PPCA and BPCA Baseline

Load Data from csv

```
data <- as.matrix(read.csv('data_unfold.csv', row.names = 1))
masked_data <- as.matrix(read.csv('masked_data_04.csv', row.names = 1))

len_masked <- sum(apply(masked_data, 1, anyNA))
```

Preprocessing

We receive the data in a shape [309, 4, 90] and subsequently unfold it into a matrix [90, 2472]. We also separate Imaginary and Real part. Therefore, 2472/4 corresponds to the number of Multipath Components for each of the 4 sub-bands. Afterwards we split the 90 into a training and validation set, 60% and and 40% of the rows respectively. We mask the validation set with hiding 3 of the bands and liberating 1 each time. The pattern for the masking is freeing bands in this fashion: 1, 3, 2, 4 .

Metrics Used

The metrics used for are the following in this order: Mean Squared Error, Mean absolute error, Normalised Root Mean Squared Error, Root Mean Squared Error

$$MSE = \frac{1}{D}\sum_{i=1}^{D}(x_{mis} - \hat{x}_{mis})^2$$

$$MAE = \frac{1}{D}\sum_{i=1}^{D}|x_{mis} - \hat{x}_{mis}|$$

$$NRMSE = \sqrt{\frac{1}{D}\Sigma_{i=1}^{D}\frac{(x_{mis_i} - \hat{x}_{mis_i})^2}{mean(x_{mis}^2)}}$$

$$RMSE = \sqrt{\frac{1}{D}\Sigma_{i=1}^{D}\Big(x_{mis} - \hat{x}_{mis}\Big)^2}$$

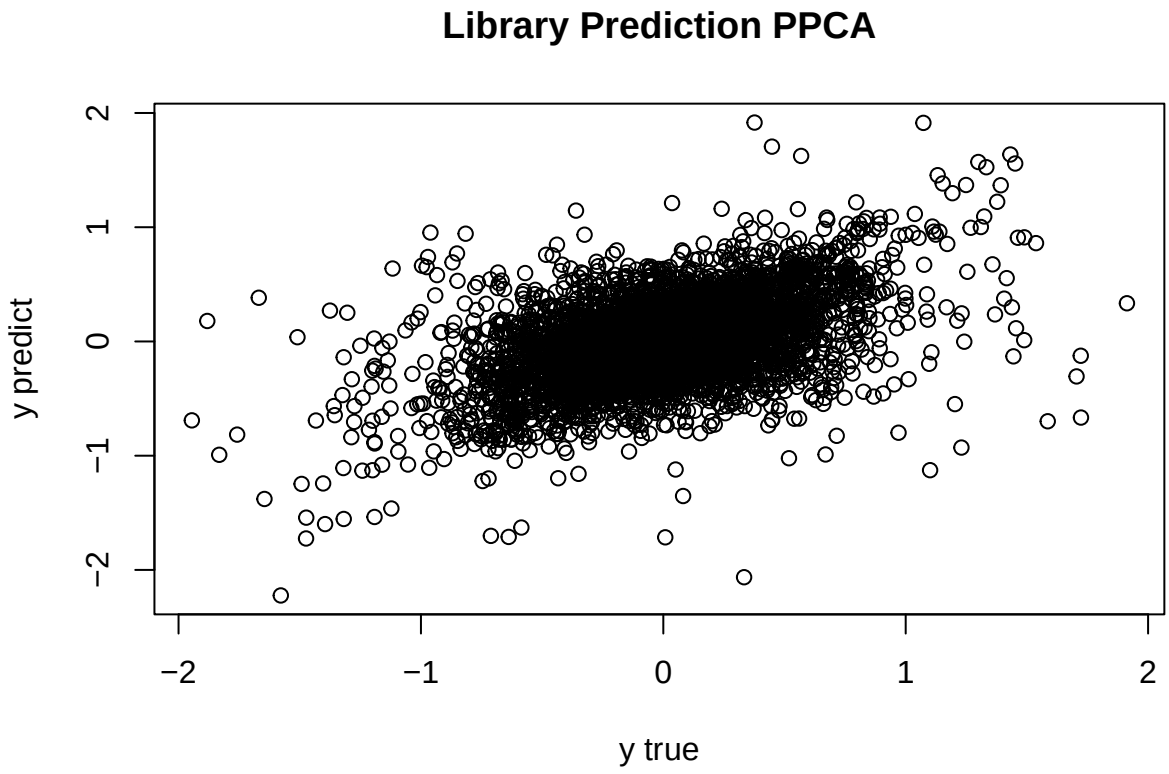
Value Imputation through library in-built functions

Hereinafter the value imputation will be done by the in-built functions of the *pcaMethods* R library. The whole dataset without splitting is fed to the algorithm where internally imputates values.

