Demo

```
library(rnaGinesis)
library(ggplot2)
library(reshape)
library(GGally)
library(NMF)
```

```
## Loading required package: pkgmaker
```

```
## Loading required package: registry
```

```
##
## Attaching package: 'pkgmaker'
```

```
## The following object is masked from 'package:base':
##
##     isNamespaceLoaded
```

```
## Loading required package: rngtools
```

```
## Loading required package: cluster
```

```
## NMF - BioConductor layer [OK] | Shared memory capabilities [NO: bigmemory] | Cores 23/24
```

```
##   To enable shared memory capabilities, try: install.extras('
## NMF
## ')
```

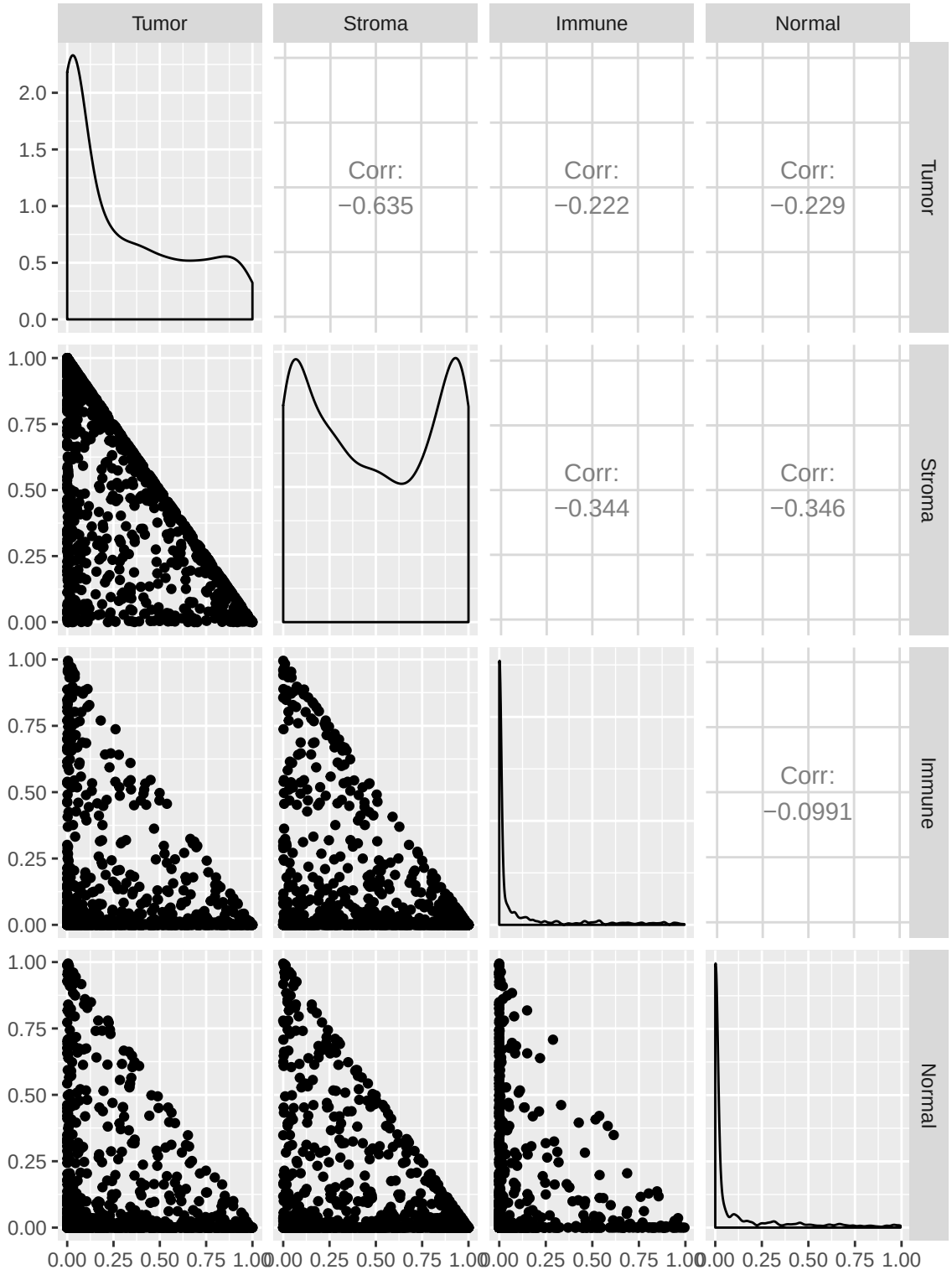
```
library(nnls)
mu <- rnaGinesis::mu
A <- rnaGinesis::A
```

