Recovering_rf_mixture

The objective of these simulations are to examine whether mixture parameters are recoverable with spatial data. We'll start by simulating some data on a grid from two gaussian random fields. We will initially assume that the fields have the same shape/scale parameters of the gaussian covariance function, but the variances are different – this allows one field to be normal, and the other fat-tailed.

```r
library(cluster)
library(mvtnorm)
library(R2jags)
library(fields)
library(ggplot2)
```

```r
grid = as.matrix(expand.grid("lon" = seq(5,15,1), "lat" = seq(5,15,1)))
nLocs = dim(grid)[1]
```

Approximating data with spatial random field

We could use RandomField, or other packages to do this. Initially we'll just simulate data using the estimation model. Use pam() to choose a number of knots on the grid. We'll specify 20 initially. We'll also jitter them slightly so knot locations don't fall exactly on stations.

```r
nKnots = 20
knots = jitter(pam(grid,nKnots)$medoids)
```

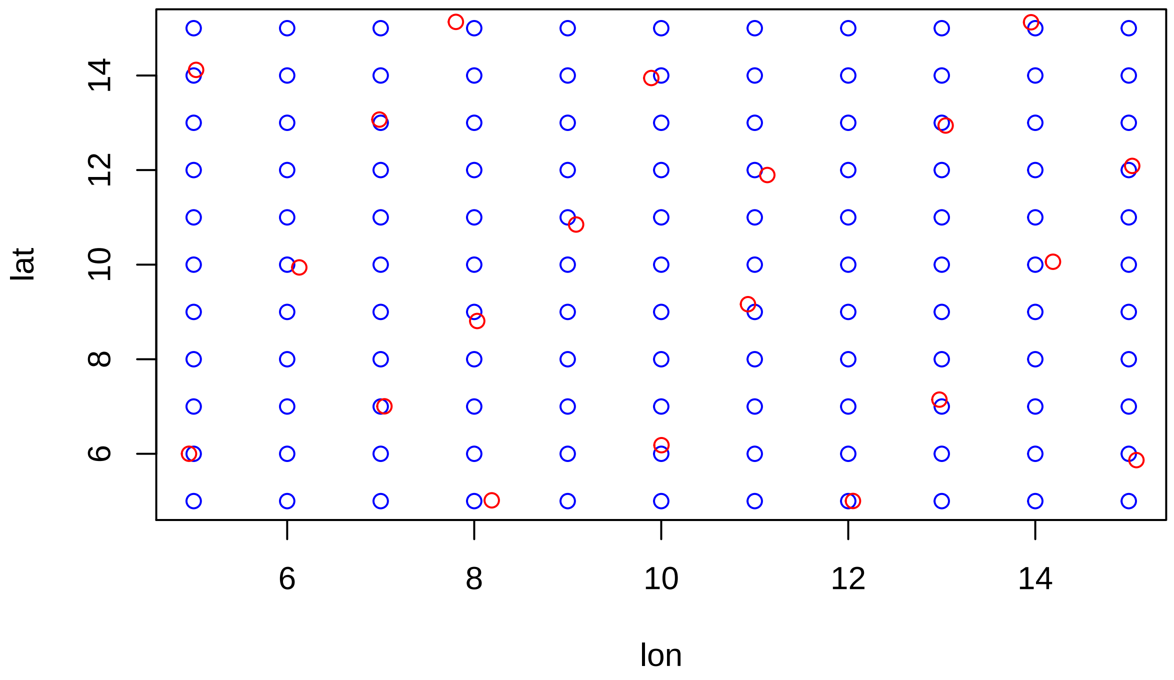

Figure 1: Simulated grid (blue) and knots for random effects (red)

Approximating data with spatial random field

Initially, we'll assume all data is observed in the same year, so there'll just be a single random effect field for normal years, and a single effect for catastrophic / abnormal years.

```r
# distance matrix of knots
distKnots = as.matrix(dist(knots))
distKnotsSq = distKnots^2 # squared distances
# note: shape parameter scaled to distance matrix
gp_scale = 0.001
sigma.norm = 0.001
sigma.fat = 0.002
corKnots = exp(-gp_scale*distKnotsSq)
Sigma.normal = corKnots * sigma.norm * sigma.norm
Sigma.fat = corKnots * sigma.fat * sigma.fat
```