PPCA

Here the function PCA is applied to use PPCA over the whole dataset. Having (NANs) inside the mentioned dataset fed to the function, NaN values are imputed through EM algorithm optimisation on for probabilistic PCA.

```
ppca_ <-  pca(masked_data, nPcs=10, method="ppca")

imputed <- completeObs(ppca_)



corrected_Abs_SE <- sqrt(mean((data[is.na(masked_data)] -
                           imputed[is.na(masked_data)])^2) /
                         mean(data[is.na(masked_data)]^2))

MAE <- mean(abs(data[is.na(masked_data)] - imputed[is.na(masked_data)]))
RMSE <- sqrt(mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2))
MSE <- mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2)
```



	NRMSE	MAE	RMSE	MSE
Error Metric	0.975472	0.0548604	0.1203877	0.0144932

BPCA

Bayesian PCA, in turn, uses the algorithm detailed in the attached notes

```
bpca_ <-  pca(masked_data, nPcs=10, method="bpca")

imputed <- completeObs(bpca_)



corrected_Abs_SE <- sum((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2) / var(data[is.na(mas

MAE <- mean(abs(data[is.na(masked_data)] - imputed[is.na(masked_data)]))
RMSE <- sqrt(mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2))
MSE <- mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2)
```

	NRMSE	MAE	RMSE	MSE
Error Metric	8965.604	0.0193711	0.045242	0.0020468

PPCA Extraction + Prediction in validation set

Here below our implementation will be outlined. We separate training (without NaNs) and validation (with NaNs) dataset. We estimate covariance parameters of the PPCA via the EM algorithm PCA of the package function feeding the training dataset to it. Later on, variance is estimated taking the variance of the noise when we reconstruct the training dataset by performing the **cross product of scores and loadings obtained from the PPCA**.

```
# Extract PCA, Latent variables and \mu vector from training data

# PCAs
ppca_ <-  pca(train, nPcs=10, method="ppca")

# Latent Varible aka Sigma in MVNorm
pc_scores <- scores(ppca_)

# Contribution of each observation to each the Latent variable
pc_loadings <- loadings(ppca_)

# \mu vector. Mass Center
pc_mu <- center(ppca_)

# Estimated Noise
epsilon <- (fitted(ppca_)-train)

# Estimated Noise Variance
var_eps <- var(c(epsilon))
```

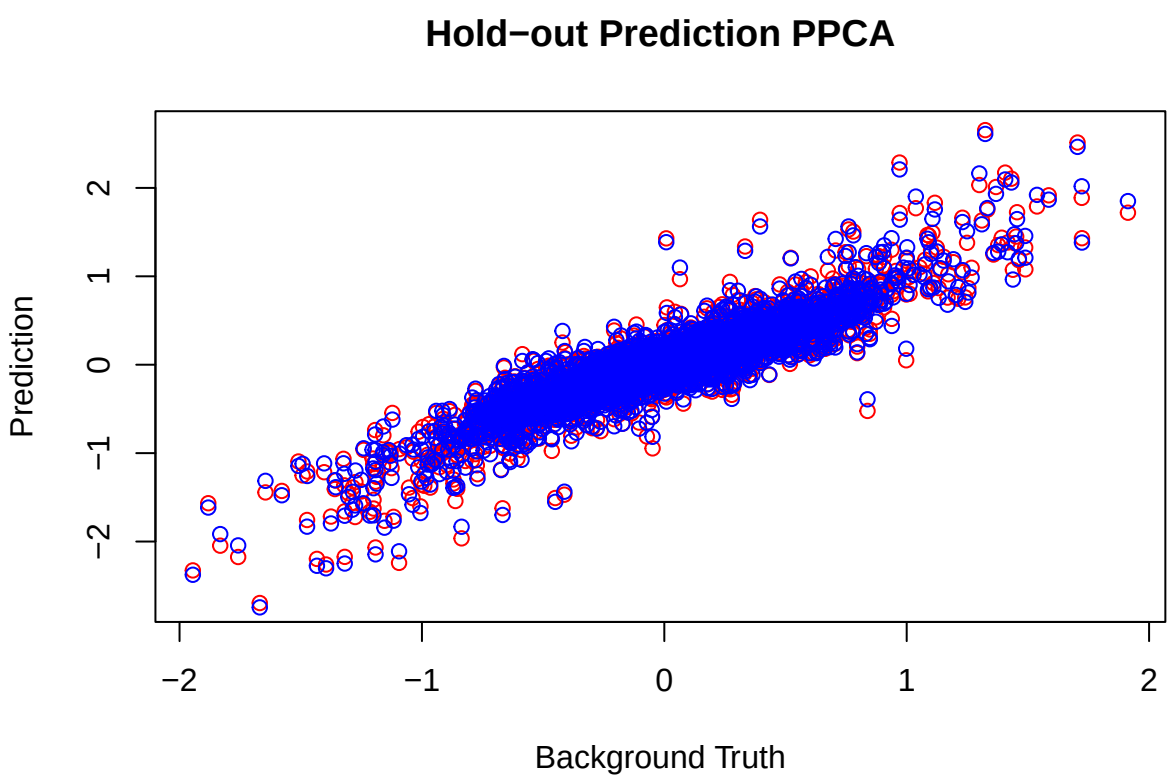
PPCA Then the prediction on the missing values is done following the formulas specified in the notes.

```
missing <- is.na(valid)

valid_pred  <- pc.predict(valid, pc_mu, pc_loadings, var_eps)

#valid_pred_var_cond <- pc.predict_varcond(valid, pc_mu, pc_loadings, var_eps)

data_rec_postpro <- prep(valid_pred, scl(ppca_), center(ppca_), reverse=TRUE)
```

