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.

```
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.

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

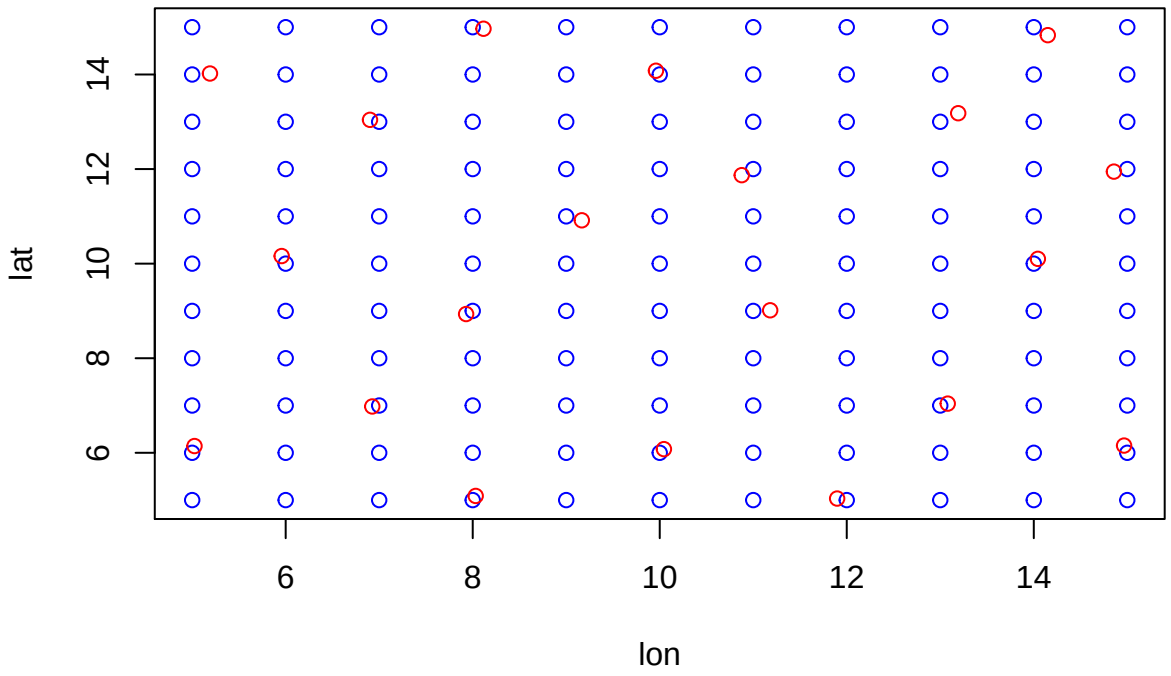

Figure 1: Simulated grid (blue) and knots for random effects (red)

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.

```
# 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,

```
# 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,

```
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(cov.norm, cov.fat)))
lines(sqrt(x), cov.fat, lwd = 3, lty = 2)
```

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