Generate simulated data

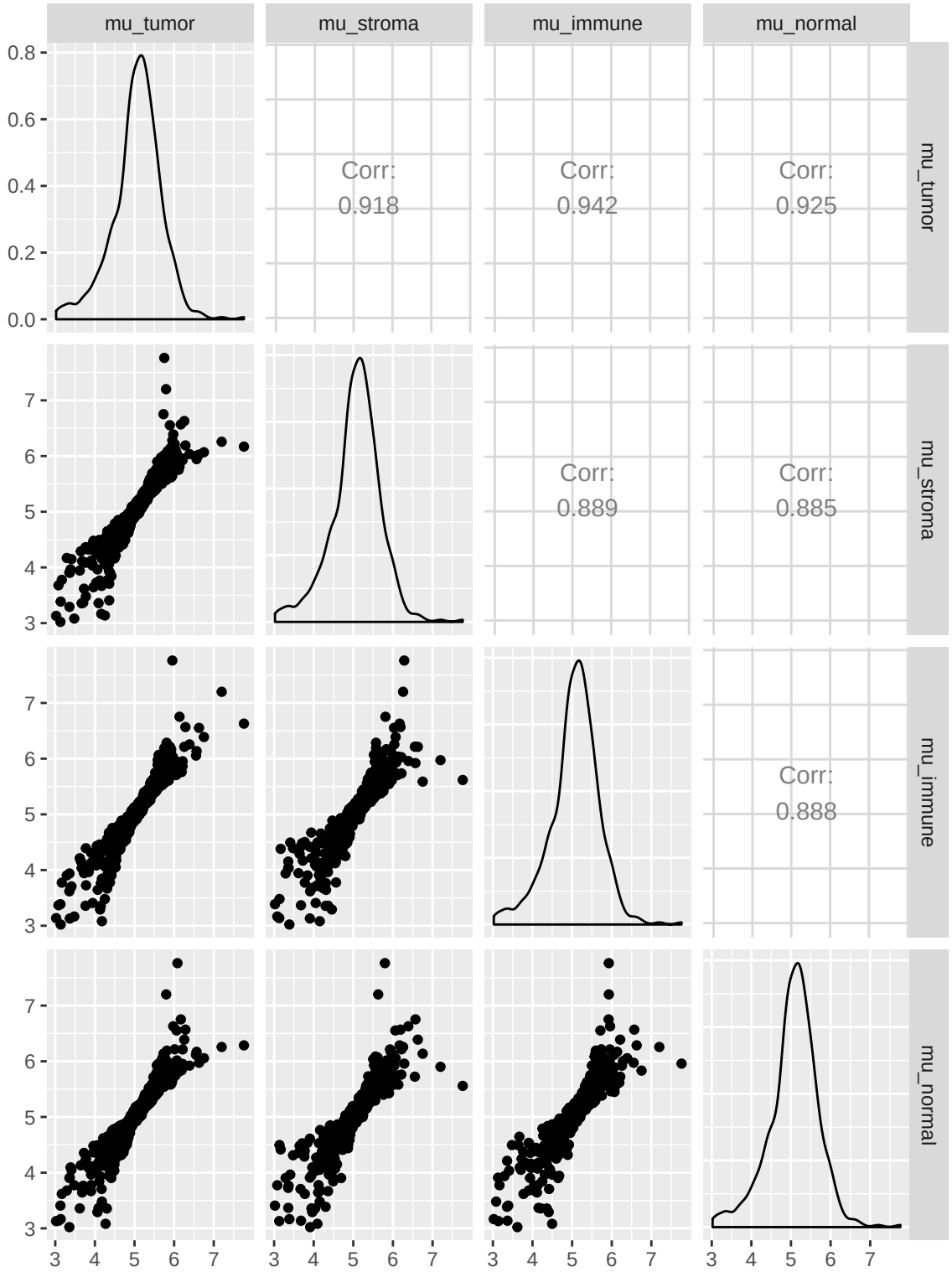
```
ngenes <- 500;
nsamples = 1000;
simresult <- Complete_simulation(A_tumor  = A[1:ngenes,1:ngenes],
                                 mu_tumor = mu[1:ngenes],
                                 Samplesize    = nsamples,
                                 scaleFactor   = 100,
                                 d.params      = c("Tumor"   = .3,
                                                   "Stroma" = .5,
                                                   "Immune"  = .1,
                                                   "Normal"  = .1),
                                 noise_setting = 1.5,
                                 seed          = 1234)
data   <- simresult[[1]]
W      <- simresult[[2]]
H      <- simresult[[3]]
rm("simresult")
rm("mu")
rm("A")
```