A plot is performed to show how the actual predicted values correlate with the predictions. The red dots correspond to the post-processing data (summing the centered means). *The pre-processing and post processing to be discussed in my own implementation due to loadings and scores use post processingin the creation of PPCA.*

	NRMSE	MAE	RMSE	MSE
Error Metric	0.4421912	0.0276837	0.054573	0.0029782

```
## integer(0)
```

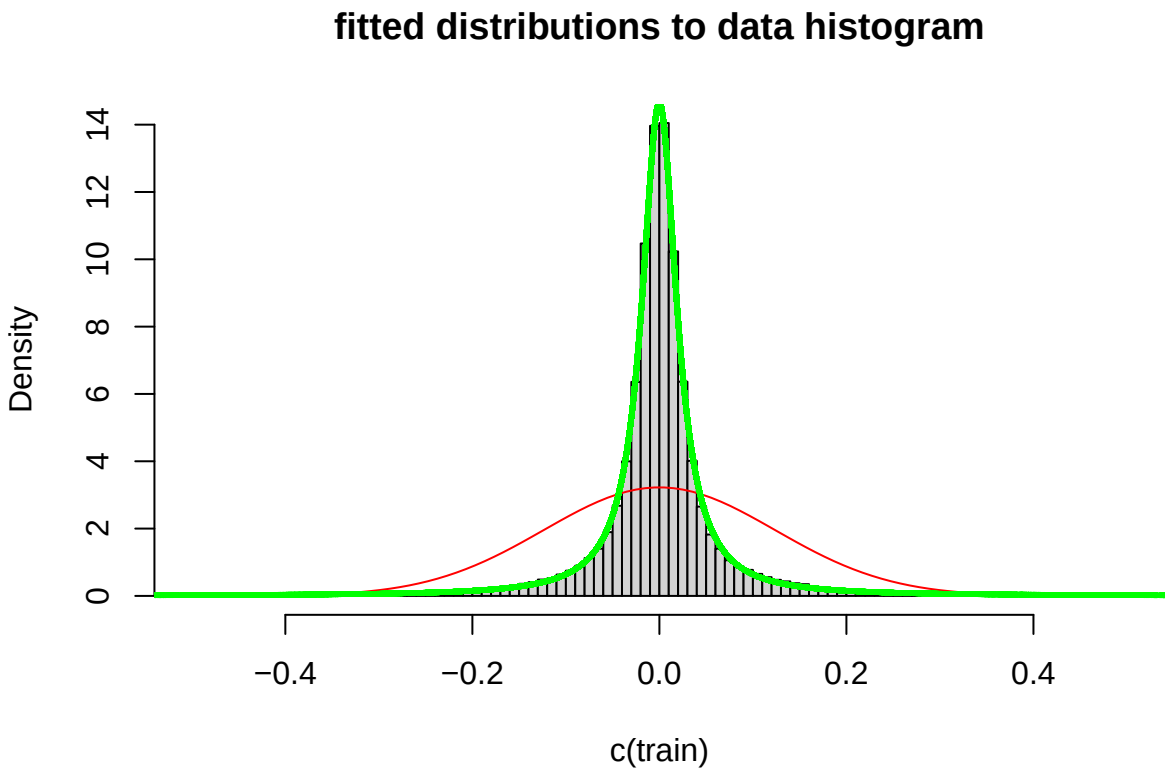


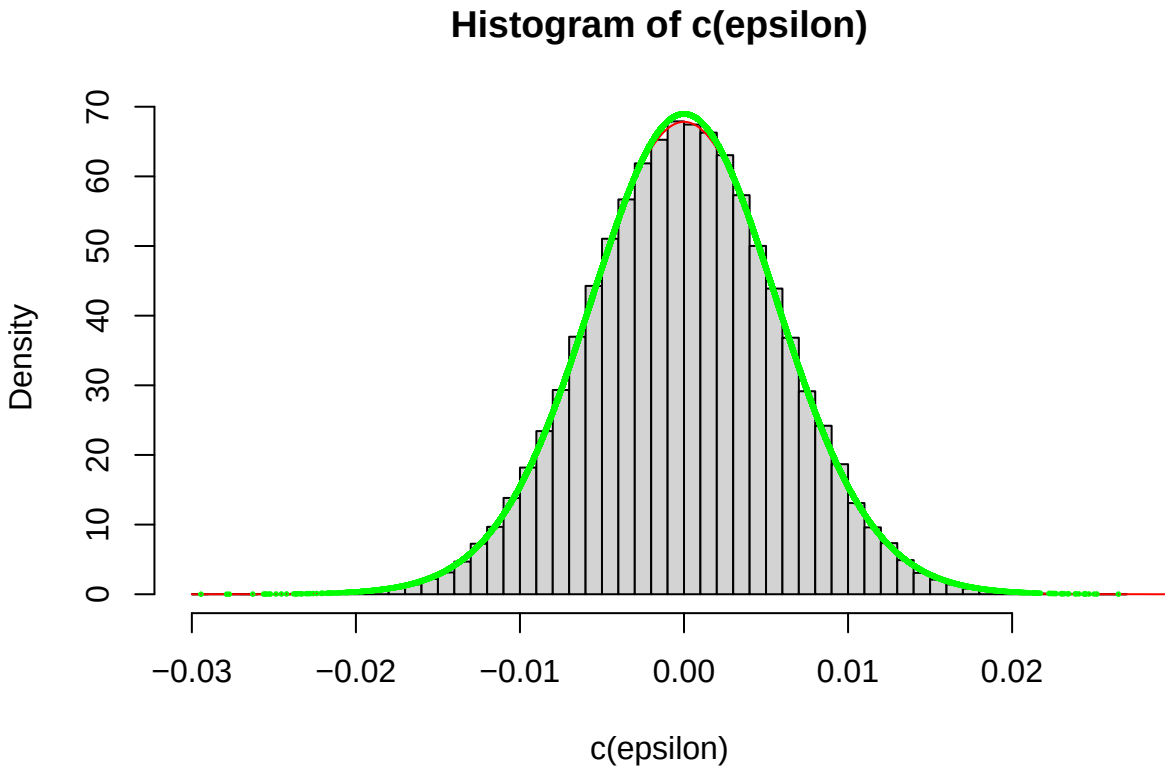
BPCA Extraction + Prediction in validation set

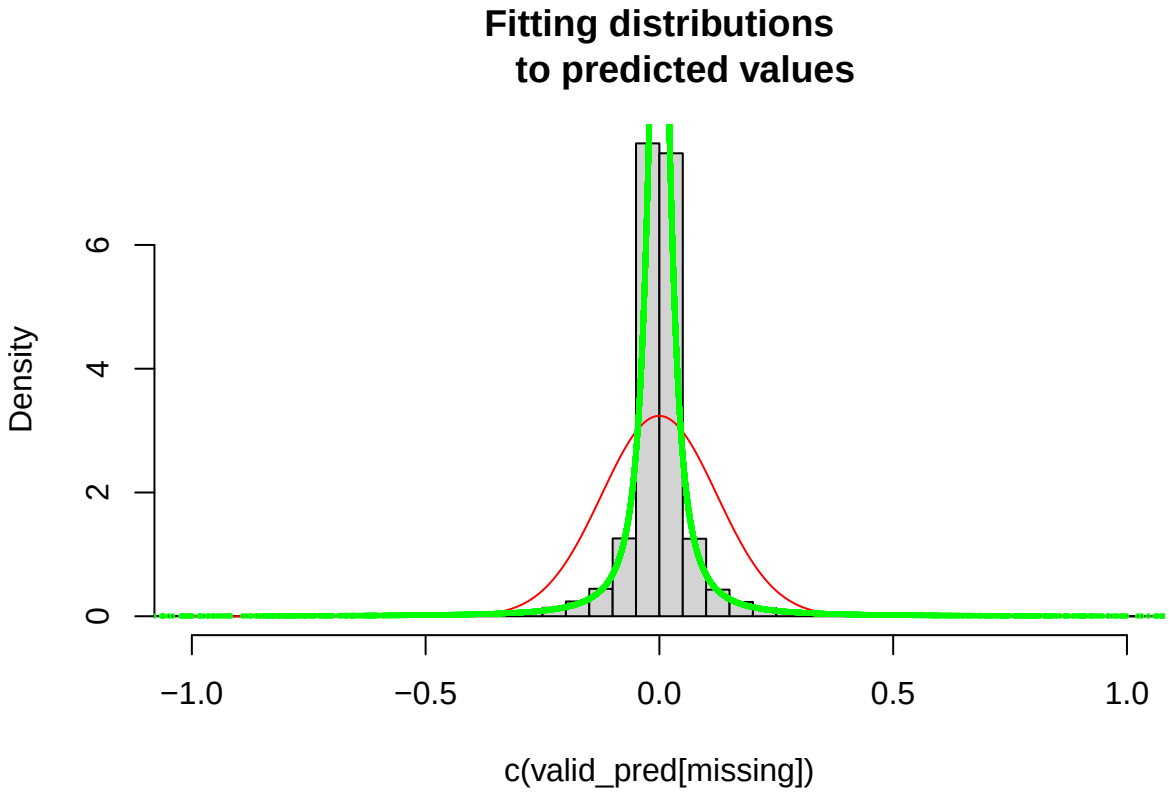
Some fix has to be done here to the library because the features are very much correlated and they use solve to invert huge covariance matrices.

