Experimental Model

Eric Bridgeford

August 26, 2016

Basis for Model

The basis for the experimental model is Expectation-Maximization of the Gaussian Mixture Model (GMM). Here, we will detail several experiments that show that our model works through data-driven simulations, demonstrate the adjustments to GMM that we make, show that they work for simulation data, and finally investigate how the experimental data fits the model.

Basic Gaussian Mixture Model

The basic Gaussian Mixture Model hinges on Expectation-Maximization, a process for finding a locally maximizing fit for mixed distributions of the exponential family. GMM has 2 steps, the expectation (or responsibility) step, and the maximization step, and repeats until our log likelihood converges.

Experimental Model (time-variant GMM, tGMM)

For our experiment, we have task fMRI data in which test subjects have a task presented before them at known points in time. We theorize that the latent signals driving thinking patterns can be represented by a gaussian mixture, and that we can adjust EM to predict the covariance matrices of each experimental task condition. We define our density function of an observed x of some latent signal z:

$$p(x_i) = \sum_c \pi_{ic} N(x_i|\mu_c, \Sigma_c) \tag{1}$$

Using our predicted covariance matrices for each experimental task condition, we can then use the Wilcoxon signed-rank test to find rois that show differential activity based on task condition.

tGMM Framework

E Step

Like traditional GMM, our expectation step consists of computing the responsibilities of each potential cluster on our individual data points.

$$r_{ic} = \frac{\pi_{ic} N(x|\mu_c, \Sigma_c)}{\sum_{c'} \pi_{ic'} N(x|\mu_{c'}, \Sigma_{c'})} \tag{2}$$

where r_ic represents the given responsibility of a cluster c on time point i. This is a relatively intuitive process, as we are essentially considering the probability that a given data point is a member of a cluster and normalizing by the probability that it is in any of the clusters.

M Step

Again, much of our process here will look very similar to traditional GMM. As we have information about how the task conditions evolve over time, however, we can skip prediction of the π vector entirely. To define some ρ vector, we can convolve a spike train f of condition onsets with a response function that gives the expected proportion of signal at each successive timestep after an impulse, such that:

$$\rho_c = f_c(i) * h(i) \tag{3}$$

For each condition c, and where ρ_c is defined for each timestep i. We normalize our ρ vector to sum to 1 at each time point, calling the normalized vector π:

$$\pi_{ic} = \frac{\rho_{ic}}{\sum_c' \rho_{ic'}} \tag{4}$$

and now our model is simply a time-variant adaptation of GMM.

The cumulative responsibility of a given cluster c

$$m_c = \sum_i r_{ic} \tag{5}$$

For updated means. The logic behind the mean maximization is that we consider how much a particular cluster "explains" our ith data point, so μ is more strongly impacted by data points that it explains more effectively.

$$\hat{\mu_c} = \frac{1}{m_c} \sum_i r_{ic} x_i \tag{6}$$

For updating the covariances. The logi here is again the same; if a point is explained well by a particular distribution, its covariance is more strongly considered than if it isn't explained by that distribution.

$$\hat{\Sigma_c} = \frac{1}{m_c} \sum_i r_{ic} (x_i - \hat{\mu_c})^T (x_i - \hat{\mu_c}) \tag{7}$$

We repeat the expectation and maximization steps until we reach convergence of the likelihood (and consequently the log likelihood) of our predicted clusters matching our data.

$$log(p(X)) = \sum_i log[\sum_c \pi_{ic} N(x_i | \mu_c, \Sigma_c)] \tag{8}$$

Wilcoxon Signed-Rank Test

Here, the data is clearly paired, as we would expect a relationship between correlation values taken at the within-subject level, and the within-scan level. We do not want to assume a Gaussian distribution for the correlation values being analyzed, so we proceed with the non-parametric wilcoxon signed-rank test.

Experiments

For our experiments, we want to show that when we linearly combine draws from a distribution for each condition, that we can predict the distributions using the known contributions from each condition at each time step. Our simple simulation can be summarized as follows:

$$x_i = \sum_c \pi_{ic} y_c \tag{9}$$

for some $y_c \in Y_c$ such that $Y_c \overset{iid}{\sim} N(\mu_c, \Sigma_c)$.

```
one_iteration <- function(nt, nc, nroi, window, verbose=TRUE) {
  conditions <- array(0, dim=c(nt, nc)) # initialize empty condition vector
  for (cond in 1:nc) {
    conditions[seq((cond-1)*2 + 1, nt, (nc-1)*2 + length(window)), cond] <- 1
  }
  source('C:/Users/ebrid/Documents/GitHub/memory/code/R/drivers/convolve_trial_onsets.R')
  response <- convolve_conditions(conditions, window)

  require('ggplot2')
  require('reshape2')
  require('Rmisc')
  func <- ggplot(data = melt(window), aes(x = Var1, y = value)) + geom_line() +
    xlab('Time Point') + ylab('Relative Response') + ggtitle('Non-Rectangular Response')
  sp <- ggplot(data = melt(conditions), aes(x = Var1, y=value, group=factor(Var2), color=factor(Var2)))
    geom_line() +
    xlab('Time Point') + ylab('Spike') + ggtitle('Spike Train of Condition Onset') +
    theme(legend.position='none')
  conv <- ggplot(data = melt(response), aes(x = Var1, y=value, color=factor(Var2), group=factor(Var2)))
    geom_line() +
    xlab('Time Point') + ylab('Relative Response') + ggtitle('Signal Present for Condition') +
    scale_colour_discrete(name='Condition') +
    theme(legend.position='bottom')
  if(verbose) {print(multiplot(plotlist=list(sp, func, conv), layout=matrix(c(1,1,2,3,3,3), nrow=2, byr

  pi_vec <- t(apply(response, 1, function(x) x/sum(x))) # normalize the response
  require('MASS')
  require('clusterGeneration')
  latent_signal <- array(0, dim=c(dim(conditions)[2], nt, nroi))
  for (i in 1:dim(conditions)[2]) {
    latent_signal[i,,] <- mvrnorm(n=nt, mu=rnorm(nroi), Sigma=genPositiveDefMat(nroi)$Sigma) # use rand
  }
  # define the observed signal as a combination of the latent signals, where the contribution
  # of each is the pi vector at that time step
  observed_signal <- t(sapply(1:dim(latent_signal)[2], function(t) {
    apply(pi_vec[t,]*latent_signal[,t,], 2, sum)
  }))
  pi_vec <- pi_vec[complete.cases(observed_signal),]
  observed_signal <- observed_signal[complete.cases(observed_signal),]
  true_means <- apply(latent_signal, c(1,3), mean)
  true_cov <- sapply(1:dim(latent_signal)[1], function(x) cov(latent_signal[x,,]), simplify='array')

  # predict a diagonal matrix for simple case
  predict_cov <- sapply(1:dim(latent_signal)[1], function(x) diag(nroi), simplify='array')
```

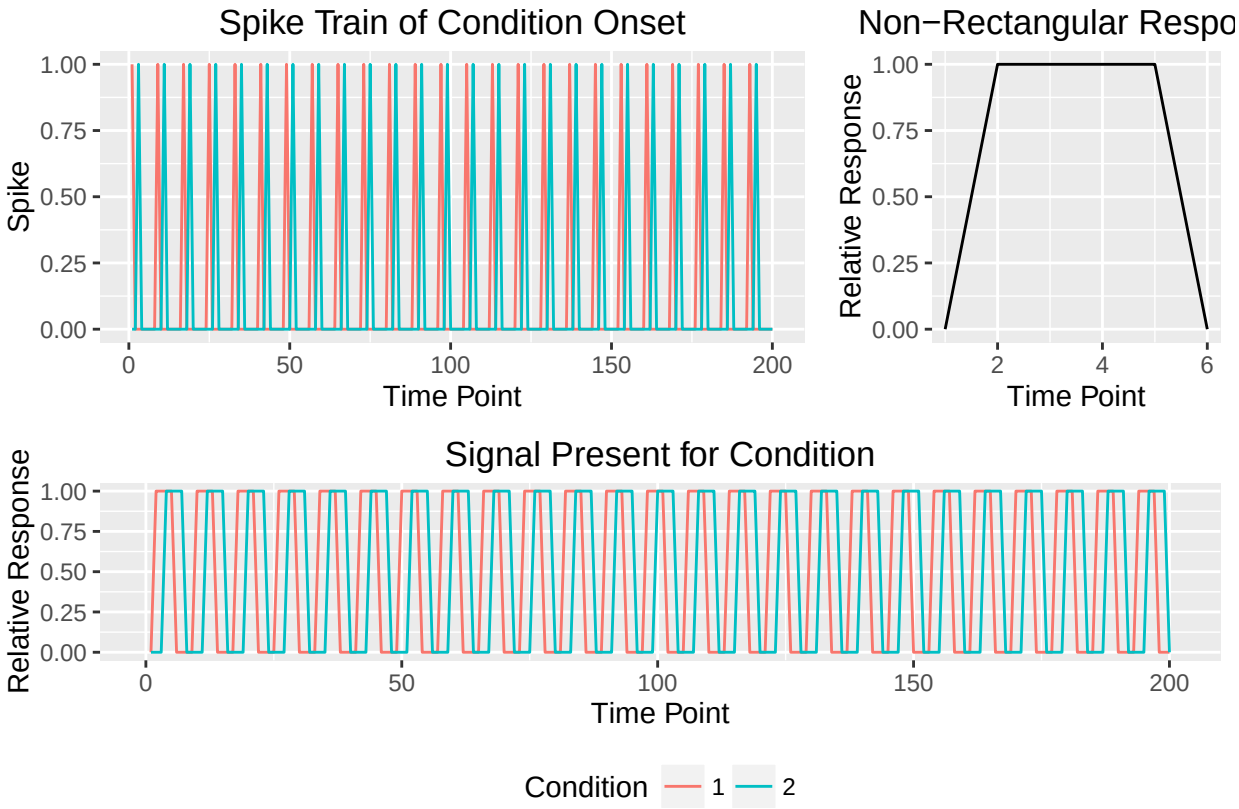
```r
source('C:/Users/ebrid/Documents/GitHub/memory/code/R/experiments/gmm_known_pi.R')
# predict zero mean
params <- gmm_known_pi(observed_signal, array(0, dim=c(nc, nroi)), predict_cov, pi_vec)
require(abind)
covs <- abind(true_cov, params$covs, along=3)

trials <- c()
labels <- c()
for (i in 1:nc) {
  trials <- c(trials, paste(i))
  labels <- c(labels, paste(i, "true", sep=" "))
}
for (i in 1:nc) {
  trials <- c(trials, paste(i))
  labels <- c(labels, paste(i, "predict", sep=" "))
}
source('C:/Users/ebrid/Documents/GitHub/Reliability/Code/FlashRupdated/functions/distance.R')
source('C:/Users/ebrid/Documents/GitHub/Reliability/Code/FlashRupdated/functions/reliability.R')
source('C:/Users/ebrid/Documents/GitHub/Reliability/Code/FlashRupdated/functions/computerank.R')

for (j in unique(trials)) {
  plotlist <- list()
  counter <- 0
  for (i in which(trials == j)) {
    counter <- counter + 1
    plotlist[[counter]] <- ggplot(melt(covs[,,i]), aes(x=Var1, y=Var2, fill=value)) +
      geom_tile() +
      scale_x_discrete("ROI", limits=c(1, ceiling(nroi/2), nroi)) +
      scale_y_discrete("ROI", limits=c(1, ceiling(nroi/2), nroi)) +
      scale_fill_gradientn(colours=c("darkblue","blue","purple","green","yellow"),name="Covariance", l

  }
  if(verbose) {print(multiplot(plotlist=plotlist, layout=matrix(c(1:counter), byrow=TRUE, nrow=1)))}
}

Dmtx <- distance(covs)
mnr <- mnr(rdf(Dmtx, trials))

D <- melt(Dmtx)
for (i in 1:length(trials)) {
  D[D$Var1 == i,]$Var1 <- labels[i]
  D[D$Var2 == i,]$Var2 <- labels[i]
}
error <- mean(sapply(unique(trials), function(x) {
  varas <- which(trials == x)
  return(Dmtx[varas[1], varas[2]])
}))
if(verbose) {print(ggplot(D, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  scale_fill_gradientn(colours=c("darkblue","blue","purple","green","yellow"),name="distance") +
  xlab("Matrix") + ylab("Matrix") + ggtitle(sprintf('MNR = %.4f, mean error = %.4f', mnr, error)))
}
return(list(covs=covs, mnr=mnr, error=error, D=Dmtx, trials=trials))
```

```
}
```