```
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.

```
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.

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

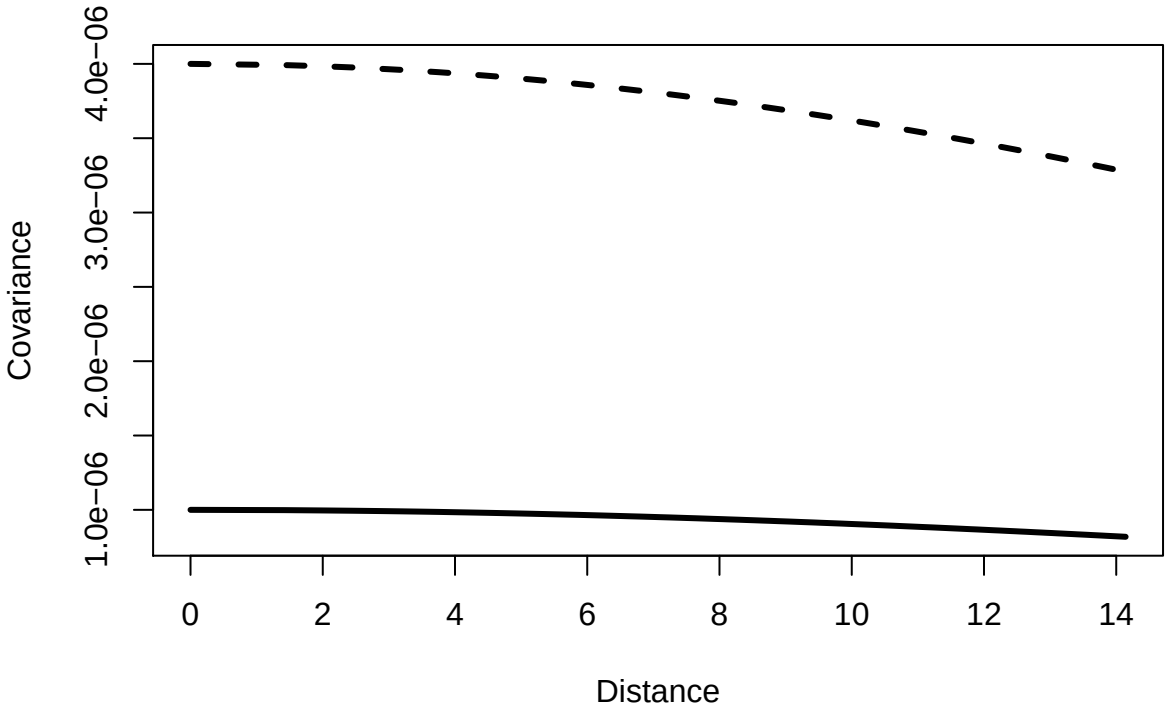

Figure 2: Gaussian covariance as function of distance for normal and fat-tailed distribution

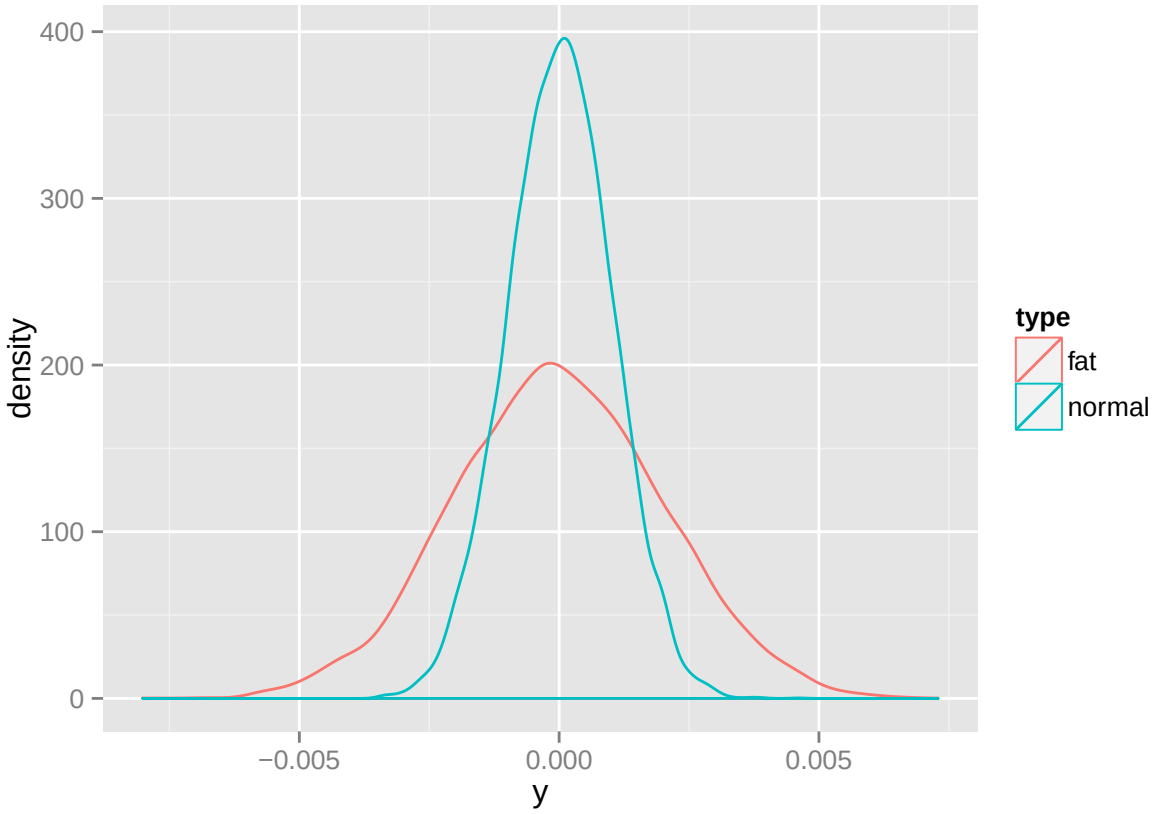

Figure 3: Normal and fat tailed distribution of the random effects