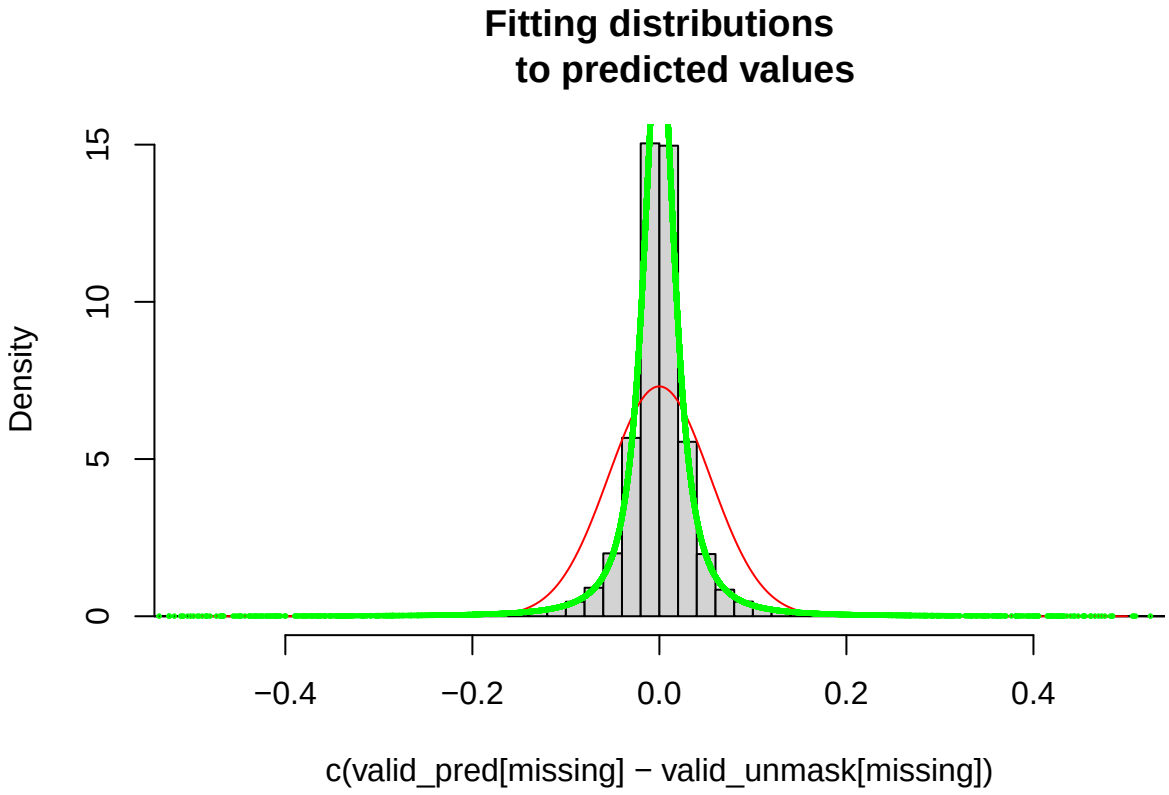
Fitting distributions on data histograms

Here below we plot the distribution of the training and the residuals of the fitting of PCA on the training data. We overlay MLE fits (Normal and T-Student Distributions) to the histograms.




Histogram of c(epsilon)
Density
0
10
20
30
40
50
60
70
−0.03
−0.02
−0.01
0.00
0.01
0.02
c(epsilon)


Fitting distributions
to predicted values
Density
0
2
4
6
−1.0
−0.5
0.0
0.5
1.0
c(valid_pred[missing])



