Discrete-Time Animal Movement Models

Perry Williams, PhD

NRES 779: Bayesian Hierarchical Modeling in Natural Resources‘11

1 Random walk

1.1 Model statement

Process Model

$$\boldsymbol{\mu}_t \sim \text{Normal}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}), \tag{1}$$

$$\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$$

$$\boldsymbol{\mu}_1 = (x_1, y_1)'$$

Prior Model

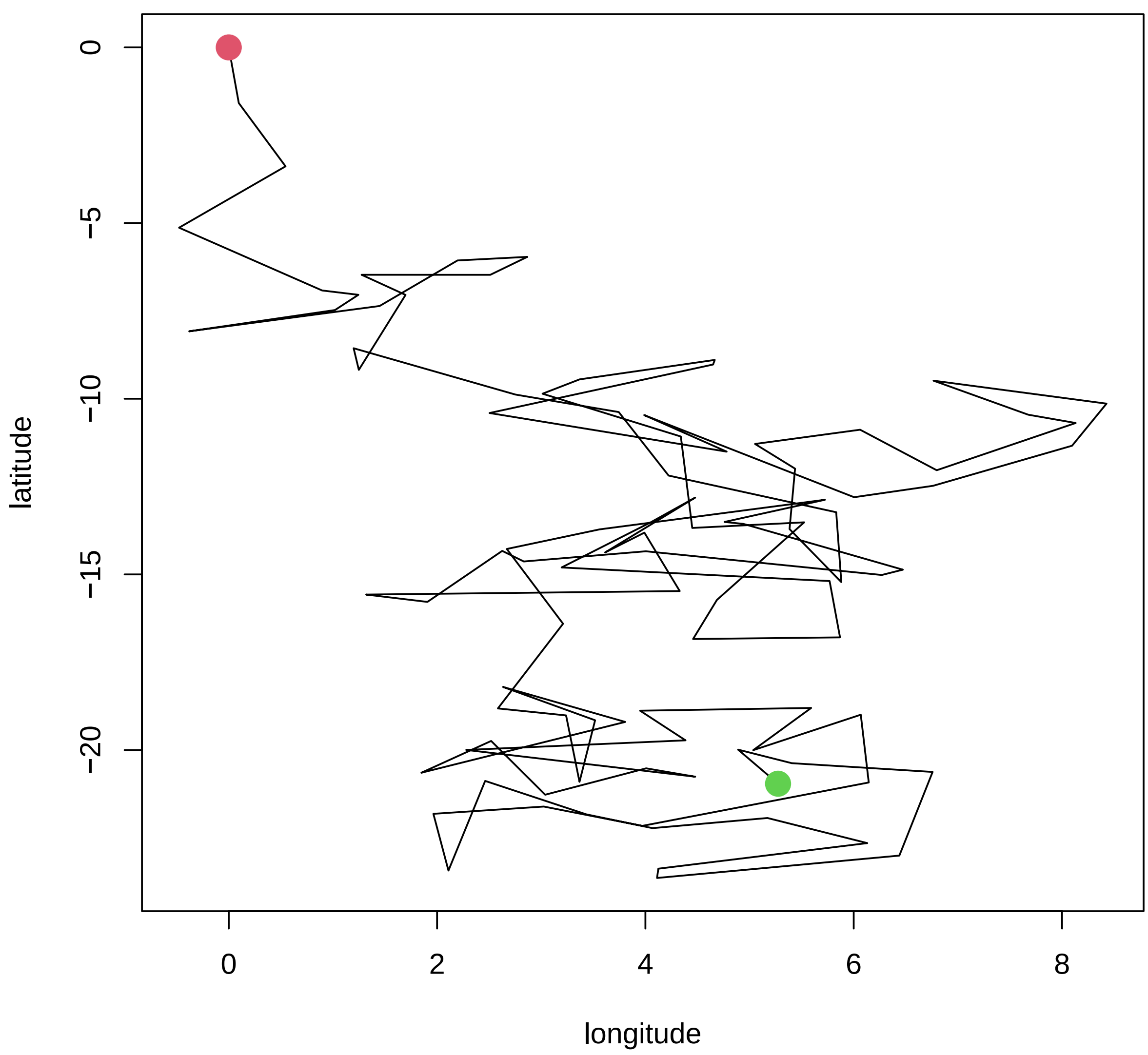
$$\sigma^2 \sim \text{IG}(s, r) \tag{2}$$

- Also known as *intrinsic conditional autoregressive model* or ICAR, because the effect of $\boldsymbol{\mu}_{t-1}$ on $\boldsymbol{\mu}_t$ not attenuated or mixed with another location-based force.
- Random Walk: $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}$.
- **Mechanism:** Implies displacement of the individual during each time step occurs in a random direction with step length governed by a univariate Weibull distribution: Step Length $\sim$ Weibull$(2, \sqrt{2\sigma^2})$.
- σ^2 controls the step lengths between successive locations.

1.2 Simulating Random Walk Data

```
##
## Load MASS library for mvrnorm() function
##

library(MASS)
library(mvtnorm)

##
## Number of time steps
##

T=100

##
## Value of sigma2
##

sigma2.truth=1.1

##
## 2x2 Identity matrix
##

I=diag(2)

##
## Create a matrix to store the data
##

mu=matrix(NA,T,2)
colnames(mu)=c("longitude","latitude")

##
## Starting point at time 1
##
```

```r
mu[1,]=c(0,0)

##
## Simulate movement using a for loop
##

for(t in 2:T){
   mu[t,]=c(mvrnorm(1,mu[t-1,],sigma2.truth*I))
}

##
## Plot movement data
##

plot(mu,type='l')
points(mu[1,1],mu[1,2],col=2,cex=2,pch=16)
points(mu[T,1],mu[T,2],col=3,cex=2,pch=16)
```