Display data properties

```
ggpairs(data = data.frame(t(H)))
```



```
ggpairs(data = as.data.frame(lapply(data.frame(W),log10)))
```



Define functions

```
my_nnls <- function(W,X){
  # X = WH
  # solve for H
  pinv <- (solve(t(W)%*%W)%*%t(W))
  H <- pinv%*%X
  H[H<0] <- 0
  residual <- X - (W %*% H)
  return(list("H" = H,
              "residuals" = residual))
}

fuzzy_nmf <- function(W,X){
  # X = WH
  # solve for H

  # start with nnls result, but don't let zeros happen
  pinv <- (solve(t(W)%*%W)%*%t(W))
  H <- pinv%*%X
  H[H<max(H)*0.1] <- max(H)*0.1

  # add a dummy factor, populate it with garbage
  ngenes = dim(W)[1]
  nfactors = dim(W)[2]
  W <- cbind(W , runif(n = ngenes) )
  H <- rbind(H , 0.1 )
  H.original <- H

  residual <- X - (W %*% H)
  exitcond <- TRUE
  while(exitcond){
    # multiplicative NMF update, modified

    # stay close to known H
    H.new <- H *  ( t(W) %*% X ) / (t(W) %*% W %*% H )
    H[nfactors + 1] <- H.new[nfactors + 1]
    H[1:nfactors]   <- 0.5 * H.new[1:nfactors] + 0.5 * H.original[1:nfactors]

    # keep known W fixed
    W.new <- W *  ( X %*% t(H) ) / (W %*% H %*% t(H))
    W[nfactors + 1] <- W.new[nfactors + 1]

    new.residual <- X - (W %*% H)
    change = sum(abs(residual)) /sum(abs(new.residual))
    residual <- new.residual
    exitcond <- change > 1.001
  }
  return(list("H" = H[1:nfactors],
              "residual" = H[nfactors+1]))
}
```

Unmix single samples

perfect knowledge

```
known.set <- 1:4
nnls.H.perfect <- H[known.set,]
nnls.resids.perfect <- numeric(nsamples)
nmf.H.perfect <- H[known.set,]
nmf.resids.perfect  <- numeric(nsamples)

for(i in 1:nsamples){
  this_sample <- data[,i]

  nnls.result <- my_nnls(W[,known.set],this_sample)
  nmf.result <- fuzzy_nmf(W[,known.set],this_sample)

  nnls.H.perfect[,i] <- nnls.result$H
  nnls.resids.perfect[i] <- sum((nnls.result$residuals[nnls.result$residuals>0])) / (ngenes * mean(mean

  nmf.H.perfect[,i] <- nmf.result$H
  nmf.resids.perfect[i] <- nmf.result$residual

}
```

