2D example using geoR to fit the emulator

David Cameron

2019-09-02

Emulating a simulator with two parameters

Since emulation in two dimensions in parameter space is essentially equivalent to geostatistics in physical space we'll use the same R package that was used before for geostatistics (geoR).

Here we imagine that we have an expensive simulator that we want to emulate that has just two uncertain parameters p1 and p2.

Imagine that can only afford to make three runs of the simulator. We feed in our three samples of the two parameters (data_p1 and data_p2) as inputs to the simulator and make the runs.

```
library(geoR)

data_p2     <- c( 0   , 4   , 4    ) ; ns <- length(data_p2)
data_p1     <- c( 3   , 0   , 3    )
```

In this simple example we will emulate just a single scaler output from the model so we now have three data values one for each run of the model.

```
data_values <- c( 0.7, 3.1, 2.2 )
```

Gaussian process (GP) prior

Mean function m(.)

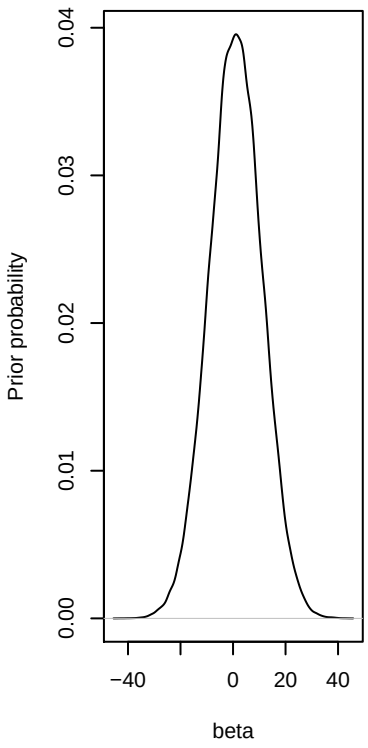
In this simple example for the prior we'll assume no trend ie that have no prior ideas about how the simulator varies in x=(p1,p2).

So our prior mean m(.) = single global mean value (β)

prior uncertainty in β

Here we will use a normal distribution

```
## mean
mb = 1.0
## standard deviation
Vb = 10.

par(mfrow = c(1, 3))
plot(density(rnorm(100000,mb,Vb)),xlab = "beta", ylab = "Prior probability",main="")
```



Covariance function c(.,.)

Stationary form of this is.

$$c(x, x') = \sigma^2 r(|x - x'|)$$

Here we will choose an exponential correlation function r.

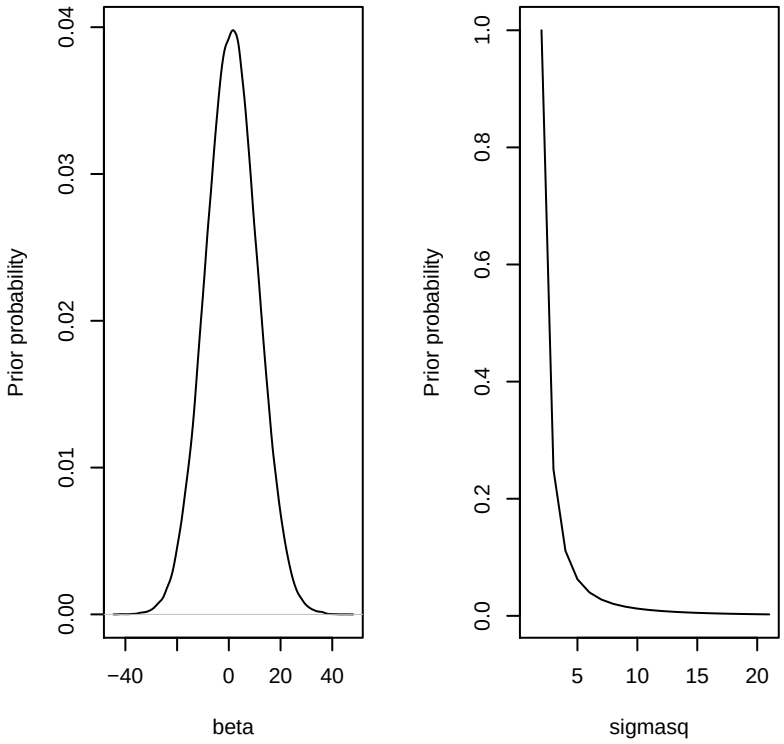
$$r = exp(-d/\phi)$$

where d is distance (= [x-x'|) and ϕ is the correlation length

prior uncertainty in σ^2

As for the spatial example we'll assume a "reciprocal" prior, where larger values become diminishingly probable, in inverse proportion to σ^2.

