Genetic Analysis of RNA-Seq Data

September 23, 2016

1 Table of Contents

19.3 Quality control on identified genes:
19.4 Identifying rhy-1 targets
19.5 Identifying egl-9 targets
19.6 Identifying vhl-1 targets
19.7 Identifying New Biology - understanding the role of rhy-1 and egl-9 in the hif-1 dependent response

2 Genetic Analysis of a Metazoan Pathway using Transcriptomic Phenotypes Supplementary and Extended Material

David Angeles-Albores, Carmie Puckett Robinson, Brian Williams, Igot Antoshechkin, and Paul W Sternberg

The purpose of this notebook is to serve as an extended, interactive document for our paper.

All code in this notebook was written, documented and generated by David Angeles-Albores. In cases when code was inspired, written or shown elsewhere first, links have been provided to the source material where possible.

100% of the data, with the exception of the raw reads, is deposited in our GitHub project. If you would like to verify our analysis, you are welcome to fork the project, although we will not allow pulls into the main branches. I hope you find this useful!

3 Folder Structure

The folder structure is a little bit important if you want to replicate the findings presented here.

Briefly, the folder structure is contained in the following major folders:

- input - contains raw FASTQ files, kallisto_all (processed reads), genmap file, TF list, cDNA file, transcripts.idx, enrichment dictionaries and hypoxia gold standard gene files
- sleuth - contains differential analysis (no *fog-2* included) results
- sleuth_batch_adjusted - contains differential analysis results with *fog-2* included
- src - all python scripts
- output - all figures
- tex - manuscript
- experimental_docs - all bioanalyzer results are placed here

4 Read alignment and differential expression analysis

The raw FASTQ reads are in input/rawseq/PROJECT_NAME/. These reads are processed by running **kallisto_bash_generator.py**, then from terminal (in the main directory) *chmod +x kallisto commands.sh; sh kallisto_commands.sh*. Reads were processed using a length of **180**bp, with a standard deviation of **60** basepairs, bootstrapped 200 times. The results from this analysis are then placed in input/kallisto_all/PROJECT_NAME/.

Sleuth analysis was performed by running **diff_exp_analyzer.R** in the **sleuth_adjusted_all** folder. This will perform the differential expression analysis. The results are stored within the *sleuth_adjusted/results/* folder.

4.1 Details of Differential Analysis

We used a Generalized Linear Model to simultaneously estimate batch effects and the effects of varying mutations relative to a wild-type genome. Therefore, the model we used can be written:

$$\log(y_{i,j,k}) = \beta_{0,i} + \beta_{\text{batch},i} * Y_{k,i} + \beta_{\text{genotype},i} * X_{j,i} \quad (1)$$

Where $Y_{k,i}$ and $X_{j,i}$ are indicator variables for batch and genotype respectively.

5 Introduction: Genetic Analysis Using Global Expression Measurements

The following sections will provide (excruciating) detail on how we performed the genetic analysis of our mutants using this data. Initially, this analysis was done blindly. For clarity, I have added all the genotype identifiers from the beginning.

To start, we should load all the python libraries that we will need:

```
In [1]: # important stuff:
        import os
        import pandas as pd
        import numpy as np
        from IPython.core.display import HTML

        # stats
        import sklearn.decomposition
        from scipy import stats as sts
        from sklearn.cluster import AgglomerativeClustering
        from scipy.cluster import hierarchy
        import statsmodels.api as sm

        # TEA and morgan
        import tissue_enrichment_analysis as tea
        import morgan as morgan

        # network graphics
        import networkx as nx

        # Graphics
        import matplotlib as mpl
        import matplotlib.ticker as plticker
        import matplotlib.pyplot as plt
        import seaborn as sns
        import matplotlib.patheffects as path_effects
        from matplotlib import rc
        rc('text', usetex=True)

        # bokeh
        import bokeh.charts
```

```
import bokeh.charts.utils
import bokeh.io
import bokeh.models
import bokeh.palettes
import bokeh.plotting
from bokeh.plotting import figure
from bokeh.resources import CDN
from bokeh.embed import file_html


# bayes and mcmc
import pymc3 as pm
import theano

# Display graphics in this notebook
bokeh.io.output_notebook()

# Magic function to make matplotlib inline;
# other style specs must come AFTER
%matplotlib inline

# This enables SVG graphics inline.
# There is a bug, so uncomment if it works.
%config InlineBackend.figure_formats = {'png', 'retina'}

# JB's favorite Seaborn settings for notebooks
rc = {'lines.linewidth': 2,
      'axes.labelsize': 18,
      'axes.titlesize': 18,
      'axes.facecolor': 'DFDFE5'}
sns.set_context('notebook', rc=rc)
sns.set_style("dark")

ft = 35 #title fontsize
import genpy
```