```r
# 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]

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(0,0.01)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.norm ~ dnorm(0,1)T(,0); # constrain mean to be negative

  # 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
  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", "indices", "nPoints")
jags.params = c("pfat", "gp_sigmaSq_fat", "gp_sigmaSq_norm", "spatialEffectsKnotsNorm",
    "spatialEffectsKnotsFat")
jagsmodel.gaussian = jags.parallel(jags.data, inits = NULL, parameters.to.save = jags.params,
    model.file = "recover_rf_gaussian.txt", n.chains = 4, n.burnin = 20000,
    n.thin = 10, n.iter = 30000, DIC = TRUE)
```

Estimating binomial mixture

The inverse logit transformation impacts how large the variance of the fat-tailed distribution needs to be. For example, in the normal mixture model, a difference in the variance between 0.001 and 0.002 may be detected. But if random effects are centered on 0, the spatial field of the probabilities of occurrence may have a very small range (e.g. 0.47 - 0.53), and the mixture parameters won't be estimable.

Extreme negative events in the mixture will make the distribution of occurrences zero-inflated; extreme positive events will make the occurrences bimodal.

```
# 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
```

```r
sigma.fat = 0.1
corKnots = exp(-gp_scale * distKnotsSq)
Sigma.normal = corKnots * sigma.norm * sigma.norm
Sigma.fat = corKnots * sigma.fat * sigma.fat
# 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
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)
# 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
mufat = 0
proj = (1 - pfat) * proj.norm + pfat * (mufat + proj.fat)

# Simulate binary data,
nPoints = nLocs
indices = seq(1, nPoints)
x = grid[indices, ]
y.binomial = rbinom(nPoints, size = 1, prob = plogis(proj[indices, 1]))
```

```r
hist(plogis(proj), xlab = "Mixture of Pr(occurrence)", col = "grey", main = "")
```