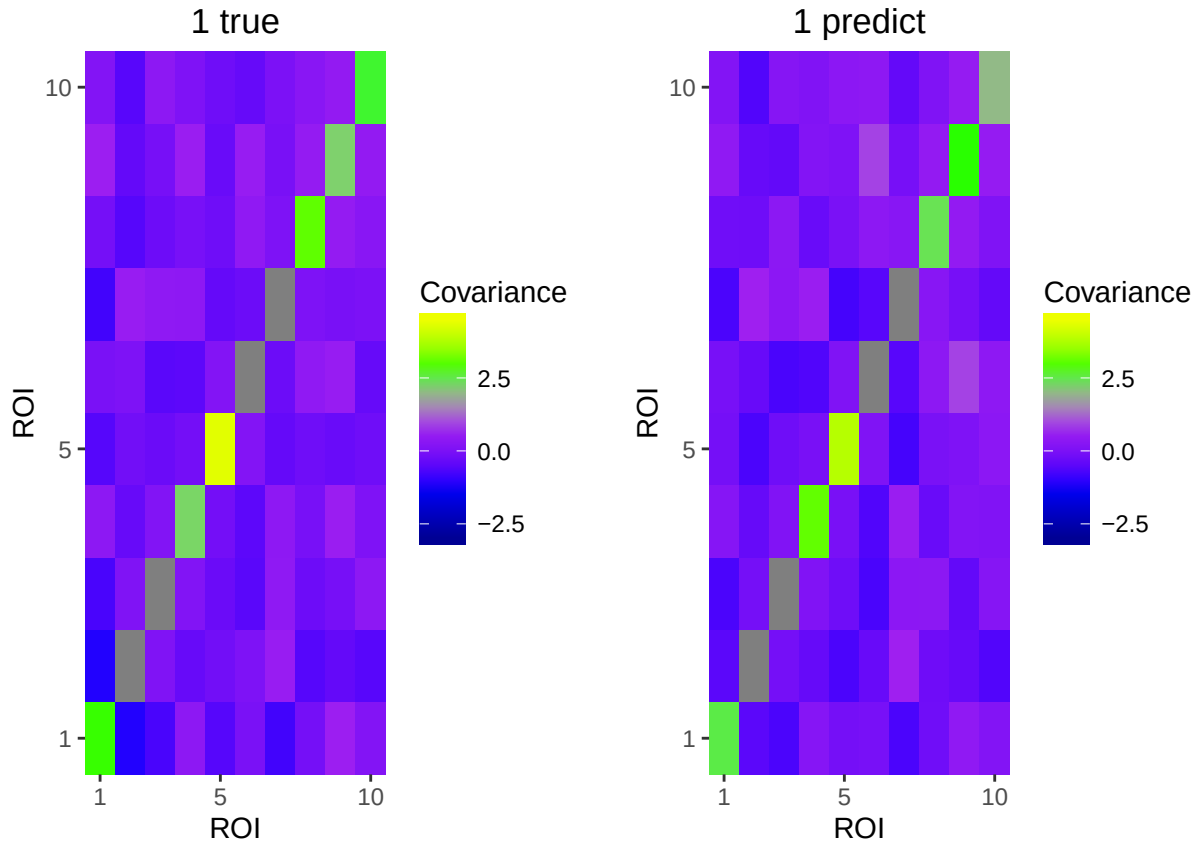
Simple Simulation 1

For our first experiment, we consider a simple $|n| = 15$ roi model, with $|c| = 2$ task conditions for this experiment, and $|i| = 200$ timesteps. We use a simple rectangular window for our expected signal lag vector.

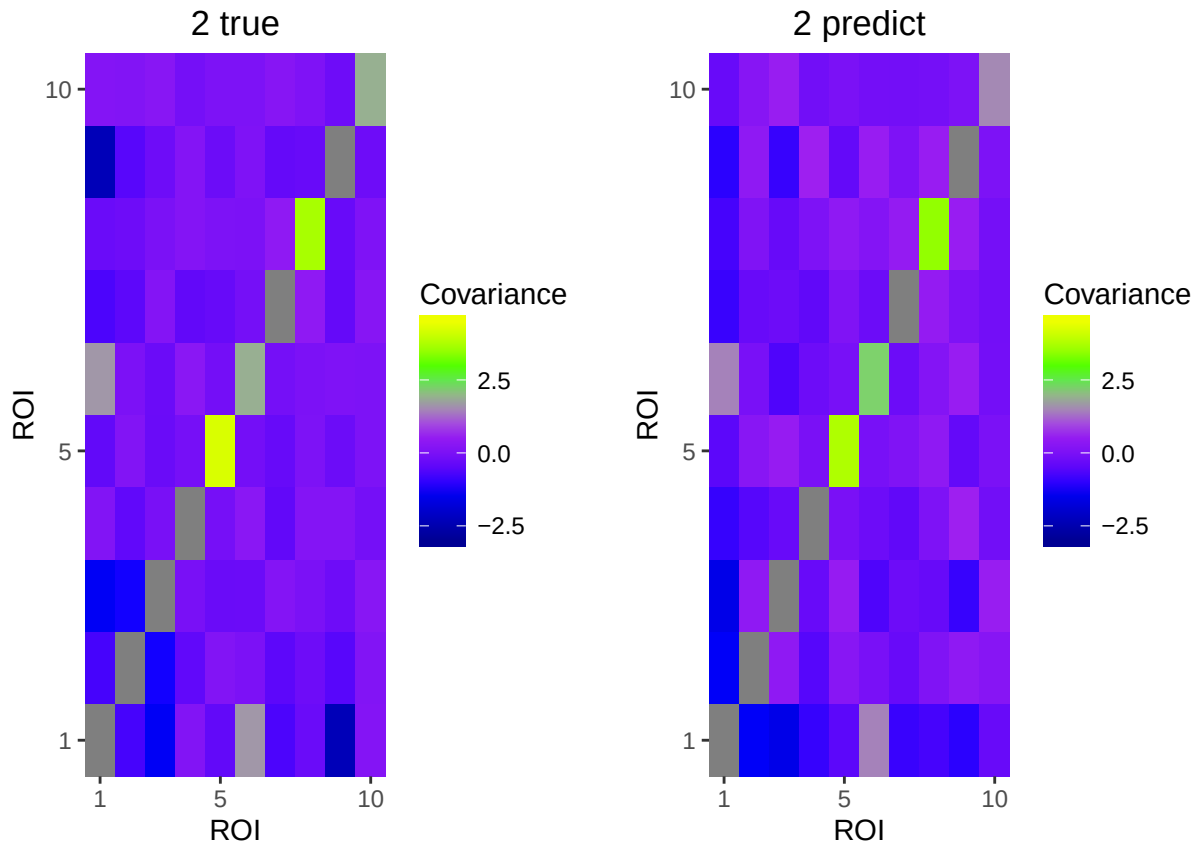
```
t <- 200
nc <- 2
nroi <- 10
tes <- one_iteration(t, nc, nroi, array(rev(c(0, 1, 1, 1, 1, 0)))) # define a simple square window)
```



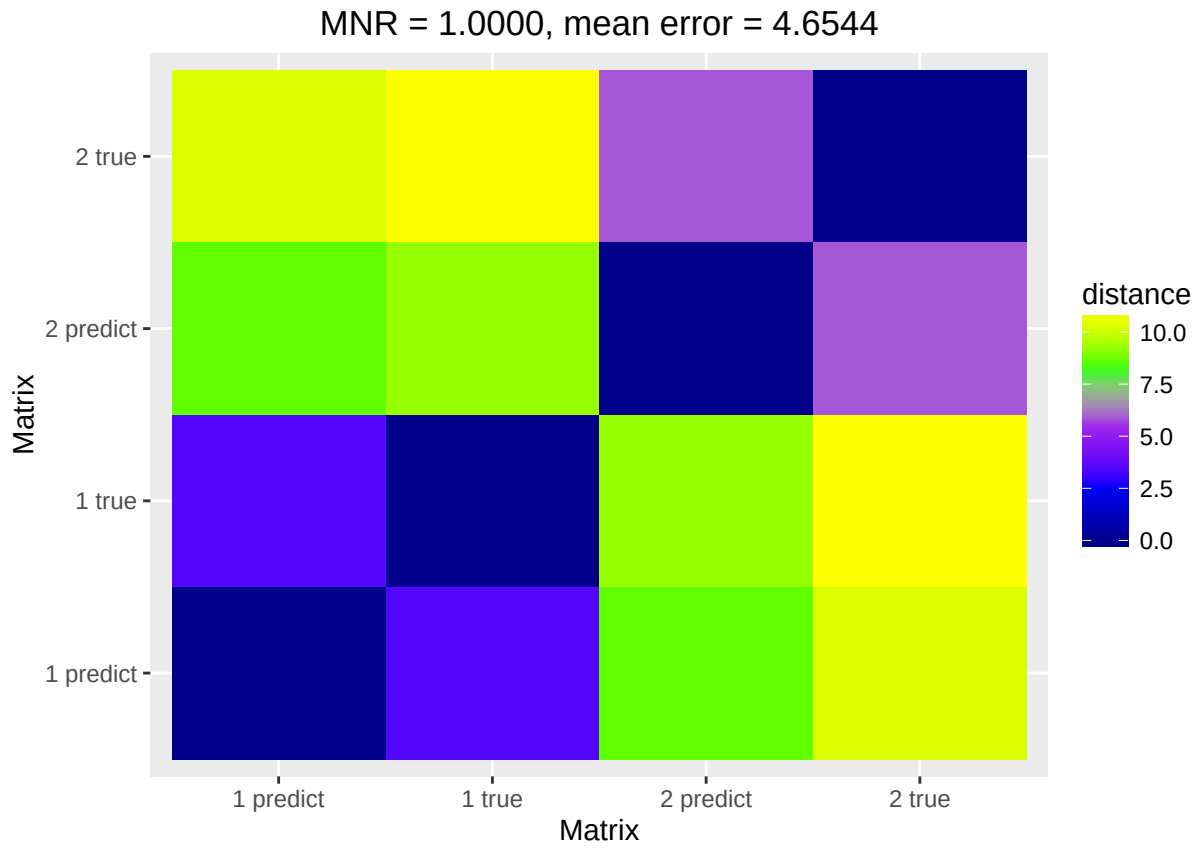

```
## NULL
```


1 true
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5
1 predict
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5

```
## NULL
```


2 true
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5
2 predict
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5

```
## NULL
```

