Auxillary Materials for

Causation without Correlation: Detecting Hidden Causal Links in Transcriptional Networks

Gerald M Pao[1,2], Ethan R Deyle[2,3], Hao Ye[2,4], Junko Ogawa[1], Marisela Guaderrama[5], Manching Ku[7], Tom Lorimer[2], Nina Tonnu[1], Erik Saberski[2], Joseph Park[6], Eugene Ke[8], Curt Wittenberg[5], Inder M Verma[9], George Sugihara[2].

Correspondence to: pao@salk.edu; gsugihara@ucsd.edu

This file takes several inputs and produces several outputs. The outputs cover all necessary files for reproducing key results of Main Text and Supplementary Text.

Pre-amble

These analyses were performed using rEDM 0.7.4. Recent rework has brought a common code-base to R and Python packages, available at https://cran.r-project.org/package=rEDM and https://pypi.org/project/pyEDM/, respectively. If the user has an existing installation of rEDM that is not 0.7.4 it is necessary to run the following to install the archieved version 0.7.4 to run these analyses.

```
if(tryCatch((packageVersion("rEDM")=="0.7.4"),error=function(e) FALSE)){
  library('rEDM')
}else{
  dir_alt_library <- paste0(.libPaths()[1],"-alt") # This can be changed by u
ser preference
  dir.create(dir_alt_library)
  withr::with_libpaths(
    new = dir_alt_library,
    code = devtools::install_github("ha0ye/rEDM@v0.7.4")
  )
  library("rEDM",lib.loc = dir_alt_library)
}
```

Load other required packages.

```
library("tidyverse")
library("gridExtra")
library("grid")
library("ggraph")
library("igraph")
```

Now check that session info matches or that any discrepencies will not cause bugs or errors.

```r
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRbla
s.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlap
ack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] igraph_1.2.4.1  ggraph_2.0.0    gridExtra_2.3   forcats_0.4.0
##  [5] stringr_1.4.0   dplyr_0.8.5     purrr_0.3.3     readr_1.3.1
##  [9] tidyr_1.0.2     tibble_2.1.3    ggplot2_3.3.0   tidyverse_1.3.0
## [13] rEDM_0.7.4
##
## loaded via a namespace (and not attached):
##  [1] ggrepel_0.8.1      Rcpp_1.0.3         lubridate_1.7.4
##  [4] lattice_0.20-38    assertthat_0.2.1   digest_0.6.19
##  [7] ggforce_0.3.1      R6_2.4.0           cellranger_1.1.0
## [10] backports_1.1.4    reprex_0.3.0       evaluate_0.14
## [13] httr_1.4.1         pillar_1.4.3       rlang_0.4.5
## [16] readxl_1.3.1       rstudioapi_0.10    rmarkdown_1.13
## [19] polyclip_1.10-0    munsell_0.5.0      broom_0.5.2
## [22] compiler_3.6.0     modelr_0.1.6       xfun_0.8
## [25] pkgconfig_2.0.2    htmltools_0.3.6    tidyselect_1.0.0
## [28] codetools_0.2-16   graphlayouts_0.5.0 viridisLite_0.3.0
## [31] crayon_1.3.4       dbplyr_1.4.2       withr_2.1.2
## [34] MASS_7.3-51.4      nlme_3.1-140       jsonlite_1.6
## [37] gtable_0.3.0       lifecycle_0.2.0    DBI_1.0.0
## [40] magrittr_1.5       scales_1.0.0       cli_1.1.0
## [43] stringi_1.4.3      farver_2.0.1       viridis_0.5.1
## [46] fs_1.3.1           xml2_1.3.1         generics_0.0.2
## [49] vctrs_0.2.4        tools_3.6.0        glue_1.3.1
## [52] tweenr_1.0.1       hms_0.5.3          yaml_2.2.0
## [55] colorspace_1.4-1   tidygraph_1.1.2    rvest_0.3.5
## [58] knitr_1.23         haven_2.2.0
```

Finally we want to source the `help_functions.R` script included in the repository. This script contains several functions to reproduce plots and pieces of plots from the Main Text from the analysis performed in this Markdown.

```
source('help_functions.R')
```

DATA

Read in raw data

The raw data for Yeast are formatted as follows:

```
- row 1 = index for which dataset
- row 2 = time ordering within each dataset
- row 3 = sample ID
- col 1 = row name
- if input has m rows and n cols, then data values occur for rows 4:m, cols 2
:n
```

We read in and apply formatting to aid understanding. This includes relabelling the datasets by the batch and strain. Batch “whi5” refers to Batch 1 in the main text, where the experimental manipulation targeted WHI5 through overexpression; batch “yhp1” refers to Batch 2 in the main text, where experimental manipulation targeted YHP1 through knockout.

```
options("stringsAsFactors" = FALSE)

read_raw_data <- function(data_file = "CorrectedYeast-UQnormalized-genes_TPM.
txt",
                          out_file = "data/gene_data_TPM.Rdata")
{


    raw_data <- read.table(data_file)
    gene_names <- raw_data[4:NROW(raw_data), 1]
    gene_names <- gsub("[\\(\\)-]", ".", gene_names)
    raw_data <- raw_data[, -1]

    dataset_idx <- as.factor(t(raw_data[1, ]))
    # levels(dataset_idx) <- c("whi5_wt", "whi5_exp", "yhp1_wt", "yhp1_exp")
    levels(dataset_idx) <- c("WT1", "ExM1", "WT2", "ExM2")
    dataset_time <- as.numeric(t(raw_data[2, ])) + 1
    raw_data <- raw_data[c(-1, -2, -3), ]

    raw_data <- t(raw_data)
    class(raw_data) <- "numeric"
    gene_data <- data.frame(raw_data)
    names(gene_data) <- gene_names
    gene_data <- data.frame(dataset = dataset_idx,
                            time = dataset_time,
                            gene_data)
    save(gene_data, file = out_file)
}
```

The raw data from the Mouse MEF are formatted differently, thus define a separate function.

```r
read_raw_mouse <- function(data_file_1 = "Main-genes_MEF_original_96_RAWTMP.c
sv",
                           data_file_2 = "Main-genes_MEF_CF1.csv",
                           data_file_3 = "mouse_data_TMP_E2FVP16.csv",
                           out_file = "Data/mouse_data_TMP.Rdata")
{
    # read in raw data from tab-delimited files
    #   row 1 = time ordering within each dataset
    #   row 2...  = samples
    #   col 1 = row (gene) name
    #   if input has m rows and n cols, then
    #     data values occur for rows 4:m, cols 2:n

    raw_data_1 <- read.table(data_file_1,sep=",")
    raw_data_2 <- read.table(data_file_2,sep=",")
    raw_data_3 <- read.table(data_file_3,sep=",")
    raw_data <- full_join(raw_data_1,raw_data_2,by="V1")
    raw_data <- full_join(raw_data,raw_data_3,by="V1")
    gene_names <- raw_data[2:NROW(raw_data), 1]
    gene_names <- gsub("ENSMUSG00000", "MUSG", gene_names)
    raw_data <- raw_data[, -1]

    # time is stored in a string describing the sample
    # Ex: c1_-1h_mm10_gencode-m5
    get_hr <- function(s) as.numeric(regmatches(s,regexec("_(.+)h",s))[[1]][2
])
    dataset_time <- sapply(t(raw_data[1, ]),get_hr,USE.NAMES = FALSE)
    raw_data <- raw_data[-1, ]

    raw_data <- t(raw_data)
    class(raw_data) <- "numeric"
    gene_block_mouse_joined <- data.frame(raw_data)

    I_nonzero <- gene_block_mouse_joined %>% summarize_all(funs(sd(.,na.rm=TR
UE))) %>% `>`(0) %>% which()
    # I_nonzero <- gene_block_mouse_joined %>% summarize_all(list(~ all(dupli
cated(. )[-1L])) ) %>% which()
    gene_block_mouse_joined <- gene_block_mouse_joined[,I_nonzero]
    names(gene_block_mouse_joined) <- gene_names[I_nonzero]

    gene_block_mouse_joined <- data.frame(dataset = "MEF_joined",
                                          time = dataset_time,
                                          gene_block_mouse_joined) %>%
      filter(time == floor(time))

    save(gene_block_mouse_joined, file = out_file)
}
```

