probability of accepting the proposed jump.
	probAccept = min( 1,
		targetRelProb( currentPosition + proposedJump , myData )
		/ targetRelProb( currentPosition , myData ) )
	# Generate a random uniform value from the interval [0,1] to
	# decide whether or not to accept the proposed jump.
	if ( runif(1) < probAccept ) {
		# accept the proposed jump
		trajectory[ t+1 ] = currentPosition + proposedJump
		# increment the accepted counter, just to monitor performance
		if ( t > burnIn ) { nAccepted = nAccepted + 1 }
	} else {
		# reject the proposed jump, stay at current position
		trajectory[ t+1 ] = currentPosition
		# increment the rejected counter, just to monitor performance
		if ( t > burnIn ) { nRejected = nRejected + 1 }
	}
}

# Extract the post-burnIn portion of the trajectory.
acceptedTraj = trajectory[ (burnIn+1) : length(trajectory) ]

# End of Metropolis algorithm.

#-----------------------------------------------------------------------
# Display the chain.

openGraph(width=4,height=8)
layout( matrix(1:3,nrow=3) )
par(mar=c(3,4,2,1),mgp=c(2,0.7,0))

# Posterior histogram:
paramInfo = plotPost( acceptedTraj , xlim=c(0,1) , xlab=bquote(theta) ,
                      cex.main=2.0 ,
                      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                      "Eff.Sz." == .(round(effectiveSize(acceptedTraj),1)) ) ) )

# Trajectory, a.k.a. trace plot, end of chain:
idxToPlot = (trajLength-100):trajLength
plot( trajectory[idxToPlot] , idxToPlot , main="End of Chain" ,
      xlab=bquote(theta) , xlim=c(0,1) , ylab="Step in Chain" ,
      type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
# Display proposal SD and acceptance ratio in the plot.
text( 0.0 , trajLength , adj=c(0.0,1.1) , cex=1.75 ,
      labels = bquote( frac(N[acc],N[pro]) ==
```

```
                        .(signif( nAccepted/length(acceptedTraj) , 3 ))))

# Trajectory, a.k.a. trace plot, beginning of chain:
idxToPlot = 1:100
plot( trajectory[idxToPlot] , idxToPlot , main="Beginning of Chain" ,
      xlab=bquote(theta) , xlim=c(0,1) , ylab="Step in Chain" ,
      type="o" , pch=20 , col="skyblue" , cex.lab=1.5 )
# Indicate burn in limit (might not be visible if not in range):
if ( burnIn > 0 ) {
  abline(h=burnIn,lty="dotted")
  text( 0.5 , burnIn+1 , "Burn In" , adj=c(0.5,1.1) )
}

saveGraph( file=paste0( fileNameRoot ,
                        "SD" , proposalSD ,
                        "Init" , trajectory[1], "Posterior" ) , type="eps" )

# Open a plot, specifying height and width
openGraph(height=7,width=3.5)
# Create 2-rows layout for two plots
layout(matrix(1:2,nrow=2))
# Plot autocorrelation function of the accepted traj
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
# Get accepted traj length
Len = length( acceptedTraj )
# Define lag
Lag = 10
# Define head of the traj, from 1 to Len-Lag
trajHead = acceptedTraj[ 1
                         : (Len-Lag) ]
# Define tail of the traj, from 1+Lag to Len
trajTail = acceptedTraj[ (1+Lag) : Len
]
# Scatterplot of the lagged trajectories
plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                         lag == .(Lag) ,
                         cor == .(round(cor(trajHead,trajTail),3)))) )

saveGraph( file=paste0( fileNameRoot ,
                        "SD" , proposalSD ,
                        "Init" , trajectory[1], "ACF", "Posterior" ) , type="eps" )