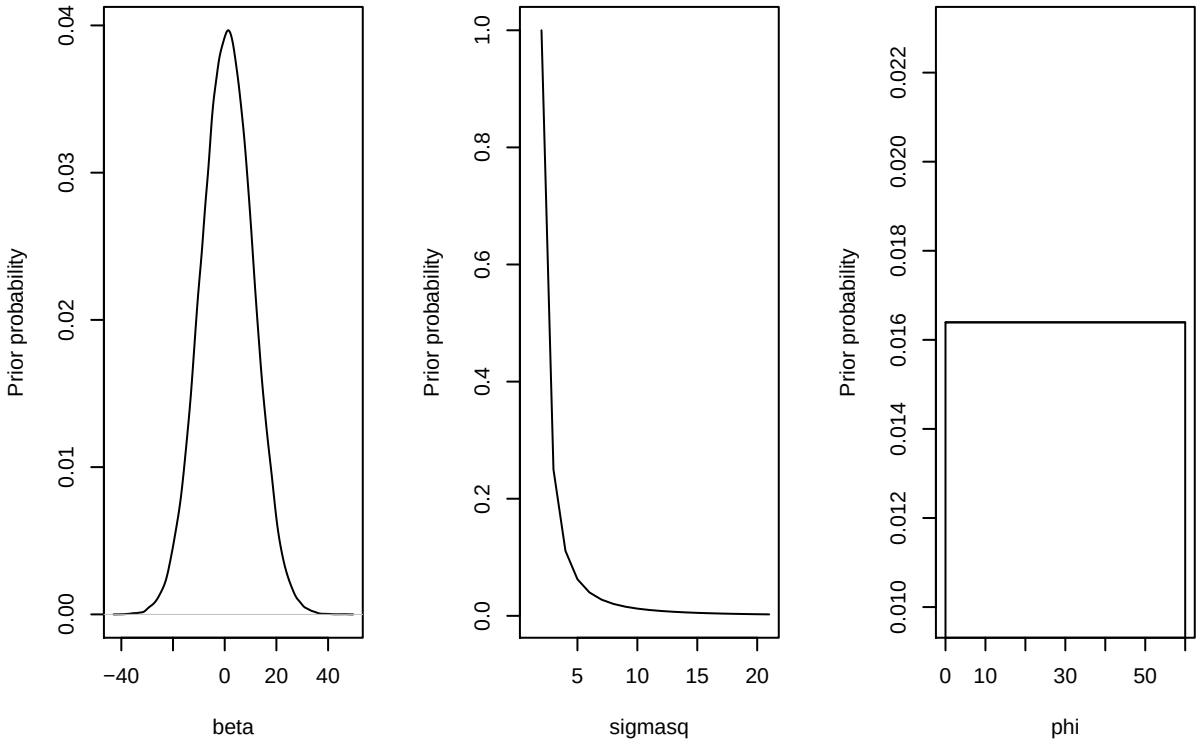
```
par(mfrow = c(1, 3))
plot(density(rnorm(100000,mb,Vb)),xlab = "beta", ylab = "Prior probability",main="")
plot((1/seq(0,2, by = 0.1)^2)/100, type = "l",
  xlab = "sigmasq", ylab = "Prior probability")
```



prior uncertainty correlation length ϕ

Here we'll assume a single correlation length for both p1 and p2 i.e. over the whole function space of the model. Again as before for geoR we'll discretise for phi and choose a uniform prior distribution.

```
seqphi_sparse <- seq(0,60,l=61)

par(mfrow = c(1, 3))
plot(density(rnorm(100000,mb,Vb)),xlab = "beta", ylab = "Prior probability",main="")
plot((1/seq(0,2, by = 0.1)^2)/100, type = "l",
  xlab = "sigmasq", ylab = "Prior probability")
plot(seqphi_sparse, rep(1/length(seqphi_sparse), length(seqphi_sparse)), type = "l",
  xlab = "phi", ylab = "Prior probability")
lines(c(0, seqphi_sparse, 60), c(0, rep(1/length(seqphi_sparse), length(seqphi_sparse)), 0), type = "l"
```



Populating GeoR

Having decided on our priors we can now populate GeoR with our choices.