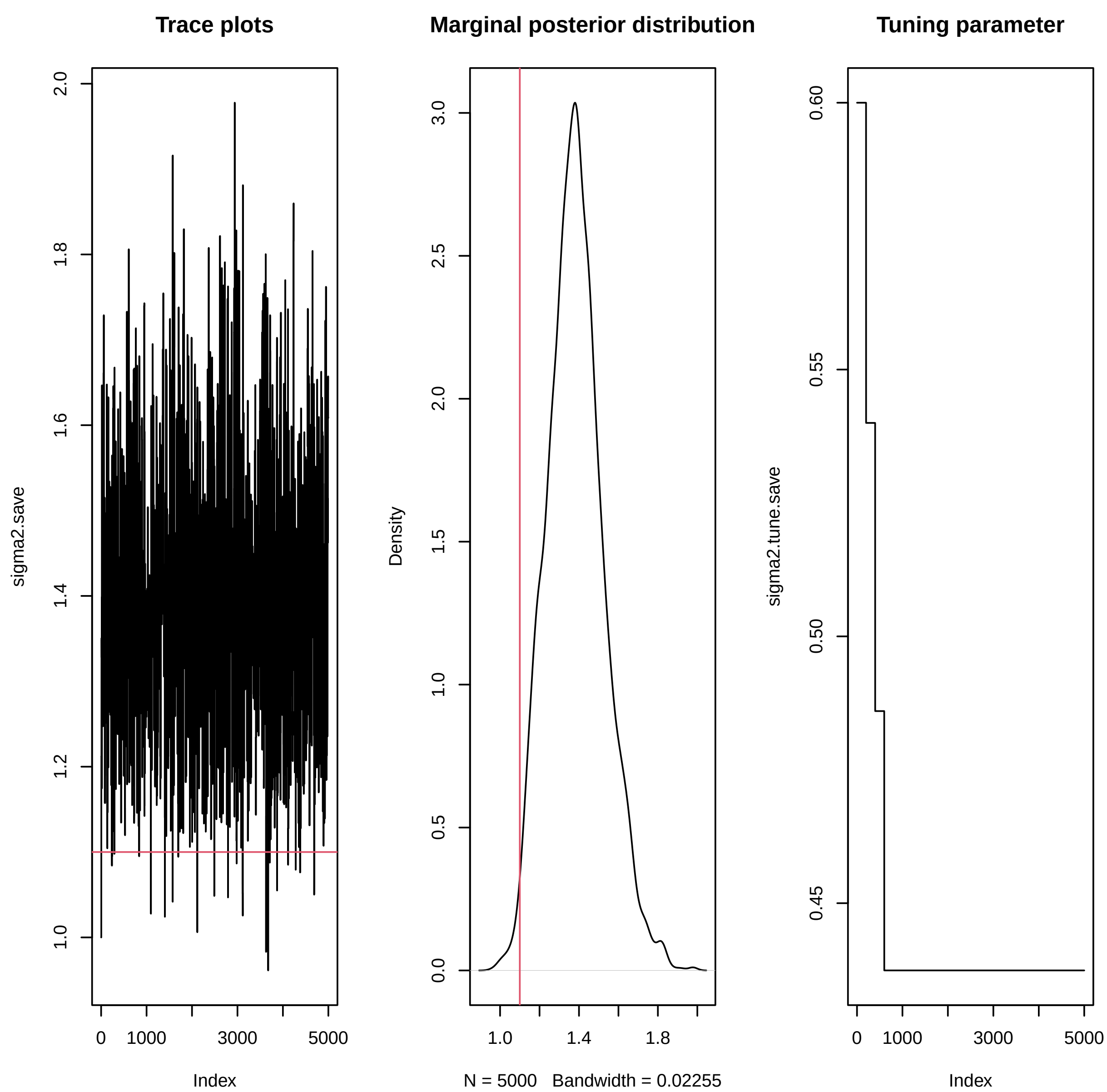


1.3 Estimate Random Walk model parameters using MCMC

```r
dmvnormal=function(mu,M,ap,sigma2){
   sapply(
      1:(T-1),
      function(x) {mvtnorm::dmvnorm(mu[x+1,],
                                    mean=M%*%mu[x,]+(I-M)%*%ap[x,],
                                    sigma=sigma2*I,
                                    log=TRUE)}
   )
}
```

```
##
## Priors on sigma2
##

s=1
r=1

##
## Number of MCMC iterations
##

n.iter=5000
checkpoint=100

##
## Matrix to store MCMC samples
##

sigma2.save=matrix(NA,n.iter,1)
sigma2.tune.save=matrix(NA,n.iter,1)

##
## Tuning parameter
##

sigma2.tune=0.6

##
## Starting value
##

sigma2=1
accept.sigma2=0

##
## Attraction point
##

ap=matrix(0,T-1,2)

##
## Begin MCMC loop
##

k=1
for(k in 1:n.iter){

   ##
   ## Sample sigma2 using Metropolis Algorithm
   ##

   sigma2.star=rnorm(1,sigma2,sigma2.tune)
   if(sigma2.star>0){
      mh1=sum(dmvnormal(mu,diag(2),ap,sigma2.star))+
         1/dgamma(sigma2.star,shape=s,rate=r,log=TRUE)
      mh2=sum(dmvnormal(mu,diag(2),ap,sigma2))+
         1/dgamma(sigma2,shape=s,rate=r,log=TRUE)
      mh=exp(mh1-mh2)
      if(mh>runif(1)){
         sigma2=sigma2.star
         accept.sigma2=accept.sigma2+1
      }
   }

   ##
   ## Autotune
   ##

   if(k%%checkpoint==0){
      if(accept.sigma2/k<0.3) sigma2.tune=sigma2.tune*0.9
      if(accept.sigma2/k>0.5) sigma2.tune=sigma2.tune*1.1
```

```
    }


    sigma2.save[k,]=sigma2
    sigma2.tune.save[k,]=sigma2.tune
}

##
## Plot output
##

par(mfrow=c(1,3))
plot(sigma2.save,type='l',main="Trace plots")
abline(h=sigma2.truth,col=2)
plot(density(sigma2.save),main="Marginal posterior distribution")
abline(v=sigma2.truth,col=2)
plot(sigma2.tune.save,type='l',main= "Tuning parameter")
```



2 Random walk with attraction

A useful generalization of the VAR(1) model allows for the inclusion of an attracting point, or central place.

2.1 Model statement

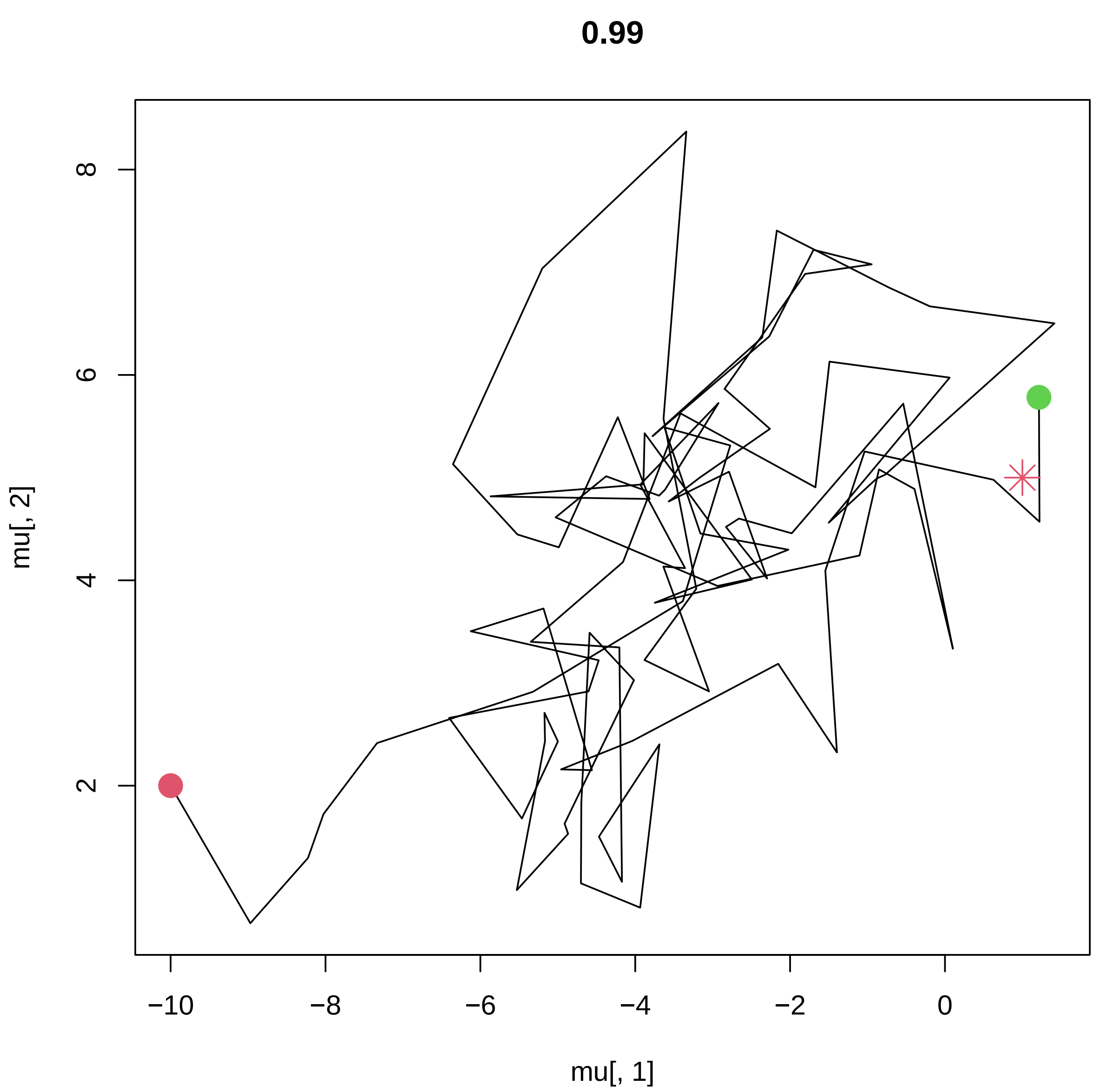
$$
\begin{aligned}
&\textbf{Process Model} \\
&\boldsymbol{\mu}_t \sim \text{Normal}(\mathbf{M}\boldsymbol{\mu}_{t-1} + (\mathbf{I} - \mathbf{M})\boldsymbol{\mu}^*, \boldsymbol{\Sigma}), && (3)\\
&\mathbf{M} \equiv \rho\mathbf{I}, && (4)\\
&\boldsymbol{\Sigma} = \sigma^2\mathbf{I}, \\
&\boldsymbol{\mu}_1 = (x_1, y_1)' \\
&\textbf{Prior Model} \\
&\sigma^2 \sim \text{IG}(s, r) \\
&\rho \sim \text{Uniform}(-1, 1) && (5)
\end{aligned}
$$

- The Random walk model described previously is a specific case of this model, where $\rho = 1$.
- ρ is interpretable as a correlation coefficient.
- σ^2 controls the step lengths between successive locations.

2.2 Simulating data

```
rm(list=ls())

##
## Load MASS library for mvrnorm() function
##

library(MASS)
library(emdbook)

##
## Attaching package:  'emdbook'
## The following object is masked from 'package:mvtnorm':
##
##     dmvnorm

##
## Number of time steps
##

T=100

##
## Value of sigma2
##

sigma2.truth=1.1

##
## Value of rho
##
# rho.truth.ind=seq(-.9999,.99999,length.out=20)
#
# for(i in 1:20){
   rho.truth=.99# rho.truth.ind[i]

   ##
   ## Value of M
   ##

   I=diag(2)
   M=rho.truth*I

   ##
   ## Attraction point
   ##

   ap=c(1,5)
```

```r
##
## Create a matrix to store the data
##

mu=matrix(NA,T,2)
colnames(mu)=c("longitude","latitude")

##
## Starting point at time 1
##

mu[1,]=c(-10,2)

##
## Simulate movement using a for loop
##

for(t in 2:T){
  mu[t,]=mvrnorm(1,M%*%mu[t-1,]+(I-M)%*%ap,sigma2.truth*I)
}

##
## Plot movement data
##

quartz()
plot(mu[,1],mu[,2],type='l',main=rho.truth)
points(mu[1,1],mu[1,2],col=2,cex=2,pch=16)
points(mu[T,1],mu[T,2],col=3,cex=2,pch=16)
points(ap[1],ap[2],pch=8,col=2,cex=2)
```