#------------------------------------------------------------------------
```

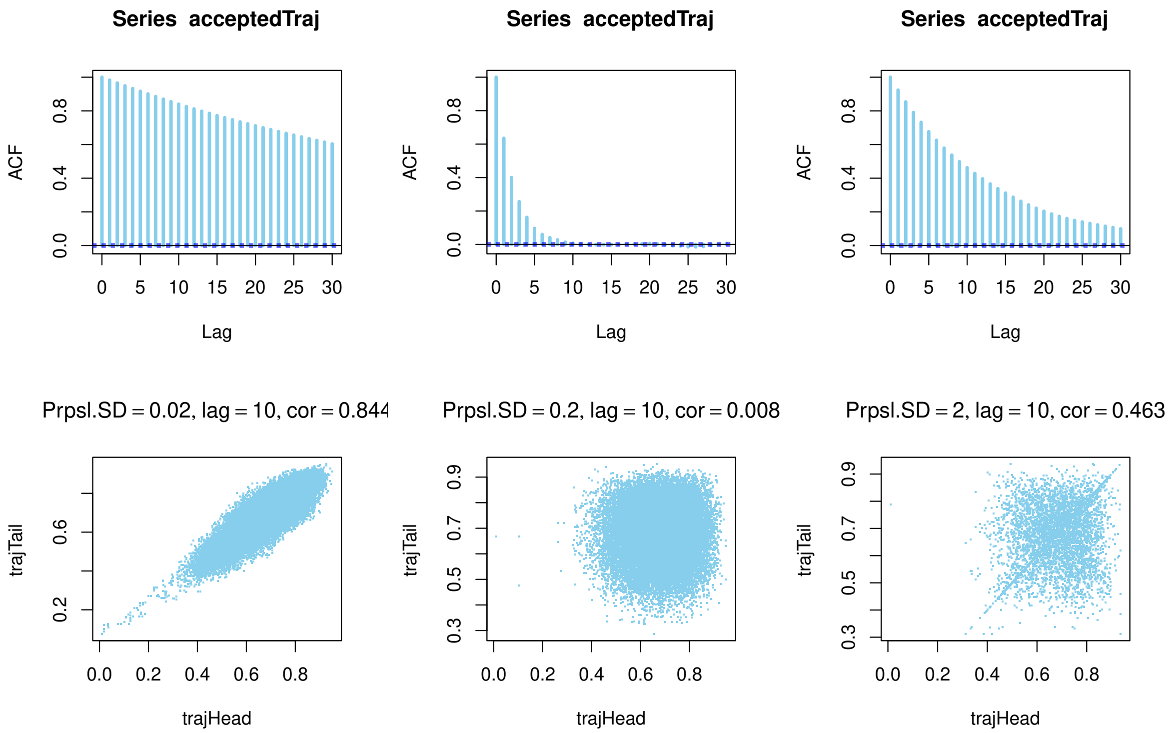
## 7.1




(a) Proposal distr. SD = 0.02 (b) Proposal distr. SD = 0.2 (c) Proposal distr. SD = 2

Figure 1: Repetition of the experiment of Figure 7.4 of the book, with a different random seed. The same considerations hold, i.e., the most efficient sampling is the one with proposal SD = 0.2. When SD = 0.02, steps in the chain are too small and almost always accepted, hence the algorithm will take long time to sample the target distribution. As a result, the ESS is low. The opposite holds for SD = 2, where steps in the chain are too large, so that the acceptance ratio is very low. Again, this leads to low ESS.

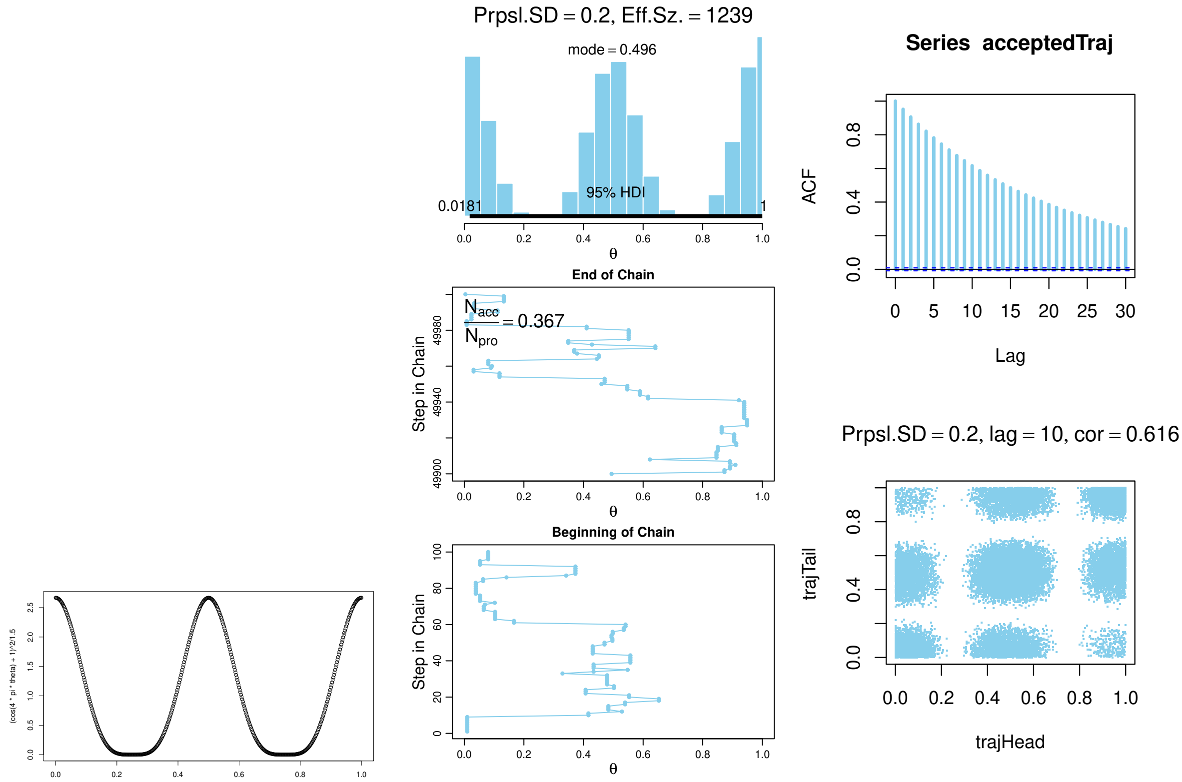
## 7.2




(a) Proposal distr. SD = 0.02 (b) Proposal distr. SD = 0.2 (c) Proposal distr. SD = 2

Figure 2: Here we show the autocorrelation function (ACF) for the trajectories described in 7.1. In the top row, ACF is plotted as a function of the lag time. In the bottom row, we show two copies of the trajectories with a lag of 10 steps, and compute the correlation for this single value. Plots in the