Calculate distance matrix from stations to knots, and covariance matrices,

```r
# Calculate distance from knots to grid
distAll = as.matrix(dist(rbind(grid, knots)))^2
distKnots21Sq = t(distAll[1:nKnots, -c(1:nKnots)])

Sigma21.normal = exp(-gp_scale*distKnots21Sq) * sigma.norm * sigma.norm
Sigma21.fat = exp(-gp_scale*distKnots21Sq) * sigma.fat * sigma.fat
```

Plot the covariance of both models as a function of distance,

```r
x = seq(0,max(distAll),length.out=100)
cov.norm = exp(-gp_scale*x)*(sigma.norm^2)
cov.fat = exp(-gp_scale*x)*(sigma.fat^2)
plot(sqrt(x), cov.norm ,xlab="Distance", ylab="Covariance", type="l", lwd=3, col="black", ylim=range(c(
lines(sqrt(x), cov.fat ,lwd=3,lty=2)
```

Another way to visualize the normal v fat-tailed distribution is looking at the densities,

```r
dat <- data.frame(type = factor(rep(c("normal","fat"), each=10000)),
                  y = c(rnorm(10000,0,sigma.norm), rnorm(10000,0,sigma.fat)))
ggplot(dat, aes(x=y, colour=type)) + geom_density()
```

Generate single MVN field for each of the normal and 'fat-tailed' distributions.

```r
re.norm = rmvnorm(1, mean = rep(0, nKnots), Sigma.normal)
re.fat = rmvnorm(1, mean = rep(0, nKnots), Sigma.fat)

# Scale
re.norm = re.norm-mean(re.norm)
re.fat = re.norm-mean(re.fat)
```

Project to the station / grid locations, and combine the random fields using some proportion of 'fat-tailed' events. Initially, we'll use 0.05.

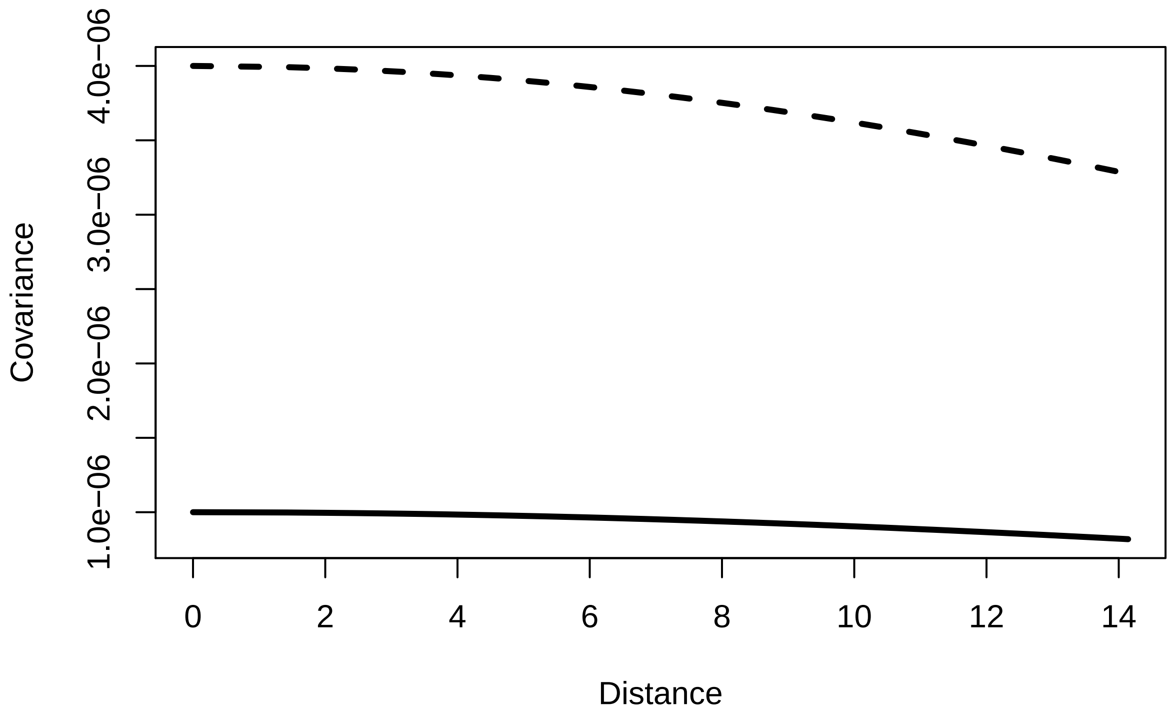

Figure 2: Gaussian covariance as function of distance for normal and fat-tailed distribution