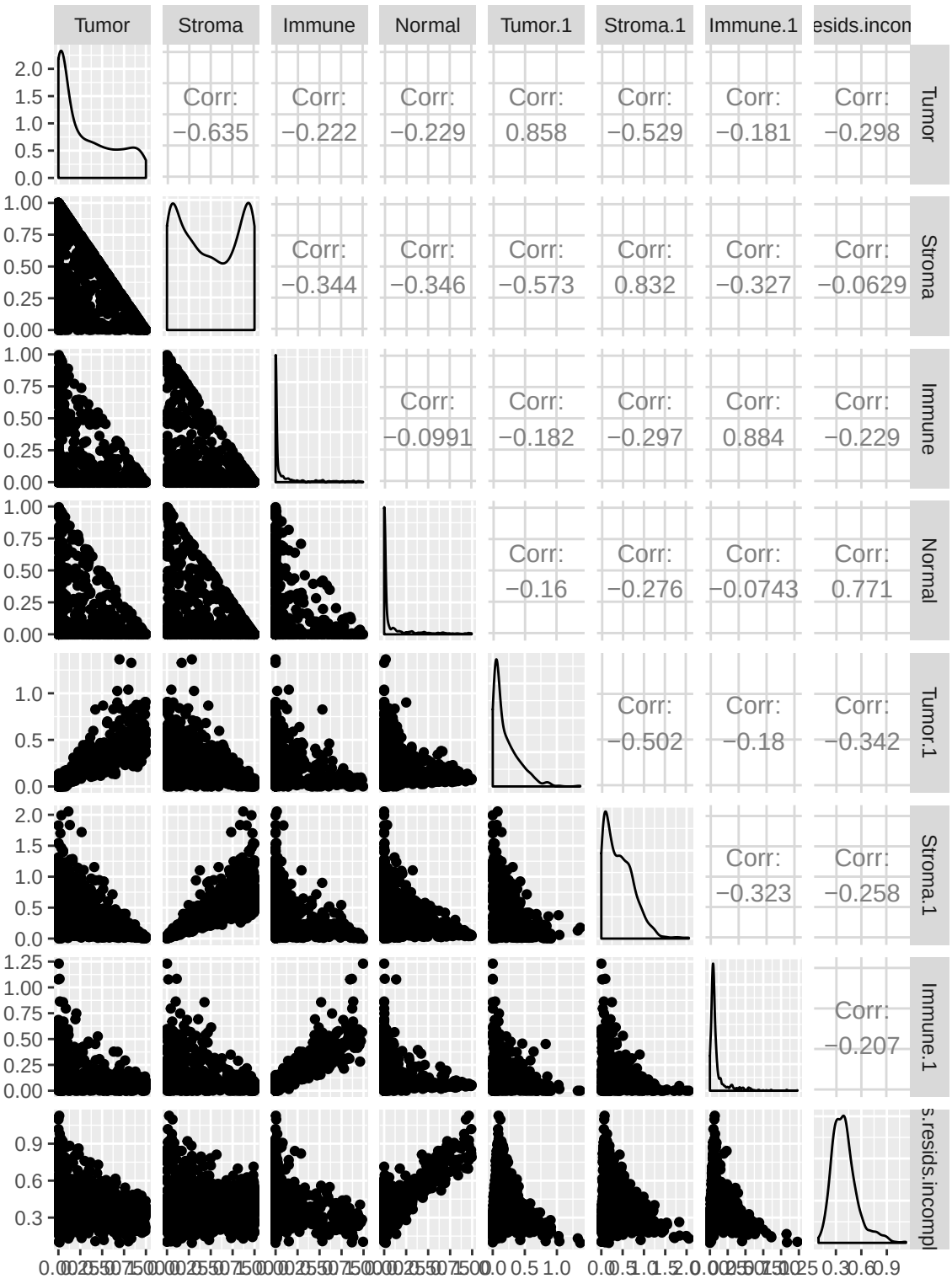
imperfect knowledge

```
known.set <- 1:3
nnls.H.incomplete <- H[known.set,]
nnls.resids.incomplete <- numeric(nsamples)
nmf.H.incomplete <- H[known.set,]
nmf.resids.incomplete <- numeric(nsamples)

for(i in 1:nsamples){
  this_sample <- data[,i]

  nnls.result <- my_nnls(W[,known.set],this_sample)
  nmf.result <- fuzzy_nmf(W[,known.set],this_sample)

  nnls.H.incomplete[,i] <- nnls.result$H
  nnls.resids.incomplete[i] <- sum((nnls.result$residuals[nnls.result$residuals>0]))/ (ngenes * mean(mea

  nmf.H.incomplete[,i] <- nmf.result$H
  nmf.resids.incomplete[i] <- nmf.result$residual

}
```