```
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(0,0.01)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) {
```

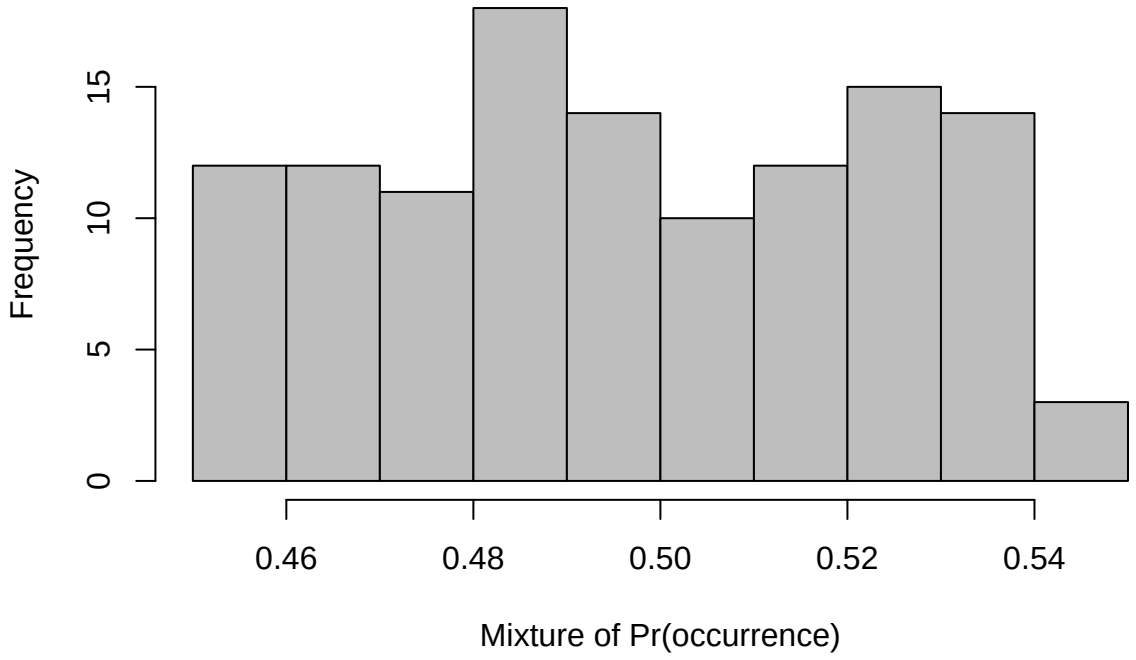

Figure 4: Simulated mixture distribution of occurence probabilities

```
 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 distribution for fat-tailed events
pfat ~ dunif(0,0.3);
mufat.norm ~ dunif(0,0.5); # change this 0.5 later
mufat.logit <- log(mufat.norm/(1-mufat.norm));

# 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) {
    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", "indices", "nPoints")
jags.params = c("pfat", "gp_sigmaSq_fat", "gp_sigmaSq_norm", "spatialEffectsKnotsNorm",
    "spatialEffectsKnotsFat")
jagsmodel.binomial = jags.parallel(jags.data, inits = NULL, parameters.to.save = jags.params,
    model.file = "recover_rf_binomial.txt", n.chains = 4, n.burnin = 20000,
    n.thin = 10, n.iter = 30000, DIC = TRUE)
```

Estimating Poisson mixture

The Poisson simulated data has the same problem as the binomial – negative events make the distribution zero-inflated, and positive will make variance > mean.

```
# 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 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
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)
# 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
mufat = 0
proj = (1 - pfat) * proj.norm + pfat * (mufat + proj.fat)

# Simulate Poisson count data,
nPoints = nLocs
```

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

```
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(0,0.01)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.norm ~ dunif(0.00000001,1); # change this upper bound later
   mufat.log <- log(mufat.norm);

   # 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(min(20, spatialEffects[indices[i], 1])));
   }

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

Run the model,

```
jags.data = list("nLocs", "nKnots", "y.poisson", "distKnotsSq", "distKnots21Sq",
    "diagKnots", "muZeros", "indices", "nPoints")
jags.params = c("pfat", "gp_sigmaSq_fat", "gp_sigmaSq_norm", "spatialEffectsKnotsNorm",
    "spatialEffectsKnotsFat")
jagsmodel.poisson = jags.parallel(jags.data, inits = NULL, parameters.to.save = jags.params,
    model.file = "recover_rf_poisson.txt", n.chains = 4, n.burnin = 20000, n.thin = 10,
    n.iter = 30000, DIC = TRUE)
```

```
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)")
```