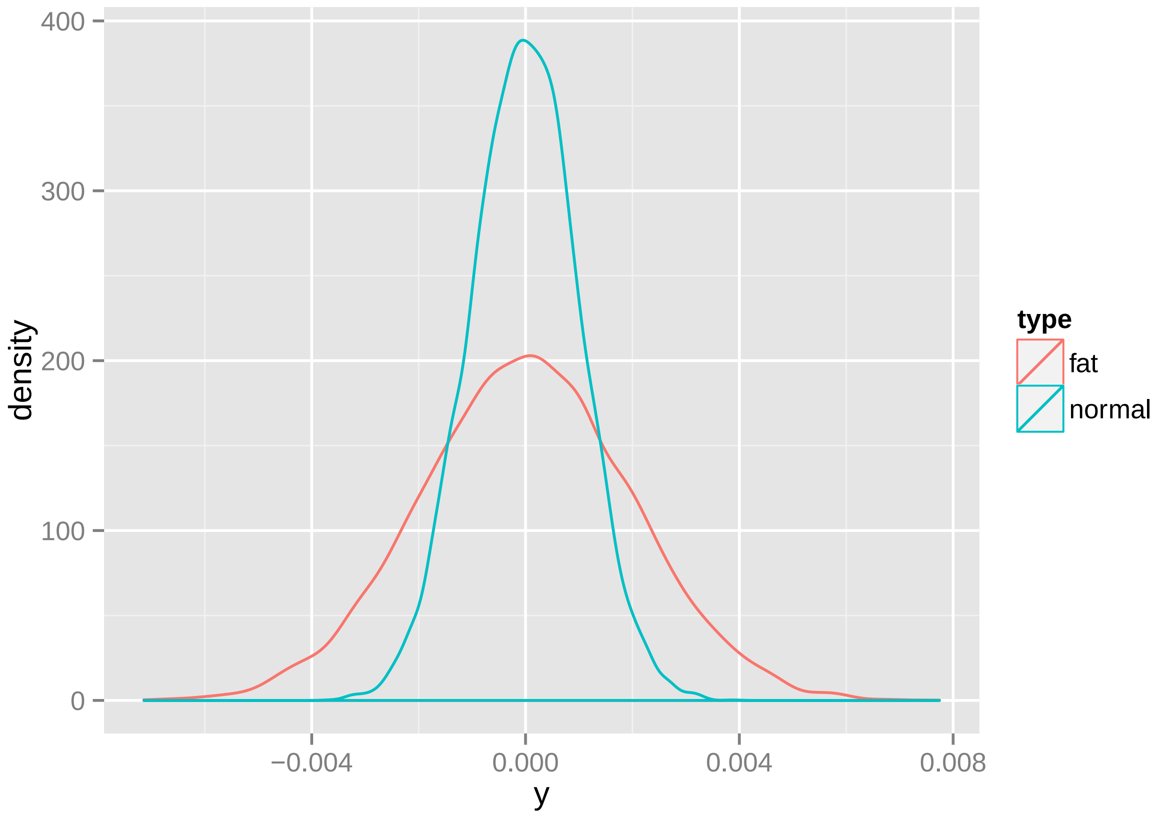

Figure 3: Normal and fat tailed distribution of the random effects

```r
# take inverse
invSigmaKnots.norm = solve(Sigma.normal)
invSigmaKnots.fat = solve(Sigma.fat)

# Project
proj.norm = Sigma21.normal %*% invSigmaKnots.norm %*% matrix(re.norm,ncol=1)
proj.fat = Sigma21.fat %*% invSigmaKnots.fat %*% matrix(re.fat,ncol=1)

# Combine to single RF
pfat = 0.1
proj = (1-pfat)*proj.norm + pfat*proj.fat
```

Next, we'll simulate 3 kinds of data from the spatial mixture: gaussian, binomial, and poisson counts.

```r
nPoints = nLocs
#indices = sample(seq(1,nLocs), size=nPoints, replace=T)
indices = seq(1,nPoints)
# assume no observation error, so observed/simulated gaussian data are same
x = grid[indices,]
y.gaussian = proj[indices,1]

y.binomial = rbinom(nPoints, size=1, prob=plogis(proj[indices,1]))

y.poisson = rpois(nPoints, lambda = exp(proj[indices,1]))

diagKnots = diag(nKnots)
muZeros = rep(0, nKnots)
```

Estimating normal mixture

The most important parameter we're interested in trying to recover is the contribution of the mixtures, and secondarily it's important to recover the variances scaling how extreme those events are.