Compare resutls

truth versus nnls incomplete

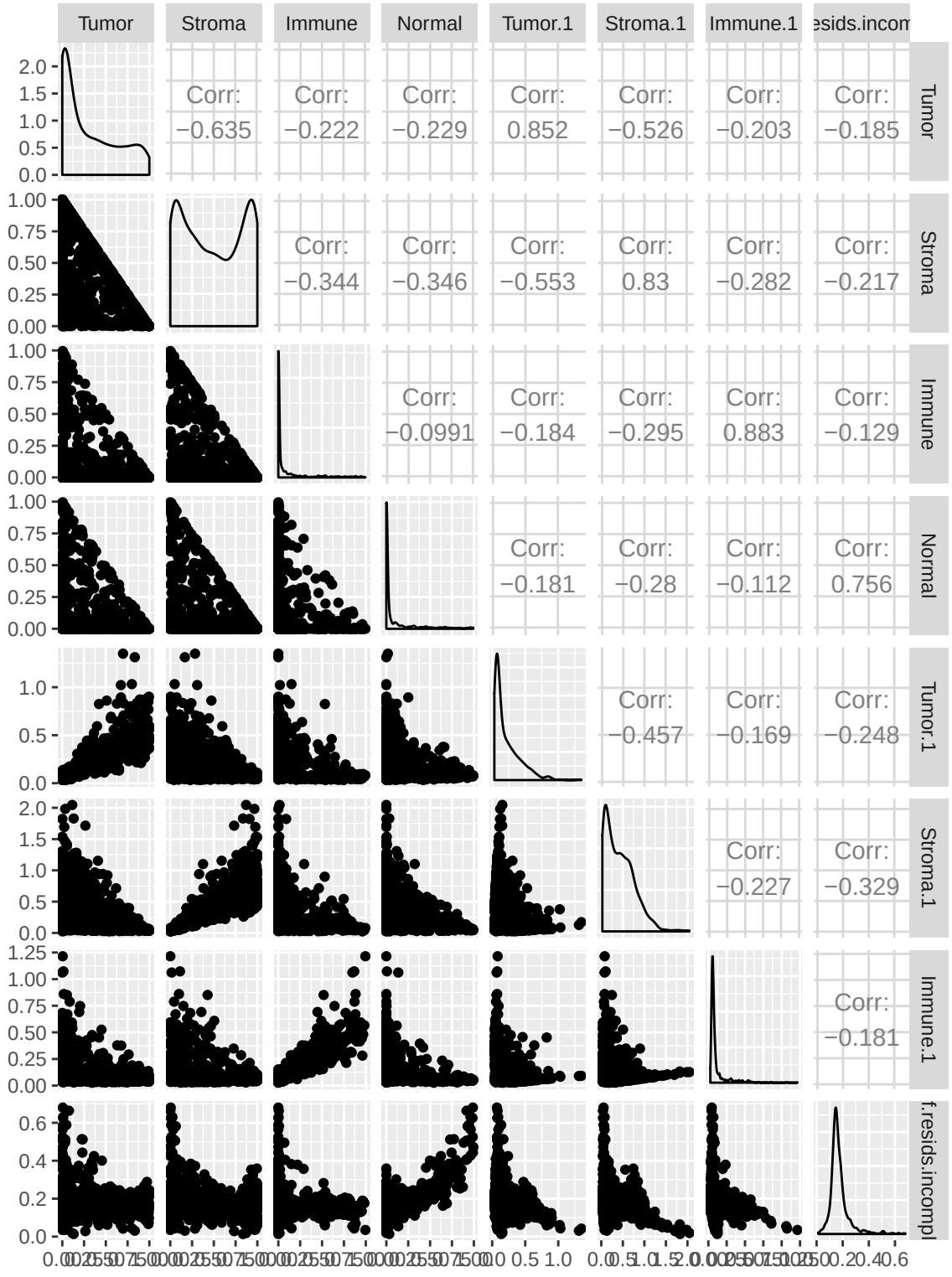
```
ggpairs(data = data.frame(t(rbind(H,
                                  nnls.H.incomplete,
                                  nnls.resids.incomplete))))
```