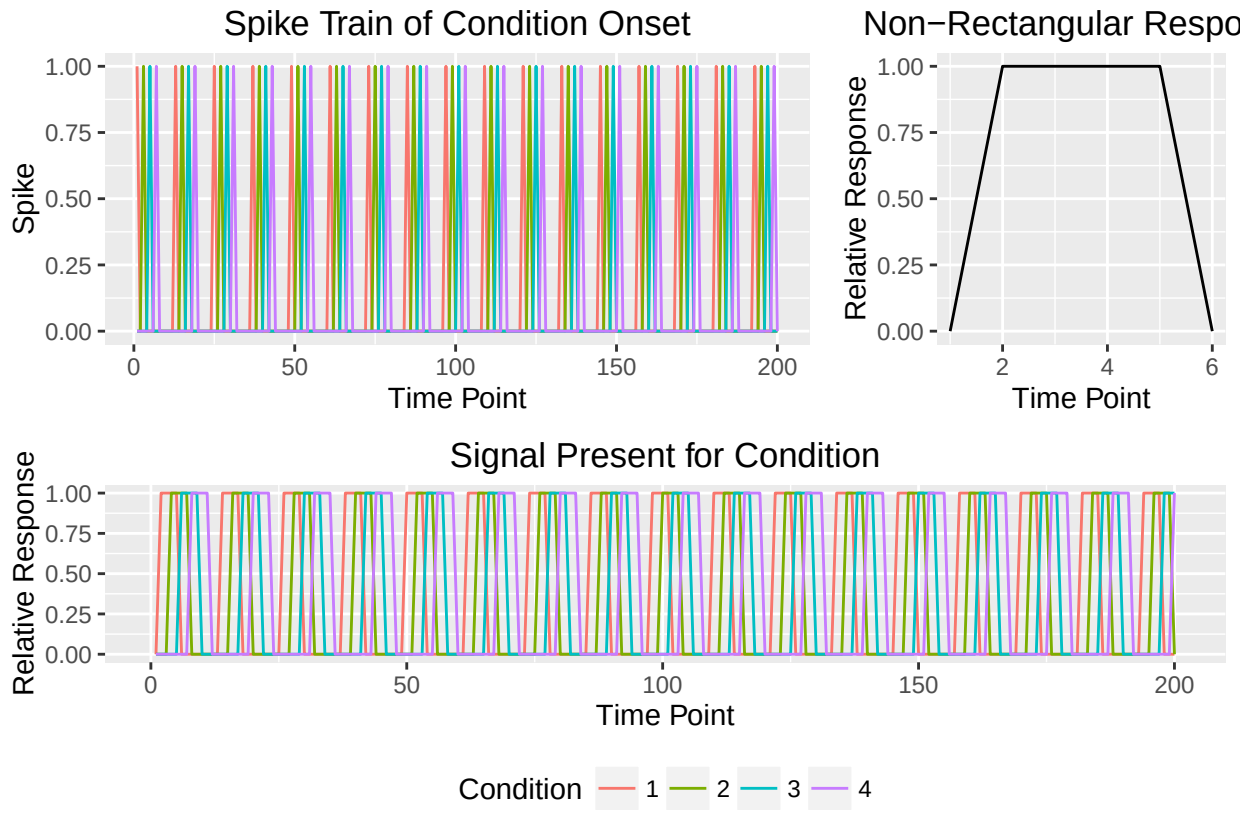



```
  print(paste("100 simulation mean MNR", mean(mnr_ar)))
```

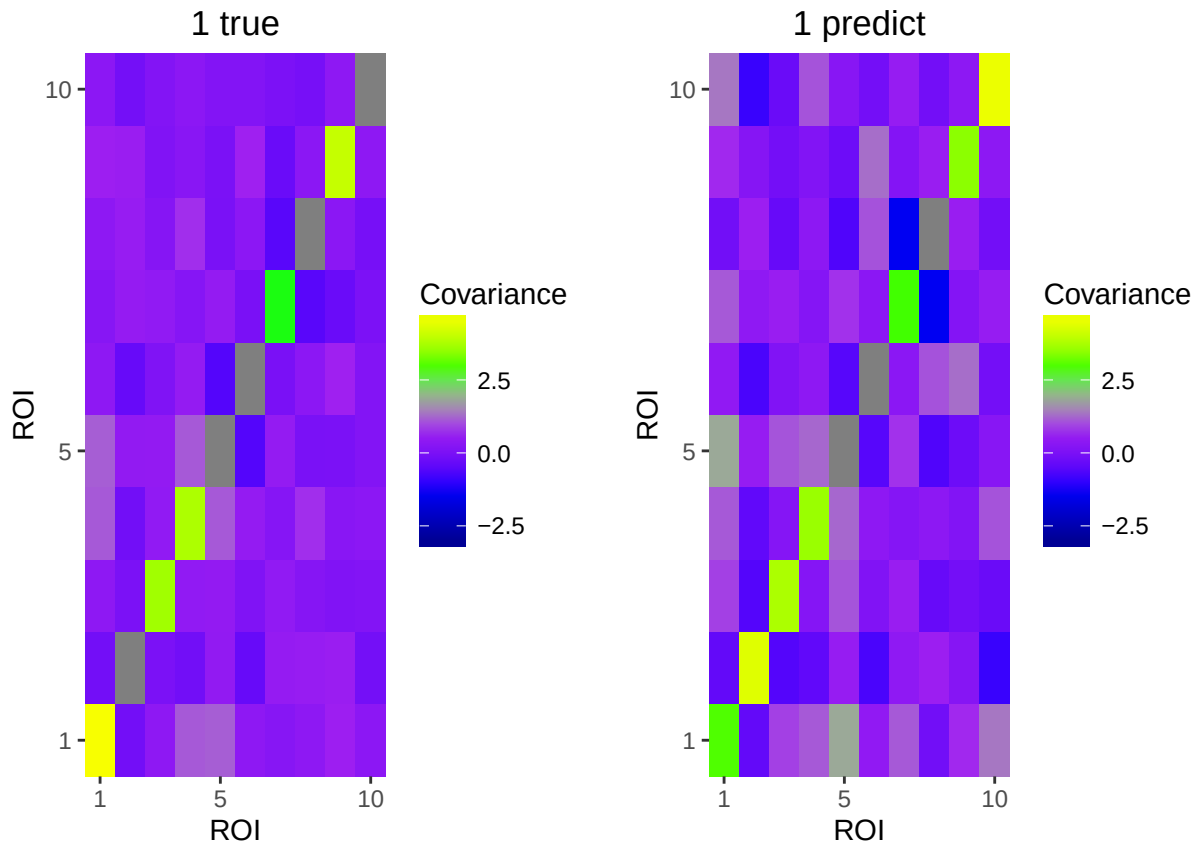
```
## [1] "100 simulation mean MNR 1"
```

4 Condition Simple Simulation

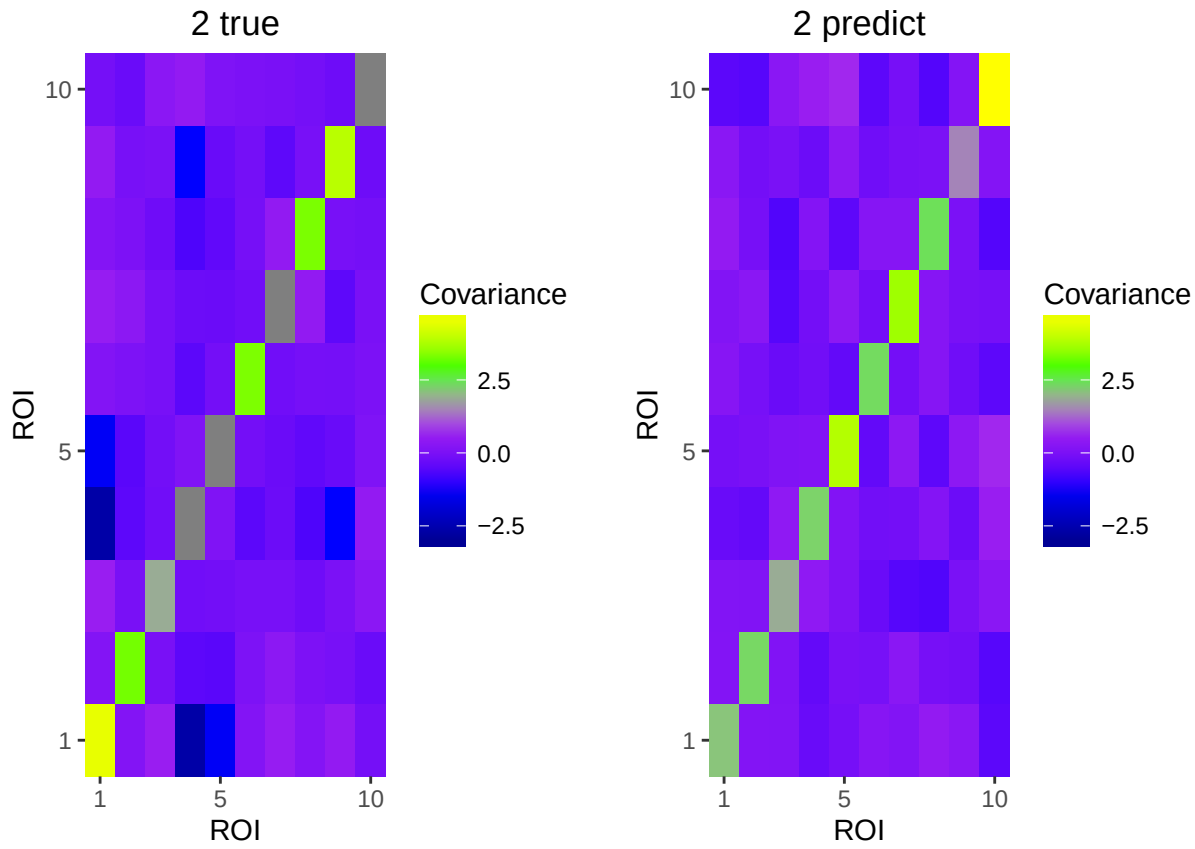
```
  t <- 200
  nc <- 4
  nroi <- 10
  tes <- one_iteration(t, nc, nroi, array(rev(c(0, 1, 1, 1, 1, 0)))) # define a simple square window)
```


Spike Train of Condition Onset
Spike
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Non−Rectangular Respo
Relative Response
1.00
0.75
0.50
0.25
0.00
2
4
6
Time Point
Signal Present for Condition
Relative Response
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Condition
1
2
3
4

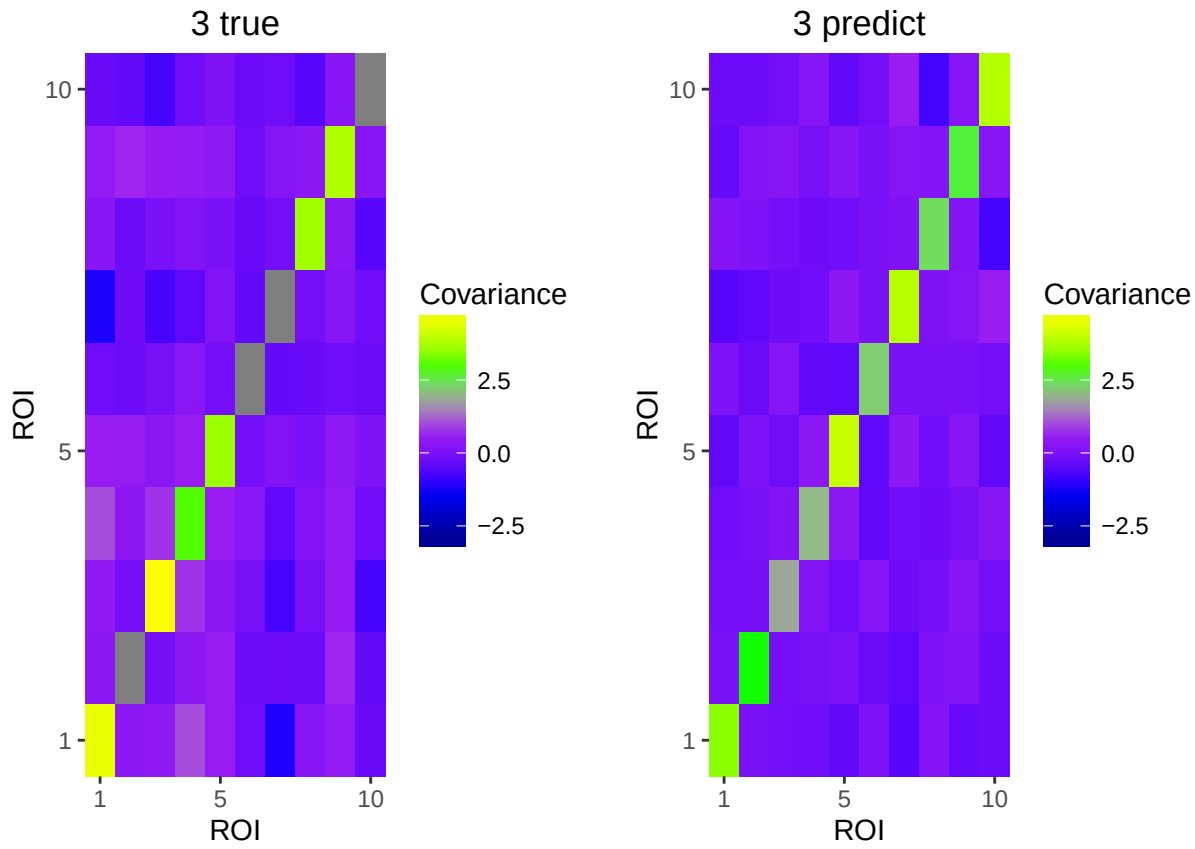
```
## NULL
```


1 true
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5
1 predict
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5