Read in and save datasets

```r
out_file <- "Data/gene_data_TPM_corrected.Rdata"
if(!file.exists(out_file)){
  read_raw_data(data_file = "CorrectedYeast-UQnormalized-genes_TPM.txt",
                out_file = out_file)
}

out_file <- "Data/mouse_data_TMP.Rdata"
if(!file.exists(out_file)){
  read_raw_mouse(data_file_1 = "Main-genes_MEF_original_96_RAWTMP.csv",
                 data_file_2 = "Main-genes_MEF_CF1.csv",
                 data_file_3 = "mouse_data_TMP_E2FVP16.csv",
                 out_file = "Data/mouse_data_TMP.Rdata")
}
```

Process raw data into gene blocks

Next the loaded data are standardized by applying a linear scaling. As described in the text Methods, the standardization is done to each gene by batch to account for any batch effects while preserving any change due to the manipulation experiment. For each gene, the wildtype time series will have min = 0, max = 1, and the corresponding gene in the experimental dataset will be transformed using the same scale.

```r
process_gene_blocks <- function(in_file = "data/gene_data_TPM_corrected.Rdata
",
                                out_file = "data/gene_blocks_TPM_corrected.Rd
ata")
{
  load(in_file)
  gene_names <- names(select(gene_data,-dataset,-time))

  # gene_block_whi5_wt <- gene_data %>% filter(dataset == "whi5_wt")
  # gene_block_whi5_exp <- gene_data %>% filter(dataset == "whi5_exp")
  # gene_block_yhp1_wt <- gene_data %>% filter(dataset == "yhp1_wt")
  # gene_block_yhp1_exp <- gene_data %>% filter(dataset == "yhp1_exp")

  gene_block_WT1 <- gene_data %>% filter(dataset == "WT1")
  gene_block_ExM1 <- gene_data %>% filter(dataset == "ExM1")
  gene_block_WT2 <- gene_data %>% filter(dataset == "WT2")
  gene_block_ExM2 <- gene_data %>% filter(dataset == "ExM2")

  for(gene in gene_names)
  {
    target_min <- min(gene_block_WT1[, gene])
    target_max <- max(gene_block_WT1[, gene])

    gene_block_WT1[, gene] <- (gene_block_WT1[, gene] - target_min) /
      (target_max - target_min)
    gene_block_ExM1[, gene] <- (gene_block_ExM1[, gene] - target_min) /
```

```
      (target_max - target_min)

    target_min <- min(gene_block_WT2[, gene])
    target_max <- max(gene_block_WT2[, gene])

    gene_block_WT2[, gene] <- (gene_block_WT2[, gene] - target_min) /
      (target_max - target_min)
    gene_block_ExM2[, gene] <- (gene_block_ExM2[, gene] - target_min) /
      (target_max - target_min)
  }

  save(gene_block_WT1, gene_block_ExM1,
       gene_block_WT2, gene_block_ExM2,
       file = out_file)
  return()
}

if(!file.exists("data/gene_blocks_TPM_corrected.Rdata")){
  process_gene_blocks()
}

load("data/gene_blocks_TPM_corrected.Rdata")
load("data/mouse_data_TMP.Rdata")
```

Univariate Analyses

Standard univariate EDM analysis is composed of sequential forecast analysis with simplex projection and locally weighted linear regression (S-map). First we define a function that performs this analysis on a single gene time-series. This function returns test statistics that are used for time-series categorization:

- "stochastic", rho > 0 with p <= 0.05
- "linear", delta_mae >= 0
- "possibly linear", delta_mae < 0
- "nonlinear 90%", delta_mae < 0 with p < 0.1
- "nonlinear 95%", delta_mae < 0 with p < 0.05

```
do_uEDM <- function(time_series,
                          lib=c(1,length(time_series)),
                          E_list=1:8){


  # Run simplex with leave-one-out cross-validation across the range of candi
date embedding dimensions, E_list.
  # Returns "out_simplex"", a data.frame with prediction statistics for each
embedding dimensions.
  out_simplex <- simplex(time_series,lib=lib,pred=lib,E=E_list)
  # Select the row with the highest forecast-skill "rho", adding "_simplex" o
nto names of prediction statistics
  # to distinguish for S-map prediction statistics, and storing in a new data
.frame "out".

  out <- out_simplex %>%
    top_n(1,rho) %>%
    select(E,tau,tp,num_pred,rho,mae,rmse,perc,p_val) %>%
    rename_at(vars(rho,mae,rmse,perc,p_val),funs(paste0(.,"_simplex")))

  # The embedding dimension saved from this single row of "out_simplex" is th
e optimal embedding dimension
  # selected as described in Sugihara & May 1990.

  E_star <- out$E[1]

  # Run S-map with leave-one-out cross-validation at the univariate optimal e
mbedding dimension, E_star,
  # across the default range of theta-values for s_map in 0.7.4.
  # Returns "out_s_map", a data.frame with prediction statistics for each the
ta value. Note that with stats_only=F
  # this also returns a column of nested data.frames with the "obs" and "pred
" columns of observed and predicted values.

  out_s_map <- s_map(time_series,lib=lib,pred=lib,E=E_star,stats_only = FALSE
)

  # Nonlinear dynamics are tested by comparing forecast-skill at theta=0 (glo
bal linear model) to the highest
  # forecast skill for theta>=0. Thus we reduce the "out_s_map" data.frame to
just the theta=0 row and minimum mae
  # row.

  out_s_map <- out_s_map %>%
    slice(1) %>%
    bind_rows(out_s_map %>% top_n(1,-mae))

  # The basic criterion of nonlinearity is that out_s_map$mae[1] - out_s_map$
mae[2] > 0. Thus we add this to the "out" data.frame.

    out$delta_mae <- out_s_map$mae[1] - out_s_map$mae[2]

  # However, a very small delta_mae could just result from random chance. Thu
s it is useful to also attach a p-value to this.
  # We can use the two-sample Wilcoxon test (more or less a non-parametric t-
test) on the distributions of forecast errors. We
  # first extract the forecast errors from out_s_map$model_output, which is a
2 element list where each element is a data.frame of
  # the prediction statistics for [1] theta=0 and [2] theta=theta_star (optim
al theta).
  # map ".$pred - .$obs"
    # NOTE: The test can be paired since the error values are ordered in time
and correspond to the same system state.

    p_val_s_map <- wilcox.test(
    out_s_map$model_output[[1]] %>% {abs(.$pred-.$obs)},
    out_s_map$model_output[[2]] %>% {abs(.$pred-.$obs)},
    paired=TRUE,alternative="greater",conf.int=TRUE
  )

#     p_val_s_map <- do.call(
#     wilcox.test,
#     c(map(out_s_map$model_output,~ abs(.$pred-.$obs)),
#       list(paired=TRUE,alternative="greater",conf.int=TRUE)
# ))

  # Finally, we add the  p-value into the "out" data.frame.


  out$p_val_s_map <- p_val_s_map$p.value

  return(out)


}
```

We then define a second wrapper function that performs the same analysis for all genes in a data block. This wrapper makes use of the package `parallel` to take advantage of multi-core processing, but can easily be rewritten without by replacing the `mclapply` call with standard `lapply`.

```
has_data <- function(ts,frac=.1){
  isGood <- sapply(ts,function(x) !is.finite(x) || isTRUE(all.equal(x,min(ts,
na.rm=TRUE))))
  return(mean( isGood ) < (1 -frac))
}

compute_univariate <- function(gene_block,lib,n.corr=1){

  require('parallel')

  gene_names <- setdiff(names(gene_block), c("dataset", "time"))

    test_stats <- function(output) {
      data.frame( has_data = TRUE,
                  stochastic = output$p_val_simplex > 0.05,
                  linear = output$delta_mae <= 0,
                  nonlinear_all = output$delta_mae > 0,
                    nonlinear_90 = output$p_val_s_map < 0.1,
                    nonlinear_95 = output$p_val_s_map < 0.05,
                    uEDM_stats = list(output))
    }

    null_stats <- function() {
      data.frame( has_data = FALSE,
                  stochastic = NA,
                  linear = NA,
                  nonlinear_all = NA,
                  nonlinear_90 = NA,
                  nonlinear_95 = NA,
                  uEDM_stats = list(NA))
    }

    uEDM_results <- mclapply(gene_names, function(gene) {
      if(has_data(gene_block[ , gene])){
        uEDM_out <- tryCatch(do_uEDM(gene_block[ , gene],lib=lib) %>% test_st
ats ,
                              error = function(e) null_stats() )
      }else{
        uEDM_out <- null_stats()
      }
      return(uEDM_out %>% mutate(gene = gene) %>% select(gene,everything()))

    }, mc.cores = n.corr)

    uEDM_results <- do.call(bind_rows,uEDM_results)
    return(uEDM_results)
}
```