truth versus nmf imperfect

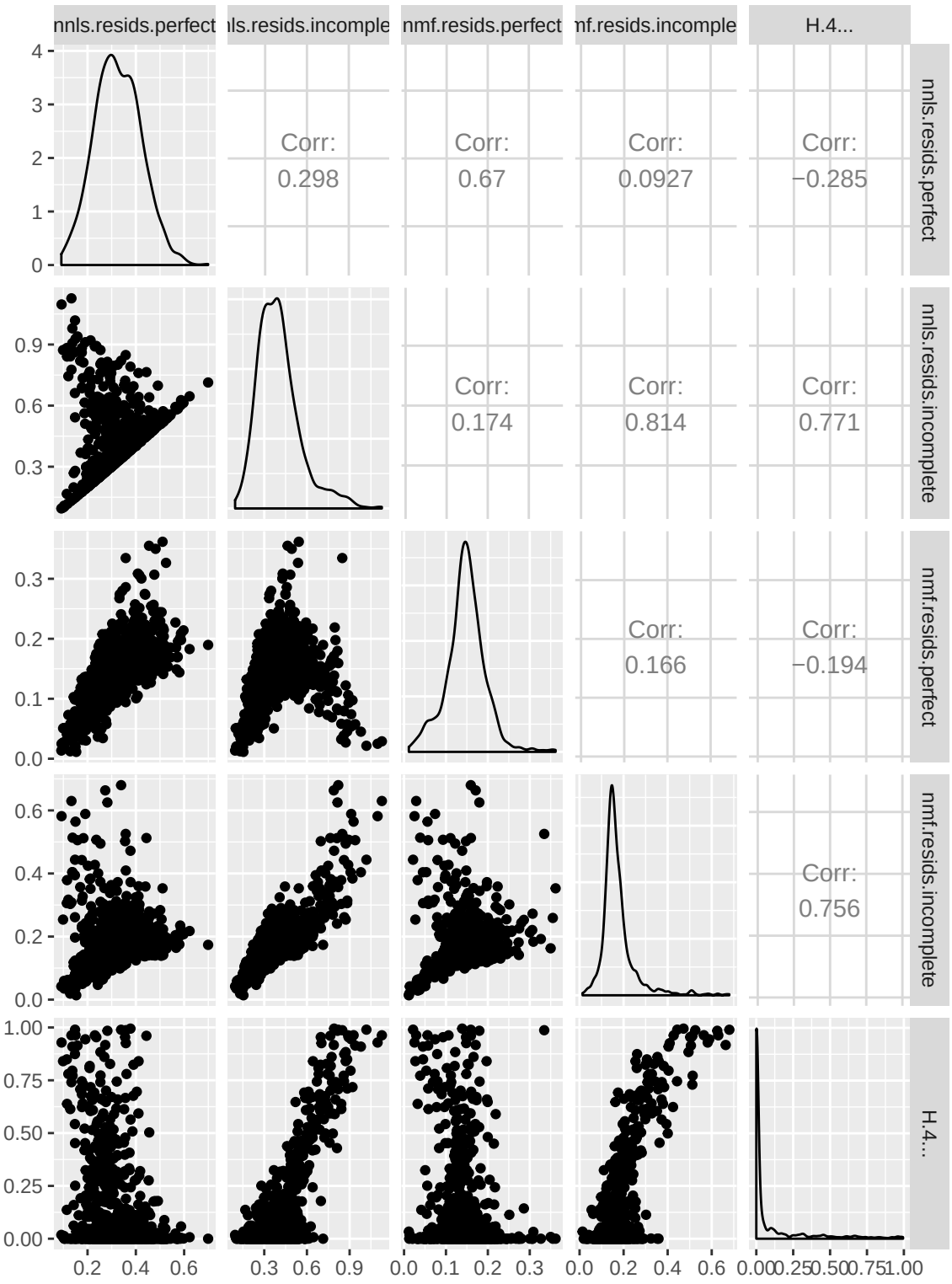
```
ggpairs(data = data.frame(t(rbind(H,
                                  nmf.H.incomplete,
```

```
nmf.resids.incomplete))))
```