Note that we set tausq to zero as the emulator should interpolate the simulator at the model points.

```
tausq         <- 0
s3            <- as.geodata( cbind(data_p2,data_p1,data_values) ) # data in geoR form
pred.grid     <- expand.grid(seq(-1,5,l=61), seq(-1,5,l=61)) # chose a predicition grid
```

We won't run the fitting just now as it will take too long but the code for it is given below.

```
emulationModel <- krige.bayes( s3,
 loc    = pred.grid,
 model  = model.control( cov.m = "exponential" ),
 prior  = prior.control(beta.prior      = "normal", beta = mb, beta.var.std = Vb,
                        sigmasq.prior   = "reciprocal",
                        phi.prior       = "uniform", phi.discrete = seqphi_sparse,
                        tausq.rel.prior = "fixed", tausq.rel    = tausq ),
  )

#setwd("~/Dropbox/CEH/courseTG15/emulation/")
#save(file="emulationModel60Recp.RData",emulationModel)
```

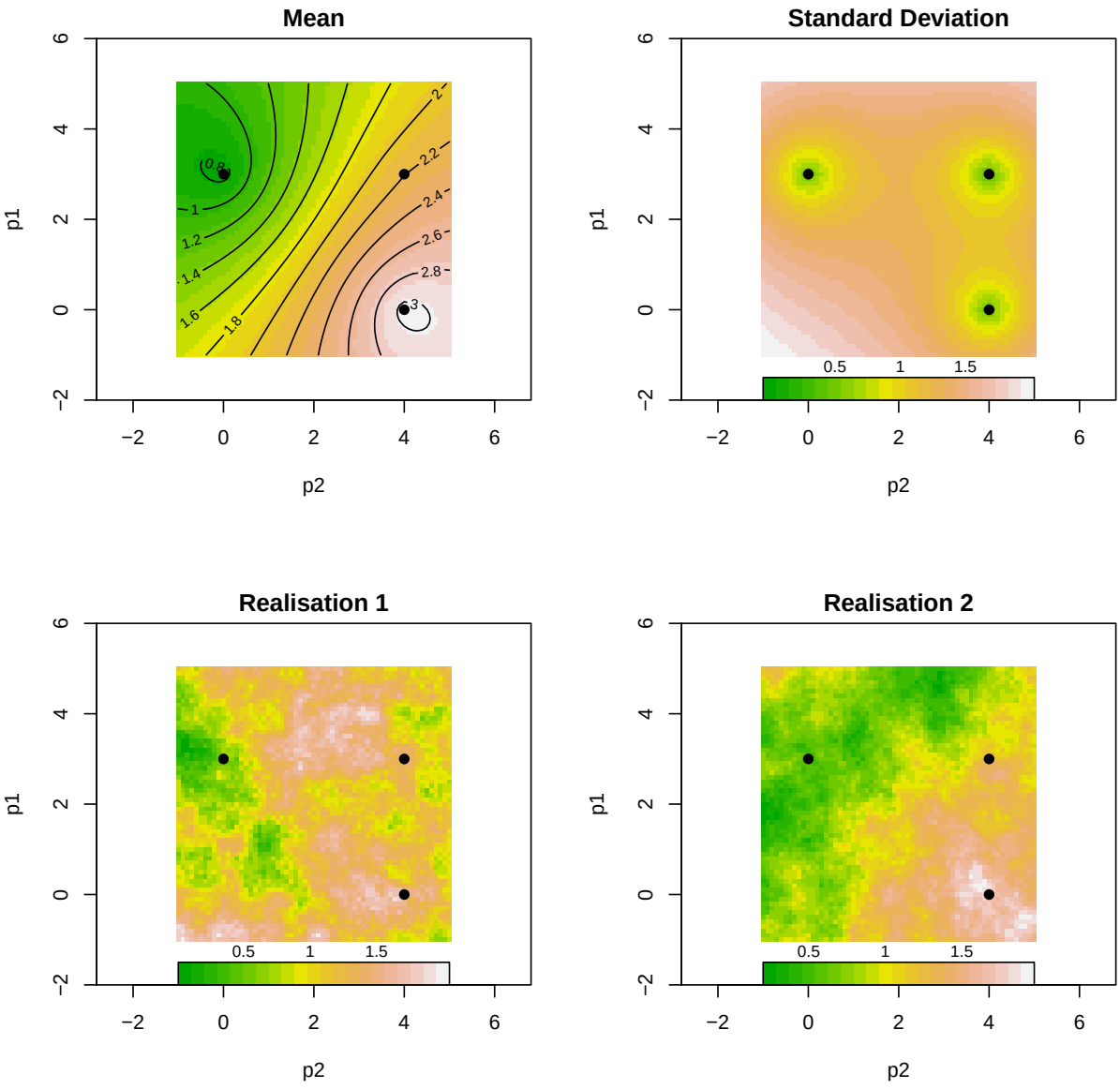
Load in an emulator fitted before. Create plots showing predictions from the emulator. The simulator runs are marked by black dots. As you can see the results look very similar to the results from spatial statistics