Next, I need to load the **phenotype_df** (please don't use this file without permission for publication purposes. This tool is unpublished [I am actively working on a paper with this] and will be available soon. I provide it here for clarity, even though it was not part of our paper), the **tissue_df** and the **tf_df**. These files contains a phenotype-gene dictionary, a tissue-gene dictionary (downloaded from TEA) and a transcription factor list (from Chris Grove).

Additionally, I will initialize several variables whose purpose will mainly be to prettify the tables and substitute in the correct names.

```
In [36]: fname = '../input/dictionaries/phenotype_dictionary.csv'
         phenotype_df = pd.read_csv(fname)
         tissue_df = tea.fetch_dictionary()
         tf_df = pd.read_csv('../input/tf_list.csv')
```

```
hypoxia_gold = pd.read_csv('../input/hypoxia_gold_standard.csv',
                           sep=',')

genotype_mapping = {'a': r'\emph{egl-9;vhl-1}',
                    'f': r'\emph{egl-9;hif-1}',
                    'b': r'\emph{egl-9}',
                    'c': r'\emph{hif-1}',
                    'd': r'\emph{vhl-1}',
                    'e': r'\emph{rhy-1}',
                    'g': r'\emph{fog-2}'
                    }

sort_pairs = {'eb': 1, 'be': 1,
              'ed': 2,'de': 2,
              'ec': 3,'ce': 3,
              'eg': 4,'ge': 4,
              'bd': 5,'db': 5,
              'cb': 6,'bc': 6,
              'bg': 7, 'gb': 7,
              'cd': 8,'dc': 8,
              'dg': 9,'gd': 9,
              'cg': 10,'gc': 10
             }

decode_pairs = {'eb': '\emph{rhy-1}, \emph{egl-9}',
                'be': '\emph{rhy-1}, \emph{egl-9}',
                'ed': '\emph{rhy-1}, \emph{vhl-1}',
                'de': '\emph{rhy-1}, \emph{vhl-1}',
                'ec': '\emph{rhy-1}, \emph{hif-1}',
                'ce': '\emph{rhy-1}, \emph{hif-1}',
                'eg': '\emph{rhy-1}, \emph{fog-2}',
                'ge': '\emph{rhy-1}, \emph{fog-2}',
                'bd': '\emph{egl-9}, \emph{vhl-1}',
                'db': '\emph{egl-9}, \emph{vhl-1}',
                'cb': '\emph{egl-9}, \emph{hif-1}',
                'bc': '\emph{egl-9}, \emph{hif-1}',
                'bg': '\emph{egl-9}, \emph{fog-2}',
                'gb': '\emph{egl-9}, \emph{fog-2}',
                'cd': '\emph{vhl-1}, \emph{hif-1}',
                'dc': '\emph{vhl-1}, \emph{hif-1}',
                'dg': '\emph{vhl-1}, \emph{fog-2}',
                'gd': '\emph{vhl-1}, \emph{fog-2}',
                'cg': '\emph{hif-1}, \emph{fog-2}',
                'gc': '\emph{hif-1}, \emph{fog-2}'
               }
```

Next, we will load up all of our files. These files will be placed within a class called morgan.hunt (funny!). All of the classes that are referenced in this tutorial are in the file morgan.py,

which you are welcome to use for your own analysis. The classes are relatively well-documented and should be usable if you are careful. That said, I am not a computer scientist, so some pathologies or bugs may pop up – if they do, please perform a pull-request on our github with a fix for the bug. Alternatively, please email me a sufficiently detailed script so I can reconstruct the failure event and correct the bug.