top and bottom row are consistent, i.e., ACF values match. The trajectory resulting from SD = 0.2 is the one showing the fastest decaying ACF. Note than when SD = 2, we observe a dense line of points on the diagonal of the scatter plot in the bottom row. Actually, being the acceptance ratio very low, the trajectory often gets stuck at some parameter value, even for long periods. When the trajectory is stuck for a number of time steps greater than the lag time (here 10), then the lagged trajectory superimposes with the non lagged one.

## 7.3

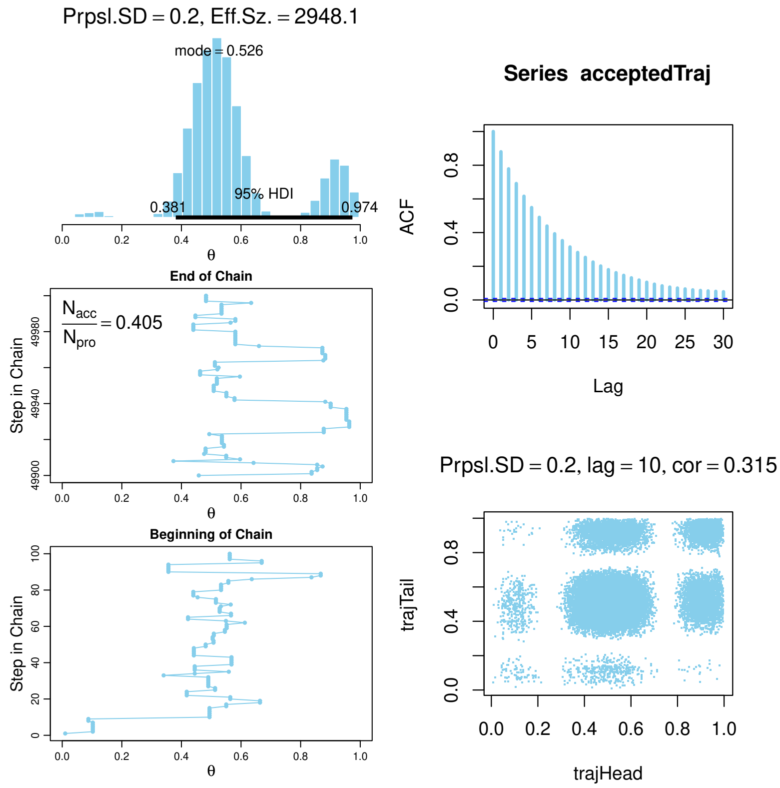



(a) Exact Prior distribution

(b) Sampled Prior distribution and sampling trajectories.

(c) Autocorrelation function of sampling trajectories.

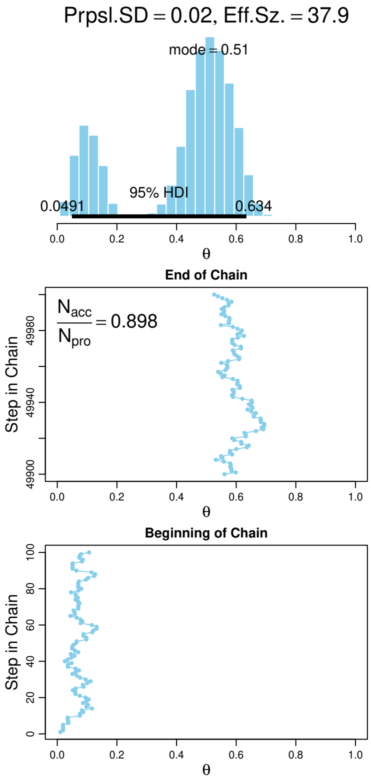
Figure 3: Exact and sampled prior. The sampling is good enough to reproduce the correct shape of the distribution, even though a larger sampling size would smoothen our estimate.



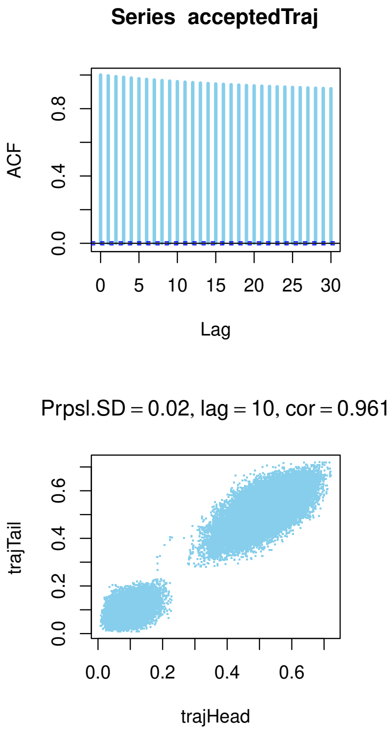

(a) Sampling of the posterior distribution.

(b) Autocorrelation function.