that we saw yesterday.

```
load(file="emulationModel60Recp.RData")

par(mfrow = c(2, 2))

## Mean
image  (emulationModel,
        val=emulationModel$predictive$mean,
        xlim=c(-2,6), ylim=c(-2,6),
        col=terrain.colors(21), main="\n\nMean", xlab="p2", ylab="p1")
points (s3, cex.max = 1.0, col = "black", add=T, pt.divide="equal" )
contour(emulationModel, add=T, nlev=11)
```

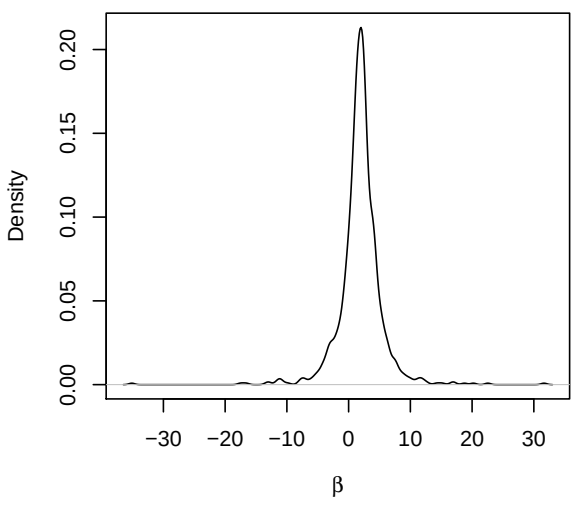
mapping the means of the predictive distribution

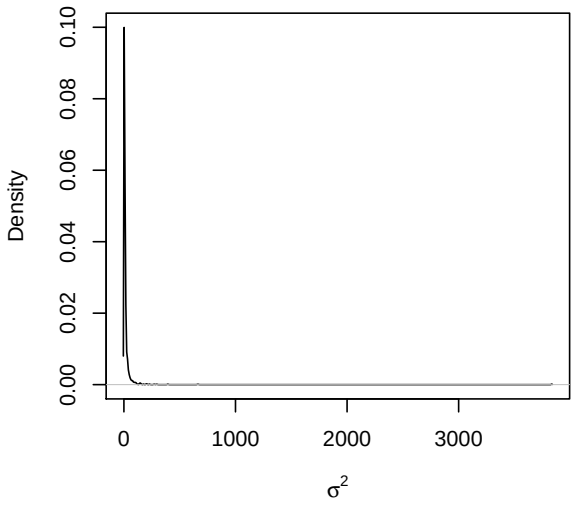
```
## Standard deviation
image  (emulationModel,
        val=sqrt(emulationModel$predictive$variance), xlim=c(-2,6), ylim=c(-2,6),
        col=terrain.colors(21), main="\n\nStandard Deviation", xlab="p2", ylab="p1")
points (s3, cex.max = 1.0, col = "black", add=T, pt.divide="equal" )
legend.krige(val=sqrt(emulationModel$predictive$variance),
             col=terrain.colors(21),
             x.leg=c(-1,5), y.leg=c(-2,-1.5) )

## Realisation 1
image  (emulationModel,
        val=emulationModel$predictive$simulations[,1],
        xlim=c(-2,6), ylim=c(-2,6),
        col=terrain.colors(21), main="\n\nRealisation 1", xlab="p2", ylab="p1")
points (s3, cex.max = 1.0, col = "black", add=T, pt.divide="equal" )
legend.krige(val=sqrt(emulationModel$predictive$simulations[,1]),
             col=terrain.colors(21),
             x.leg=c(-1,5), y.leg=c(-2,-1.5) )

## Realisation 2
image  (emulationModel,
        val=emulationModel$predictive$simulations[,2],
        xlim=c(-2,6), ylim=c(-2,6),
        col=terrain.colors(21), main="\n\nRealisation 2", xlab="p2", ylab="p1")
points (s3, cex.max = 1.0, col = "black", add=T, pt.divide="equal" )
legend.krige(val=sqrt(emulationModel$predictive$simulations[,2]),
             col=terrain.colors(21),
             x.leg=c(-1,5), y.leg=c(-2,-1.5) )
```