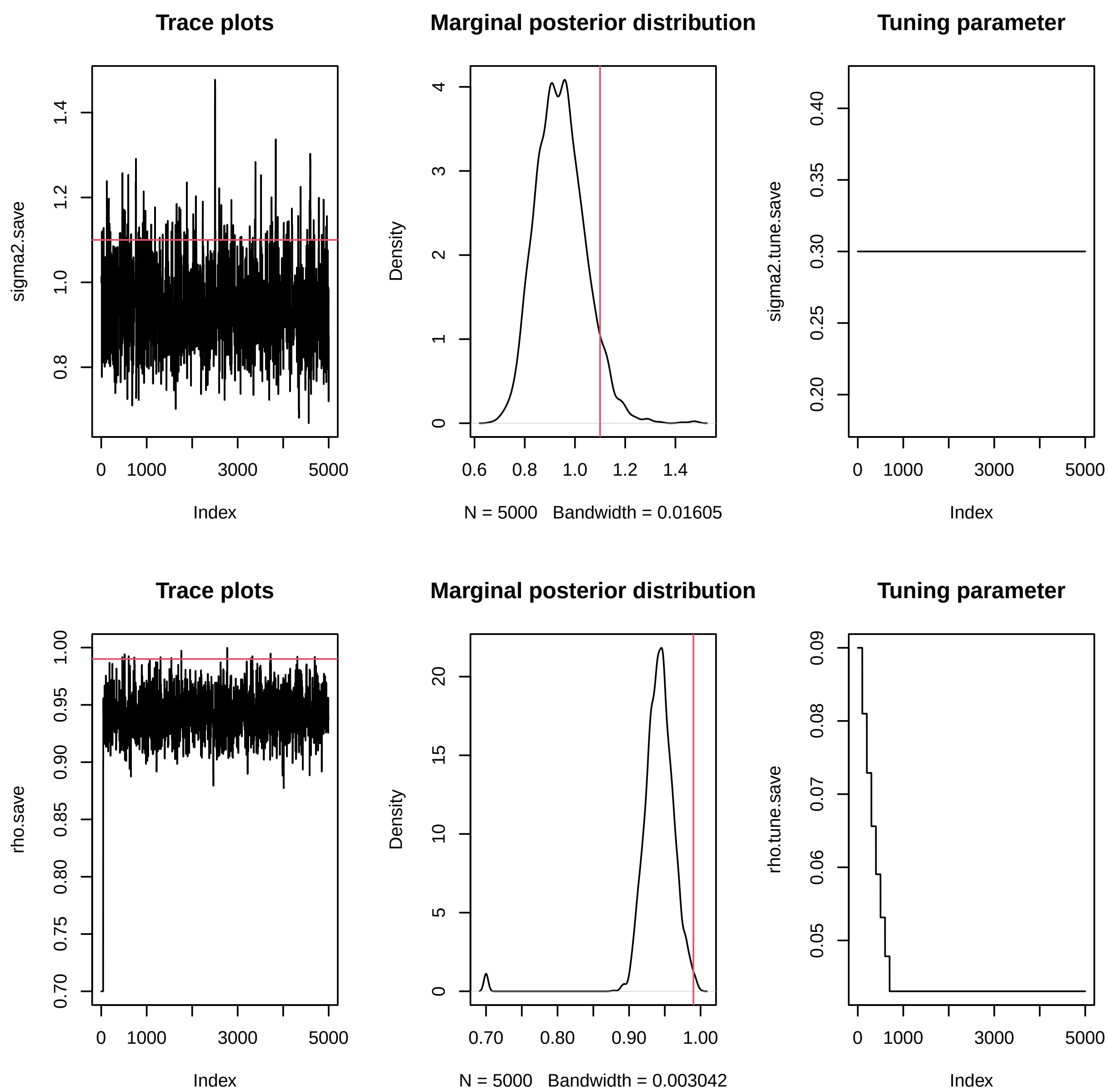


```
# }
```

2.3 Estimate Random Walk with Attraction Model Parameters using MCMC

```
##
## Priors on sigma2
##

s=1
r=1

##
## Priors on rho
##

rho.l=-1
rho.u=1

##
## Number of MCMC iterations and tuning checkpoint
##

n.iter=5000
checkpoint=100

##
## Matrix to store MCMC samples
##

sigma2.save=matrix(NA,n.iter,1)
sigma2.tune.save=matrix(NA,n.iter,1)

rho.save=matrix(NA,n.iter,1)
rho.tune.save=matrix(NA,n.iter,1)

##
## Tuning parameters
##

sigma2.tune=0.3
rho.tune=0.09

##
## Starting value
##

sigma2=1
accept.sigma2=0

rho=0.7
accept.rho=0

##
## Begin MCMC loop
##

for(k in 1:n.iter){

  ##
  ## Sample sigma2 using Metropolis Algorithm
  ##

  sigma2.star=rnorm(1,sigma2,sigma2.tune)
  if(sigma2.star>0){
    mh1=sum(emdbook::dmvnorm(mu[-1,],t(M%*%t(mu[-T,]))+
                               matrix(t(rep((I-M)%*%ap,each=T-1)),T-1,2),
                             sigma2.star*I,log=TRUE))+
      1/dgamma(sigma2.star,shape=s,rate=r,log=TRUE)
    mh2=sum(emdbook::dmvnorm(mu[-1,],t(M%*%t(mu[-T,]))+
                               matrix(t(rep((I-M)%*%ap,each=T-1)),T-1,2),
```

```
                                sigma2*I,log=TRUE))+
         1/dgamma(sigma2,shape=s,rate=r,log=TRUE)
      mh=exp(mh1-mh2)
      if(mh>runif(1)){
         sigma2=sigma2.star
         accept.sigma2=accept.sigma2+1
      }
   }

   ##
   ## Sample rho using Metropolis Algorithm
   ##

   rho.star=rnorm(1,rho,rho.tune)
   M.star=rho.star*I
   if(rho.star>rho.l&rho.star<rho.u){
      mh1=sum(dmvnorm(mu[-1,],t(M.star%*%t(mu[-T,]))+
                         matrix(t(rep((I-M.star)%*%ap,each=T-1)),T-1,2),
                      sigma2*I,log=TRUE))
      mh2=sum(dmvnorm(mu[-1,],t(M%*%t(mu[-T,]))+
                         matrix(t(rep((I-M)%*%ap,each=T-1)),T-1,2),
                      sigma2*I,log=TRUE))
      mh=exp(mh1-mh2)
      if(mh>runif(1)){
         rho=rho.star
         accept.rho=accept.rho+1
         M=M.star
      }
   }

   ##
   ## Autotune
   ##

   if(k%%checkpoint==0){
      if(accept.sigma2/k<0.3) sigma2.tune=sigma2.tune*0.9
      if(accept.sigma2/k>0.5) sigma2.tune=sigma2.tune*1.1
      if(accept.rho/k<0.3) rho.tune=rho.tune*0.9
      if(accept.rho/k>0.5) rho.tune=rho.tune*1.1
   }

   sigma2.save[k,]=sigma2
   sigma2.tune.save[k,]=sigma2.tune

   rho.save[k,]=rho
   rho.tune.save[k,]=rho.tune

}

##
## Plot output
##

par(mfrow=c(2,3))
plot(sigma2.save,type='l',main="Trace plots")
abline(h=sigma2.truth,col=2)
plot(density(sigma2.save),main="Marginal posterior distribution")
abline(v=sigma2.truth,col=2)
plot(sigma2.tune.save,type='l',main= "Tuning parameter")

plot(rho.save,type='l',main="Trace plots")
abline(h=rho.truth,col=2)
plot(density(rho.save),main="Marginal posterior distribution")
abline(v=rho.truth,col=2)
plot(rho.tune.save,type='l',main= "Tuning parameter")
```


Trace plots
Marginal posterior distribution
Tuning parameter
sigma2.save
Index
Density
N = 5000 Bandwidth = 0.01605
sigma2.tune.save
Index
Trace plots
Marginal posterior distribution
Tuning parameter
rho.save
Index
Density
N = 5000 Bandwidth = 0.003042
rho.tune.save
Index

3 Random walk with multiple attraction points and change point

3.1 Model statement

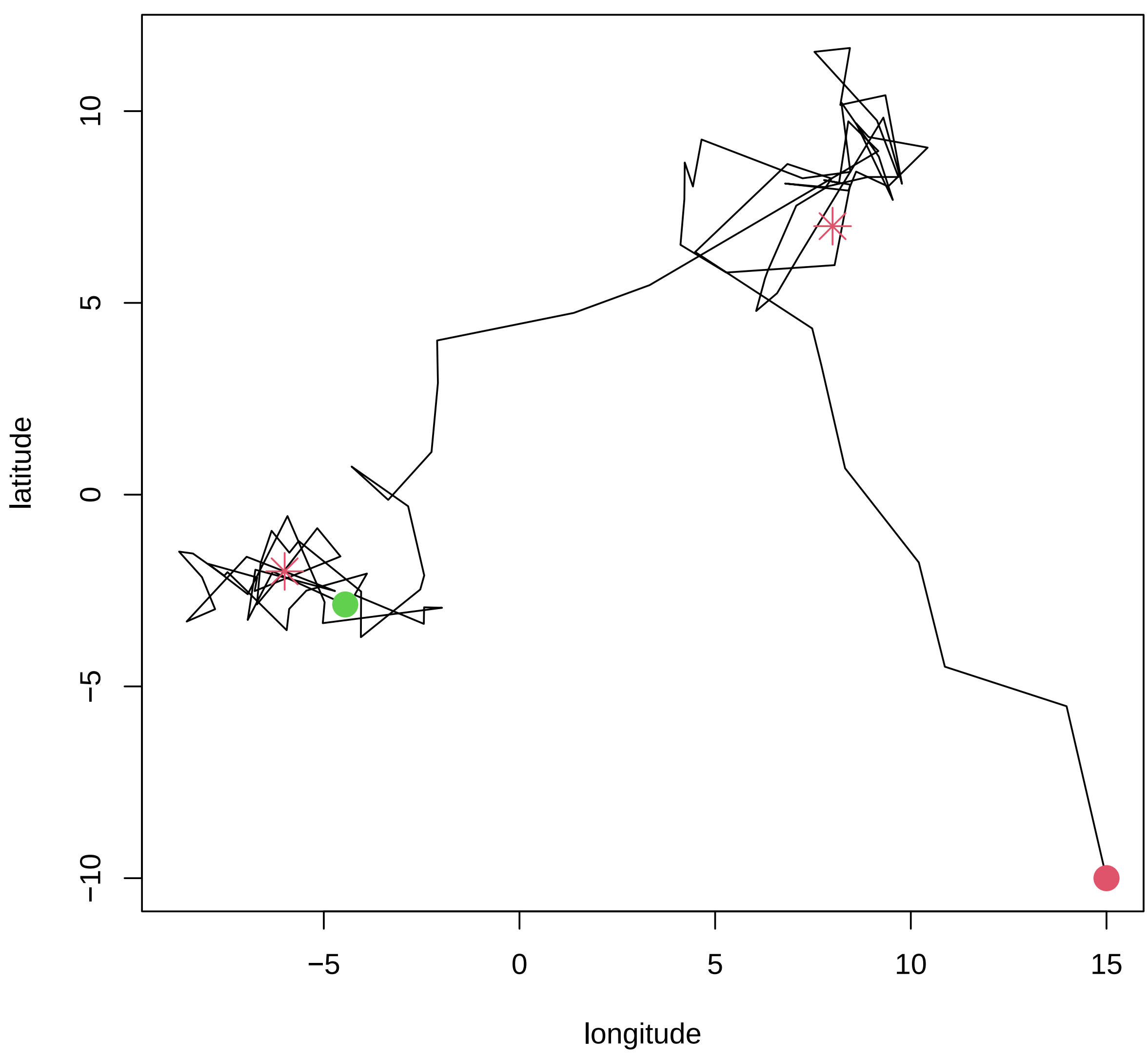
$$
\begin{aligned}
&\textbf{Process Model} \\
&\boldsymbol{\mu}_t \sim \text{Normal}(\mathbf{M}\boldsymbol{\mu}_{t-1} + (\mathbf{I}-\mathbf{M})\boldsymbol{\mu}^*, \boldsymbol{\Sigma}), \qquad (6)\\
&\boldsymbol{\mu}_t^* = \begin{cases} \boldsymbol{\mu}_1^* & t < t^* \\ \boldsymbol{\mu}_2^* & t \geq t^* \end{cases} \\
&\mathbf{M} \equiv \rho\mathbf{I}, \\
&\boldsymbol{\Sigma} = \sigma^2\mathbf{I}, \\
&\boldsymbol{\mu}_1 = (x_1, y_1)' \\
&\textbf{Prior Model} \\
&\sigma^2 \sim \text{IG}(s, r) \\
&\rho \sim \text{Uniform}(-1, 1) \\
&t^* \sim \text{Uniform}(0, T)
\end{aligned}
$$