Now we apply this wrapper to multiple divisions of the yeast and mouse data.

```
file_out_uEDM <- "./output/univariate_EDM_results.Rdata"

if(!file.exists(file_out_uEDM)){
  load("Data/gene_blocks_TPM_corrected.Rdata")
  load("Data/mouse_data_TMP.Rdata")

  lib_mouse_joined <- rbind(c(1,69),c(70,82),c(83,95))
  lib_mouse_fastonly <- c(1,69)

  n_WT1 <- NROW(gene_block_WT1)
  n_WT2 <- NROW(gene_block_WT2)
  gene_block_wt <- rbind(gene_block_WT1, gene_block_WT2)
  lib_WT_composite <- matrix(c(1, n_WT1, n_WT1 + 1, n_WT1 + n_WT2), ncol = 2,
byrow = TRUE)

  out_uEDM <- bind_rows(
    compute_univariate(gene_block_WT1,c(1,n_WT1),n.corr = 8) %>% mutate(data_
set = "yeast_WT1"),
    compute_univariate(gene_block_WT2,c(1,n_WT2),n.corr = 8) %>% mutate(data_
set = "yeast_WT2"),
    compute_univariate(gene_block_wt,lib_WT_composite,n.corr = 8) %>% mutate(
data_set = "yeast_WT_full"),
    compute_univariate(gene_block_mouse_joined,lib_mouse_joined,n.corr=8) %>%
mutate(data_set = "mouse_joined"),
    compute_univariate(gene_block_mouse_joined,lib_mouse_fastonly,n.corr=8) %
>% mutate(data_set = "mouse_fastonly")
  )

  save(out_uEDM,file=file_out_uEDM)
}
```

We can organize these results into a table with the information displayed in supplemental Figure S3.

```
load("./output/univariate_EDM_results.Rdata")

table_uEDM_totals <- out_uEDM %>%
  group_by(data_set) %>%
  filter(has_data) %>%
  mutate(predictable = !stochastic) %>%
  mutate_at(vars(c("linear",starts_with("nonlinear"))), funs((predictable)*(.
))) %>%
  summarise_at(vars(c("has_data","stochastic","predictable","linear",starts_w
ith("nonlinear"))),funs( sum(.,na.rm=TRUE))) %>%
  mutate(stat = "total") %>%
  select(data_set,stat,everything()) %>%
  mutate_at(vars(-(1:3)),funs(round(.,digits=0)))

table_uEDM_frac <- table_uEDM_totals %>%
  group_by(data_set) %>%
  mutate_at(vars(c("linear",starts_with("nonlinear"))), funs(./predictable))
%>%
  mutate_at(vars(c("stochastic","predictable")),
                funs( ./has_data ) )%>%
  mutate(stat = "fraction") %>%
  select(data_set,stat,everything()) %>%
  mutate_at(vars(-(1:3)),funs(round(.,digits=2)))


knitr::kable(bind_rows(table_uEDM_totals,table_uEDM_frac) %>% arrange(data_se
t,desc(stat)), digits=4)
```

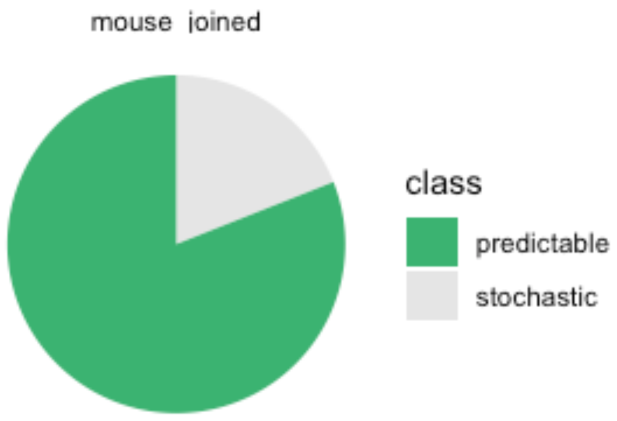
data_set	stat	has_data	stochastic	predictable	linear	nonlinear_all	nonlinear_90	nonlinear_95
mouse_fastonly	total	22509	7149.00	15360.00	3724.00	11636.00	3064.00	1730.00

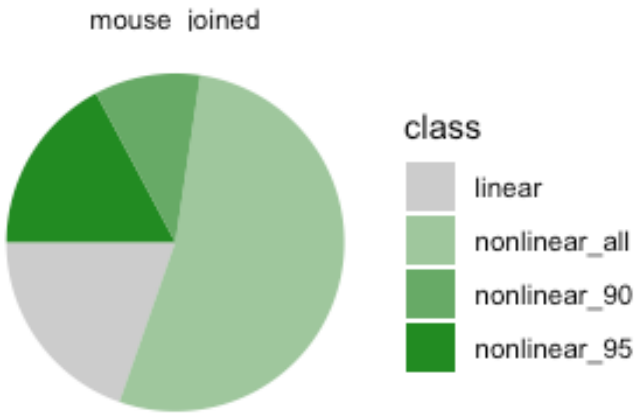
mouse_fastonly	fraction	22509	0.32	0.68	0.24	0.76	0.20	0.11
mouse_joined	total	22532	4281.00	18251.00	3585.00	14666.00	4994.00	3143.00
mouse_joined	fraction	22532	0.19	0.81	0.20	0.80	0.27	0.17
yeast_WT_full	total	6189	503.00	5686.00	899.00	4787.00	2006.00	1359.00
yeast_WT_full	fraction	6189	0.08	0.92	0.16	0.84	0.35	0.24
yeast_WT1	total	6189	949.00	5240.00	1225.00	4015.00	1241.00	750.00
yeast_WT1	fraction	6189	0.15	0.85	0.23	0.77	0.24	0.14
yeast_WT2	total	6161	875.00	5286.00	1310.00	3976.00	1214.00	694.00
yeast_WT2	fraction	6161	0.14	0.86	0.25	0.75	0.23	0.13

Then we plot the key fractions reported in the table above as pie-charts. These form the basis of Figure 2B.

```
plot_sets <- c("mouse_joined","yeast_wt_full")

df_uEDM_totals <- out_uEDM %>%
  group_by(data_set) %>%
  filter(has_data) %>%
  mutate(predictable = !stochastic) %>%
  mutate(nonlinear_all = nonlinear_all & !nonlinear_90) %>%
  mutate(nonlinear_90 = nonlinear_90 & !nonlinear_95) %>%
  mutate_at(vars(c("linear",starts_with("nonlinear"))), funs((predictable)*(.
))) %>%
  summarise_at(vars(c("has_data","stochastic","predictable","linear",starts_w
ith("nonlinear"))),funs( sum(.,na.rm=TRUE))) %>%
  mutate(stat = "total") %>%
  select(data_set,stat,everything())

g_pie1 <- df_uEDM_totals %>%
  filter(data_set %in% plot_sets) %>%
      mutate_at(vars(c("stochastic","predictable")),
               funs( ./has_data ) )%>%
  select(data_set,stochastic,predictable) %>%
  gather(key = "class",value="fraction",stochastic:predictable) %>%
  ggplot(aes(x=factor(1),y=fraction,fill=class)) + geom_bar(width = 1,stat="i
dentity") +
    coord_polar(theta="y") +
```

```r
    scale_fill_manual(values=c(stochastic="grey90",predictable="mediumseagree
n")) +
    theme_void() + facet_wrap(~data_set)

g_pie2 <- df_uEDM_totals %>%
  filter(data_set %in% plot_sets) %>%
  mutate_at(vars(c("linear",starts_with("nonlinear"))), funs(./predictable))
%>%
  select(data_set,linear,starts_with("nonlinear")) %>%
  gather(key = "class",value="fraction",-data_set) %>%
  mutate(class=factor(class,levels=c("linear","nonlinear_all","nonlinear_90",
"nonlinear_95"))) %>%
  ggplot(aes(x=factor(1),y=fraction,fill=class,alpha=class)) + geom_bar(width
= 1,stat="identity") +
    coord_polar(theta="y",star=-pi/2) +
    scale_fill_manual(values=c(linear="grey80",
                               nonlinear_all="forestgreen",
                               nonlinear_90="forestgreen",
                               nonlinear_95="forestgreen")) +
      scale_alpha_manual(values=c(linear=1,
                               nonlinear_all=.5,
                               nonlinear_90=.75,
                               nonlinear_95=1)) +
    theme_void() + facet_wrap(~data_set)

gridExtra::grid.arrange(gridExtra::arrangeGrob(g_pie1),gridExtra::arrangeGrob
(g_pie2),nrow=2)
```