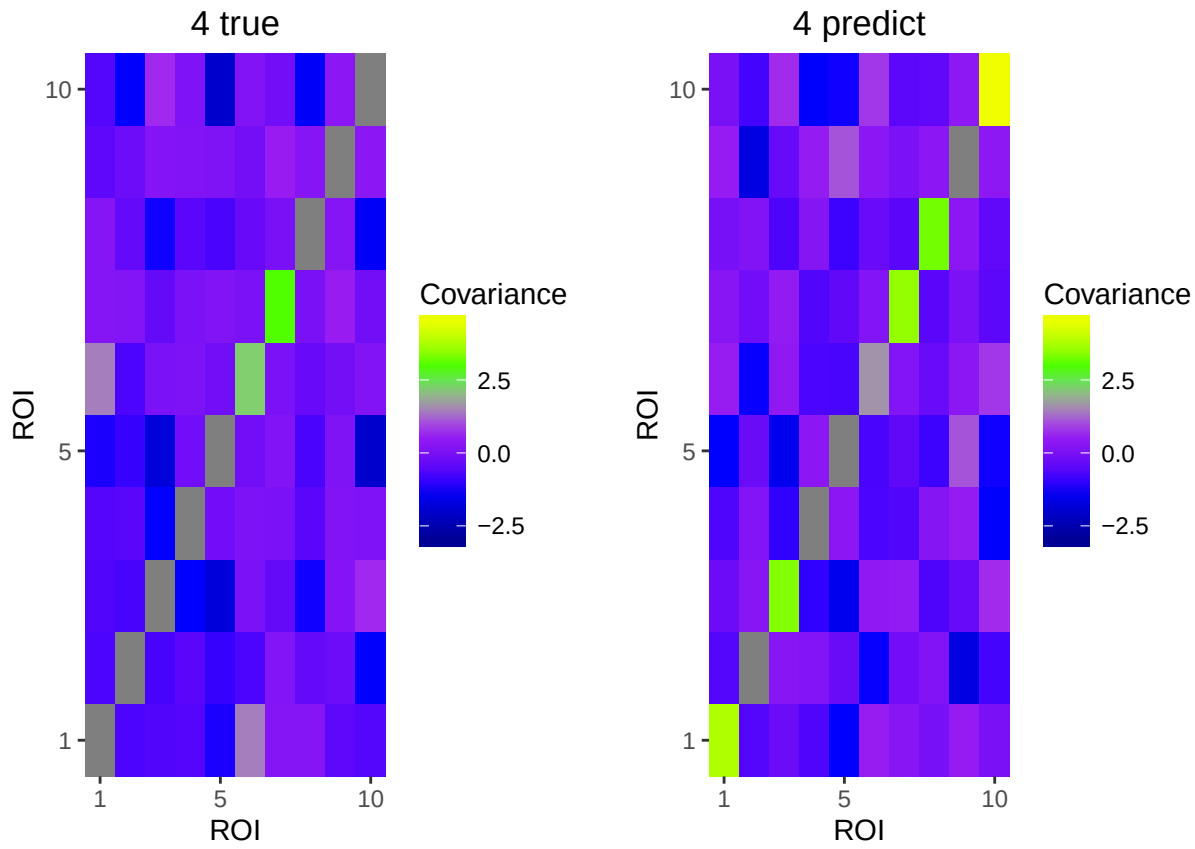
```
## NULL
```



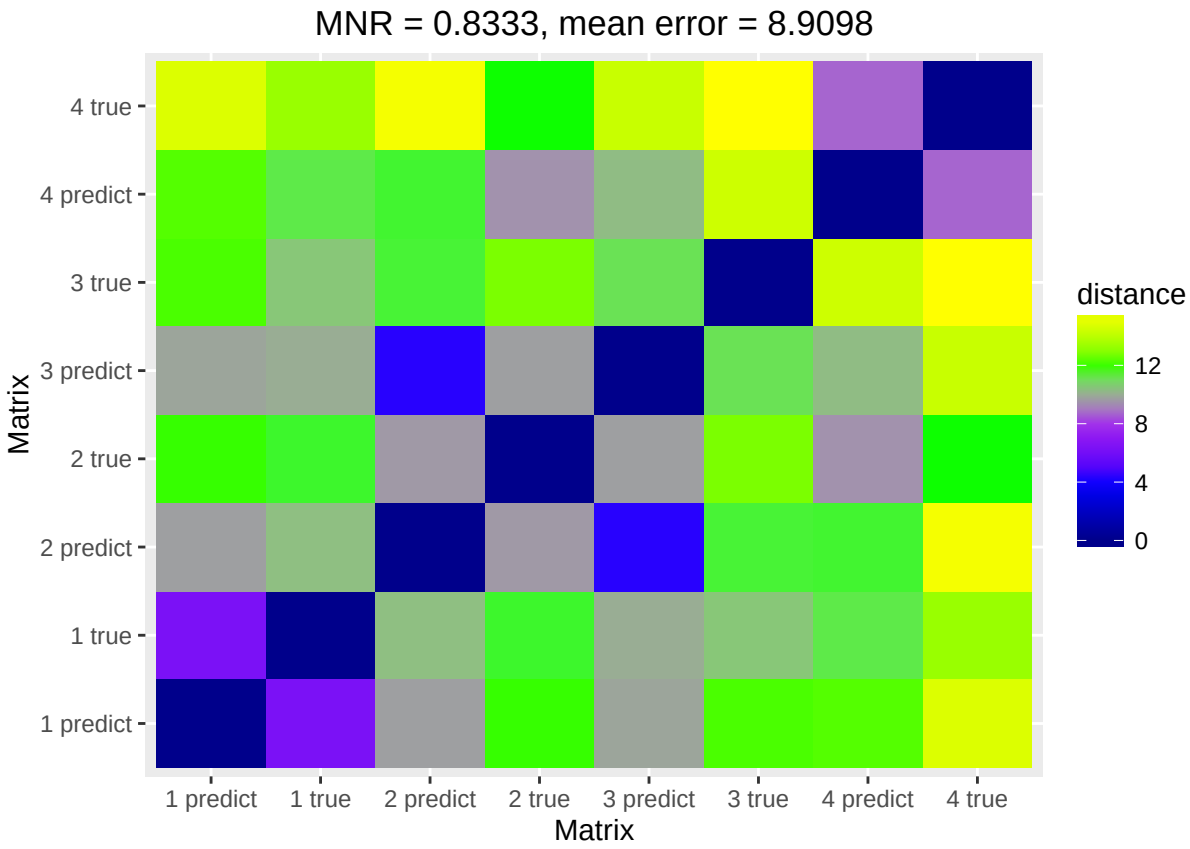

```
## NULL
```


3 true
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5
3 predict
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5

```
## NULL
```


4 true
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5
4 predict
10
5
1
ROI
1
5
10
ROI
Covariance
2.5
0.0
−2.5

```
## NULL
```

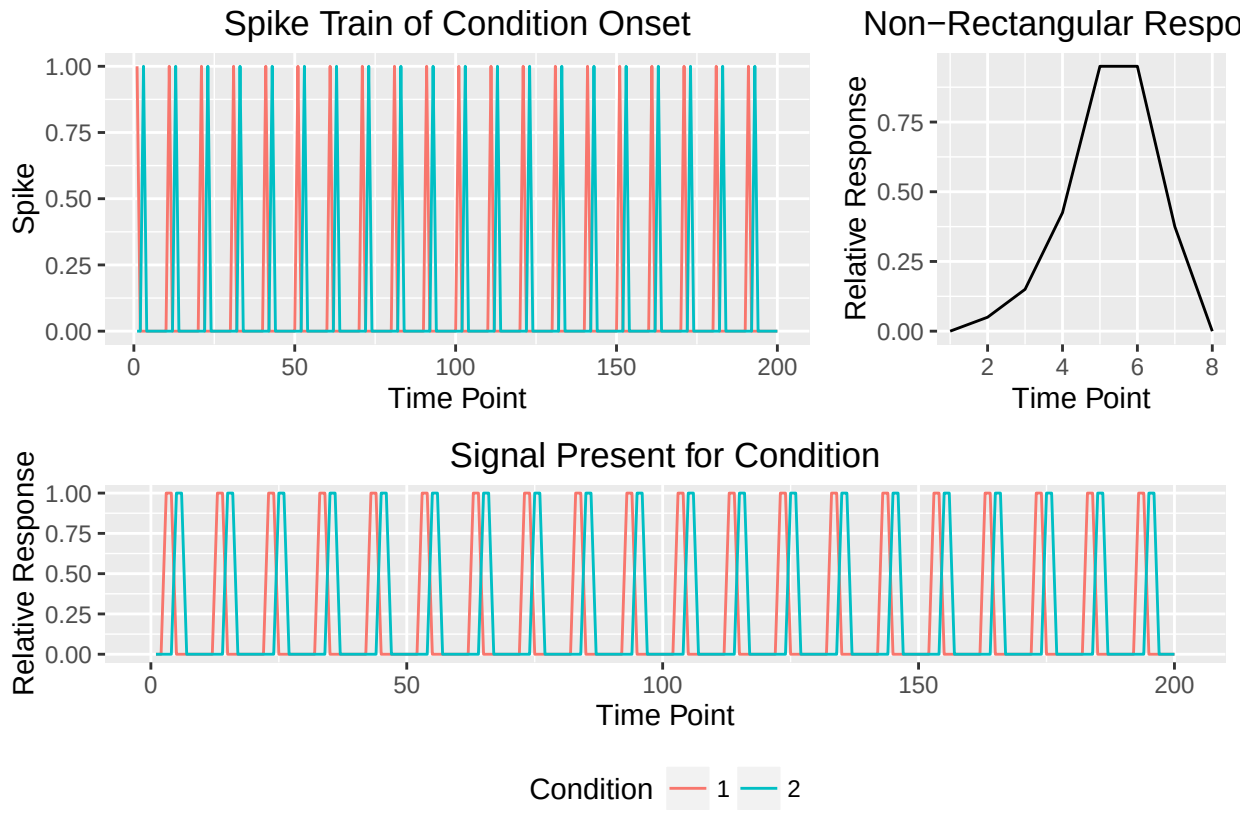



```
  print(paste("100 simulation mean MNR", mean(mnr_ar)))
```