- t^* is the change point to be estimated

3.2 Simulating Random Walk with multiple attraction points and a change point

```
rm(list=ls())

##
## Load MASS library for mvrnorm() function
##

library(MASS)

##
## Number of time steps
##

T=100

##
## Value of sigma2
##

sigma2.truth=1.1

##
## Value of rho
##

rho.truth=0.7

##
## Value of M
##

I=diag(2)
M=rho.truth*I

##
## change.point
##

t.cp.truth=50

##
## Attraction point
##

ap1=c(8,7)
ap2=c(-6,-2)
ap=matrix(NA,T-1,2)
ap[,1]=ifelse(1:(T-1)<=t.cp.truth,ap1[1],ap2[1])
```

```
ap[,2]=ifelse(1:(T-1)<=t.cp.truth,ap1[2],ap2[2])

##
## Create a matrix to store the data
##

mu=matrix(NA,T,2)
colnames(mu)=c("longitude","latitude")

##
## Starting point at time 1
##

mu[1,]=c(15,-10)

##
## Simulate movement using a for loop
##

for(t in 2:T){
   mu[t,]=mvtnorm::rmvnorm(1,M%*%mu[t-1,]+(I-M)%*%ap[t-1,],sigma2.truth*I)
}

##
## Plot movement data
##

plot(mu,type='l')
points(mu[1,1],mu[1,2],col=2,cex=2,pch=16)
points(mu[T,1],mu[T,2],col=3,cex=2,pch=16)
points(ap1[1],ap1[2],pch=8,col=2,cex=2)
points(ap2[1],ap2[2],pch=8,col=2,cex=2)
```



3.3 Estimate model parameters using MCMC

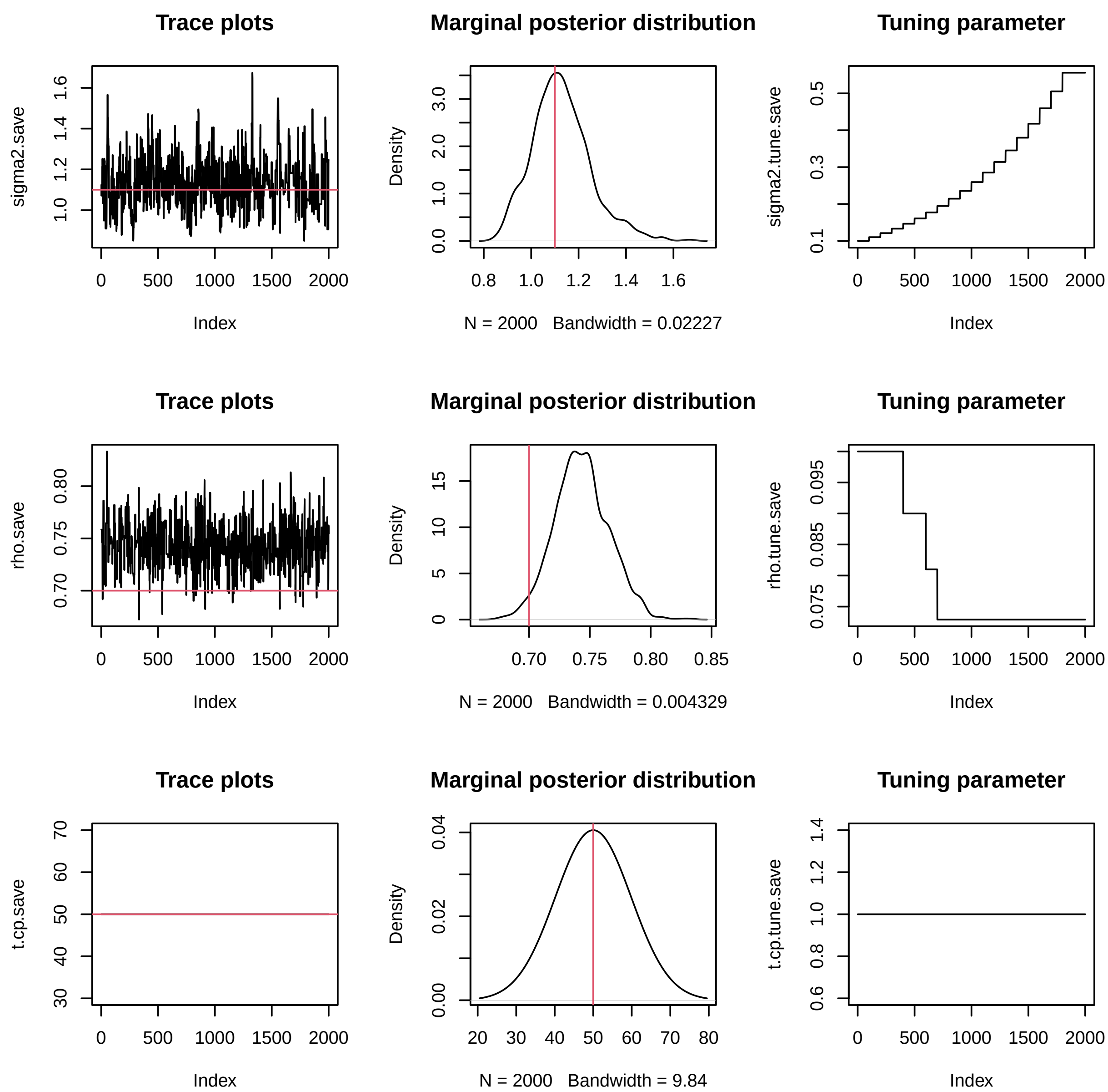
```
###
### Function to calculate lots of multivariate normal densities
###

dmvnormal=function(mu,M,ap,sigma2){
   sapply(
      1:(T-1),
      function(x) {mvtnorm::dmvnorm(mu[x+1,],
                                    mean=M%*%mu[x,]+(I-M)%*%ap[x,],
                                    sigma=sigma2*I,
                                    log=TRUE)}
   )
}


##
## Priors on sigma2
##

s=1
r=1

##
## Priors on rho
##

rho.l=-1
rho.u=1

##
## Priors on t.cp
##

t.cp.l=25
t.cp.u=75

##
## Number of MCMC iterations and tuning checkpoint
##

n.iter=2000
checkpoint=100

##
## Matrix to store MCMC samples
##

sigma2.save=matrix(NA,n.iter,1)
sigma2.tune.save=matrix(NA,n.iter,1)

rho.save=matrix(NA,n.iter,1)
rho.tune.save=matrix(NA,n.iter,1)

t.cp.save=matrix(NA,n.iter,1)
t.cp.tune.save=matrix(NA,n.iter,1)

##
## Tuning parameters
##

sigma2.tune=0.1
rho.tune=0.1
t.cp.tune=1

##
## Starting value
##
```

```
sigma2=sigma2.truth
accept.sigma2=0
rho=rho.truth
accept.rho=0
t.cp=t.cp.truth
accept.t.cp=0
ap1=c(8,7)
ap2=c(-6,-2)
ap=matrix(NA,T-1,2)
ap[,1]=ifelse(1:(T-1)<=t.cp.truth,ap1[1],ap2[1])
ap[,2]=ifelse(1:(T-1)<=t.cp.truth,ap1[2],ap2[2])

ap.star=ap

##
## Begin MCMC loop
##

for(k in 1:n.iter){
  if(k%%100==0) cat(k," ")
   ##
   ## Sample sigma2 using Metropolis Algorithm
   ##

   sigma2.star=rnorm(1,sigma2,sigma2.tune)
   if(sigma2.star>0){
      mh1=sum(dmvnormal(mu,M,ap,sigma2.star)) +
         1/dgamma(sigma2.star,shape=s,rate=r,log=TRUE)
      mh2=sum(dmvnormal(mu,M,ap,sigma2))+
         1/dgamma(sigma2,shape=s,rate=r,log=TRUE)
      mh=min(1,exp(mh1-mh2))
      if(mh>runif(1)){
         sigma2=sigma2.star
         accept.sigma2=accept.sigma2+1
      }
   }

   ##
   ## Sample rho using Metropolis Algorithm
   ##

   rho.star=rnorm(1,rho,rho.tune)
   M.star=rho.star*I
   if(rho.star>rho.l&rho.star<rho.u){
      mh1=sum(dmvnormal(mu,M.star,ap,sigma2))
      mh2=sum(dmvnormal(mu,M,ap,sigma2))
      mh=min(1,exp(mh1-mh2))
      if(mh>runif(1)){
         rho=rho.star
         accept.rho=accept.rho+1
         M=M.star
      }
   }

   ##
   ## Sample t.cp using Metropolis Algorithm
   ##

   t.cp.star=t.cp+sample(-t.cp.tune:t.cp.tune,1)
   ap.star[,1]=ifelse(1:(T-1)<=t.cp.star,ap1[1],ap2[1])
   ap.star[,2]=ifelse(1:(T-1)<=t.cp.star,ap1[2],ap2[2])

   if(t.cp.star>t.cp.l&t.cp.star<t.cp.u){
      mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
      mh2=sum(dmvnormal(mu,M,ap,sigma2))
      mh=min(1,exp(mh1-mh2))
      if(mh>runif(1)){
         t.cp=t.cp.star
         ap=ap.star
         accept.t.cp=accept.t.cp+1
      }
```

```
  }

  ##
  ## Autotune
  ##

  if(k%%checkpoint==0){
    if(accept.sigma2/k<0.3) sigma2.tune=sigma2.tune*0.9
    if(accept.sigma2/k>0.5) sigma2.tune=sigma2.tune*1.1

    if(accept.rho/k<0.3) rho.tune=rho.tune*0.9
    if(accept.rho/k>0.5) rho.tune=rho.tune*1.1

    if(accept.t.cp/k<0.3) t.cp.tune=max(1,t.cp.tune-1)
    if(accept.t.cp/k>0.5) t.cp.tune=t.cp.tune+1

    }

  sigma2.save[k,]=sigma2
  sigma2.tune.save[k,]=sigma2.tune

  rho.save[k,]=rho
  rho.tune.save[k,]=rho.tune

  t.cp.save[k,]=t.cp
  t.cp.tune.save[k]=t.cp.tune
}
```

```
## 100  200  300  400  500  600  700  800  900  1000  1100  1200  1300  1400  1500  1600  1700  1800  19
```

```
##
## Plot output
##

par(mfrow=c(3,3))
plot(sigma2.save,type='l',main="Trace plots")
abline(h=sigma2.truth,col=2)
plot(density(sigma2.save),main="Marginal posterior distribution")
abline(v=sigma2.truth,col=2)
plot(sigma2.tune.save,type='l',main= "Tuning parameter")

plot(rho.save,type='l',main="Trace plots")
abline(h=rho.truth,col=2)
plot(density(rho.save),main="Marginal posterior distribution")
abline(v=rho.truth,col=2)
plot(rho.tune.save,type='l',main= "Tuning parameter")

plot(t.cp.save,type='l',main="Trace plots")
abline(h=t.cp.truth,col=2)
plot(density(t.cp.save),main="Marginal posterior distribution")
abline(v=t.cp.truth,col=2)
plot(t.cp.tune.save,type='l',main= "Tuning parameter")
```