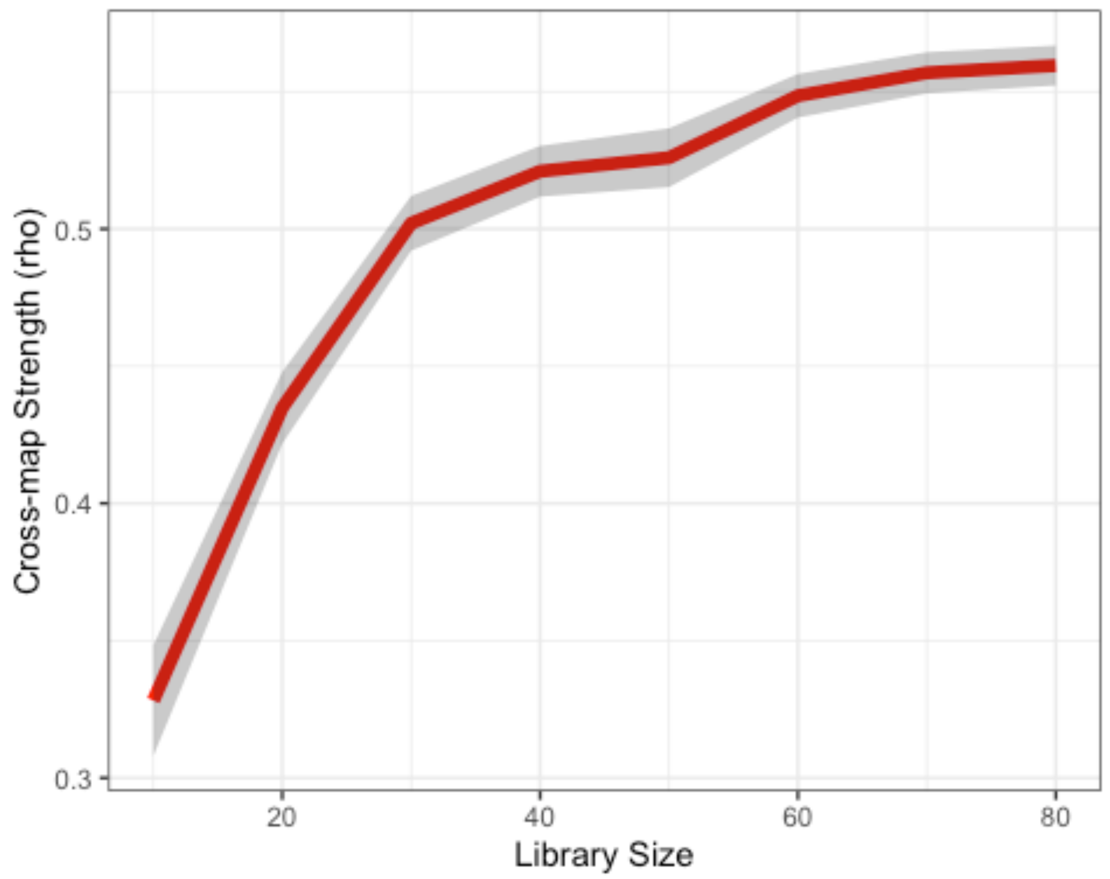
Specific Gene-Gene Analyses Yeast

Most of the figures and results relating to gene-gene interactions in the Main Text focus on the targets of the two experimental manipulations. This next section reproduces these results. First we define variables identifying the two target genes.

```
ExM1_target <- c(WHI5="YOR083W")
ExM2_target <- c(YHP1="YDR451C")
```

Before performing comprehensive analysis across all possible targets, we reproduce Main Text Figure 3 that shows the specific case of the gene-gene relationship between WHI5 and SWI4. The raw data reveal an absence of linear cross-correlation. However, lack of correlation does not indicate a lack of causation in nonlinear systems. Causation can instead be estimated by looking at ability to cross-map states from one manifold to another. The third panel of Figure 3 involves paired attractors with nearest neighbor time points selected on the SWI4 manifold and cross-identified on the WHI5 manifold.

```
genes <- c(SWI4="YER111C", WHI5="YOR083W")

n1 <- NROW(gene_block_WT1)
n2 <- NROW(gene_block_WT2)

lib_fig3c <- rbind(c(1, n1),
             n1 + c(1, n2))
block_fig3c <- rbind(gene_block_WT1[, genes],
               gene_block_WT2[, genes])

## Calculate full library hindcast cross-map skill (tp=-1) across E=1:8
ccm_tp_1 <- map_df(1:8, function(E) {
    ccm(block_fig3c, lib = lib_fig3c, tp = -1, E = E,
        lib_sizes = n1 + n2, random_libs = FALSE,
        lib_column = genes[1], target_column = genes[2],
        silent = TRUE)
})

## The best E should be E = 5
best_E <- ccm_tp_1$E[which.max(ccm_tp_1$rho)]

## Now perform instanteneous cross-map skill (tp=0) at best_E across library
sizes.
SWI4_xmap_WHI5 <- ccm(block_fig3c, lib = lib_fig3c, tp = 0, E = best_E,
                      lib_sizes = seq(10, 80, by = 10),
                      num_samples = 200,
                      lib_column = genes[1], target_column = genes[2],
                      silent = TRUE, RNGseed = 42)

g_SW4_xmap_WHI5 <- ggplot(SWI4_xmap_WHI5,
                  aes(x = lib_size, y = rho)) +
    labs(x = "Library Size", y = "Cross-map Strength (rho)") +
    stat_summary(fun.y = mean, geom = "line", col = "red", size = 2) +
    stat_summary(fun.data = mean_cl_normal, geom = "ribbon", alpha = 0.3) +
    theme_bw()

## Warning: `fun.y` is deprecated. Use `fun` instead.

print(g_SW4_xmap_WHI5)
```



We can repeat CCM analysis across the other WHI5 interactors explored in the early figures of the Main Text. First we write a function to compute the full-library CCM skill between all pairs of genes in a subnetwork.