Appendix: All code for this report

```
knitr::opts_chunk$set(echo = TRUE)
library(R.matlab)
library(BiocManager)
library(pcaMethods)
library(MetabolAnalyze)
library(MASS)

data <- readMat("1ue_quandriga_90TTI_10dB_data.mat")

mat <- data$amp

# Truncation of data
mat <- mat[1:(dim(mat)[1]/2),,]

unfold <- function(mat){

  # Unfolds Matrix "mat" with dimensions [p x sb x tti] into [tti x sb·p·2].
  # This 2 is stands for Re and Im Part Separated
    # p: Path
    # sb: Number of Freq Subband
    # tti: Time Transmitting Interval

  # Inputs
    # mat: matrix [p x sb x tti]
```

```
  # Outputs
    # unfolded: matrix [tti x sb·p·2]

  p <- dim(mat)[1] # Paths per sb per Re or Im

  sb <- dim(mat)[2]

  # empty matrix to be filled
  unfolded <- matrix(NA, dim(mat)[3], dim(mat)[1] * sb * 2)

  fold_iterate <- c(1:p, (2*p+1):(3*p), (4*p+1):(5*p), (6*p+1):(7*p))
  # Iterates unfolded mat

  fold_x <- 1

  for (i in 1:sb){ # Iterates sb (4)

    for  (j in 1:p){ # Iterates p

      subset <- mat[j,i,]

      unfolded[, fold_iterate[fold_x]]  <- Re(subset)
      unfolded[, fold_iterate[fold_x] + p]  <- Im(subset)

      fold_x <- fold_x + 1

      }
  }
  return(unfolded)
}

#data <- unfold(mat)

#write.csv(data,'data_unfold.csv')


mask_data <- function(mat,mask){

  # Masking of the data "mat" with a mask object given

   # Inputs
    # data: matrix [tti x sb·p·2]
    # mask: matrix [mask_length x sb·p·2]

   # Outputs
    # mat: matrix [tti x sb·p·2]

  bracket_mask <- nrow(mask)

  mat[(nrow(mat)-bracket_mask+1):nrow(mat), ] <-  mask *
    mat[(nrow(mat)-bracket_mask+1):nrow(mat), ]
  mat
```

```
}

len_masked <- 0
pilot_masking <- function(mat,perc){

  # It masks given a percentage of TTI beginning by last TTI. Periodically
  # masking 75% of the paths except for pilot band
    # p_sb: Paths per sb per param x 2

  # Inputs
    # data: matrix [tti x sb·p·2]
    # perc: % of TTI masked

  # Outputs
    # data_masked: matrix [tti x sb·p·2]

  p_sb <- ncol(mat)/4

  nomask_1 <- 1:p_sb
  nomask_2 <- p_sb:(2*p_sb)
  nomask_3 <- (2*p_sb):(3*p_sb)
  nomask_4 <- (3*p_sb):ncol(mat)

  ones_row <- rep(1, ncol(mat))

  mask_1 <- ones_row
  mask_1[-nomask_1] <- NA

  mask_2 <- ones_row
  mask_2[-nomask_2] <- NA

  mask_3 <- ones_row
  mask_3[-nomask_3] <- NA

  mask_4 <- ones_row
  mask_4[-nomask_4] <- NA

  maskblock <- rbind(mask_1,  mask_3, mask_2, mask_4)

  multiplicator <- masked_bracket_len/4

  mask <- do.call(rbind, replicate(multiplicator, maskblock, simplify=FALSE))

  len_masked <<- nrow(mask)

  mask_data(mat, mask)

}



#masked_data <- pilot_masking(data,0.4)
```

```r
#write.csv(masked_data, 'masked_data_04.csv')

data <- as.matrix(read.csv('data_unfold.csv', row.names = 1))
masked_data <- as.matrix(read.csv('masked_data_04.csv', row.names = 1))

len_masked <- sum(apply(masked_data, 1, anyNA))



split <- function(masked_data, len_masked){

  # Split Train and Validation Data in terms of the mask length

  train <- masked_data[1:(nrow(masked_data) - len_masked), ]

  valid <- masked_data[(nrow(masked_data) - len_masked + 1):nrow(masked_data), ]

  list(train=train, valid=valid)
}

train_valid <- split(masked_data, len_masked)

train <- train_valid$train
valid <- train_valid$valid

train_valid_unmasked  <- split(data, len_masked)
valid_unmask <- train_valid_unmasked$valid



ppca_ <-  pca(masked_data, nPcs=10, method="ppca")

imputed <- completeObs(ppca_)



corrected_Abs_SE <- sqrt(mean((data[is.na(masked_data)] -
                          imputed[is.na(masked_data)])^2) /
                         mean(data[is.na(masked_data)]^2))

MAE <- mean(abs(data[is.na(masked_data)] - imputed[is.na(masked_data)]))
RMSE <- sqrt(mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2))
MSE <- mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2)


table <- matrix(c(corrected_Abs_SE, MAE, RMSE, MSE), 1, 4)
rownames(table) <- c("Error Metric")
colnames(table) <- c("NRMSE", "MAE", "RMSE", "MSE")

plot(data[is.na(masked_data)], imputed[is.na(masked_data)], ylab = 'y predict', xlab = 'y true', main="

knitr::kable(table)
```

```r
bpca_ <-  pca(masked_data, nPcs=10, method="bpca")

imputed <- completeObs(bpca_)



corrected_Abs_SE <- sum((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2) / var(data[is.na(mas

MAE <- mean(abs(data[is.na(masked_data)] - imputed[is.na(masked_data)]))
RMSE <- sqrt(mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2))
MSE <- mean((data[is.na(masked_data)] - imputed[is.na(masked_data)])^2)



table <- matrix(c(corrected_Abs_SE, MAE, RMSE, MSE), 1, 4)