Trace plots
Marginal posterior distribution
Tuning parameter
sigma2.save
Index
Density
N = 2000 Bandwidth = 0.02227
sigma2.tune.save
Index
Trace plots
Marginal posterior distribution
Tuning parameter
rho.save
Index
Density
N = 2000 Bandwidth = 0.004329
rho.tune.save
Index
Trace plots
Marginal posterior distribution
Tuning parameter
t.cp.save
Index
Density
N = 2000 Bandwidth = 9.84
t.cp.tune.save
Index

4 Random walk estimating the point of attraction

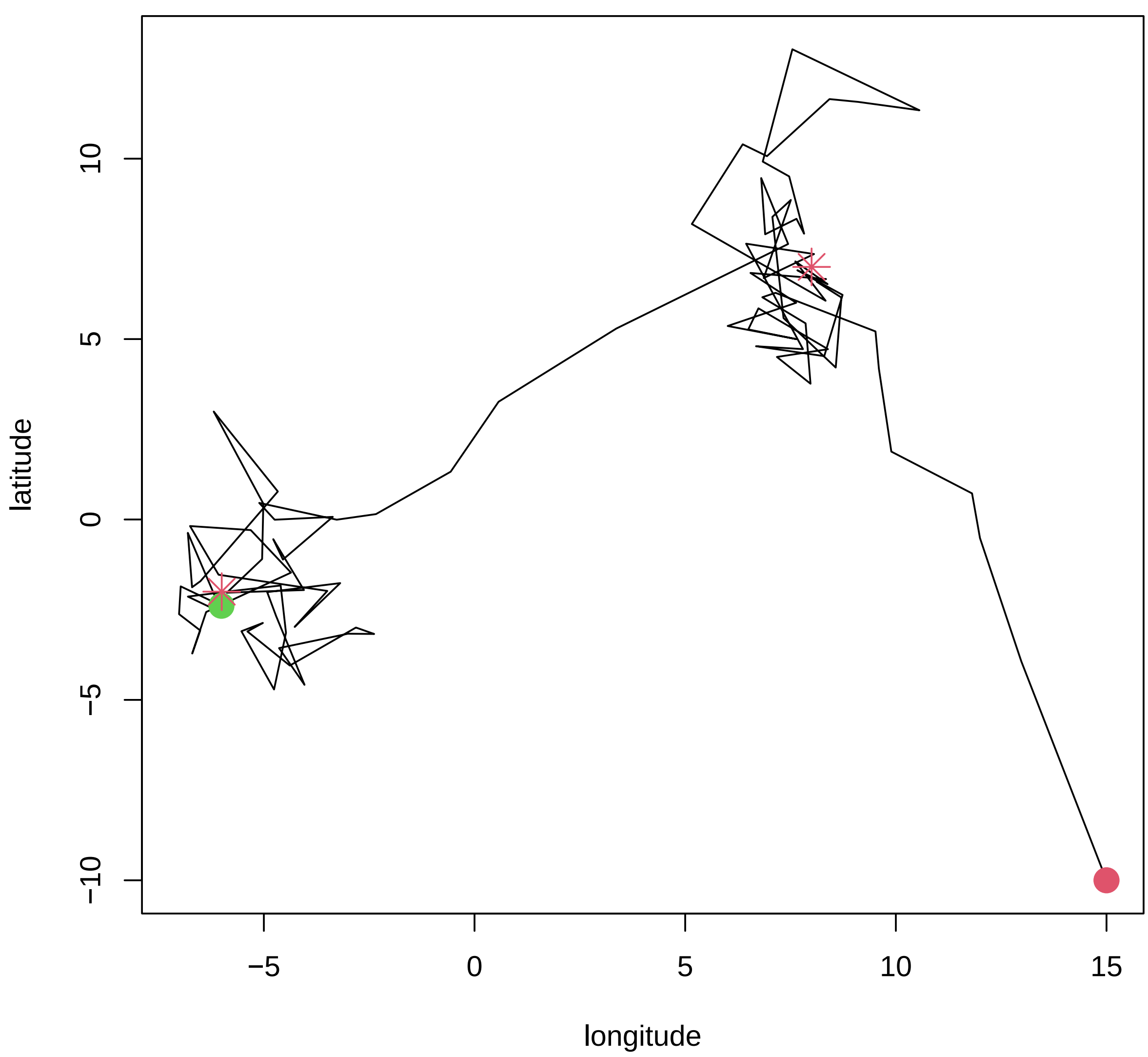
4.1 Model statement

$$
\begin{aligned}
&\textbf{Process Model} \\
&\qquad \boldsymbol{\mu}_t \sim \text{Normal}(\mathbf{M}\boldsymbol{\mu}_{t-1} + (\mathbf{I} - \mathbf{M})\boldsymbol{\mu}^*, \boldsymbol{\Sigma}), \\
&\qquad \mathbf{M} \equiv \rho \mathbf{I}, \\
&\qquad \boldsymbol{\Sigma} = \sigma^2 \mathbf{I}, \\
&\qquad \boldsymbol{\mu}_1 = (x_1, y_1)' \\
&\textbf{Prior Model} \\
&\qquad \sigma^2 \sim \text{IG}(s, r) \\
&\qquad \rho \sim \text{Uniform}(-1, 1) \\
&\qquad \mu^* \sim \text{Uniform}(0, 10)
\end{aligned} \tag{7}
$$

- The Random walk model described previously is a specific case of this model, where $\rho = 1$.
- ρ is interpretable as a correlation coefficient.
- σ^2 controls the step lengths between successive locations.

```
rm(list=ls())

##
## Load MASS library for mvrnorm() function
##

library(MASS)

##
## Number of time steps
##

T=100

##
## Value of sigma2
##

sigma2.truth=1.1

##
## Value of rho
##

rho.truth=0.7

##
## Value of M
##

I=diag(2)
M=rho.truth*I

##
## change.point
##

t.cp.truth=50

##
## Attraction point
##

ap1=c(8,7)
ap2=c(-6,-2)
ap=matrix(NA,T-1,2)
ap[,1]=ifelse(1:(T-1)<=t.cp.truth,ap1[1],ap2[1])
ap[,2]=ifelse(1:(T-1)<=t.cp.truth,ap1[2],ap2[2])

##
```

```r
## Create a matrix to store the data
##

mu=matrix(NA,T,2)
colnames(mu)=c("longitude","latitude")

##
## Starting point at time 1
##

mu[1,]=c(15,-10)

##
## Simulate movement using a for loop
##

for(t in 2:T){
   mu[t,]=mvtnorm::rmvnorm(1,M%*%mu[t-1,]+(I-M)%*%ap[t-1,],sigma2.truth*I)
}

##
## Plot movement data
##

plot(mu,type='l')
points(mu[1,1],mu[1,2],col=2,cex=2,pch=16)
points(mu[T,1],mu[T,2],col=3,cex=2,pch=16)
points(ap1[1],ap1[2],pch=8,col=2,cex=2)
points(ap2[1],ap2[2],pch=8,col=2,cex=2)
```