As a first pass, we'll do it in JAGS, and we'll switch to TMB for the faster models linked with INLA.

```r
jagsscript = cat("
model {
   # priors on parameters for gaussian process
   gp_scaleInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_scale <- 1/gp_scaleInv;
   gp_sigmaSqInv_norm ~ dgamma(0.01,0.01); # prior on normal var parameter
   gp_sigmaSq_norm <- 1/gp_sigmaSqInv_norm;
   gp_sigmaSq_offset ~ dnorm(0,1)T(0,); # offset making fat tail > normal
   gp_sigmaSq_fat <- gp_sigmaSq_norm + gp_sigmaSq_offset;
     gp_sigmaSqInv_fat <- 1/gp_sigmaSq_fat;
   gp_jitterSqInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_jitterSq <- 1/gp_jitterSqInv;

   # This builds the 2 cov matrices needed
   # SigmaKnots is the COV matrix btween knots
   # SigmaOffDiag is the COV matrix between new locations and knots, e.g. it's (100 x 10)
   for(i in 1:nKnots) {
    for(j in 1:nKnots) {
     # this adds some jitter to the diagonal but not the off-diags
     SigmaKnotsNorm[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_norm * exp(-gp_scale * distKnotsSq[i,j]) + d
     SigmaKnotsFat[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_fat * exp(-gp_scale * distKnotsSq[i,j]) + dia
    }
   }
   for(i in 1:nLocs) {
    for(j in 1:nKnots) {
     SigmaOffDiagNorm[i,j] <- gp_sigmaSq_norm * exp(-gp_scale * distKnots21Sq[i,j]);
     SigmaOffDiagFat[i,j] <- gp_sigmaSq_fat * exp(-gp_scale * distKnots21Sq[i,j]);
    }
   }
   invSigmaKnotsNorm <- inverse(SigmaKnotsNorm[,]); # inverse of matrix for projection and mvn distribut
   invSigmaKnotsFat <- inverse(SigmaKnotsFat[,]); # inverse of matrix for projection and mvn distributio

   # mixture contribution of fat tailed events
   pfat ~ dunif(0,0.3);
   mufat ~ dnorm(0,1)T(,0);

   # Spatial random effects
   spatialEffectsKnotsNorm[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsNorm);
   spatialEffectsKnotsFat[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsFat);
   spatialEffects[1:nLocs,1] <- (1-pfat)*((SigmaOffDiagNorm %*% invSigmaKnotsNorm) %*% spatialEffectsKno

   # evaluate the likelihood
   resid.tau ~ dgamma(0.001,0.001);

   for(i in 1:nPoints) {
       pred[i] <- spatialEffects[indices[i], 1];
       y.gaussian[i] ~ dnorm(pred[i], resid.tau);
   }

}  ",file="recover_rf_gaussian.txt")
```

Run the model,

```
jags.data = list("nLocs","nKnots", "y.gaussian", "distKnotsSq", "distKnots21Sq","diagKnots","muZeros","
   jags.params=c("pfat","gp_sigmaSq_fat","gp_sigmaSq_norm","spatialEffectsKnotsNorm","spatialEffectsKnot
jagsmodel.gaussian = jags.parallel(jags.data, inits=NULL, parameters.to.save=jags.params, model.file="re
```