Figure 4: Here we evaluate the posterior distribution given outcomes 0,1,1. Our prior distribution is a three-modal distribution with peaks located at 0, 0.5 and 1 respectively. Our data suggests that, among the three corresponding 'hills', the one on the left is the least likely and the one in the middle is the most likely.

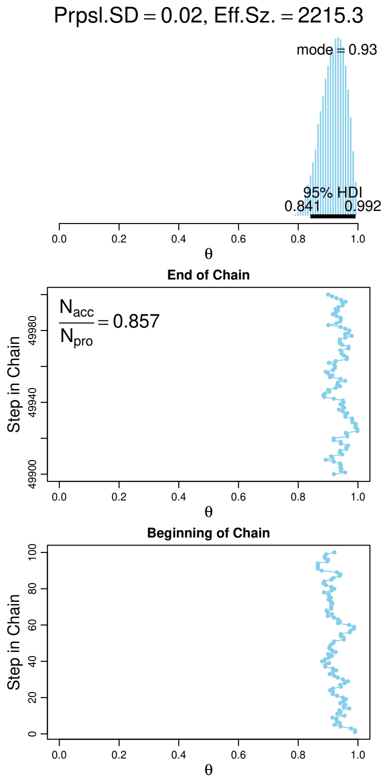



(a) Sampling of the posterior distribution.

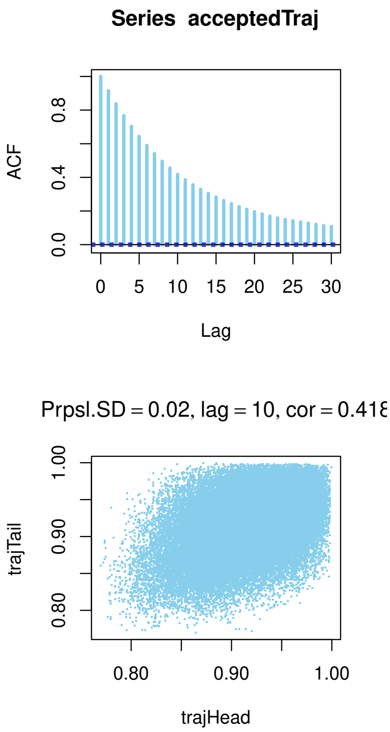



(b) Autocorrelation function.

Figure 5: Here we replicate the experiment of the previous figure, but the chosen proposal SD is too small. We are not correctly sampling the posterior distribution, and we get a wrong estimate of it. There are many ways to realize something is going wrong, including a very small ESS and very slowly decaying ACF. Also, if you look at the next figure...




(a) Sampling of the posterior distribution.




(b) Autocorrelation function.

Figure 6: ...changing the initial condition we get a completely different estimate! Also notice that in this case the random walk remains stuck in the right peak, and is so unaware of the unobserved part of the distribution that ACF behaves well, and the effective size is consequently large. Trying different initial conditions turns out to be crucial.