rownames(table) <- c("Error Metric")
colnames(table) <- c("NRMSE", "MAE", "RMSE", "MSE")

knitr::kable(table)



# Extract PCA, Latent variables and \mu vector from training data

# PCAs
ppca_ <-  pca(train, nPcs=10, method="ppca")

# Latent Varible aka Sigma in MVNorm
pc_scores <- scores(ppca_)

# Contribution of each observation to each the Latent variable
pc_loadings <- loadings(ppca_)

# \mu vector. Mass Center
pc_mu <- center(ppca_)

# Estimated Noise
epsilon <- (fitted(ppca_)-train)

# Estimated Noise Variance
var_eps <- var(c(epsilon))



pc.predict_1_sample <- function(sample, mu, Sigma, var_eps){

  # This function predicts the missing data for one sample x_i by means of the
  # conditional predictive distribution

  # Split Observations x_i
  obs_ind <- which(!is.na(sample))
  mis_ind <- which(is.na(sample))
```

```
  sample_obs <- sample[obs_ind]

  # Split Mus
  mu_obs <- mu[obs_ind]
  mu_mis <- mu[mis_ind]

  # Split Sigma (W)
  Sigma_obs <- Sigma[obs_ind, ]
  Sigma_mis <- Sigma[mis_ind, ]

  # chol2inv(chol(M)) where M is cov matrix

  # Compute x_mis|x_obs
  sample[mis_ind] <- mu_mis + Sigma_mis %*% solve(t(Sigma_obs) %*% Sigma_obs + var_eps*diag(ncol(Sigma_

  #Sigma_cond <- var_eps * Sigma_mis %*% solve(t(Sigma_obs) %*% Sigma_obs + var_eps*diag(ncol(Sigma_obs
  sample

}

pc.predict <- function(valid_set, mu, Sigma, var_eps){

  # Run pc.predict_1_sample for all functions

  val_pred <- apply(valid_set, 1, pc.predict_1_sample, mu, Sigma, var_eps)
  t(val_pred)

}
missing <- is.na(valid)

valid_pred  <- pc.predict(valid, pc_mu, pc_loadings, var_eps)

#valid_pred_var_cond <- pc.predict_varcond(valid, pc_mu, pc_loadings, var_eps)

data_rec_postpro <- prep(valid_pred, scl(ppca_), center(ppca_), reverse=TRUE)




# Corrected_Abs_SE <- sqrt(mean((valid_unmask[missing] - valid_pred[missing]))^2 / var(valid_unmask[mis

# Jinliangs
corrected_Abs_SE <- sqrt(mean(abs(valid_unmask[missing] - valid_pred[missing])^2) / mean(abs(valid_unmas

MAE <- mean(abs(valid_unmask[missing] - valid_pred[missing]))
RMSE <- sqrt(mean((valid_unmask[missing] - valid_pred[missing])^2))
MSE <- mean((valid_unmask[missing] - valid_pred[missing])^2)

table <- matrix(c(corrected_Abs_SE, MAE, RMSE, MSE), 1, 4)
rownames(table) <- c("Error Metric")
colnames(table) <- c("NRMSE", "MAE", "RMSE", "MSE")
```

```r
knitr::kable(table)

plot(valid_unmask[missing], valid_pred[missing], col='red', ylab= 'Prediction',
     xlab= 'Background Truth', main="Hold-out Prediction PPCA") +
  points(valid_unmask[missing], data_rec_postpro[missing], col='blue')

legend(1,95, legend=c("Predictions", "Post-Processed prediction"),
       col=c("red", "blue"), lty=1:2, cex=0.8)

#```{r }

################################################################################
######################### system is computationally r#############################
#########################singular: reciprocal condition number####################
################################################################################

## Extract BPCA, Latent variables and \mu vector from training data
#
## BPCAs
#bpca_ <-  pca(train, nPcs=10, method="bpca")
#
## Latent Varible aka Sigma in MVNorm
#pc_scores <- scores(bpca_)
#
## Contribution of each observation to each the Latent variable
#pc_loadings <- loadings(bpca_)
#
## \mu vector. Mass Center
#pc_mu <- center(bpca_)
#
## Estimated Noise
#epsilon <- (completeObs(ppca_)-train)
#
## Estimated Noise Variance
#var_eps <- var(c(epsilon))
#
#
#valid_pred  <- pc.predict(valid, pc_mu, pc_loadings, var_eps)
#data_rec_postpro <- prep(valid_pred, scl(ppca_), center(ppca_), reverse=TRUE)
#
#
#corrected_Abs_SE <- sum((valid_unmask[missing] - valid_pred[missing])^2) / var(valid_unmask[missing])
#
#MAE <- mean(abs(valid_unmask[missing] - valid_pred[missing]))
#RMSE <- sqrt(mean((valid_unmask[missing] - valid_pred[missing])^2))
#MSE <- mean((valid_unmask[missing] - valid_pred[missing])^2)
#
#```{r }
#table <- matrix(c(corrected_Abs_SE, MAE, RMSE, MSE), 1, 4)
#rownames(table) <- c("Error Metric")
#colnames(table) <- c("NRMSE", "MAE", "RMSE", "MSE")
#
#knitr::kable(table)
```

```
#
#plot(valid_unmask[missing], valid_pred[missing], col='red', ylab= 'Prediction',
#     xlab= 'Background Truth', main="Hold-out Prediction BPCA") +
#  points(valid_unmask[missing], data_rec_postpro[missing], col='blue')
#
#legend(1,95, legend=c("Predictions", "Post-Processed prediction"),
#       col=c("red", "blue"), lty=1:2, cex=0.8)
#
#

################################################################################