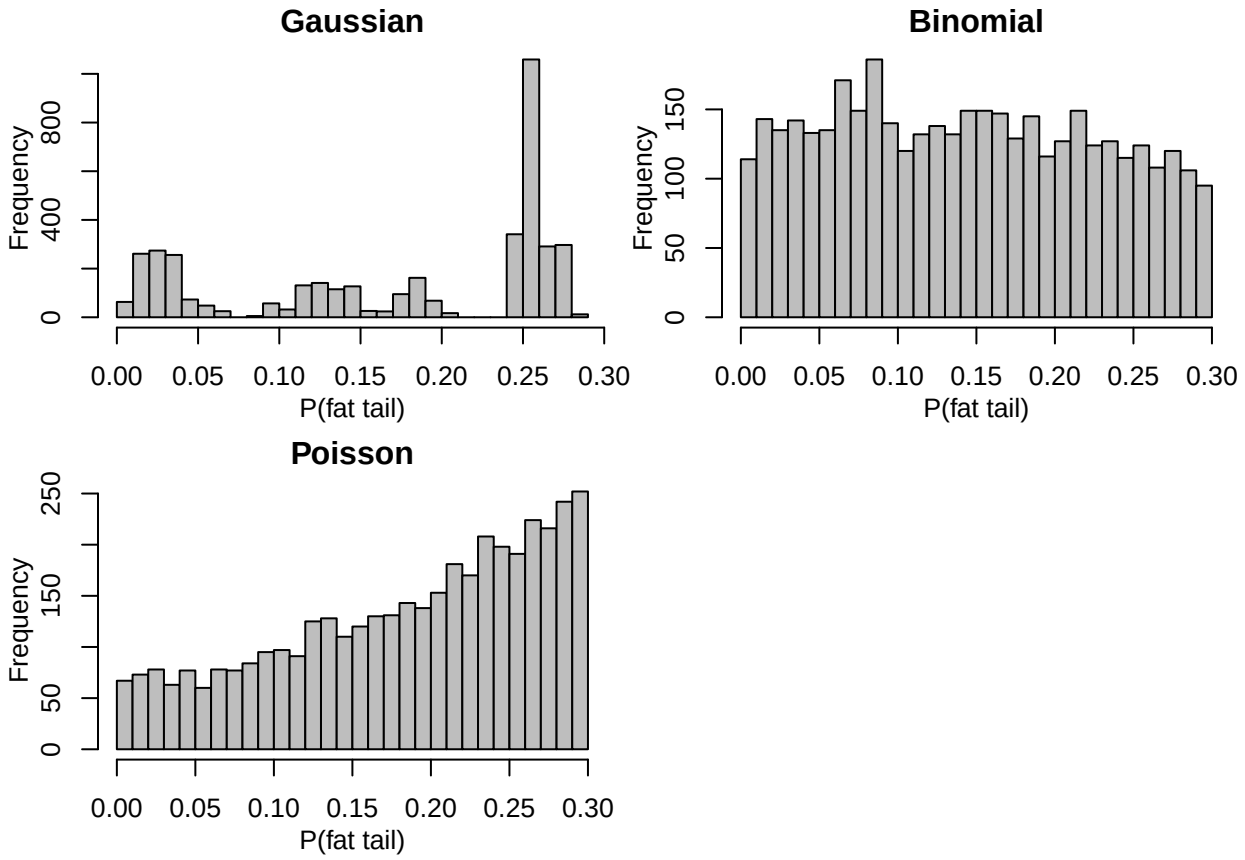

Figure 5: Estimates of contribution of fat-tailed distribution to mixture

```
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.norm", "G.fat", "B.norm", "B.fat", "P.norm", "P.fat"), col = rep(c("white",
        "grey"), 3))
```