comparison of residuals

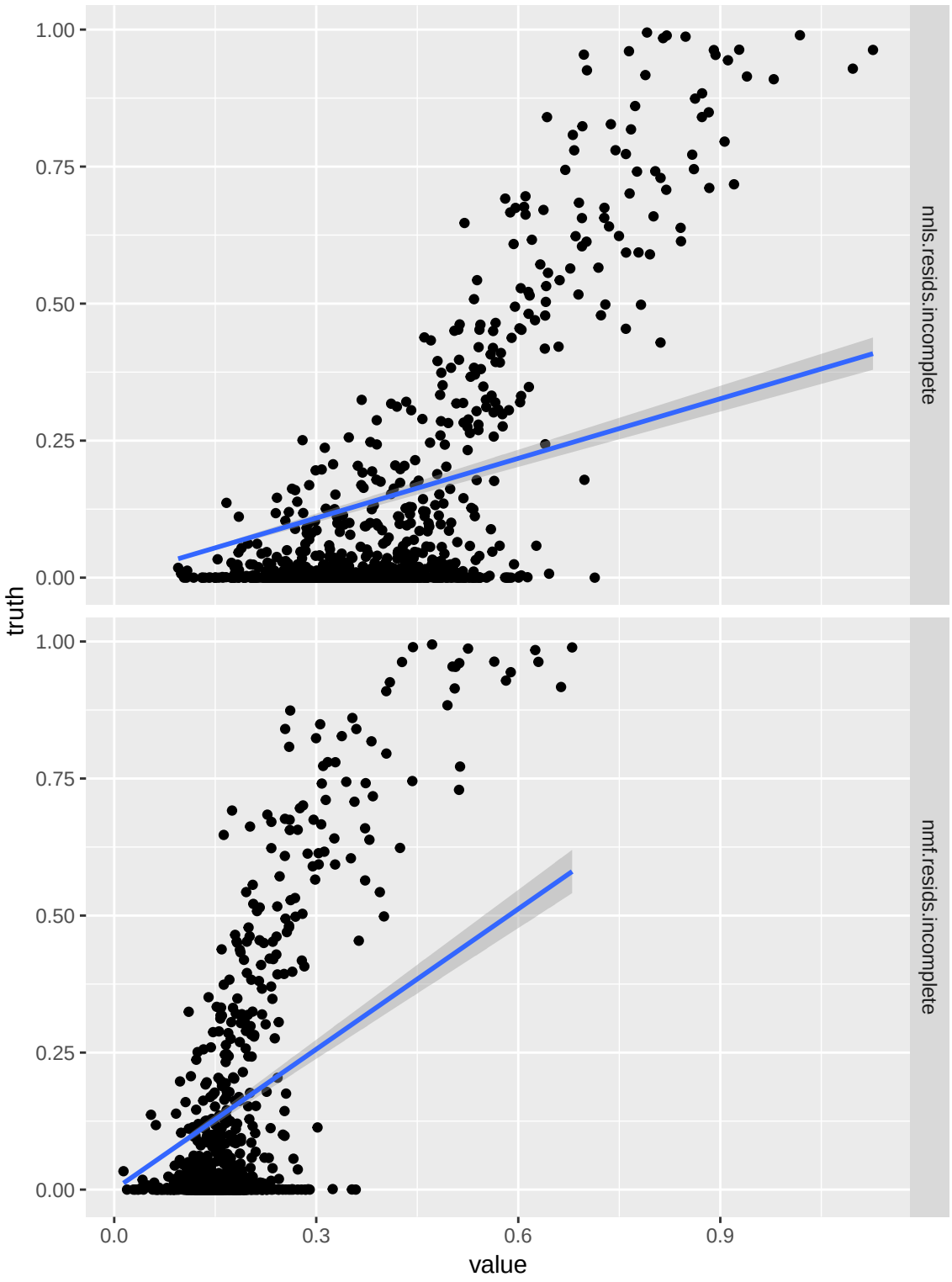
```
ggpairs(data = data.frame(nnls.resids.perfect,
                          nnls.resids.incomplete,
                          nmf.resids.perfect,
                          nmf.resids.incomplete,
                          H[4,]))
```



Correlation to unknown

```
my_df <- data.frame(nnls.resids.incomplete,
                    nmf.resids.incomplete,
```