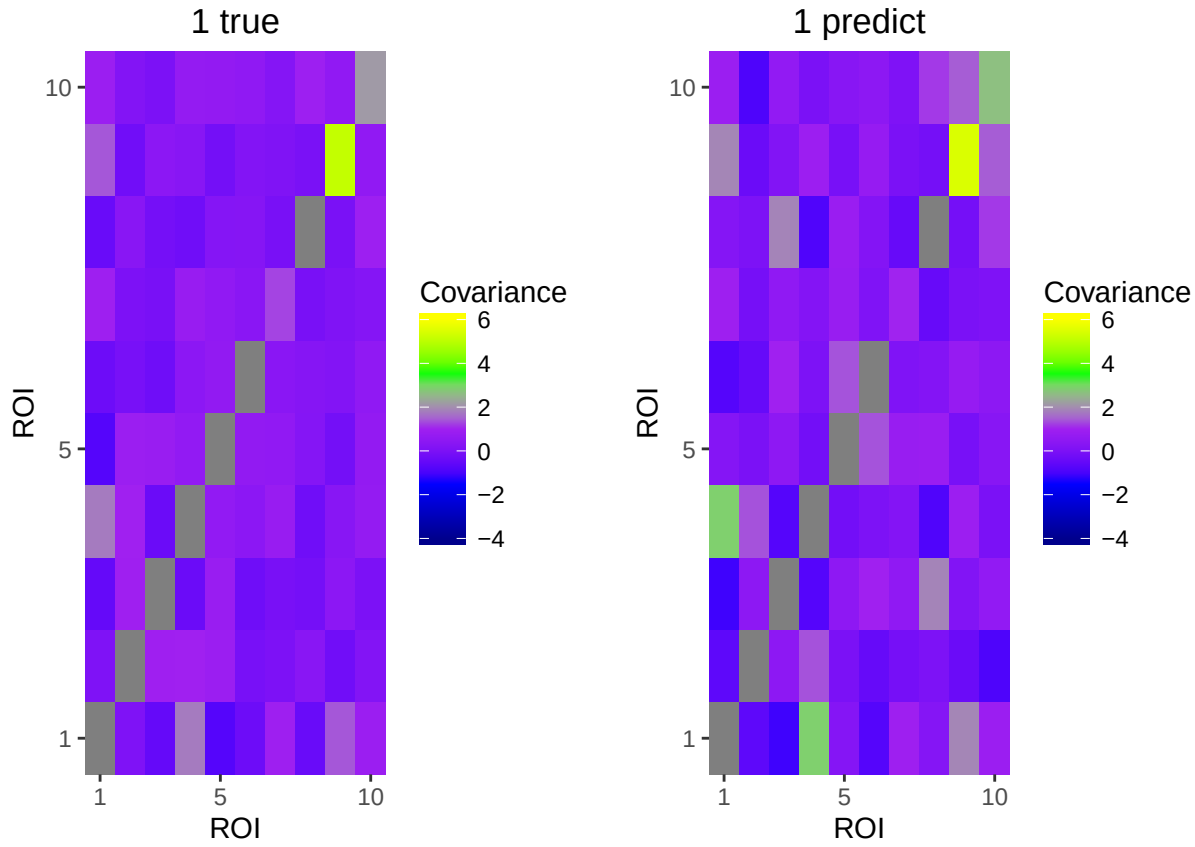
```
## [1] "100 simulation mean MNR 0.837202380952381"
```

2 Condition Simulation, Non-Rectangular Window

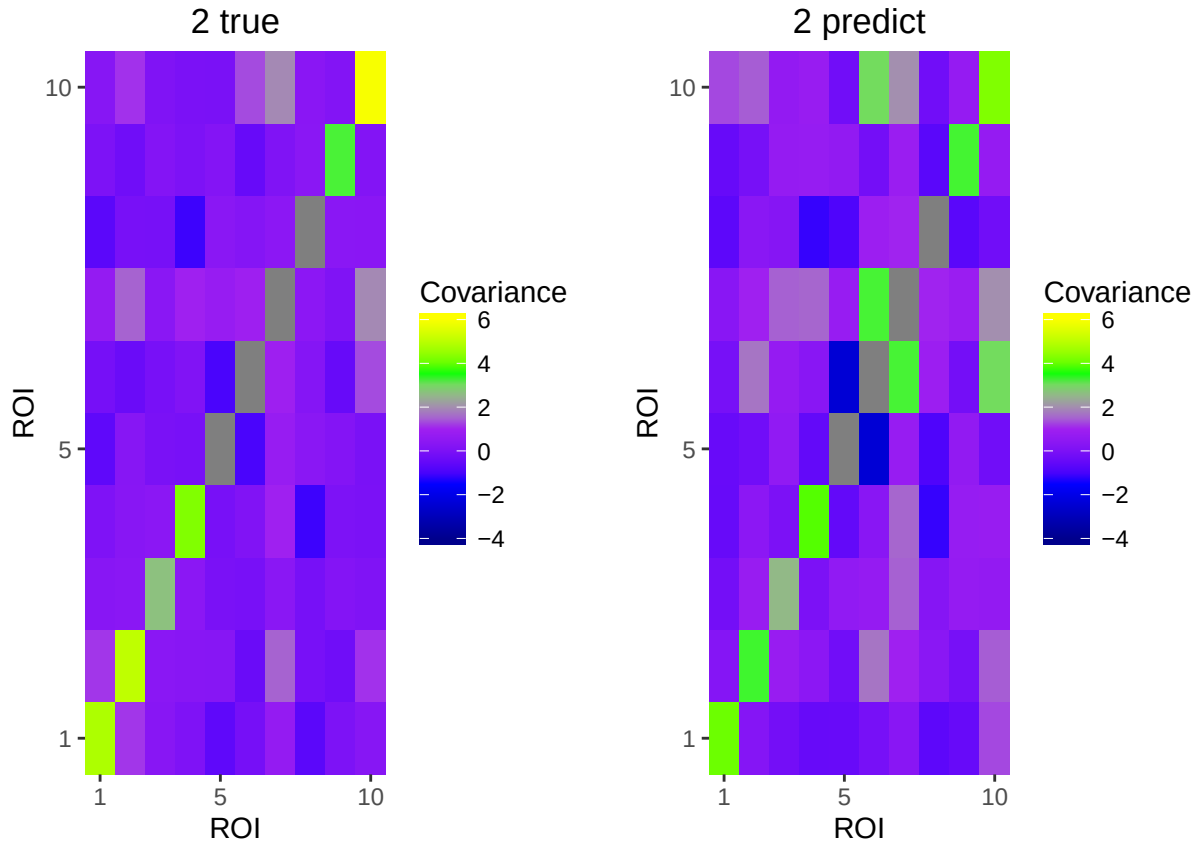
```
  t <- 200
  nc <- 2
  nroi <- 10
  tes <- one_iteration(t, nc, nroi, array(rev(c(0, .375, .95, .95, .425, .15, .05, 0)))) # define a sim
```


Spike Train of Condition Onset
Spike
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Non−Rectangular Respo
Relative Response
0.75
0.50
0.25
0.00
2
4
6
8
Time Point
Signal Present for Condition
Relative Response
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Condition
1
2

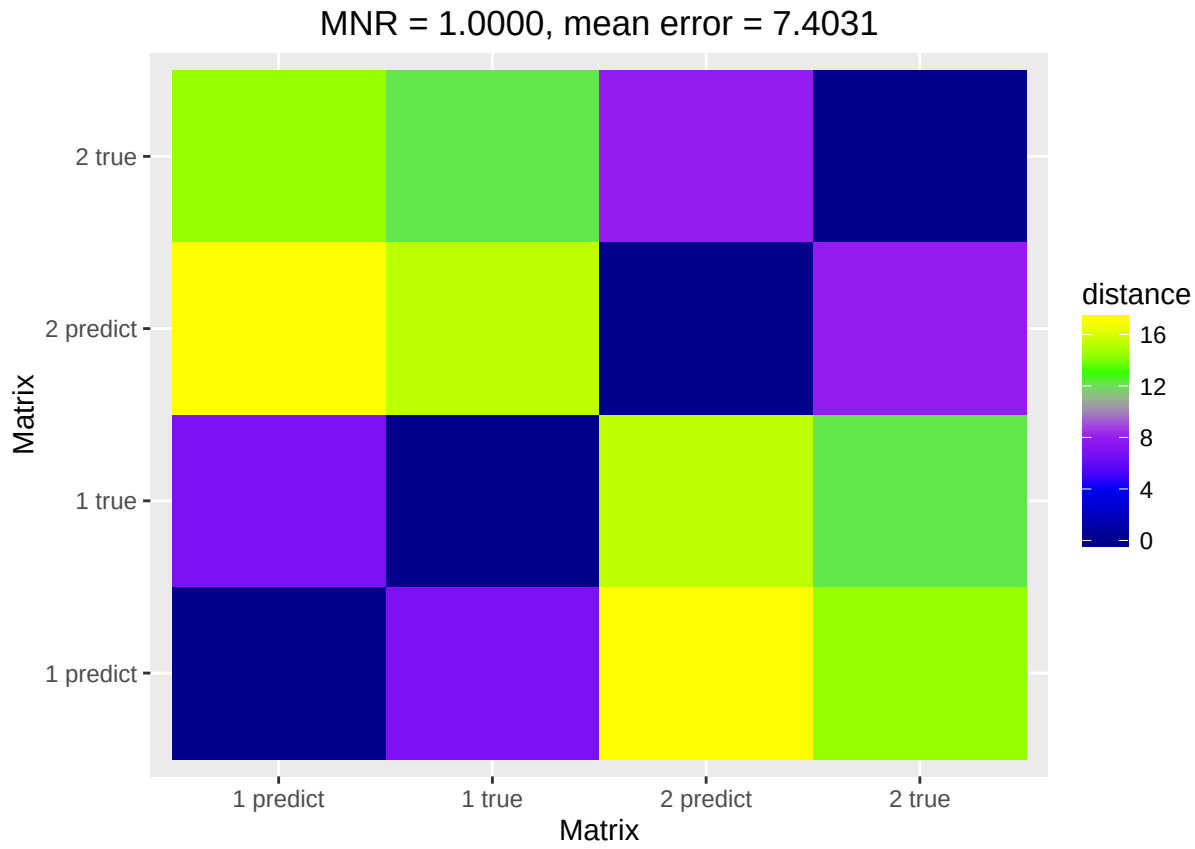
```
## NULL
```




```
## NULL
```




```
## NULL
```

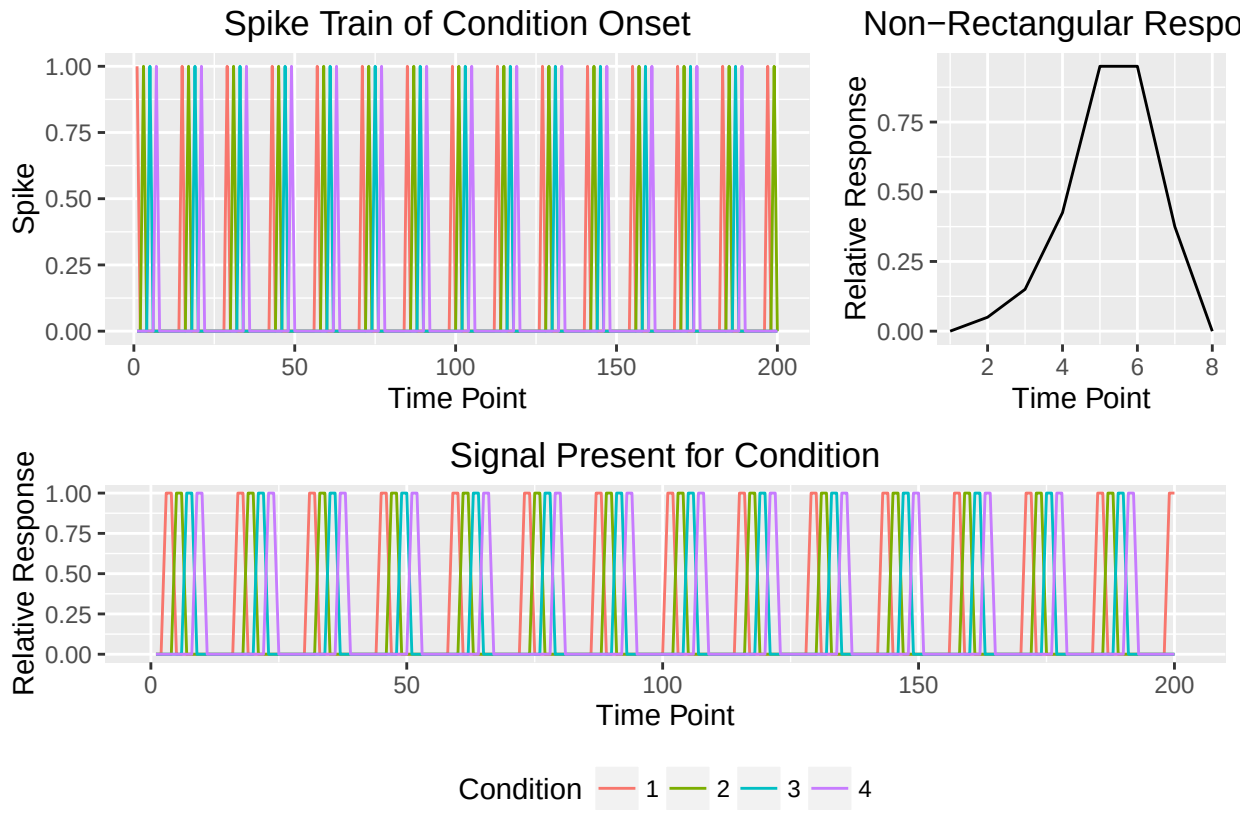



```
  print(paste("100 simulation mean MNR", mean(mnr_ar)))
```