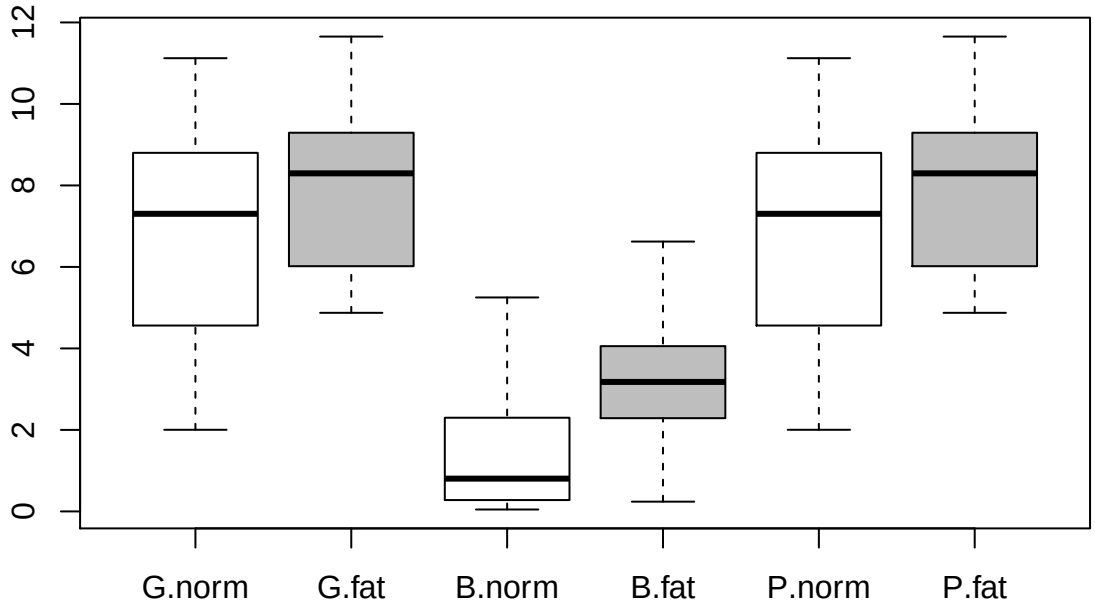

Figure 6: Estimates of contribution of fat-tailed distribution to mixture

Spatial replication

It may be possible to improve estimation of the catastrophic mixture parameters if we include spatial replication. In the context of the fish datasets we're working with, these are like combining samples from multiple years (the underlying gaussian random field is assumed to be constant and not changing over time).

We'll initially explore this with the binomial and Poisson distributions, and simulating 5 years of data

Binomial

```
# 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 = 0.1
corKnots = exp(-gp_scale * distKnotsSq)
Sigma.normal = corKnots * sigma.norm * sigma.norm
Sigma.fat = corKnots * sigma.fat * sigma.fat
# 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
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)
# 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
mufat = 0
proj = (1 - pfat) * proj.norm + pfat * (mufat + proj.fat)

# Simulate binary data,
nPoints = nLocs * 5
indices = rep(seq(1, nLocs), 5)
x = grid[indices, ]
y.binomial = rbinom(nPoints, size = 1, prob = plogis(proj[indices, 1]))
```

```
jags.data = list("nLocs", "nKnots", "y.binomial", "distKnotsSq", "distKnots21Sq",
    "diagKnots", "muZeros", "indices", "nPoints")
jags.params = c("pfat", "gp_sigmaSq_fat", "gp_sigmaSq_norm", "spatialEffectsKnotsNorm",
    "spatialEffectsKnotsFat")
jagsmodel.binomial.rep = jags.parallel(jags.data, inits = NULL, parameters.to.save = jags.params,
    model.file = "recover_rf_binomial.txt", n.chains = 4, n.burnin = 20000,
    n.thin = 10, n.iter = 30000, DIC = TRUE)
```

Compare the densities of the estimated fat-tailed contribution between the model with a single replicate, and 5 years of data

```
attach.jags(jagsmodel.binomial, overwrite = TRUE)
par(mfrow = c(2, 1), mgp = c(2, 1, 0), mai = c(0.5, 0.6, 0.4, 0.05))
hist(pfat, 40, col = "grey", main = "Single replicate")
attach.jags(jagsmodel.binomial.rep, overwrite = TRUE)
hist(pfat, 40, col = "grey", main = "Multiple replicates")
```

Poisson

```
# distance matrix of knots
distKnots = as.matrix(dist(knots))
distKnotsSq = distKnots^2  # squared distances
```