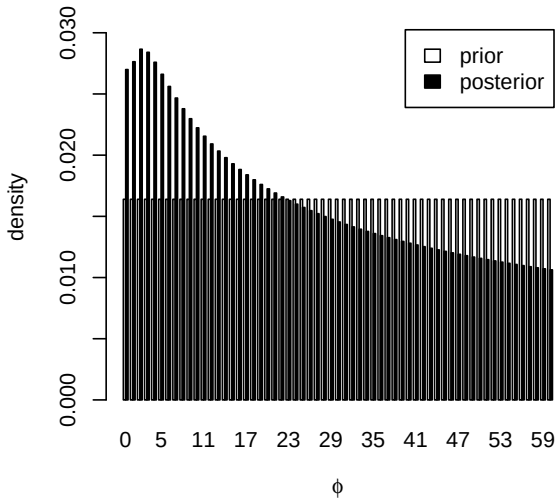



```
par(mfrow = c(2,2))
plot(density(emulationModel$posterior$sample$beta), main = "", xlab = expression(beta))
plot(density(emulationModel$posterior$sample$sigmasq), main = "", xlab = expression(sigma^2))
plot(emulationModel)
```

```
## parameter `tausq.rel` is fixed
```







Calibration using the emulator

Now we can calibrate the parameters p1 and p2 with the emulator. As before we can use BayesianTools for this.

Here we'll calibrate with the emulator mean taking also the mean value of the hyperparameters that were fitted above.

We have a single calibration data point 2.6. We choose here a normal likelihood function with a σ observation error of 0.1.

```
library(BayesianTools)
```

```
set.seed(123)

samplePost     = emulationModel$posterior$sample
mb             = mean(samplePost$beta)
sigmasq_star   = mean(samplePost$sigmasq)
phi_star       = mean(samplePost$phi)

likelihood <- function(par){

u0       <- c(par[2],par[1]) # We shall predict the value of z at the point u0 = {0,0}.
ex.bayes <- krige.bayes( s3,
  loc    = u0,
  model  = model.control( cov.m = "exponential" ),
  prior  = prior.control( beta.prior      = "fixed", beta      = mb,
                          sigmasq.prior   = "fixed", sigmasq   = sigmasq_star,
                          phi.prior       = "fixed", phi       = phi_star,
                          tausq.rel.prior = "fixed", tausq.rel = tausq ) )

llValue <- dnorm(ex.bayes$predictive$mean - 2.6, sd = 0.1, log = TRUE)
return(llValue)
}
```

We choose the same prior distribution for both of the parameters p1 and p2, the uniform distribution from -1 to 5.

```
prior <- createUniformPrior(lower = c(-1.0,-1.0), upper = c(5.0,5.0))
```

Since there are only two parameters the dimensions space we are sampling is small so a chain of 3000 is sufficient.

```
bayesianSetup <- createBayesianSetup(likelihood, prior, names = c('par1','par2'))

settings <- list(iterations = 3000, nrChains = 2)

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "DEzs", settings = settings)
```