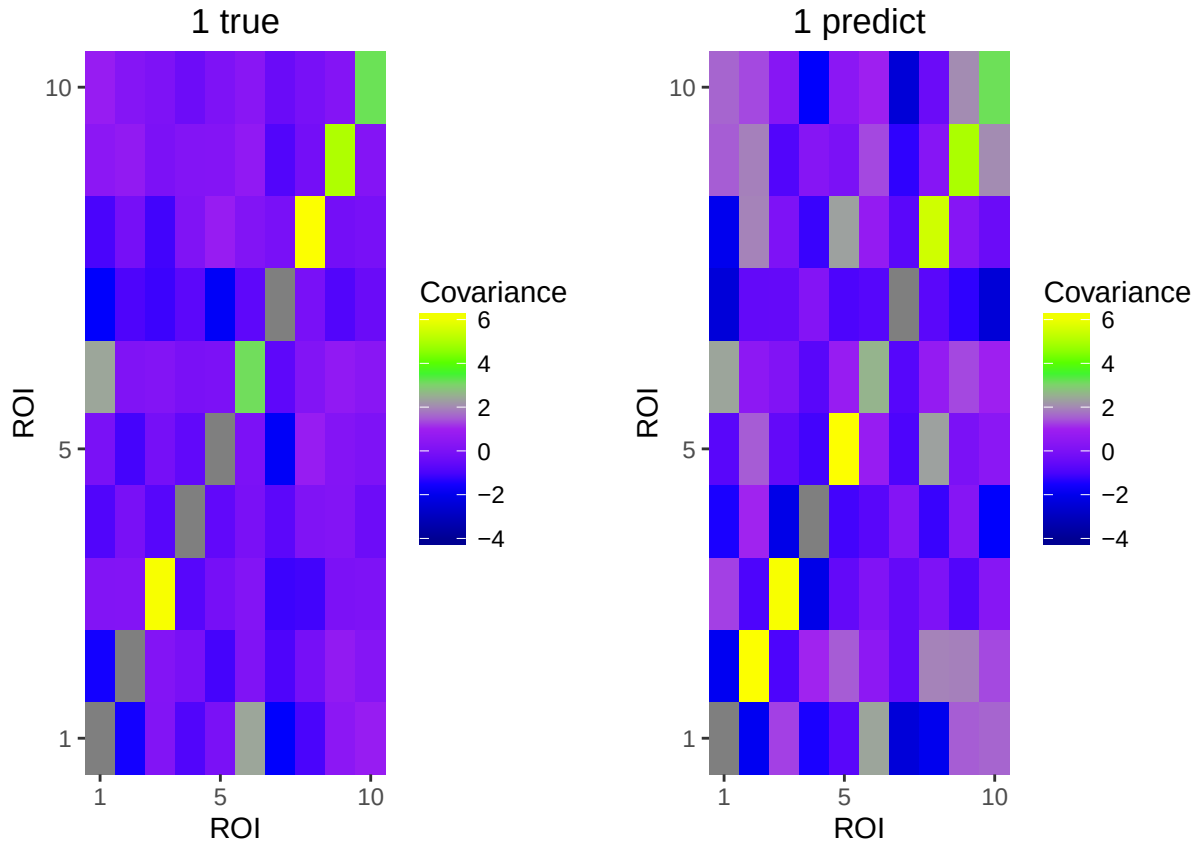
```
## [1] "100 simulation mean MNR 0.9975"
```

4 Condition Simple Simulation, Non-Rectangular Window

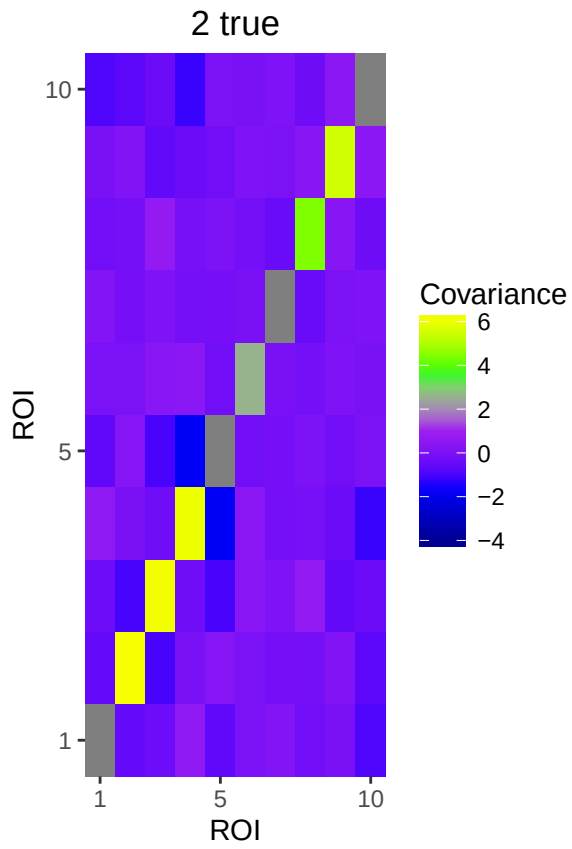
```
  t <- 200
  nc <- 4
  nroi <- 10
  tes <- one_iteration(t, nc, nroi, array(rev(c(0, .375, .95, .95, .425, .15, .05, 0)))) # define a sim
```

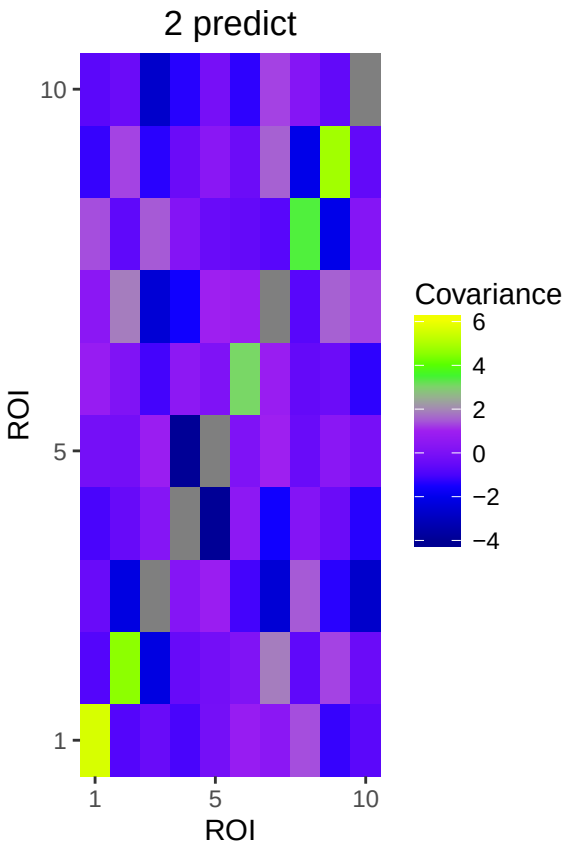

Spike Train of Condition Onset
Spike
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Non−Rectangular Respo
Relative Response
0.75
0.50
0.25
0.00
2
4
6
8
Time Point
Signal Present for Condition
Relative Response
1.00
0.75
0.50
0.25
0.00
0
50
100
150
200
Time Point
Condition
1
2
3
4

```
## NULL
```

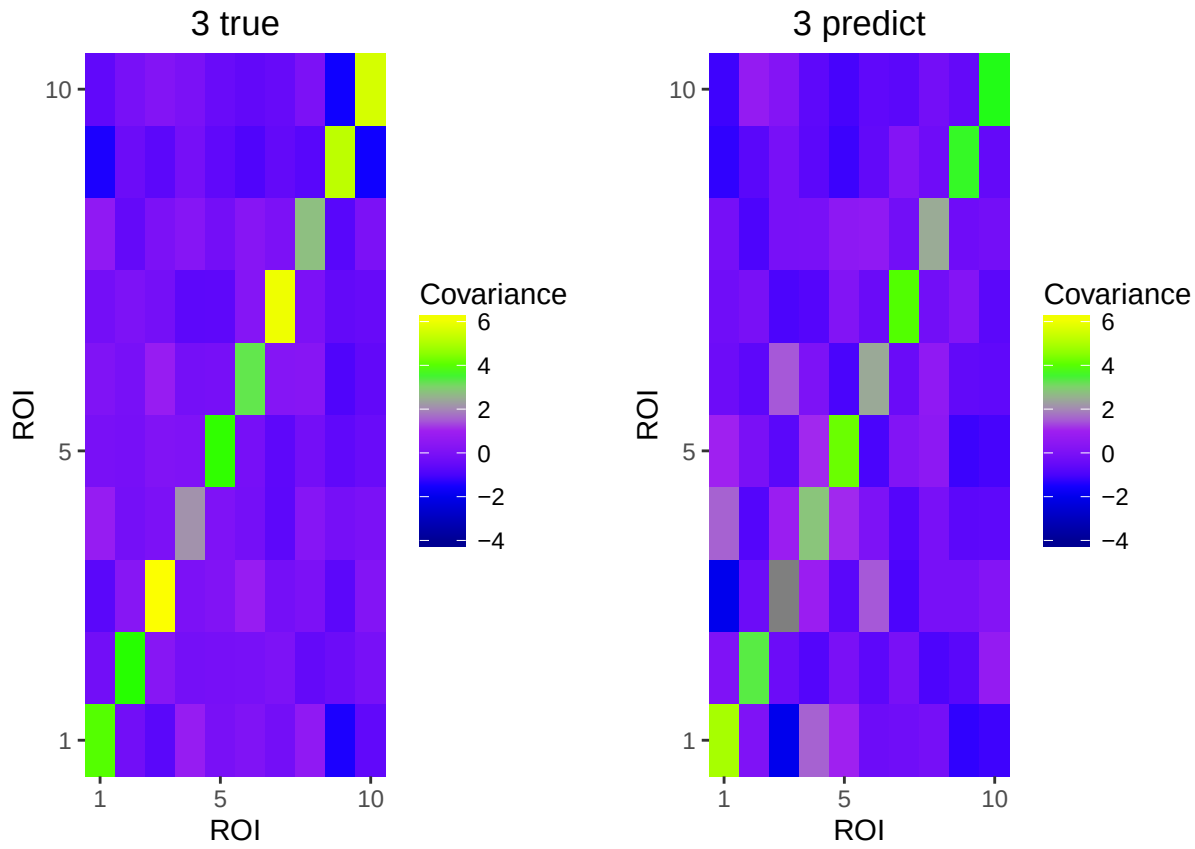

1 true
10
5
1
ROI
1
5
10
ROI
Covariance
6
4
2
0
−2
−4
1 predict
10
5
1
ROI
1
5
10
ROI
Covariance
6
4
2
0
−2
−4

```
## NULL
```