In [3]:
```
# Specify the genotypes to refer to:
single_mutants = ['b', 'c', 'd', 'e', 'g']
# Specify which genotypes are double mutants
# and of what single mutants:
double_mutants = {'a' : 'bd', 'f':'bc'}

# initialize the morgan.hunt object:
# target_id is the column with isoform specific names
# b is the name of the column with the GLM regression coefficients
# tpm is the name of the column with the TPM numbers
# qval is the name of the column with the FDR corrected q-values
thomas = morgan.hunt('target_id', 'b', 'tpm', 'qval')

# input the genmap file:
thomas.add_genmap('../input/library_genotype_mapping.txt', comment='#')

# add the names of the single mutants
thomas.add_single_mutant(single_mutants)

# add the names of the double mutants
thomas.add_double_mutants(['a', 'f'], ['bd', 'bc'])

# set the q-value threshold for significance to its default value, 0.1
thomas.set_qval()

# Add the tpm files:
kallisto_loc = '../input/kallisto_all/'
thomas.add_tpm(kallisto_loc, '/kallisto/abundance.tsv', '')

# Make all possible combinations of WT, X
combs = {}
for gene in thomas.genmap.genotype.unique():
    if gene != 'wt':
        combs[gene] = 'WT_'+gene+'/'

# # load all the beta values for each genotype:
# sleuth_loc = '../sleuth/'
# thomas.add_betas(sleuth_loc, 'betas.csv', combs)

# load all the beta values for each genotype:
sleuth_loc = '../sleuth_all_adjusted/kallisto/'
for file in os.listdir("../sleuth_all_adjusted/kallisto"):
```

```
        if file[:4] == 'beta':
            letter = file[-5:-4].lower()
            thomas.add_beta(sleuth_loc + file, letter)
            thomas.beta[letter].sort_values('target_id', inplace=True)
            thomas.beta[letter].reset_index(inplace=True)
```

Next, I will filter the data, removing any genes that don't show up in all the files and removing the bottom 10% of the genes by expression level. This is an aggressive cutoff, I know.

```
In [4]: thomas.filter_data(0, 0.1)

Number of na genes: 232
```