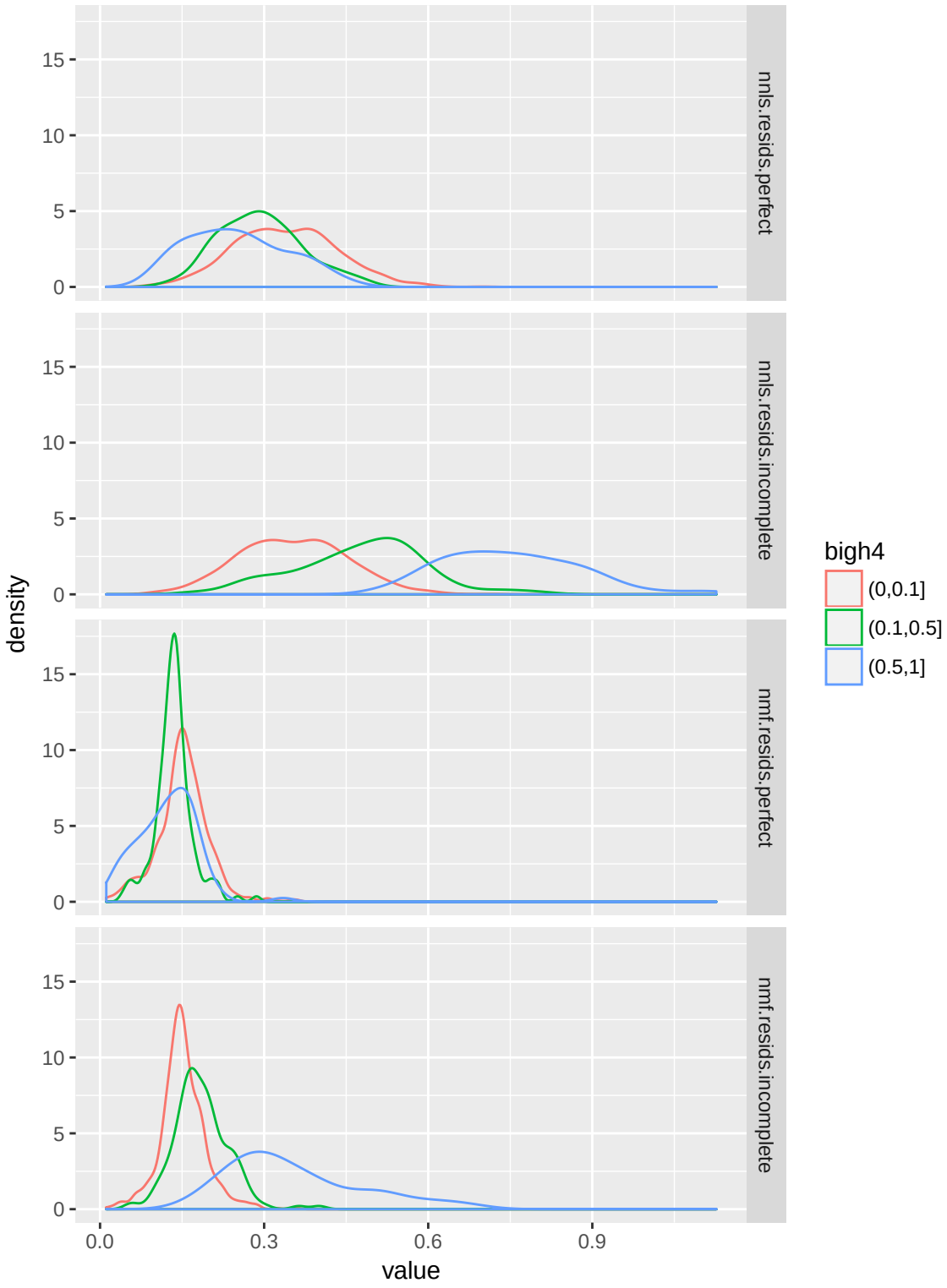
```
                    truth = H[4,])
my_df <- melt(my_df,id.vars = "truth")
p <- ggplot(data = my_df, aes(y = truth,x = value))
p <- p + facet_grid(facets = variable ~ .)
p <- p + geom_point()
p <- p + geom_smooth(method = "glm",formula = y ~ x -1)
print(p)
```



predictive power

```
my_df <- data.frame(nnls.resids.perfect,
                    nnls.resids.incomplete,
```

```
                    nmf.resids.perfect,
                    nmf.resids.incomplete,
                    bigh4 = cut(H[4,],breaks = c(0, 0.1, 0.5 ,1)))
my_df <- melt(my_df,id.vars = "bigh4")
p <- ggplot(data = my_df, aes(x = value,color = bigh4))
p <- p + facet_grid(facets = variable ~ .)
p <- p + geom_density()
print(p)
```


density
value
0.0
0.3
0.6
0.9
0
5
10
15
nnls.resids.perfect
nnls.resids.incomplete
nmf.resids.perfect
nmf.resids.incomplete
bigh4
(0,0.1]
(0.1,0.5]
(0.5,1]