

Janus function

Coefficients:

Pearson Correlation:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Sample covariance:

$$Cov(x, y) = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{N - 1}$$

Covariance contribution function:

R function

```
janus <- \(matrix, small_set_samples, normalize_by = "none") {  
  
  matrix <- t(matrix)  
  matrix <- scale(.input$concatenated, center = T, scale = F) ①  
  output <- list(large = matrix[,setdiff(colnames(matrix), small_set_samples)], ②  
                small = matrix[,small_set_samples])  
  output <- map(matrix, ~ {# the following steps are individually performed on both sets  
    .x <- t(.x) # to compute a genes x genes matrix, the object need to be transposed  
    .x[is.na(.x)] <- 0 # Zero-impute NAs, otherwise genes with NAs wont be quantified  
    .x <- .x %*% t(.x) ③  
  
    if (normalize_by == "sd") {##### rework  
      .n <- apply(.x, 1, \(.) {sum(!(is.na(.) | . == 0))}) # number of observations per gene  
      .x <- .x / .n
```

```

}

diag(.x) <- NA
.x})

if (normalize_by == "N") {
  output$large <- output$large / length(setdiff(colnames(matrix), small_set_samples))
  output$small <- output$small / length(small_set_samples)
}
#output <- abind::abind(output$large, output$small, along = 3, new.names = c("large", "small"))
return(output)
}

```

- ① Subtract the sample-wise mean to center the object.
- ② The matrix is split along the large and small set.
- ③ Compute Gram-matrix of mean-centered fitness effects. This is the equivalent of the numerator of the covariance formula $\sum (x_i - \bar{x})(y_i - \bar{y})$.
- ④ Set self-contributions to NA
- ⑤ Normalize the covariance contribution by the number of samples of both sets. Equivalent to dividing by N .

Python function (Yasir)

```

def get_contributions(df, sample_size_separation):

    #std = df.T.std(ddof=1)
    #std_matrix = pd.DataFrame(np.outer(std, std), columns=df.index, index=df.index)

    # Mean expression across all samples
    global_mean = df.T.mean()
    mean_centered = df.apply(lambda x: x - global_mean)

    # Split into big and small groups
    big_mean = mean_centered.iloc[:, :sample_size_separation]
    small_mean = mean_centered.iloc[:, sample_size_separation:]

    # Dot products for each group (co-deviation matrix)
    contr_big = pd.DataFrame(np.dot(big_mean, big_mean.T), columns=df.index, index=df.index)
    contr_small = pd.DataFrame(np.dot(small_mean, small_mean.T), columns=df.index, index=df.index)

```

```

# Normalize by Euclidean norm of mean-centered expression vectors
norms = mean_centered.T.apply(np.linalg.norm)
denom = pd.DataFrame(np.outer(norms, norms), columns=df.index, index=df.index)

# Final normalized contribution matrices
contr_big /= denom
contr_small /= denom

# Remove diagonal (self-contribution)
np.fill_diagonal(contr_big.values, np.nan)
np.fill_diagonal(contr_small.values, np.nan)

return contr_big, contr_small

```

- ① Subtract the sample-wise mean to center the object.
- ② The matrix is split along the large and small set.
- ③ Compute Gram-matrix of mean-centered fitness effects. This is the equivalent of the numerator of the covariance formula $\sum (x_i - \bar{x})(y_i - \bar{y})$.
- ④ Set self-contributions to NA