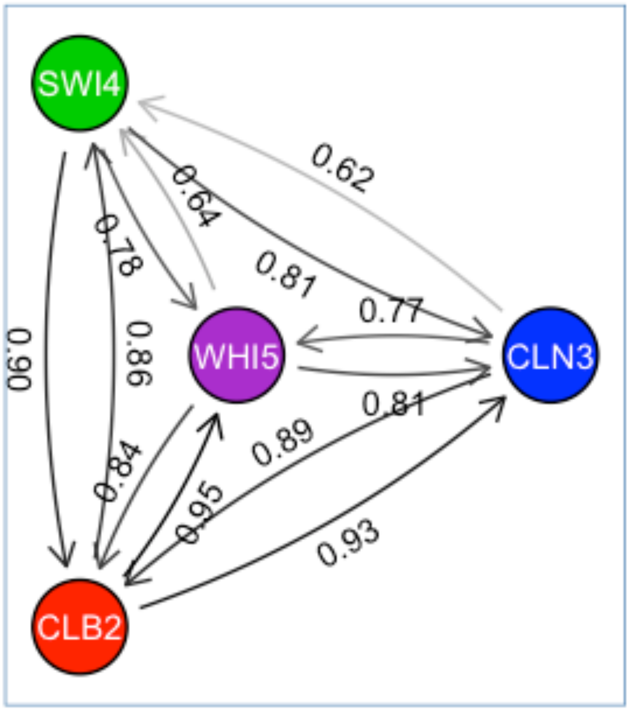
```r
compute_ccm_on_block <- function( block,
                                   lib = c(1,NROW(block)),
                                 max_E = 8,
                                 results_file = NULL)
{
    genes = names(block)

    params <- expand.grid(from_gene = genes, to_gene = genes) %>%
        filter(from_gene != to_gene)

    ccm_results <- map_df(1:NROW(params), function(i) {
        from_gene <- params$from_gene[i]
        to_gene <- params$to_gene[i]

        ccm_tp_1 <- map_df(1:max_E, function(E) {
            ccm(block, lib = lib, tp = -1, E = E,
                lib_sizes = NROW(block), random_libs = FALSE,
                lib_column = from_gene, target_column = to_gene,
                silent = TRUE)
        })
        best_E <- ccm_tp_1$E[which.max(ccm_tp_1$rho)]

        if(length(best_E) == 1) # compute CCM using tp = 0
        {
            ccm_tp_0 <- ccm(block, lib = lib, tp = 0, E = best_E,
                            lib_sizes = NROW(block), random_libs = FALSE,
                            lib_column = from_gene, target_column = to_gene,
                            silent = TRUE)
        } else { # if invalid, return NA results
            ccm_tp_0 <- data.frame(E = NA, tau = NA, tp = NA,
                                   num_neighbors = NA,
                                   lib_column = from_gene, target_column = to
_gene,
                                   lib_size = NA, num_pred = NA,
                                   rho = NA, mae = NA, rmse = NA)
        }

        return(ccm_tp_0)
    })

    if(!is.null(results_file))
    {
        save(ccm_results, file = results_file)
    }
    return(ccm_results)
}
```

We apply this to a four-gene subnetwork around WHI5.

```
genes_whi5_network <- c("WHI5"="YOR083W",
                        "CLN3"="YAL040C",
                        "SWI4"="YER111C",
                        "CLB2"="YPR119W")


  n1 <- NROW(gene_block_WT1)
  n2 <- NROW(gene_block_WT2)

  lib_WT_composite <- rbind(c(1, n1),
                 n1 + c(1, n2))

    block_whi5_network <- rbind(gene_block_WT1[, genes_whi5_network],
                   gene_block_WT2[, genes_whi5_network])

    names(block_whi5_network) <- names(genes_whi5_network)

out_whi5_network_xmap <- compute_ccm_on_block( block_whi5_network,
                               lib = lib_WT_composite,
                               max_E = 8,
                               results_file = NULL)
```

These are then plotted in ggraph. Note that the direction of arrows in the plot correspond to the causal direction, which is reciprocal to the cross-map calculation. That is, if Gene Y cross-maps states of Gene X, this indicates a causal influence of Gene X on Gene Y.

```
color_key_whi5_network <- list("CLN3" = "#0000FF", # "blue",
                               "SWI4"="#00CC00", # "green",
                               "WHI5"="#AA00CC", # purple",
                               "CLB2"="#FF0000") # "red")

g <- graph_from_data_frame(out_whi5_network_xmap[, c("target_column","lib_col
umn")])
E(g)$weight <- 0.5 + out_whi5_network_xmap$rho/2
E(g)$rho_val <- format(out_whi5_network_xmap$rho, digits = 2)
V(g)$color <- color_key_whi5_network[V(g)$name]

node_size <- 0.15
g_whi5_network <- ggraph(g, layout = "star", center = "WHI5") +
  geom_edge_fan(aes(label = rho_val, colour = weight),
              # geom_edge_fan(aes(label = rho_val, alpha = weight),
              show.legend = FALSE,
              angle_calc = "along",
              label_dodge = unit(-3, "mm"),
              arrow = arrow(length = unit(3, "mm")),
              start_cap = circle(node_size/1.5, "npc"),
              end_cap = circle(node_size/1.5, "snpc")) +
  geom_node_circle(aes(r = node_size, fill = color), show.legend = FALSE) +
  scale_edge_colour_gradient(low="#BBBBBB",high="#000000") +
  geom_node_text(aes(label = name), color = "white") +
  theme_graph(foreground = 'steelblue', fg_text_colour = 'white') +
  coord_fixed()


print(g_whi5_network)
```



Experimental Validation Analyses

Proceeding to the comprehensive analysis across all possible interactors with WHI5 and YHP1, we define a function to drop genes from analysis that do not have sufficient data. These are generally genes that have all 0 reads in the raw data.

```
is_bad_data <- function(ts) (sum(is.finite(ts)) < 2 ||
              var(ts, na.rm = TRUE) < 0.00001 ||
              var(ts[-1], na.rm = TRUE) < 0.00001)

is_bad_data_2 <- function(ts) ( !is.finite(var(ts)) || var(ts) < 0.00001 || v
ar(ts) < 0.00001 )
```

Define the function to compute the pairwise correlation between the manipulated target of a batch (WHI5 or YHP1) and all other targets.

```
compute_correlation_to_target_gene <- function(gene_block,
                                               target_gene)
{
    target_ts <- gene_block[, target_gene]

    gene_names <- setdiff(names(gene_block), c("dataset", "time"))
    data_block <- gene_block[, gene_names]
    corr_results <- data.frame(gene = gene_names,
                               metric = "corr_rho",
                               value = NA)

    for(i in seq_len(NCOL(data_block)))
    {
        ts <- data_block[, i]
          if(!is_bad_data(ts))
            corr_results$value[i] <- cor(ts, target_ts, use = "pairwise")
    }
    return(corr_results)
}
```

Next, define the function to compute the pairwise cross-map skill between the manipulated target of a batch (WHI5 or YHP1) and all other targets. We are interested in the casual relationships in wildtype expression, which means we have two realizations to work with, WT1 and WT2. For identifying causal genes, we would like the most accurate cross-map skill measurement from the densist (in terms of observations) possible attractor. As described in the Methods, since WT1 and WT2 should be separate realizations of the same underlying rules, we can concatenate these together to form a composite attractor with twice as many points. However, we are also interested in checking how robust our CCM measurements are. Thus we also run CCM calculations separately for each gene with just WT1 data and just WT2 data. In all cases, we use hindcast cross-map skill (at tp = -1) to select an optimal embedding dimension between 1 and 8, before measuring instanteneous cross-map skill (at tp = 0) at the optimal embedding dimension.

```
compute_ccm_to_target_gene <- function(gene_block_1,
                                       gene_block_2,
                                       target_gene,
                                       max_E = 8)
{
    gene_block_wt <- rbind(gene_block_1, gene_block_2)
    gene_names <- setdiff(names(gene_block_1), c("dataset", "time"))
    n1 <- NROW(gene_block_1)
    n2 <- NROW(gene_block_2)

    lib <- rbind(c(1, n1),
                 n1 + c(1, n2))
    columns <- lapply(1:max_E, function(E) {
        seq(from = 2, length.out = E)
    })

    ccm_results <- map_df(setdiff(gene_names, target_gene), function(gene) {
        block <- gene_block_wt[, c(target_gene, gene)]
        for(lag in 1:(max_E-1))
        {
            to_lag <- block[, NCOL(block)]
            block[, NCOL(block)+1] <- c(NA, to_lag[1:(n1-1)],
                                        NA, to_lag[n1 + (1:(n2-1))])
        }

        # determine best E
        ccm_tp_1 <- map(1:2,function(i){
          block_lnlp(block, lib = lib[i,], pred = lib[i,], tp = -1,
                              target_column = 1, columns = columns,
                              silent = TRUE)
          })

        best_E <- map(ccm_tp_1, ~which.max(.x$rho))
        # if any best_E elements invalid, return NA results
        if(reduce(map(best_E,~`!=`(length(.x),1)),`|`)){
            return(data.frame(lib_column = gene, target_column = target_gene,
                              rho = NA, mae = NA, rmse = NA,
                              E_1 = NA, rho_1 = NA, mae_1 = NA, rmse_1 = NA,
                              E_2 = NA, rho_2 = NA, mae_2 = NA, rmse_2 = NA))
        }

        ## compute CCM to each WT realization separately
        out <- map(1:2,function(i) {
          block_lnlp(block, lib = lib[i,], pred = lib[i,],tp = 0,
                     target_column = 1, columns = columns[[ best_E[[i]] ]],
                     silent = TRUE,stats_only = FALSE)
          })

        ccm_half <- map(out,~select(.x,-model_output))

        ccm_tp_0 <- invoke(compute_stats,
                           do.call(bind_rows,map(out,~.x$model_output[[1]]))
%>%
                             select(obs,pred) %>%
                             rename(observed=obs,predicted=pred))

        return(data.frame(lib_column = gene, target_column = target_gene,
                          rho = ccm_tp_0$rho, mae = ccm_tp_0$mae, rmse = ccm_
tp_0$rmse,
                          E_1=best_E[[1]], rho_1 = ccm_half[[1]]$rho,
                          mae_1 = ccm_half[[1]]$mae, rmse_1 = ccm_half[[1]]$r
mse,
                          E_2=best_E[[2]], rho_2 = ccm_half[[2]]$rho,
                          mae_2 = ccm_half[[2]]$mae, rmse_2 = ccm_half[[2]]$r
mse))
    })

    return(ccm_results)
}
```