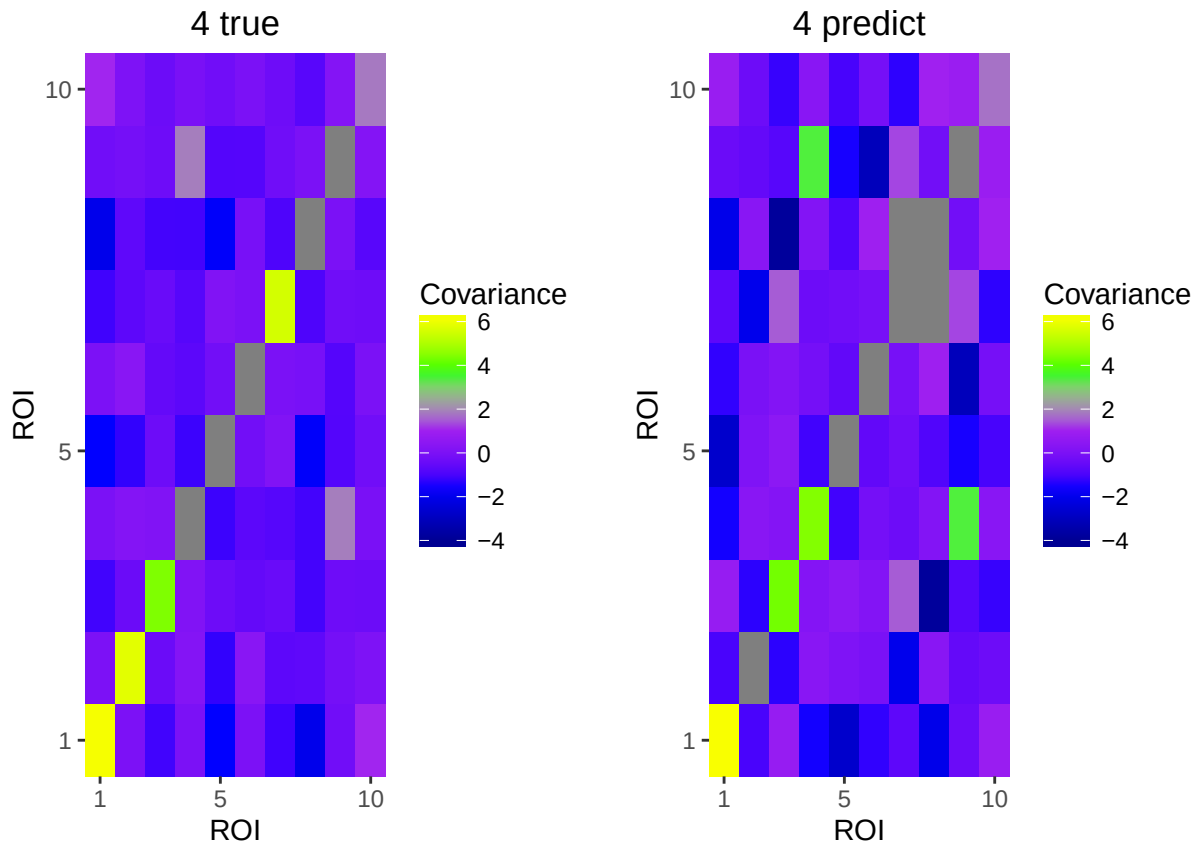

2 true
10
5
1
ROI
1
5
10
ROI
Covariance
6
4
2
0
−2
−4


2 predict
10
5
1
ROI
1
5
10
ROI
Covariance
6
4
2
0
−2
−4

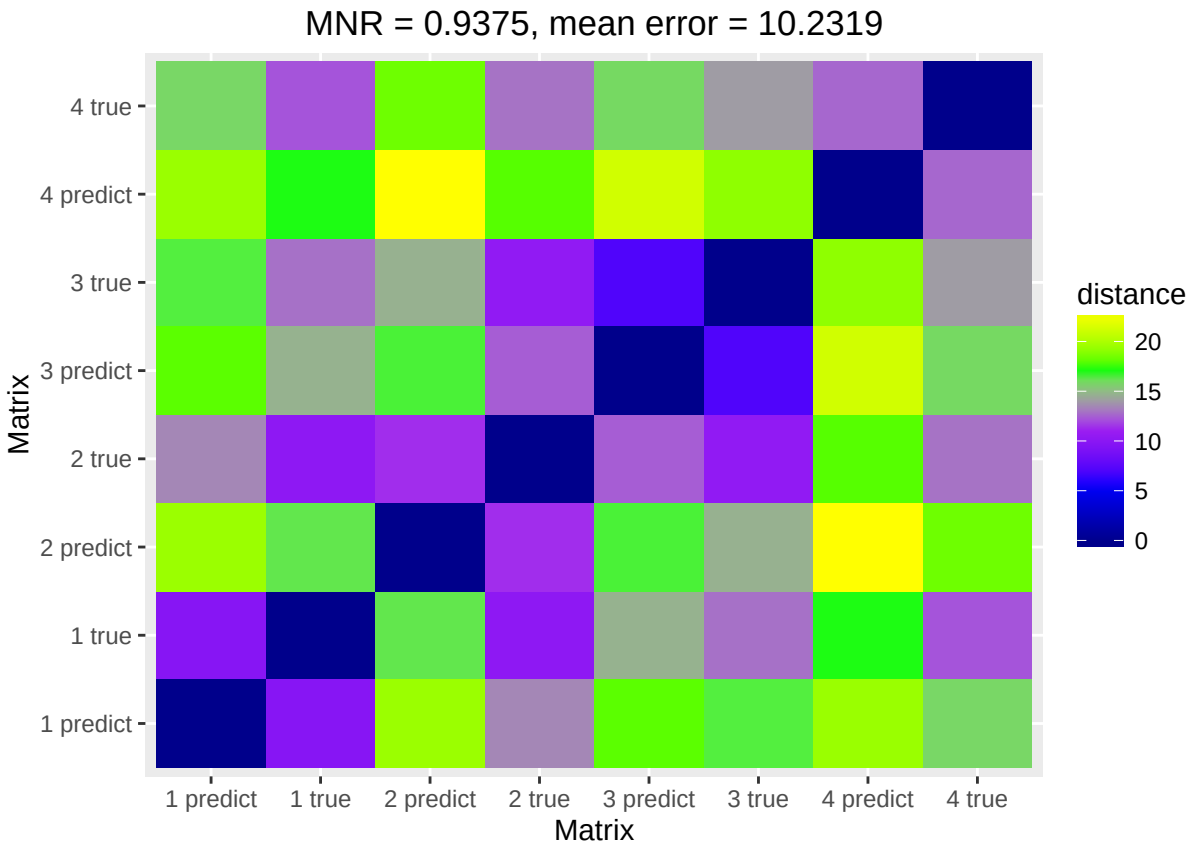
```
## NULL
```




```
## NULL
```




```
## NULL
```




```
  print(paste("100 simulation mean MNR", mean(mnr_ar)))
```

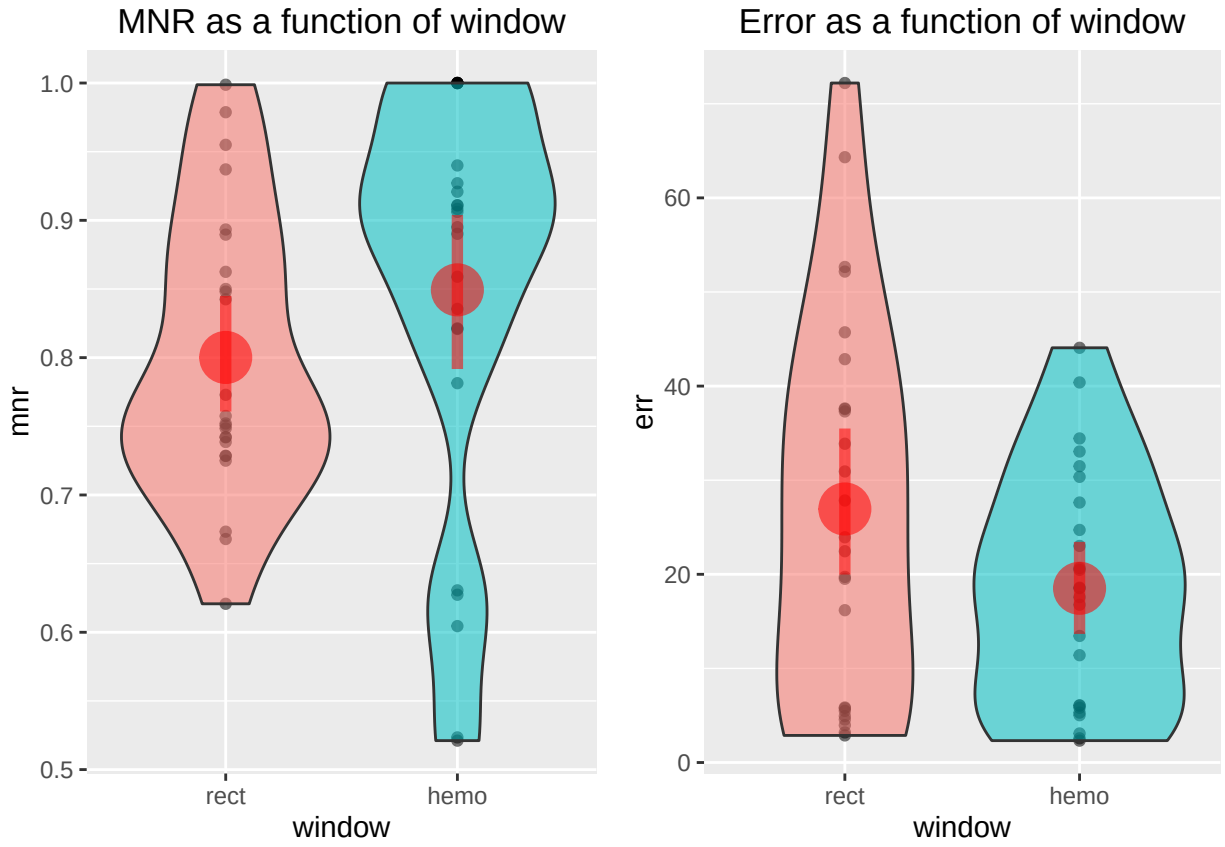
```
## [1] "100 simulation mean MNR 0.969315476190476"
```

Exploring Parameters

Here, we will do a large scale parameter sweep of our model, to see under which conditions it performs best (and where it performs worst, for future data collection). We will vary the number of timesteps, conditions, and rois, as well as change the window type, and look at the resultant impact on the dataset mnr and mean error. We also show violin plots, which show the relative probability density of the data taking different output values of the mnr or error.