4.2 Estimate rw and ap model parameters using MCMC

```
###
### Function to calculate lots of multivariate normal densities
###

dmvnormal=function(mu,M,ap,sigma2){
      sapply(
         1:(T-1),
         function(x) {mvtnorm::dmvnorm(mu[x+1,],
                                       mean=M%*%mu[x,]+(I-M)%*%ap[x,],
                                       sigma=sigma2*I,
                                       log=TRUE)}
      )
   }

##
## Priors on ap.x
##

ap.x.l=-15
ap.x.u=20

##
## Priors on ap.y
##

ap.y.l=-15
ap.y.u=20

##
## Priors on sigma2
##

s=1
r=1

##
## Priors on rho
##

rho.l=-1
rho.u=1

##
## Priors on t.cp
##

t.cp.l=25
t.cp.u=75

##
## Number of MCMC iterations and tuning checkpoint
##

n.iter=1000
checkpoint=100

##
## Matrix to store MCMC samples
##

ap1.save=matrix(NA,n.iter,2)
ap2.save=matrix(NA,n.iter,2)
ap1.tune.save=matrix(NA,n.iter,2)
ap2.tune.save=matrix(NA,n.iter,2)

sigma2.save=matrix(NA,n.iter,1)
sigma2.tune.save=matrix(NA,n.iter,1)

rho.save=matrix(NA,n.iter,1)
```

```
rho.tune.save=matrix(NA,n.iter,1)

t.cp.save=matrix(NA,n.iter,1)
t.cp.tune.save=matrix(NA,n.iter,1)

##
## Tuning parameters
##

ap1.x.tune=0.1
ap1.y.tune=0.1
ap2.x.tune=0.1
ap2.y.tune=0.1
sigma2.tune=0.1
rho.tune=0.1
t.cp.tune=1

##
## Starting value
##

sigma2=sigma2.truth
accept.sigma2=0
rho=rho.truth
accept.rho=0
t.cp=t.cp.truth
accept.t.cp=0
ap1.truth=ap1=c(8,7)
ap2.truth=ap2=c(-6,-2)
ap=matrix(NA,T-1,2)
ap[,1]=ifelse(1:(T-1)<=t.cp.truth,ap1[1],ap2[1])
ap[,2]=ifelse(1:(T-1)<=t.cp.truth,ap1[2],ap2[2])
ap.star=ap
ap1.x=ap[1,1]
ap1.y=ap[1,2]
ap2.x=ap[t.cp+1,1]
ap2.y=ap[t.cp+1,2]
accept.ap1.x=0
accept.ap1.y=0
accept.ap2.x=0
accept.ap2.y=0

##
## Begin MCMC loop
##

for(k in 1:n.iter){
   if(k%%100==0) cat(k," ")

   ##
   ## Sample ap1.x using Metropolis Algorithm
   ##

   ap.star=ap
   ap1.x.star=rnorm(1,ap1.x,ap1.x.tune)
   ap1.star=c(ap1.x.star,ap1.y)
   ap.star[,1]=ifelse(1:(T-1)<=t.cp,ap1.star[1],ap2[1])
   ap.star[,2]=ifelse(1:(T-1)<=t.cp,ap1.star[2],ap2[2])

   if(ap1.x.star>ap.x.l & ap1.x.star<ap.x.u){
      mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
      mh2=sum(dmvnormal(mu,M,ap,sigma2))
      mh=exp(mh1-mh2)
      if(mh>runif(1)){
         ap1.x=ap1.x.star
         ap1=ap1.star
         ap=ap.star
         accept.ap1.x=accept.ap1.x+1
      }
   }
```

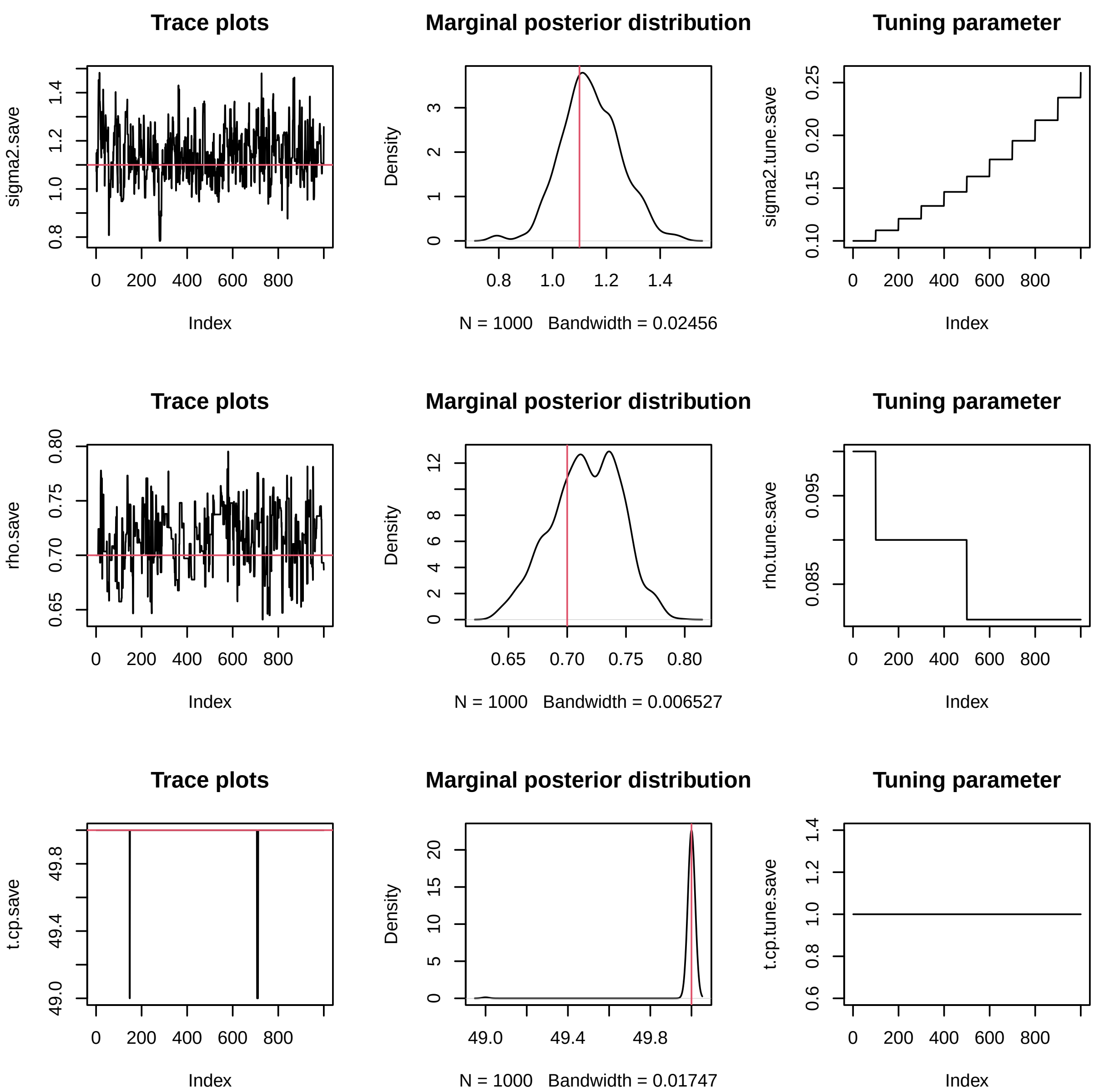
```
##
## Sample ap1.y using Metropolis Algorithm
##

ap.star=ap
ap1.y.star=rnorm(1,ap1.y,ap1.y.tune)
ap1.star=c(ap1.x,ap1.y.star)
ap.star[,1]=ifelse(1:(T-1)<=t.cp,ap1.star[1],ap2[1])
ap.star[,2]=ifelse(1:(T-1)<=t.cp,ap1.star[2],ap2[2])

if(ap1.y.star>ap.y.l & ap1.y.star<ap.y.u){
   mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
   mh2=sum(dmvnormal(mu,M,ap,sigma2))
   mh=exp(mh1-mh2)
   if(mh>runif(1)){
      ap1.y=ap1.y.star
      ap1=ap1.star
      ap=ap.star
      accept.ap1.y=accept.ap1.y+1
   }
}

##
## Sample ap2.x using Metropolis Algorithm
##

ap.star=ap
ap2.x.star=rnorm(1,ap2.x,ap2.x.tune)
ap2.star=c(ap2.x.star,ap2.y)
ap.star[,1]=ifelse(1:(T-1)<=t.cp,ap1[1],ap2.star[1])
ap.star[,2]=ifelse(1:(T-1)<=t.cp,ap1[2],ap2.star[2])

if(ap2.x.star>ap.x.l & ap2.x.star<ap.x.u){
   mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
   mh2=sum(dmvnormal(mu,M,ap,sigma2))
   mh=exp(mh1-mh2)
   if(mh>runif(1)){
      ap2.x=ap2.x.star
      ap2=ap2.star
      ap=ap.star
      accept.ap2.x=accept.ap2.x+1
   }
}

##
## Sample ap2.y using Metropolis Algorithm
##

ap.star=ap
ap2.y.star=rnorm(1,ap2.y,ap2.y.tune)
ap2.star=c(ap2.x,ap2.y.star)
ap.star[,1]=ifelse(1:(T-1)<=t.cp,ap1[1],ap2.star[1])
ap.star[,2]=ifelse(1:(T-1)<=t.cp,ap1[2],ap2.star[2])

if(ap2.y.star>ap.y.l & ap2.y.star<ap.y.u){
   mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
   mh2=sum(dmvnormal(mu,M,ap,sigma2))
   mh=exp(mh1-mh2)
   if(mh>runif(1)){
      ap2.y=ap2.y.star
      ap2=ap2.star
      ap=ap.star
      accept.ap2.y=accept.ap2.y+1
   }
}

##
## Sample sigma2 using Metropolis Algorithm
##

sigma2.star=rnorm(1,sigma2,sigma2.tune)
```

```
if(sigma2.star>0){
   mh1=sum(dmvnormal(mu,M,ap,sigma2.star)) +
      1/dgamma(sigma2.star,shape=s,rate=r,log=TRUE)
   mh2=sum(dmvnormal(mu,M,ap,sigma2))+
      1/dgamma(sigma2,shape=s,rate=r,log=TRUE)
   mh=min(1,exp(mh1-mh2))
   if(mh>runif(1)){
      sigma2=sigma2.star
      accept.sigma2=accept.sigma2+1
   }
}

##
## Sample rho using Metropolis Algorithm
##

rho.star=rnorm(1,rho,rho.tune)
M.star=rho.star*I
if(rho.star>rho.l&rho.star<rho.u){
   mh1=sum(dmvnormal(mu,M.star,ap,sigma2))
   mh2=sum(dmvnormal(mu,M,ap,sigma2))
   mh=min(1,exp(mh1-mh2))
   if(mh>runif(1)){
      rho=rho.star
      accept.rho=accept.rho+1
      M=M.star
   }
}

##
## Sample t.cp using Metropolis Algorithm
##

t.cp.star=t.cp+sample(-t.cp.tune:t.cp.tune,1)
ap.star[,1]=ifelse(1:(T-1)<=t.cp.star,ap1[1],ap2[1])
ap.star[,2]=ifelse(1:(T-1)<=t.cp.star,ap1[2],ap2[2])

if(t.cp.star>t.cp.l&t.cp.star<t.cp.u){
   mh1=sum(dmvnormal(mu,M,ap.star,sigma2))
   mh2=sum(dmvnormal(mu,M,ap,sigma2))
   mh=min(1,exp(mh1-mh2))
   if(mh>runif(1)){
      t.cp=t.cp.star
      ap=ap.star
      accept.t.cp=accept.t.cp+1
   }
}

##
## Autotune
##

if(k%%checkpoint==0){

   if(accept.ap1.x/k<0.3) ap1.x.tune=ap1.x.tune*0.9
   if(accept.ap1.x/k>0.5) ap1.x.tune=ap1.x.tune*1.1

   if(accept.ap1.y/k<0.3) ap1.y.tune=ap1.y.tune*0.9
   if(accept.ap1.y/k>0.5) ap1.y.tune=ap1.y.tune*1.1

  if(accept.ap2.x/k<0.3) ap2.x.tune=ap2.x.tune*0.9
   if(accept.ap2.x/k>0.5) ap2.x.tune=ap2.x.tune*1.1

   if(accept.ap2.y/k<0.3) ap2.y.tune=ap2.y.tune*0.9
   if(accept.ap2.y/k>0.5) ap2.y.tune=ap2.y.tune*1.1

   if(accept.sigma2/k<0.3) sigma2.tune=sigma2.tune*0.9
   if(accept.sigma2/k>0.5) sigma2.tune=sigma2.tune*1.1

   if(accept.rho/k<0.3) rho.tune=rho.tune*0.9
   if(accept.rho/k>0.5) rho.tune=rho.tune*1.1
```