Define the function to compute the difference in co-prediction between wildtype and experimentally manipulated dynamics. As described in the Main Text and Methods, to evaluate changes under the experimental manipulation in Batch 1 (wildtype control to experimental manipulation), we use the wildtype dynamics of Batch 2 as a reference, and vice versa. We do a traditional univariate lag-coordinate embedding of the data using separate `lib` and `pred` sets. The `lib` corresponds to the reference attractor, and the `pred` to either the control wildtype or experimental manipulation time series for the target batch. For each we select the maximum simplex projection forecast skill across embedding dimensions between 1 and 8. These two prediction skills are compared to compute `rho_diff`, a measurement of the change in dynamics under the experimental manipulation. The significance of the difference is assessed by two-sample Wilcoxon test on the mean absolute errors of observed and co-predicted values. For comparison, we also compute a simple change in levels by calculating the difference in median expression in the control wildtype and experimental manipulation time series, and assess the significance by a two-sample Wilcoxon test of the expression levels themselves.

```r
compute_copred <- function(gene_block_reference,
                           gene_block_control,
                           gene_block_experiment)
{
  # this function computes co-prediction using cross-batch library and predic
tion sets.
  n_reference <- NROW(gene_block_reference)
  n_control <- NROW(gene_block_control)
  n_experiment <- NROW(gene_block_experiment)

  lib_reference <- c(1, n_reference)
  lib_control <- c(1,n_control) + n_reference
  lib_experiment <- c(1, n_experiment) + n_reference + n_control

  gene_names <- setdiff(names(gene_block_reference), c("dataset", "time"))

  copred_results <- map_df(gene_names, function(gene) {

    ts <- c(gene_block_reference[,gene],gene_block_control[,gene],gene_block_
experiment[,gene])

    # check for bad time series
    if(is_bad_data_2(ts))
      return(data_frame(gene = gene,
                        E = NA,
                        simplex_wt = list(NA),
                        copred_mae_pval = NA,
                        copred_rmse_pval = NA,
                        copred_to_wt = list(NA),
                        copred_to_exp = list(NA)))


    # compute simplex for wildtype to wildtype
    copred_to_wt <- simplex(ts,
                           lib = lib_reference,
                           pred = lib_control,
                            E = 1:8,
                            silent = TRUE,
                            stats_only = FALSE) %>%
      filter(rank(-rho, ties.method="first")==1)

    # compute simplex for wildtype to mutant
    copred_to_exp <- simplex(ts,
                             lib = lib_reference,
                             pred = lib_experiment,
                             E = 1:8,
                             stats_only = FALSE) %>%
      filter(rank(-rho, ties.method="first")==1)

    copred_mae_pval <- wilcox.test(
      abs(copred_to_wt$model_output[[1]]$pred - copred_to_wt$model_output[[1]
]$obs),
      abs(copred_to_exp$model_output[[1]]$pred - copred_to_exp$model_output[[
1]]$obs),
      alternative="less"
    )$p.value


    lib2index <- function(lib) unlist(apply(as.matrix(lib),2,function(x) seq(
x[1],x[2])))

    distr_change <- wilcox.test(x = ts[lib2index(lib_control)],
                                y = ts[lib2index(lib_experiment)],
                                paired = FALSE, conf.int = TRUE)
        dmedian <- median(ts[lib2index(lib_control)])-median(ts[lib2index(lib
_experiment)])
    dmedian_pval <- distr_change$p.value

    return(data_frame(gene = gene, E = copred_to_exp$E[1],
                      simplex_wt = NA,
                      copred_mae_pval,
                      copred_to_wt = list(copred_to_wt),
                      copred_to_exp = list(copred_to_exp),
                      dmedian = dmedian,
                      dmedian_pval = dmedian_pval))
  })

  return(copred_results)
}
```

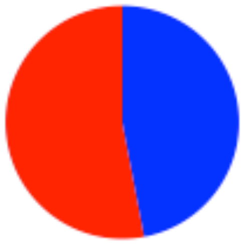
The final function definitions for the gene-gene analysis are to aggregate results of the correlation, cross-map, and co-prediction analyses and bin for comparison between high-CCM-low-corr and low-CCM-low-corr genes as in Main Text Figure 5.

```r
aggregate_results <- function(corr_results,
                              ccm_results,
                              copred_results)
{
  ccm_output <- data.frame(gene = as.character(ccm_results$lib_column),
                           ccm_results[, c("rho", "mae", "rmse")]) %>%
    tidyr::gather(metric, value, rho:rmse)
  ccm_output$metric <- paste0("ccm_", ccm_output$metric)

  output <- bind_rows(ccm_output, corr_results)
  output <- tidyr::spread(output, metric, value)

  copred_output <- copred_results %>%
    filter(is.finite(.$E)) %>%
    mutate(copred_rho_diff = map_dbl(copred_to_wt, ~.x$rho) -
             map_dbl(copred_to_exp, ~.x$rho),
           copred_rho_max =  pmax(map_dbl(copred_to_wt, ~.x$rho),
                                  map_dbl(copred_to_exp, ~.x$rho)),
           copred_rho_min = pmin(map_dbl(copred_to_wt, ~.x$rho),
                                 map_dbl(copred_to_exp, ~.x$rho)),
           simplex_rho = NA,
           copred_rho_wt = map_dbl(copred_to_wt, ~.x$rho),
           copred_rho_exp = map_dbl(copred_to_exp, ~.x$rho)) %>%
    select(-copred_to_wt,-copred_to_exp,-simplex_wt)

  output <- full_join(output,copred_output,by='gene')

  return(output)
}

bin_copred_topbot <- function(output,
                              thresh_corr=0.1,
                              min_copred_thresh=0.1,
                              max_copred_thresh = min_copred_thresh,
                              thresh_ccm_high = NULL,
                              thresh_ccm_low = NULL,
                              n_compare=100)
{
  output %>%
    select_if(~sum(!is.na(.)) > 0) %>%
    ungroup() %>%
    filter(complete.cases(.)) %>%
    filter(abs(corr_rho) <= thresh_corr) %>%
    filter( copred_rho_min > min_copred_thresh) %>%
    filter( copred_rho_max > max_copred_thresh) %>%
    mutate(bins=cut(rank(ccm_rho),
                    breaks=c(0,n_compare,n()-n_compare,n()))) %>%
    mutate(bins=factor(bins,labels=c("low_ccm","mid_ccm","high_ccm"))) %>%
    filter(as.numeric(bins) != 2)
}
```