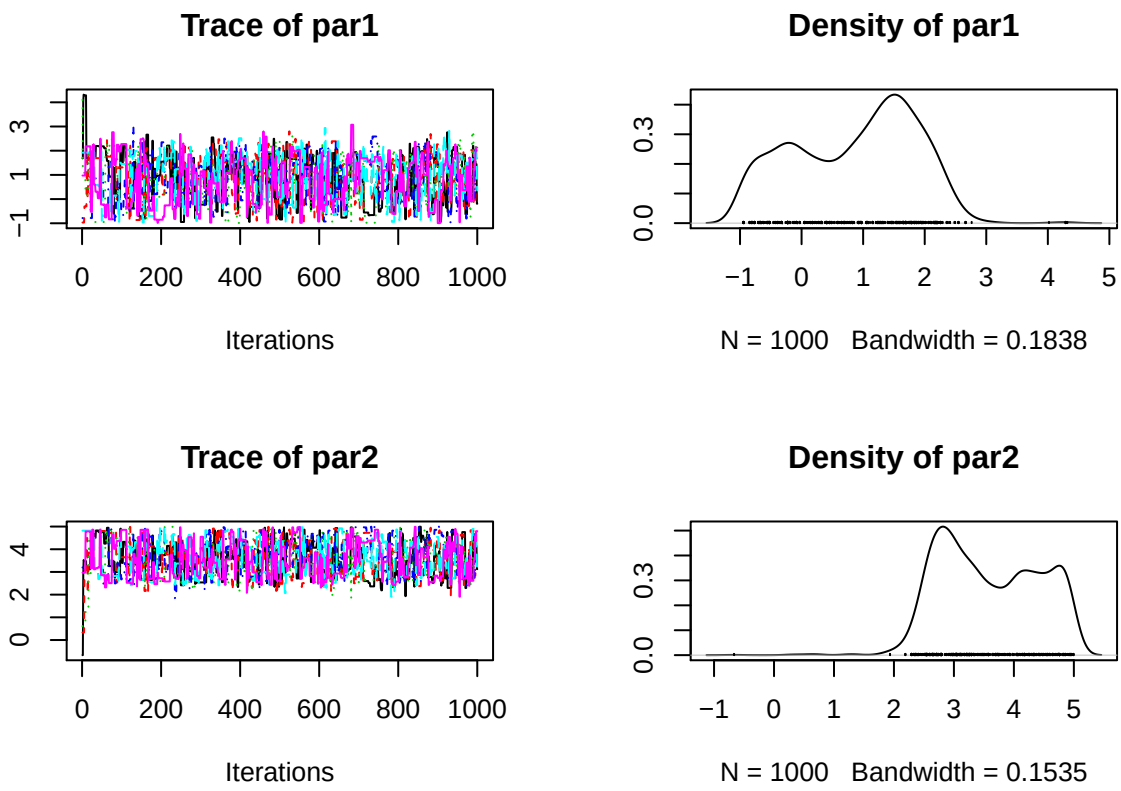
As before we can check that the calibration has converged.

```
gelmanDiagnostics(out)
```

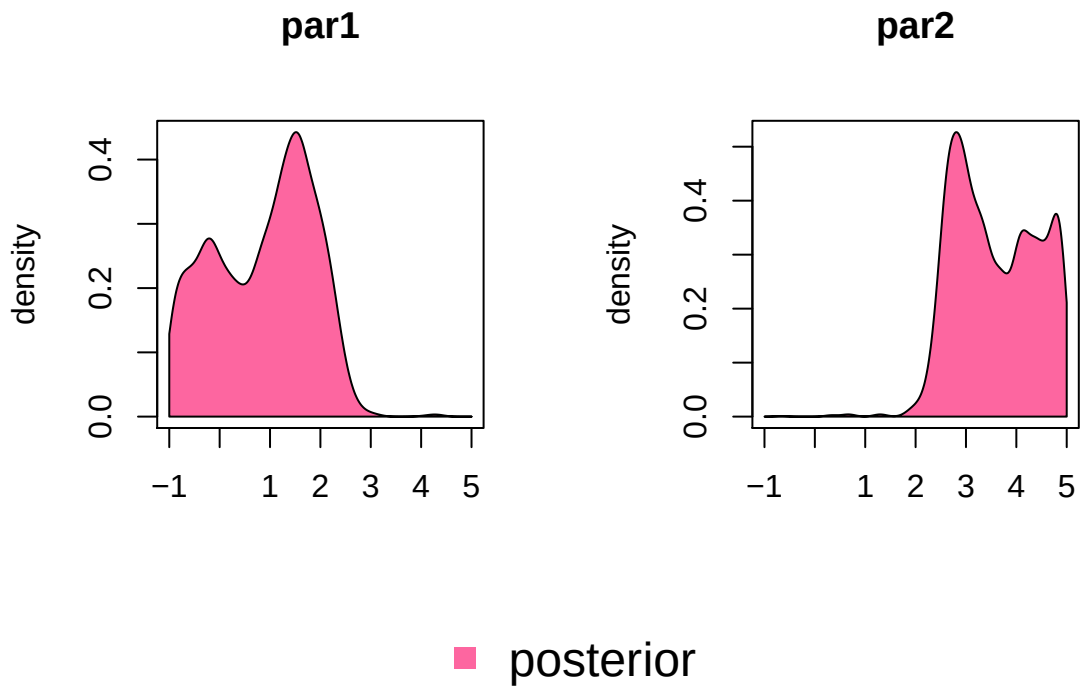
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## par1       1.01       1.02
## par2       1.02       1.05
##
## Multivariate psrf
##
## 1.04
```