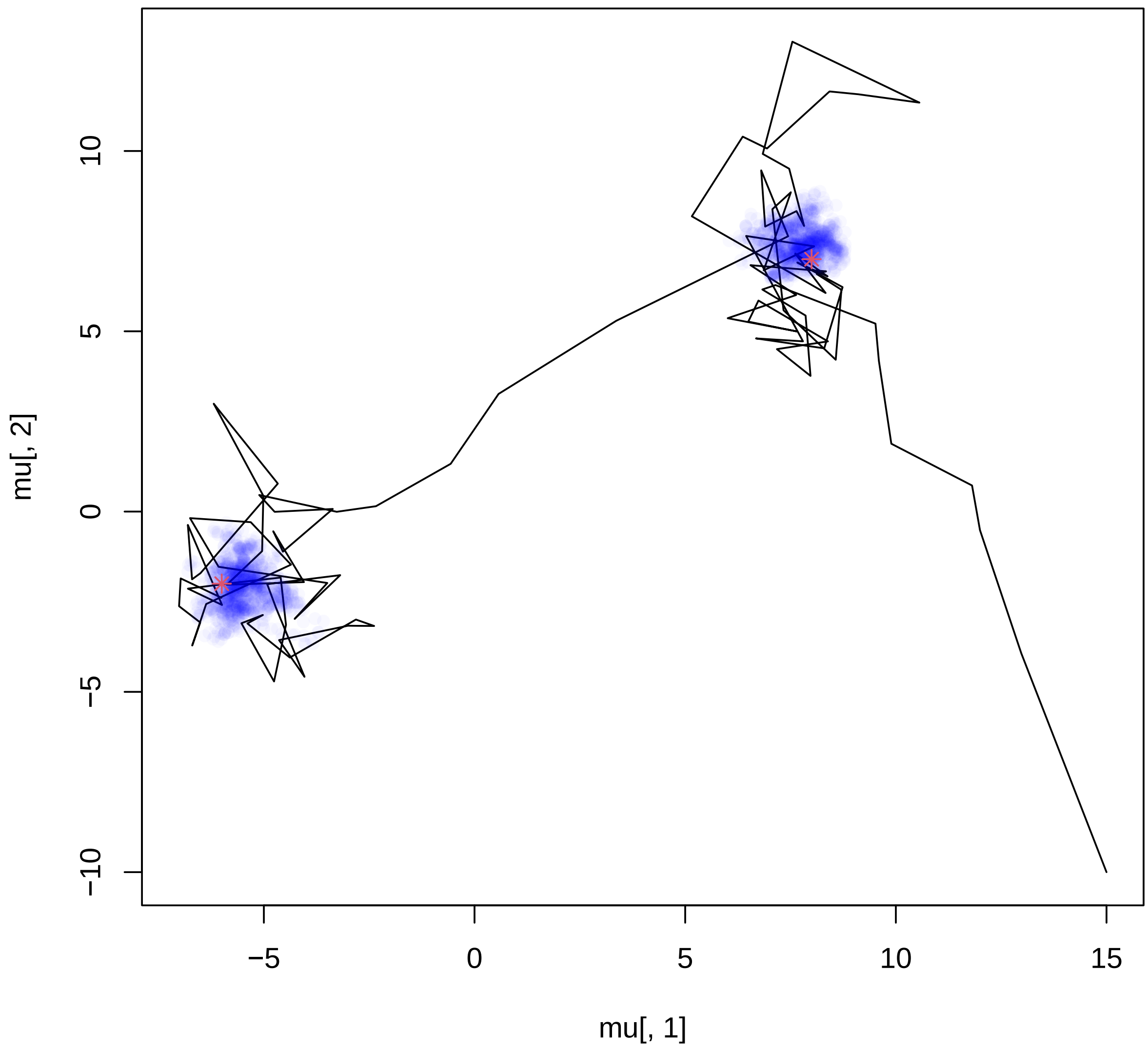
```
      if(accept.t.cp/k<0.3) t.cp.tune=max(1,t.cp.tune-1)
      if(accept.t.cp/k>0.5) t.cp.tune=t.cp.tune+1

   }

   ap1.save[k,]=ap1
   ap2.save[k,]=ap2
   ap1.tune.save[k,]=c(ap1.x.tune,ap1.y.tune)
   ap2.tune.save[k,]=c(ap2.x.tune,ap2.y.tune)

   sigma2.save[k,]=sigma2
   sigma2.tune.save[k,]=sigma2.tune

   rho.save[k,]=rho
   rho.tune.save[k,]=rho.tune

   t.cp.save[k,]=t.cp
   t.cp.tune.save[k]=t.cp.tune
}
```

```
## 100  200  300  400  500  600  700  800  900  1000
```

```
##
## Plot output
##

par(mfrow=c(3,3))
plot(sigma2.save,type='l',main="Trace plots")
abline(h=sigma2.truth,col=2)
plot(density(sigma2.save),main="Marginal posterior distribution")
abline(v=sigma2.truth,col=2)
plot(sigma2.tune.save,type='l',main= "Tuning parameter")

plot(rho.save,type='l',main="Trace plots")
abline(h=rho.truth,col=2)
plot(density(rho.save),main="Marginal posterior distribution")
abline(v=rho.truth,col=2)
plot(rho.tune.save,type='l',main= "Tuning parameter")

plot(t.cp.save,type='l',main="Trace plots")
abline(h=t.cp.truth,col=2)
plot(density(t.cp.save),main="Marginal posterior distribution")
abline(v=t.cp.truth,col=2)
plot(t.cp.tune.save,type='l',main= "Tuning parameter")
```




```
par(mfrow=c(1,1))
plot(mu[,1],mu[,2],type="l")
points(ap1.save,pch=16,col=rgb(0,0,1,0.03))
points(ap1.truth[1],ap1.truth[2],col=2,pch=8)
points(ap2.save,pch=16,col=rgb(0,0,1,0.03))
points(ap2.truth[1],ap2.truth[2],col=2,pch=8)
```



5 Velocity Models

Let $\mathbf{v}_t \equiv \boldsymbol{\mu}_t - \boldsymbol{\mu}_{t-1}$, Then the position model:

$$\begin{aligned} &\textbf{Process Model} \\ &\boldsymbol{\mu}_t \sim \text{Normal}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}), \qquad (8) \\ &\boldsymbol{\Sigma} = \sigma^2 \mathbf{I} \\ &\boldsymbol{\mu}_1 = (x_1, y_1)' \\ &\textbf{Prior Model} \\ &\sigma^2 \sim \text{IG}(s, r) \qquad (9) \end{aligned}$$