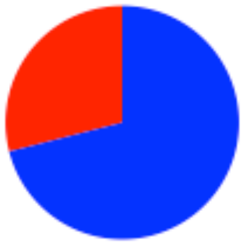
We apply these three analyses with WHI5 as the target gene (the manipulated gene of Batch 1), saving the results for further analysis.

```r
out_file <-  "./output/whi5_TPM_analysis.RData"
out_file_aggregate <- "output/whi5_TPM_deviation_calcs.Rdata"

if(!file.exists(out_file)){
  load("data/gene_blocks_TPM_corrected.Rdata")

  target_gene <- ExM1_target

  corr_results <- compute_correlation_to_target_gene(rbind(gene_block_WT1,
                                                           gene_block_WT2),
                                                     target_gene)

  ccm_results <- compute_ccm_to_target_gene(gene_block_WT1,
                                            gene_block_WT2,
                                            target_gene)

  copred_results <- compute_copred(gene_block_reference=gene_block_WT2,
                                   gene_block_control=gene_block_WT1,
                                   gene_block_experiment=gene_block_ExM1)

  save(corr_results, ccm_results, copred_results, file = out_file)
}

if(!file.exists(out_file_aggregate)){
  load(out_file)
  output <- aggregate_results(corr_results,
                              ccm_results,
                              copred_results)

  output_binned <- bin_copred_topbot(output,
                                     thresh_corr = 0.1,
                                     min_copred_thresh = -1)

  save(output, output_binned, file = out_file_aggregate)
}

exp1 <- new.env()
load("output/whi5_TPM_deviation_calcs.Rdata",envir = exp1)

plot_pie_charts(exp1$output_binned)
```

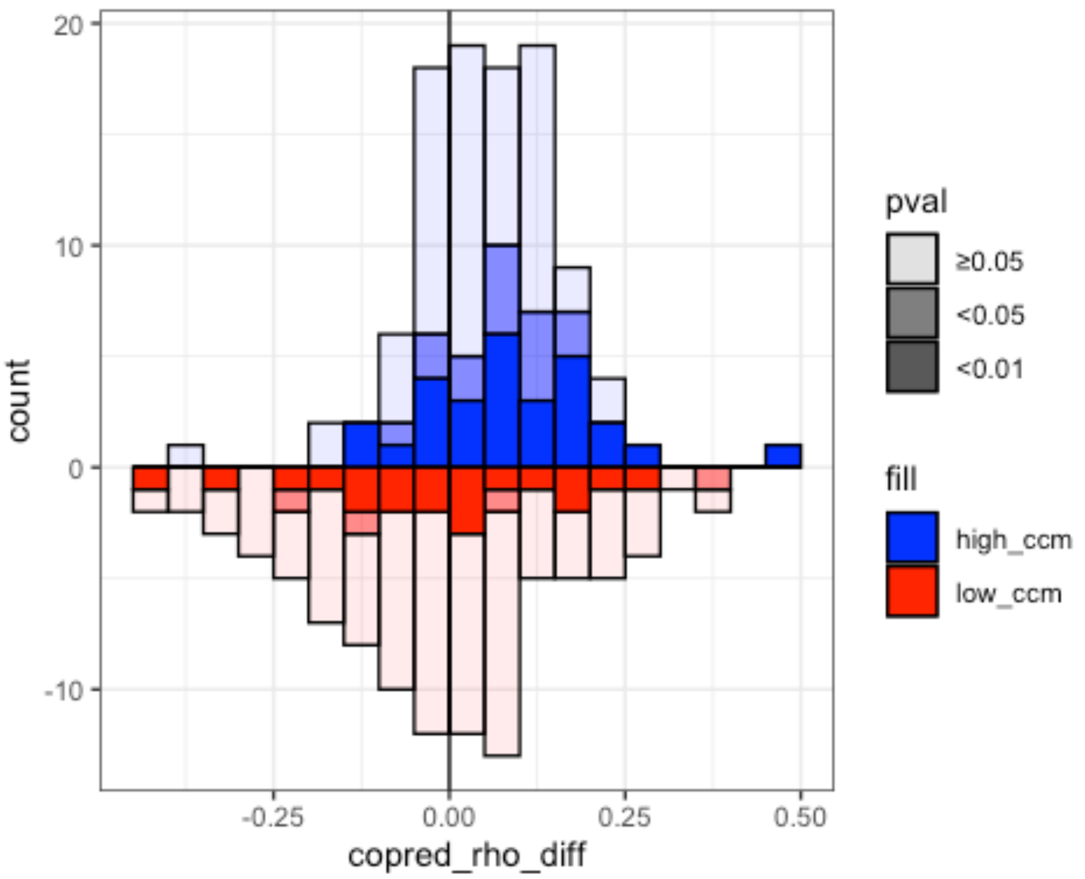
Low ccm, low corr
dynamic change
47%



High ccm, low corr
dynamic change
71%



```r
plot_copred_top_bottom_pval(exp1$output_binned,
                            x_var = "copred_rho_diff")
```



We can also compute a significance for this comparison, again using two-sample Wilcoxon test as the distributions have no reason to conform to something parametric.

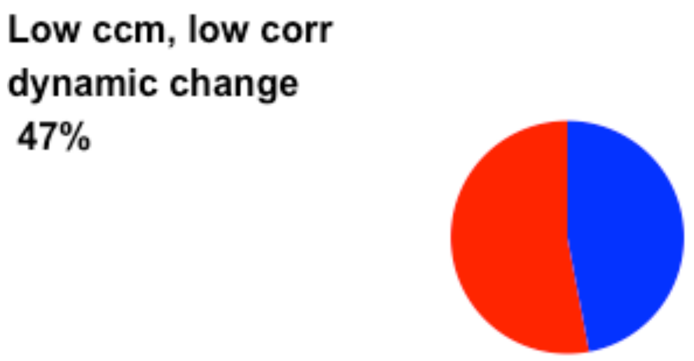
```
wilcox.test(exp1$output_binned$copred_rho_diff[exp1$output_binned$bins=="low_
ccm"],exp1$output_binned$copred_rho_diff[exp1$output_binned$bins=="high_ccm"]
,paired = FALSE)

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  exp1$output_binned$copred_rho_diff[exp1$output_binned$bins ==  and
exp1$output_binned$copred_rho_diff[exp1$output_binned$bins ==     "low_ccm"]
and     "high_ccm"]
## W = 3553, p-value = 0.0004088
## alternative hypothesis: true location shift is not equal to 0
```

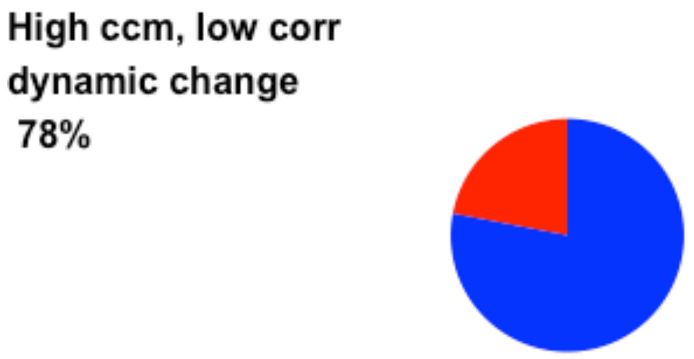
And we also apply the three analyses with YHP1 as the target gene.

```
out_file <-  "./output/yhp1_TPM_analysis.RData"
out_file_aggregate <- "output/yhp1_TPM_deviation_calcs.Rdata"

if(!file.exists(out_file)){
  load("data/gene_blocks_TPM_corrected.Rdata")
  target_gene <- ExM2_target

  corr_results <- compute_correlation_to_target_gene(rbind(gene_block_WT1,
                                                          gene_block_WT2),
                                                    target_gene)

  ccm_results <- compute_ccm_to_target_gene(gene_block_WT1,
                                            gene_block_WT2,
                                            target_gene)

  copred_results <- compute_copred(gene_block_ref=gene_block_WT1,
                                   gene_block_control=gene_block_WT2,
                                   gene_block_experiment=gene_block_ExM2)

  save(corr_results, ccm_results, copred_results, file = out_file)
}

if(!file.exists(out_file_aggregate)){
  load(out_file)
  output <- aggregate_results(corr_results,
                              ccm_results,
                              copred_results)

  output_binned <- bin_copred_topbot(output,
                                     thresh_corr = 0.1,
                                     min_copred_thresh = -1)

  save(output, output_binned, file = out_file_aggregate)
}

exp2 <- new.env()
load("output/yhp1_TPM_deviation_calcs.Rdata",envir = exp2)
```

After binning the 100 high CCM low corr and 100 low CCM low corr, we we reproduce the pie and bar charts of Main Text Figure 5.