Estimating binomial mixture

We can also extend this to the binomial distribution by changing a single line in the JAGS script. The inverse logit transformation impacts how large the variance of the fat-tailed distribution needs to be.

```
jagsscript = cat("
model {
   # priors on parameters for gaussian process
   gp_scaleInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_scale <- 1/gp_scaleInv;
   gp_sigmaSqInv_norm ~ dgamma(0.01,0.01); # prior on normal var parameter
   gp_sigmaSq_norm <- 1/gp_sigmaSqInv_norm;
   gp_sigmaSq_offset ~ dnorm(0,1)T(0,); # offset making fat tail > normal
   gp_sigmaSq_fat <- gp_sigmaSq_norm + gp_sigmaSq_offset;
     gp_sigmaSqInv_fat <- 1/gp_sigmaSq_fat;
   gp_jitterSqInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_jitterSq <- 1/gp_jitterSqInv;

   # This builds the 2 cov matrices needed
   # SigmaKnots is the COV matrix btween knots
   # SigmaOffDiag is the COV matrix between new locations and knots, e.g. it's (100 x 10)
   for(i in 1:nKnots) {
    for(j in 1:nKnots) {
     # this adds some jitter to the diagonal but not the off-diags
     SigmaKnotsNorm[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_norm * exp(-gp_scale * distKnotsSq[i,j]) + d
     SigmaKnotsFat[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_fat * exp(-gp_scale * distKnotsSq[i,j]) + dia
    }
   }
   for(i in 1:nLocs) {
    for(j in 1:nKnots) {
     SigmaOffDiagNorm[i,j] <- gp_sigmaSq_norm * exp(-gp_scale * distKnots21Sq[i,j]);
     SigmaOffDiagFat[i,j] <- gp_sigmaSq_fat * exp(-gp_scale * distKnots21Sq[i,j]);
    }
   }
   invSigmaKnotsNorm <- inverse(SigmaKnotsNorm[,]); # inverse of matrix for projection and mvn distribut
   invSigmaKnotsFat <- inverse(SigmaKnotsFat[,]); # inverse of matrix for projection and mvn distributio
   # mixture distribution for fat-tailed events
   pfat ~ dunif(0,0.3);
   mufat ~ dnorm(0,1)T(,0);

   # Spatial random effects
   spatialEffectsKnotsNorm[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsNorm);
   spatialEffectsKnotsFat[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsFat);
   spatialEffects[1:nLocs,1] <- (1-pfat)*((SigmaOffDiagNorm %*% invSigmaKnotsNorm) %*% spatialEffectsKno

   # evaluate the likelihood
   for(i in 1:nPoints) {
       logit(pred[i]) <- min(max(spatialEffects[indices[i], 1],-20),20);
       y.binomial[i] ~ dbern(pred[i]);
   }

}  ",file="recover_rf_binomial.txt")
```

Run the model,

```
jags.data = list("nLocs","nKnots", "y.binomial", "distKnotsSq", "distKnots21Sq","diagKnots","muZeros","
   jags.params=c("pfat","gp_sigmaSq_fat","gp_sigmaSq_norm","spatialEffectsKnotsNorm","spatialEffectsKno
jagsmodel.binomial = jags.parallel(jags.data, inits=NULL, parameters.to.save=jags.params, model.file="re
```

Estimating Poisson mixture

We can also extend this to the Poisson distribution by changing a single line in the JAGS script.

```
jagsscript = cat("
model {
   # priors on parameters for gaussian process
   gp_scaleInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_scale <- 1/gp_scaleInv;
   gp_sigmaSqInv_norm ~ dgamma(0.01,0.01); # prior on normal var parameter
   gp_sigmaSq_norm <- 1/gp_sigmaSqInv_norm;
   gp_sigmaSq_offset ~ dnorm(0,1)T(0,); # offset making fat tail > normal
   gp_sigmaSq_fat <- gp_sigmaSq_norm + gp_sigmaSq_offset;
     gp_sigmaSqInv_fat <- 1/gp_sigmaSq_fat;
   gp_jitterSqInv ~ dgamma(0.01,0.01); # shared btw normal/fat
   gp_jitterSq <- 1/gp_jitterSqInv;

   # This builds the 2 cov matrices needed
   # SigmaKnots is the COV matrix btween knots
   # SigmaOffDiag is the COV matrix between new locations and knots, e.g. it's (100 x 10)
   for(i in 1:nKnots) {
    for(j in 1:nKnots) {
     # this adds some jitter to the diagonal but not the off-diags
     SigmaKnotsNorm[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_norm * exp(-gp_scale * distKnotsSq[i,j]) + d
     SigmaKnotsFat[i,j] <- (1-diagKnots[i,j]) * gp_sigmaSq_fat * exp(-gp_scale * distKnotsSq[i,j]) + dia
    }
   }
   for(i in 1:nLocs) {
    for(j in 1:nKnots) {
     SigmaOffDiagNorm[i,j] <- gp_sigmaSq_norm * exp(-gp_scale * distKnots21Sq[i,j]);
     SigmaOffDiagFat[i,j] <- gp_sigmaSq_fat * exp(-gp_scale * distKnots21Sq[i,j]);
    }
   }
   invSigmaKnotsNorm <- inverse(SigmaKnotsNorm[,]); # inverse of matrix for projection and mvn distribu
   invSigmaKnotsFat <- inverse(SigmaKnotsFat[,]); # inverse of matrix for projection and mvn distributi

   # mixture contribution of fat tailed events
   pfat ~ dunif(0,0.3);
   mufat ~ dnorm(0,1)T(,0);
   # Spatial random effects
   spatialEffectsKnotsNorm[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsNorm);
   spatialEffectsKnotsFat[1:nKnots,1] ~ dmnorm(muZeros, invSigmaKnotsFat);
   spatialEffects[1:nLocs,1] <- (1-pfat)*((SigmaOffDiagNorm %*% invSigmaKnotsNorm) %*% spatialEffectsKn

   # evaluate the likelihood
   for(i in 1:nPoints) {
       y.poisson[i] ~ dpois(exp(spatialEffects[indices[i], 1]));
   }

}  ",file="recover_rf_poisson.txt")
```