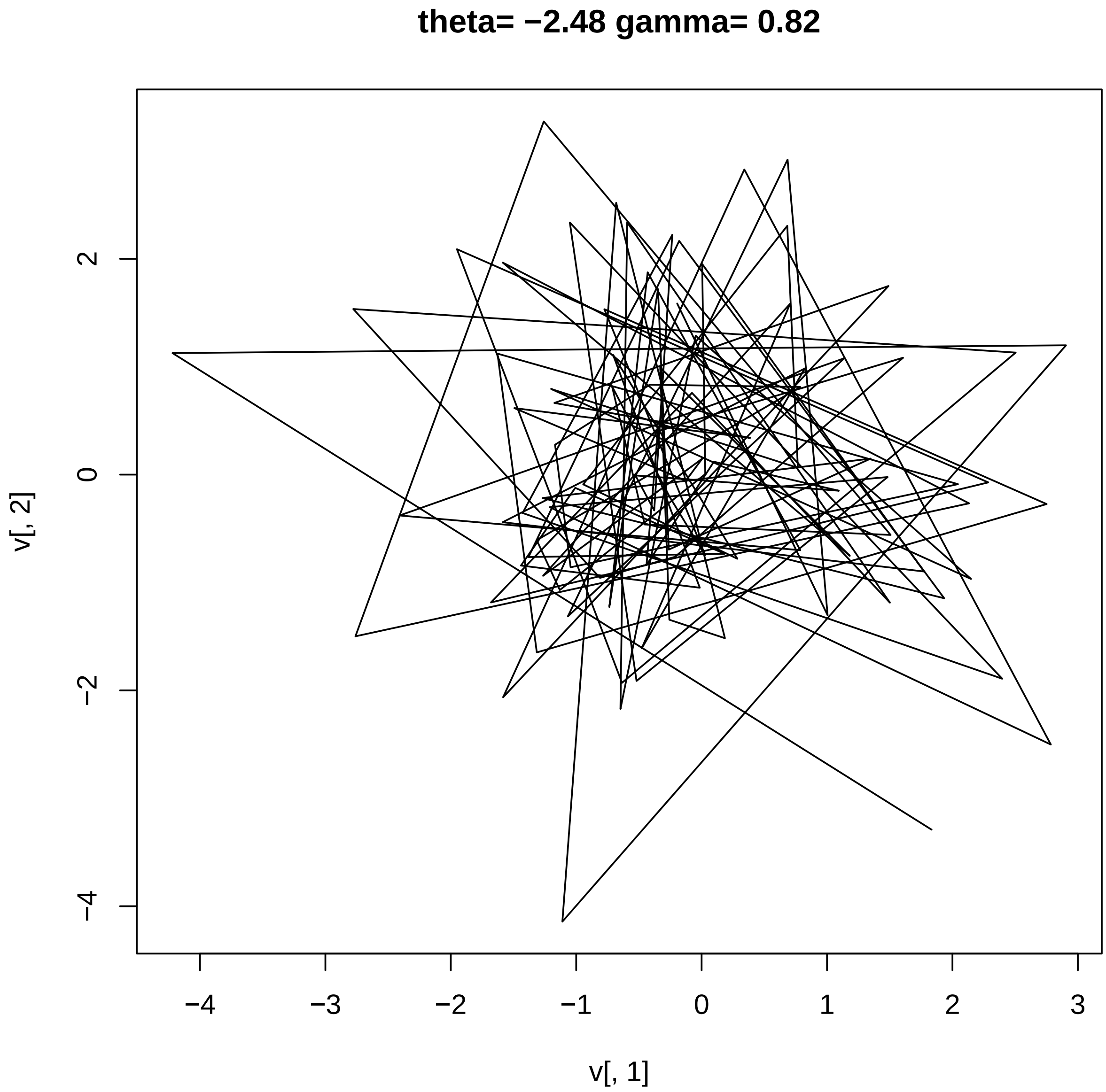
becomes:

$$\begin{aligned} &\textbf{Process Model} \\ &\mathbf{v}_t \sim \text{Normal}(\mathbf{0}, \sigma^2 \mathbf{I}), \qquad (10) \\ &\textbf{Prior Model} \\ &\sigma^2 \sim \text{IG}(s, r) \qquad (11) \end{aligned}$$

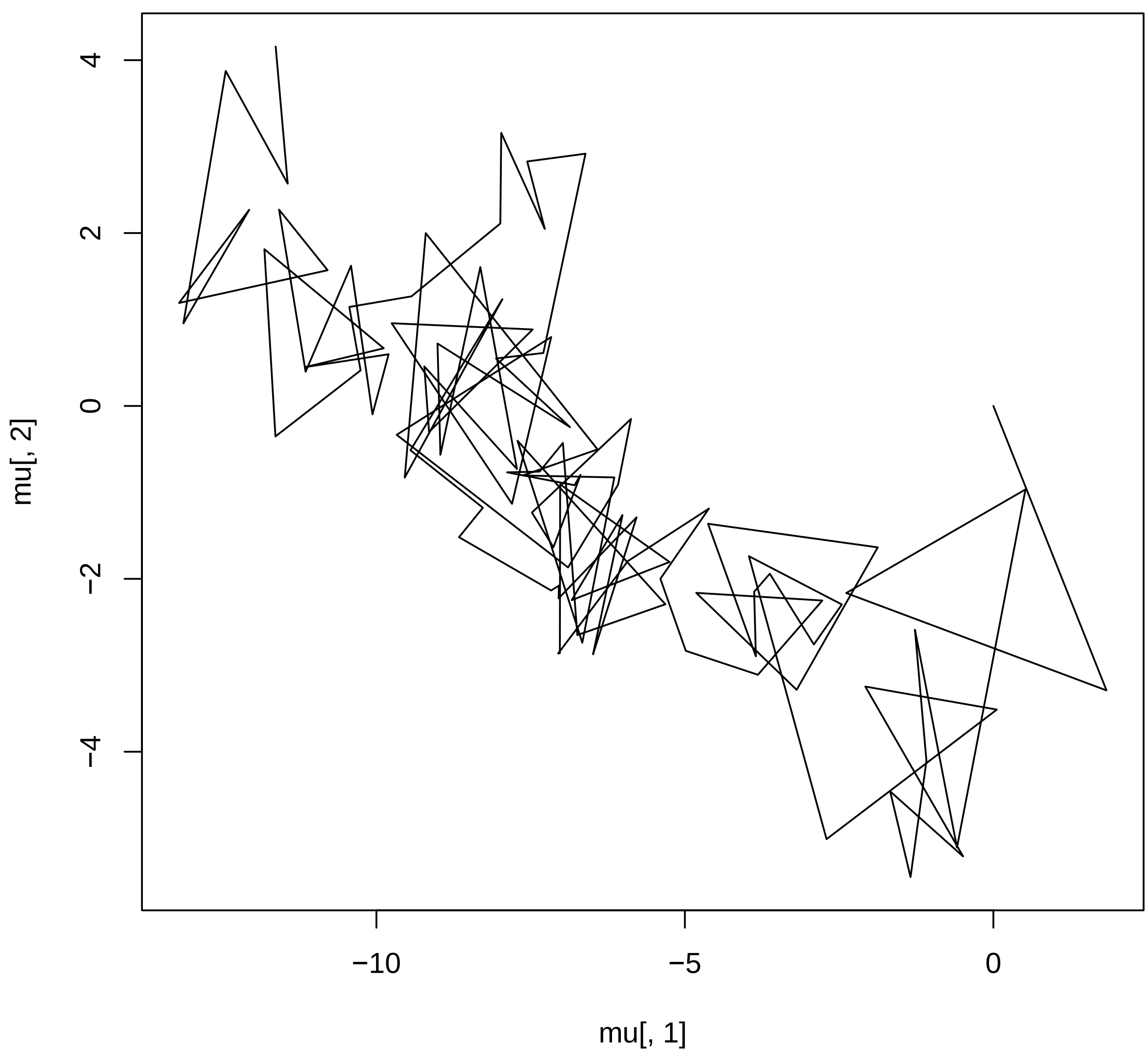
5.1 Turning angle

$$
\begin{aligned}
&\textbf{Process Model} \\
&\mathbf{v}_t \sim \text{Normal}(\mathbf{M}\mathbf{v}_{t-1}, \mathbf{\Sigma}), \qquad (12) \\
&\mathbf{\Sigma} = \sigma^2 \mathbf{I} \\
&\mathbf{M} \equiv \gamma \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \\
&\textbf{Prior Model} \\
&\theta \sim \text{Unif}(-\pi, \pi) \\
&\gamma \sim \text{Unif}(0, 1) \\
&\sigma^2 \sim \text{IG}(s, r)
\end{aligned}
$$

```r
library(MASS)

#########################################################################
### Turning angle velocity model
#########################################################################

n.steps=100
sigma2=1/rgamma(1,1,1)
v1=rnorm(2,0,sqrt(sigma2))
theta=runif(1,-pi,pi)
theta.truth=theta
gamma=runif(1,0.7,1)
gamma.truth=gamma
R=matrix(c(
   cos(theta),-sin(theta),
   sin(theta),cos(theta)),
   2,2,byrow=TRUE)
M=gamma*R
v=matrix(,n.steps,2)
v[1,]=v1

for(i in 2:n.steps){
    v[i,]=mvrnorm(1,M%*%v[i-1,],diag(2))
}
plot(v[,1],v[,2],type='l',main=paste("theta=",round(theta,2),"gamma=",round(gamma,2)))
```




```r
###
### Converting turning angle velocity model to position data
###

mu=matrix(0,n.steps+1,2)
for(i in 2:(n.steps+1)){
    mu[i,]=mu[i-1,]+v[i-1,]
}
plot(mu[,1],mu[,2],type='l')
```



4
2
0
−2
−4
mu[, 2]
−10
−5
0
mu[, 1]