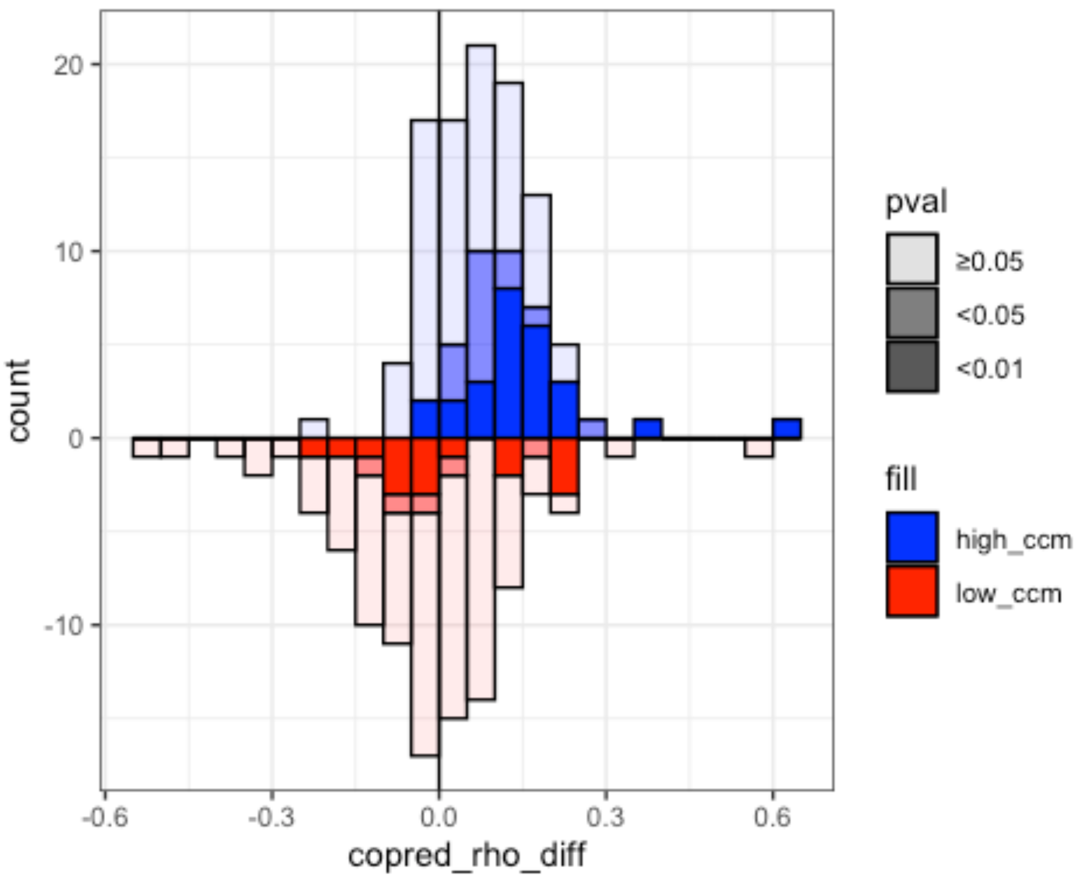
```
plot_pie_charts(exp2$output_binned)
```





```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[arrange]
## 2 2 (2-2,1-1) arrange gtable[arrange]

## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[arrange]
## 2 2 (2-2,1-1) arrange gtable[arrange]
```

```
plot_copred_top_bottom_pval(exp2$output_binned,
                       x_var = "copred_rho_diff")
```



As with WHI5, we also compute a significance for this comparison using two-sample Wilcoxon test.

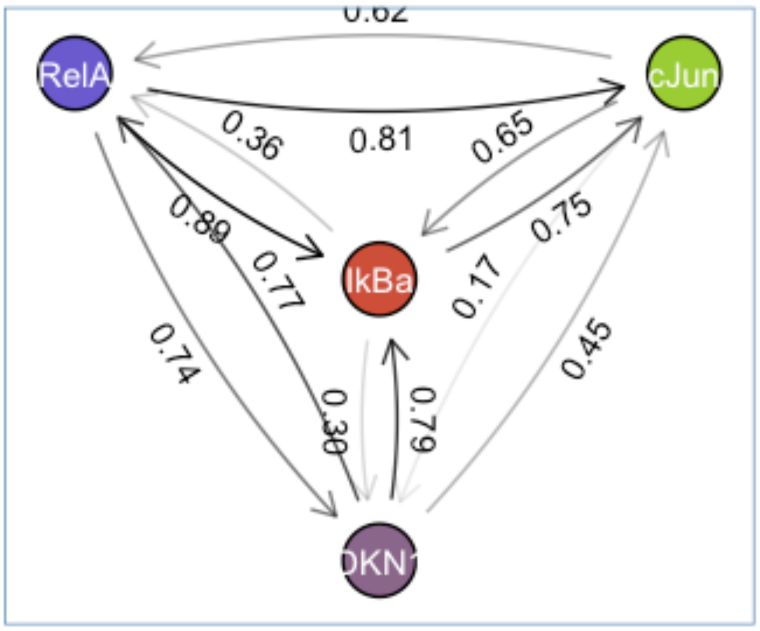
```
wilcox.test(exp2$output_binned$copred_rho_diff[exp2$output_binned$bins=="low_
ccm"],exp2$output_binned$copred_rho_diff[exp2$output_binned$bins=="high_ccm"]
,paired = FALSE)

##
##  Wilcoxon rank sum test with continuity correction
##
## data:  exp2$output_binned$copred_rho_diff[exp2$output_binned$bins ==  and
exp2$output_binned$copred_rho_diff[exp2$output_binned$bins ==     "low_ccm"]
and     "high_ccm"]
## W = 2887, p-value = 2.448e-07
## alternative hypothesis: true location shift is not equal to 0
```

Mouse Gene Network

```
genes_IkBa_network <- c(IkBa="MUSG021025.7",
                 cJun="MUSG052684.4",
                 RelA="MUSG024927.7",
                 CDKN1A="MUSG023067.10")
```

```r
block_IkBa <- gene_block_mouse_joined[,genes_IkBa_network]
names(block_IkBa) <- names(genes_IkBa_network)

lib_IkBa <- c(1,69)

out_IkBa_network_xmap <- compute_ccm_on_block(block_IkBa,lib=lib_IkBa,max_E =
6) %>%
  select(target_column,lib_column,rho) %>%
  # filter(rho > .4) %>%
  mutate_at(vars(ends_with("column")),as.character) %>%
  arrange(lib_column,target_column)

color_key_IkBa_network <- list("IkBa" = "tomato3",
                               # "ACTB" = "orangered",
                               "cJun" = "yellowgreen",
                               "RelA" = "slateblue",
                               "CDKN1A" = "plum4")

layout_key_IkBa_network <- list("IkBa" = c(1.25,2/sqrt(3)),
                                "RelA" = c(0,2),
                                "cJun" = c(2.5,2),
                                "CDKN1A" = c(1.25,0))

g_IkBa_network <- plot_net_from_rho(out_IkBa_network_xmap,color_key_IkBa_netw
ork,layout_key_IkBa_network)

print(g_IkBa_network + ggtitle("CCM causality") + theme(plot.title=element_te
xt(family="sans")))
```

CCM causality



0.62
RelA
cJun
0.36
0.81
0.65
0.89
0.77
IkBa
0.75
0.17
0.74
0.30
0.79
0.45
DKN1