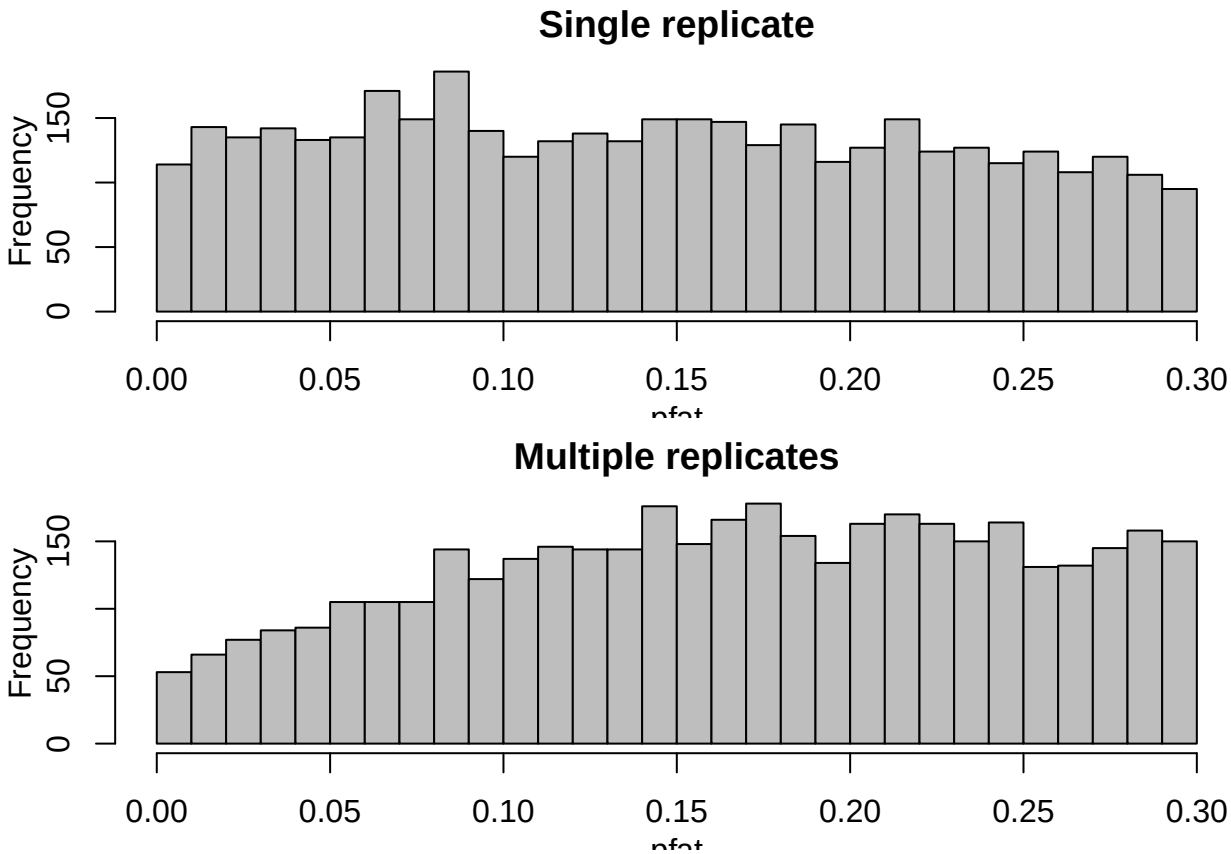

Figure 7: Estimated contribution of wider tailed distribution for binomial data

```
# 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 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
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)
# 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
mufat = 0
proj = (1 - pfat) * proj.norm + pfat * (mufat + proj.fat)

# Simulate poisson data,
nPoints = nLocs * 5
indices = rep(seq(1, nLocs), 5)
x = grid[indices, ]
y.poisson = rpois(nPoints, lambda = exp(proj[indices, 1]))
```

```
jags.data = list("nLocs", "nKnots", "y.poisson", "distKnotsSq", "distKnots21Sq",
    "diagKnots", "muZeros", "indices", "nPoints")
jags.params = c("pfat", "gp_sigmaSq_fat", "gp_sigmaSq_norm", "spatialEffectsKnotsNorm",
    "spatialEffectsKnotsFat")
jagsmodel.poisson.rep = jags.parallel(jags.data, inits = NULL, parameters.to.save = jags.params,
    model.file = "recover_rf_poisson.txt", n.chains = 4, n.burnin = 20000, n.thin = 10,
    n.iter = 30000, DIC = TRUE)
```