```
tracePlot<-function (sampler, thin = "auto", ...)
{
    codaChain = getSample(sampler, coda = T, thin = thin, ...)
    plot(codaChain,smooth=FALSE)
}
```

```
tracePlot(out)
```

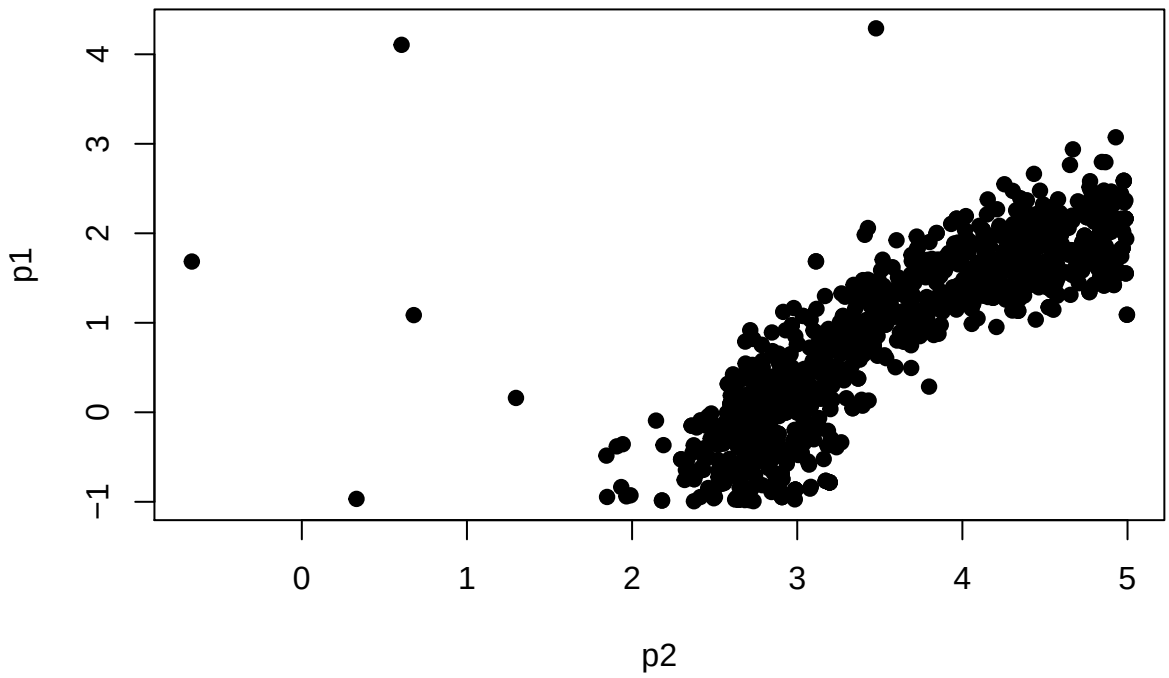



```
marginalPlot(out, singlePanel=FALSE,xrange = matrix(c(-1, 5, -1, 5),nrow=2,ncol=2,byrow=FALSE))
```



As before we can take a sample from the MCMC chain and plot the joint posterior.

```
parMatrix  = getSample(out, numSamples = 1002)

plot(parMatrix[,2],parMatrix[,1],pch=19,xlab="p2",ylab="p1")
```


p1
p2
−1
0
1
2
3
4
0
1
2
3
4
5