################### Fitting distributions on data histograms ###################
################################################################################

fitdistnorm <- fitdistr(c(train), densfun = "normal")
fitdistt <- fitdistr(c(train), densfun = "t")

x <- seq(-.5, .5, length=100)

normfit <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])
hx <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])

tfit <- dt(x, df =  fitdistt$estimate[[3]])

yvals <- dt((c(train) - fitdistt$estimate[[1]])/fitdistt$estimate[[2]],
            df = fitdistt$estimate[[3]])/fitdistt$estimate[[2]]

################################## Plots ######################################

hist(c(train), breaks = 500, xlim = c(-0.5, .5), freq =FALSE,
     main = 'fitted distributions to data histogram')
lines(x, hx,type = 'l', # density plot ,
      col = "red")
points((c(train)), yvals, cex=.25, col='green')
legend(1, 95, legend=c("MLE Student-T", "MLE Normal"),
       col=c("green", "blue"), lty=1:2, cex=0.8)


################################################################################
###################### Fitting distributions on the residuals ##################
################################################################################

fitdistnorm <- fitdistr(c(epsilon), densfun = "normal")
fitdistt <- fitdistr(c(epsilon), densfun = "t")

x <- seq(-.03, .03, length=1000 )

normfit <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])
hx <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])

tfit <- dt(x, df =  fitdistt$estimate[[3]])
yvals <- dt((c(epsilon) - fitdistt$estimate[[1]])/fitdistt$estimate[[2]],
            df = fitdistt$estimate[[3]])/fitdistt$estimate[[2]]
```

```
################################## Plots ########################################

hist(c(epsilon), breaks = 70, probability = TRUE)
lines(x, hx,type = 'l', # density plot ,
      col = "red", cex=2)
points((c(epsilon)), yvals, cex=.25, col='green')
legend(1, 95, legend=c("MLE Student-T", "MLE Normal"),
       col=c("green", "blue"), lty=1:2, cex=0.8)

#################################################################################
###################### Fitting distributions on the predicted ##################
#################################################################################

fitdistnorm <- fitdistr(c(valid_pred[missing]), densfun = "normal")
fitdistt <- fitdistr(c(valid_pred[missing]), densfun = "t")

x <- seq(-1, 1, length=1000)

normfit <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])
hx <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])

tfit <- dt(x, df =  fitdistt$estimate[[3]])
yvals <- dt((c(valid_pred[missing]) - fitdistt$estimate[[1]])/fitdistt$estimate[[2]],
            df = fitdistt$estimate[[3]])/fitdistt$estimate[[2]]

################################## Plots ########################################

hist(c(valid_pred[missing]), breaks = 130, probability = TRUE, main = 'Fitting distributions
     to predicted values', xlim=c(-1,1))
lines(x, hx,type = 'l', # density plot ,
      col = "red", cex=2)
points((c(valid_pred[missing])), yvals, cex=.25, col='green')
legend(1, 95, legend=c("MLE Student-T", "MLE Normal"),
       col=c("green", "blue"), lty=1:2, cex=0.8)

#################################################################################
################ Fitting distributions on the predicted error ##################
#################################################################################

fitdistnorm <- fitdistr(c(valid_pred[missing] - valid_unmask[missing]), densfun = "normal")
fitdistt <- fitdistr(c(valid_pred[missing] - valid_unmask[missing]), densfun = "t")

x <- seq(-.5, .5, length=1000)

normfit <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])
hx <- dnorm(x, fitdistnorm$estimate[[1]], fitdistnorm$estimate[[2]])

tfit <- dt(x, df =  fitdistt$estimate[[3]])
yvals <- dt((c(valid_pred[missing]- valid_unmask[missing]) - fitdistt$estimate[[1]])/fitdistt$estimate[
            df = fitdistt$estimate[[3]])/fitdistt$estimate[[2]]

################################## Plots ########################################
```

```
hist(c(valid_pred[missing]- valid_unmask[missing]), breaks = 130, probability = TRUE, main = 'Fitting di
     to predicted values', xlim=c(-.5, .5))
lines(x, hx,type = 'l', # density plot ,
      col = "red", cex=2)
points((c(valid_pred[missing]- valid_unmask[missing])), yvals, cex=.25, col='green')
legend(1, 95, legend=c("MLE Student-T", "MLE Normal"),
       col=c("green", "blue"), lty=1:2, cex=0.8)
```