Compare the densities of the estimated fat-tailed contribution between the model with a single replicate, and 5 years of data

```
attach.jags(jagsmodel.poisson, overwrite = TRUE)
par(mfrow = c(2, 1), mgp = c(2, 1, 0), mai = c(0.5, 0.6, 0.4, 0.05))
hist(pfat, 40, col = "grey", main = "Single replicate")
attach.jags(jagsmodel.poisson.rep, overwrite = TRUE)
hist(pfat, 40, col = "grey", main = "Multiple replicates")
```

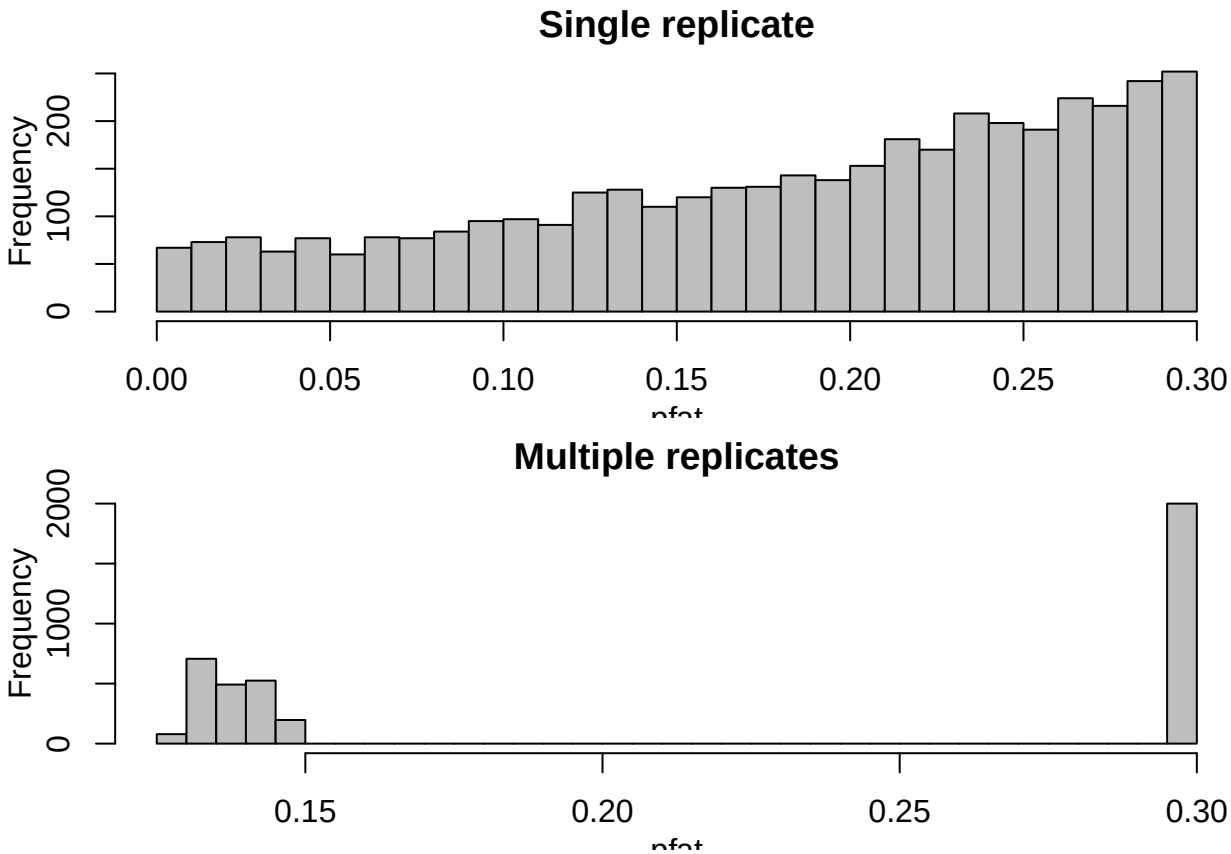

Figure 8: Estimated contribution of wider tailed distribution for Poisson data