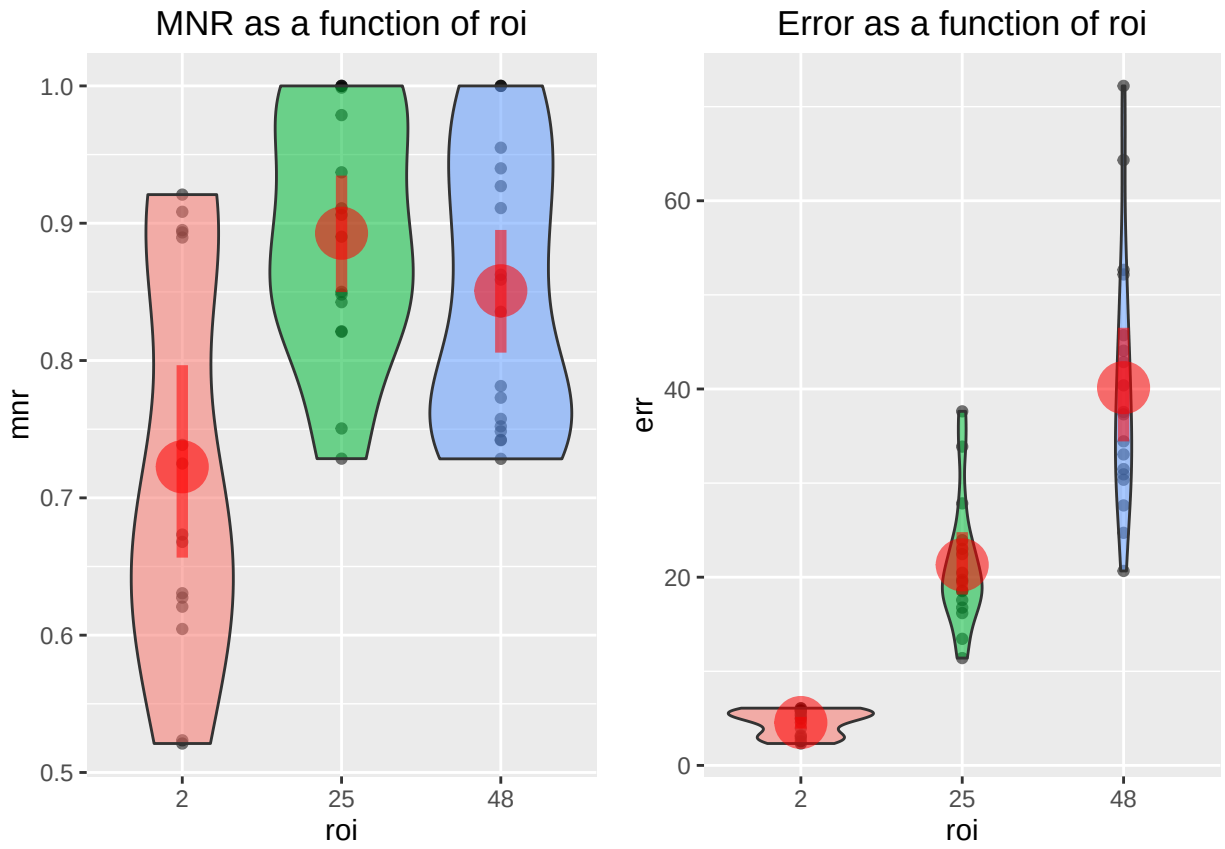
```
  niter <- 100
  window_id <- c("rect", "hemo")
  windows <- list(array(rev(c(0, .375, .95, .95, .425, .15, .05, 0))), c(array(rev(c(0, 1, 1, 1, 1, 0))
  param_explore <- data.frame(timesteps=c(), conditions=c(), window=c(), roi=c(), mnr=c(), err=c())
  for (nt in c(100, 200, 300)) {
    for (nc in c(2, 4, 6)) {
      for (w in 1:length(windows)) {
        for (nroi in c(2, 25, 48)) {
          mnr_ar <- array(NaN, dim=c(niter))
          err_ar <- array(NaN, dim=c(niter))
          for (i in 1:niter) {
            results <- one_iteration(nt, nc, nroi, window = windows[[w]], verbose=FALSE)
            mnr_ar[i] <- results$mnr
            err_ar[i] <- results$error
```

```
        }
        param_explore = rbind(param_explore, data.frame(timesteps=nt, conditions=nc, window=window_id
                                                        mnr=mean(mnr_ar), err=mean(err_ar)))
      }
    }
  }
}
```



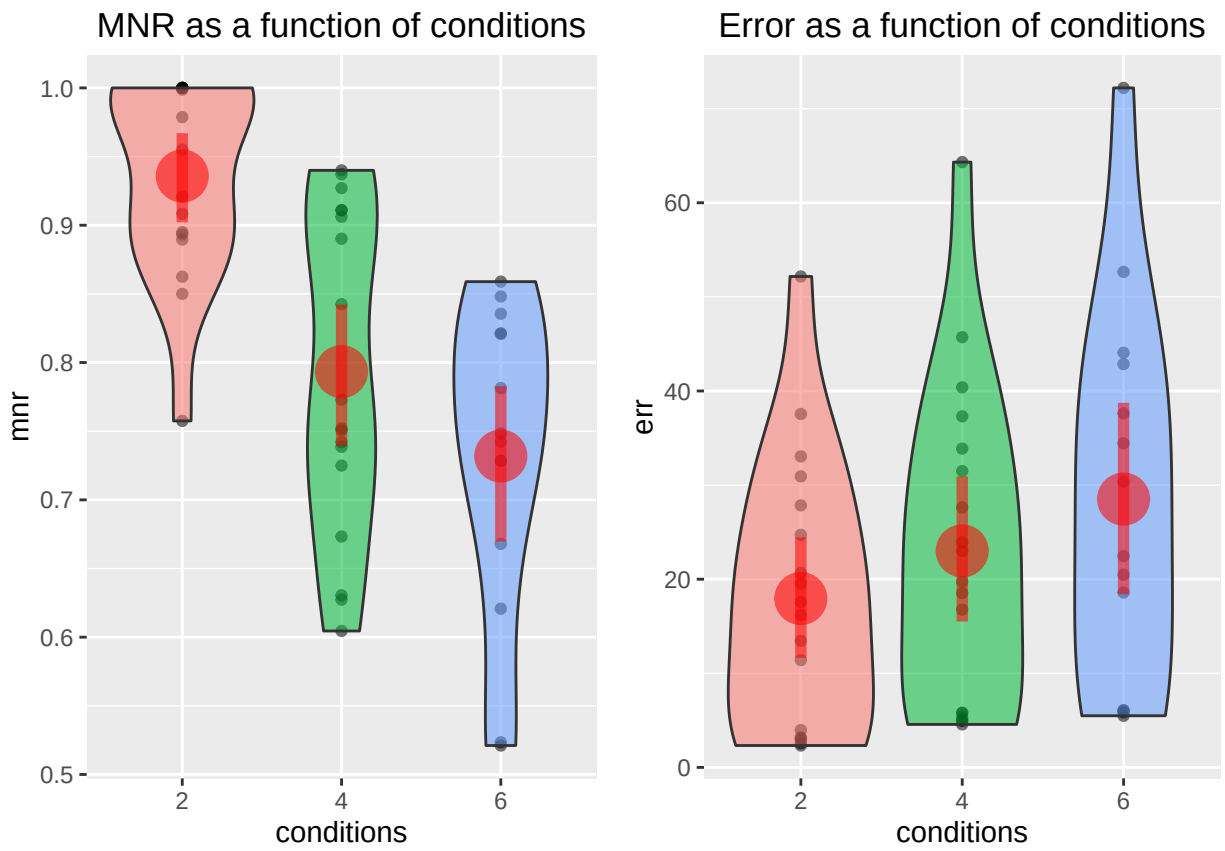

```
## NULL
```

As we can see, choosing the hemodynamic response window gives us a better estimate of our state than the rectangular window. This intuitively makes sense, as recall that the hemodynamic response window is approximately a triangle, so there will be particular periods that will have very high π_{ic} s than the rectangular window (where π_{ic} would show fewer points with one condition contributing significantly more to the mixing than the others).



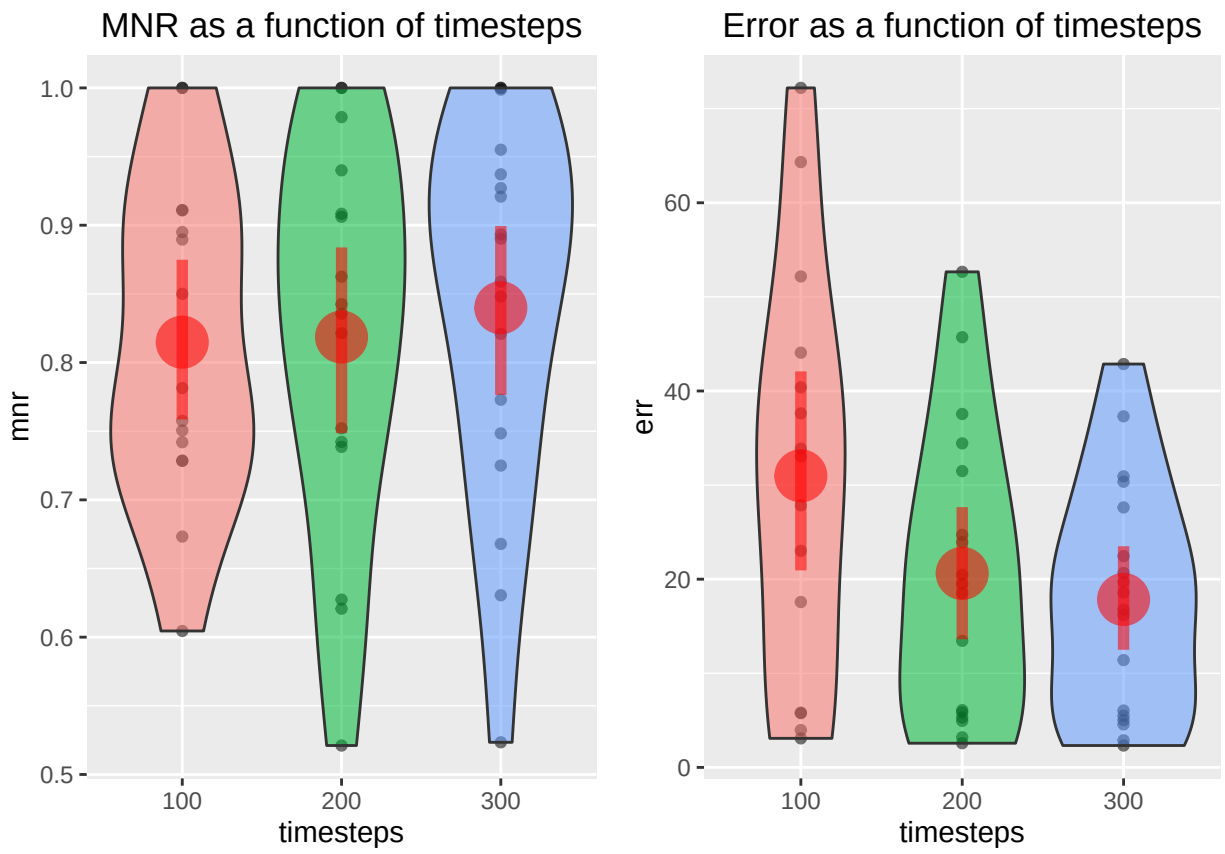

```
## NULL
```

We would expect performance to be relatively constant for different roi counts, which is confirmed here (does not appear to be any sort of pattern).




```
## NULL
```

As could be expected, as number of conditions increases, our mnr and error correspondingly increase. This is because with more possible conditions at each time step, our approximation of the contribution of any one condition at that time point is going to deteriorate.




```
## NULL
```

Here, we would intuitively expect more time steps to produce a better result, and this appears as though it will probably hold (need to test more points, but there is definitely a slight difference in the means here).

Known Criticisms

The main problem I am seeing with a lot of the initial investigations is that the actual explored parameters are very limited for the time-variant vs. time stagnant (normal GMM) model. In particular, for the n=6 case anyway, the parameters were always expected on one sweep and were determined to be the optimal parameters based on the likelihood really not changing compared to the simple guess of a diagonal matrix (our initial guesses are always diagonal matrices). The critism is relatively prominent for large numbers of conditions, so it is necessary to a) find a way to improve the performance at large numbers of conditions, or b) looking at other ways to characterize the working memory paradigm (maybe some sort of bayes network, maybe some sort of state space model focusing on signal propogation, etc). It is interesting to note that when the number of conditions is higher, our predictions always match our truth covariance matrices best, however, the truth matrices match the other truth matrices best (the predictions are always very far from the truths, even if the predictions are closest to their respective truths)