Run the model,

```
jags.data = list("nLocs","nKnots", "y.poisson", "distKnotsSq", "distKnots21Sq","diagKnots","muZeros","i
   jags.params=c("pfat","gp_sigmaSq_fat","gp_sigmaSq_norm","spatialEffectsKnotsNorm","spatialEffectsKno
jagsmodel.poisson = jags.parallel(jags.data, inits=NULL, parameters.to.save=jags.params, model.file="re
```

```r
par(mfrow=c(2,2),mgp=c(2,1,0),mai=c(0.5,0.5,0.3,0.1))
attach.jags(jagsmodel.gaussian, overwrite=TRUE)
hist(pfat, 40, col="grey",main="Gaussian",xlab="P(fat tail)")
attach.jags(jagsmodel.binomial, overwrite=TRUE)
hist(pfat, 40, col="grey",main="Binomial",xlab="P(fat tail)")
attach.jags(jagsmodel.poisson, overwrite=TRUE)
hist(pfat, 40, col="grey",main="Poisson",xlab="P(fat tail)")
```

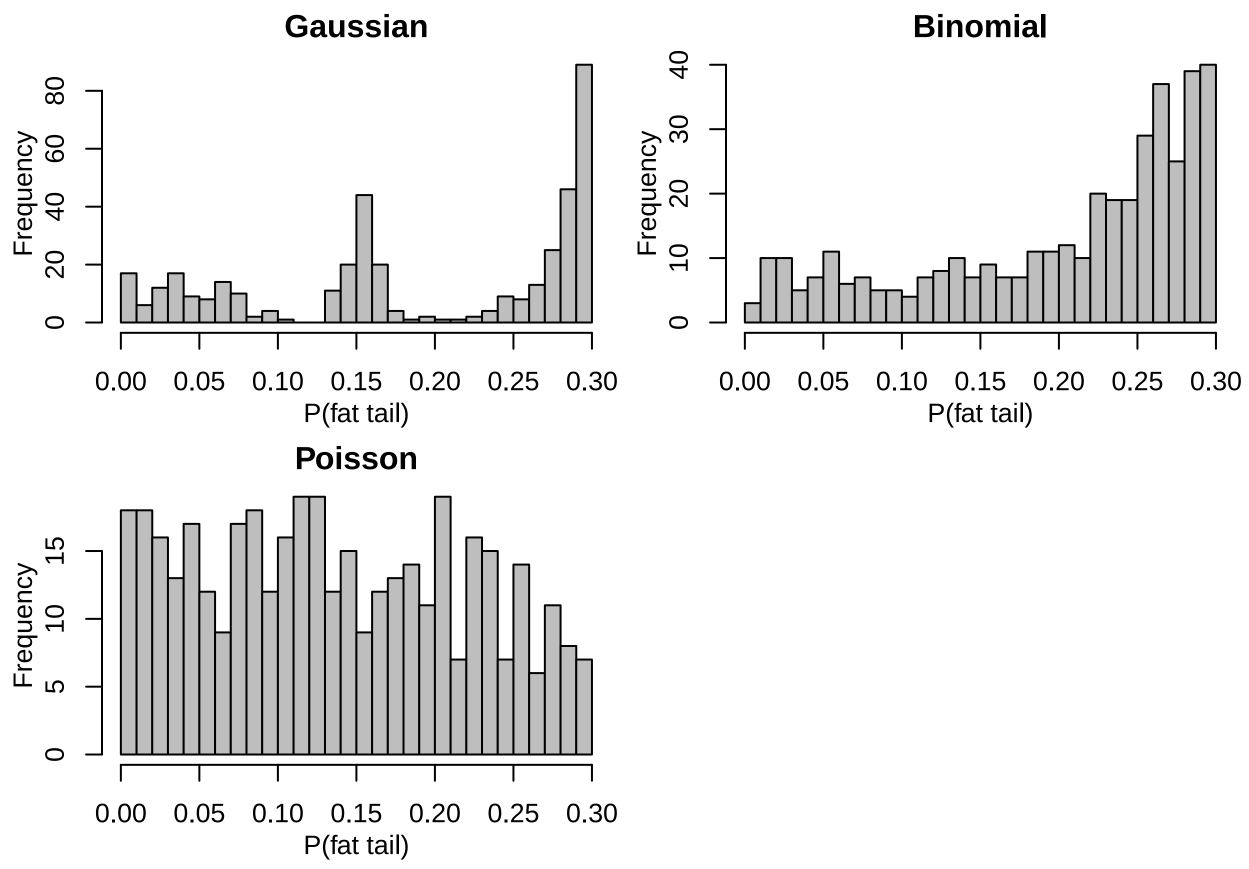

Figure 4: Estimates of contribution of fat-tailed distribution to mixture

```r
attach.jags(jagsmodel.gaussian, overwrite=TRUE)
sd.gaussian = sqrt(c(gp_sigmaSq_norm, gp_sigmaSq_fat))
n.mcmc = length(gp_sigmaSq_norm)
attach.jags(jagsmodel.binomial, overwrite=TRUE)
sd.binomial = sqrt(c(gp_sigmaSq_norm, gp_sigmaSq_fat))

attach.jags(jagsmodel.gaussian, overwrite=TRUE)
sd.poisson = sqrt(c(gp_sigmaSq_norm, gp_sigmaSq_fat))

boxplot(c(sd.gaussian, sd.binomial, sd.poisson) ~ sort(rep(1:6, n.mcmc)), outline = FALSE, names = c("G
```

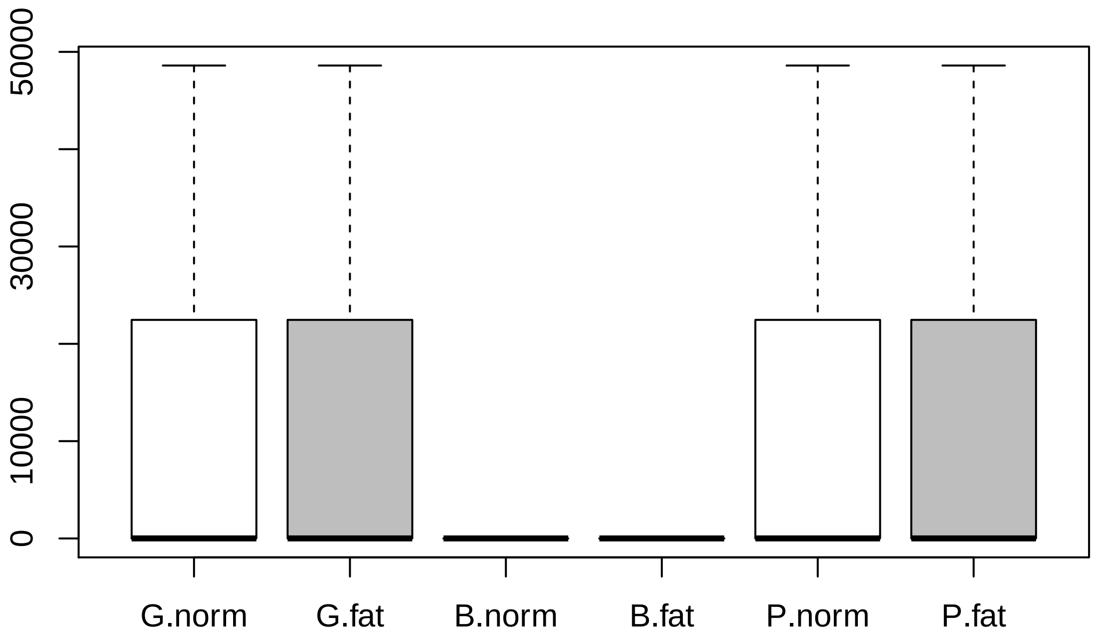

Figure 5: Estimates of contribution of fat-tailed distribution to mixture