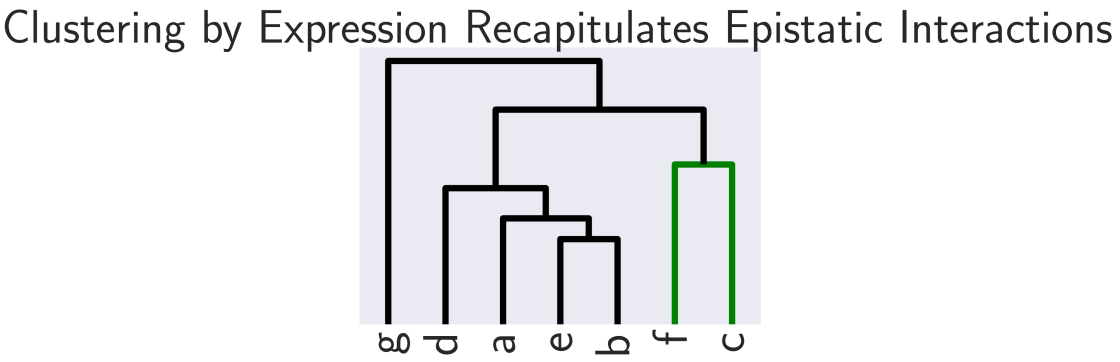
6 Figure 1. Dendrogram Clustering

Having performed the above analysis, let's go ahead and make the (almost finished) dendrogram figure in the paper. The only post-processing this will undergo is manually changing the letters to gene names and removing some of the labels for clarity.

```
In [5]: # find the set of all genes that are DE in any category
        max_overlap = np.array([])
        for df in thomas.beta_filtered.values():
            ind = df.qval < thomas.q
            if len(max_overlap) == 0:
                max_overlap = df[ind].target_id.values
            else:
                max_overlap = np.concatenate((max_overlap,
                                df[ind].target_id.values))

        max_overlap = max_overlap.tolist()
        max_overlap = list(set(max_overlap))
        print(len(max_overlap))

        bvals = np.array([])
        for df in thomas.beta_filtered.values():
            temp = df[df.target_id.isin(max_overlap)].b.values
            temp = (temp - temp.mean())/temp.std()
            if len(bvals) == 0:
                bvals = temp
            else:
                bvals = np.vstack((bvals, temp))

        # Perform agglomerative clustering
        sklearn_pca, n = genpy.pca(bvals)
        model = hierarchy.linkage(sklearn_pca.transform(bvals), 'ward')
        # extract the labels:
        labels = list(thomas.beta_filtered.keys())
```

```
        # rename the double mutant labels (i.e. instead of 'a' --> 'bd')
        for i, label in enumerate(labels):
            if label in double_mutants:
                labels[i] = double_mutants[label]

        # Linewidth parameter, temporarily set to 7
        plt.rcParams['lines.linewidth'] = 7

        # set colors (black; blue)
        hierarchy.set_link_color_palette(['k', 'g', 'b'])

        # draw the dendrogram
        hierarchy.dendrogram(
            model,
            truncate_mode='level',  # show only the last p merged clusters
            labels=list(thomas.beta_filtered.keys()),
            p=21,  # show only the last p merged clusters
            show_leaf_counts=False,  # otherwise numbers in brackets are counts
            leaf_rotation=90.,
            leaf_font_size=12.,
            show_contracted=True,
            above_threshold_color='k'
        )

        # get the current axis
        ax = plt.gca()

        # add in the orange and green boxes
        height = ax.get_ylim()[1]*.36

        plt.xticks(fontsize=55)
        plt.gca().yaxis.set_major_locator(plt.NullLocator())

        title = 'Clustering by Expression Recapitulates Epistatic Interactions'
        plt.title(title, fontsize=ft*1.5)
        plt.savefig('../output/tpm_dendrogram.pdf', bbox_inches='tight')

        # return linewidth to a reasonable setting
        plt.rcParams['lines.linewidth'] = 2
```

```
4674
The first 2 principal components explain >=90% of the data
```



7 Bayesian versus Spearman Regressions

Ok! Having verified that we still have plenty of power to identify these mutants, we need to go ahead and perform some statistics. Next, I will define some useful functions to make plotting easier. After that, I will begin a detailed analysis of the different prediction methods, specifically attempting to compare Spearman rank regression with a Bayesian linear regression of ranked data.

```
In [51]: def pathify(title, xlabel, ylabel, xticks=True, yticks=True, **kwargs):
             """
             A function to pathify the labels, titles and ticks in a plot.
             """
             labelsize = kwargs.pop('labelsize', 20)
             titlesize = kwargs.pop('titlesize', 25)

             # make the labels and title into paths
             effect = [path_effects.Normal()]
             plt.ylabel(ylabel,
                        fontsize=labelsize).set_path_effects(effect)
             plt.xlabel(xlabel,
                        fontsize=labelsize).set_path_effects(effect)
             plt.title(title,
                       fontsize=titlesize).set_path_effects(effect)

             ax = plt.gca()
             # go through each xtick or ytick and make
             # it a path if user specified to do so.
             if xticks == True:
                 for i, label in enumerate(ax.get_xticklabels()):
                     ax.get_xticklabels()[i].set_path_effects(effect)
             if yticks == True:
                 for i, label in enumerate(ax.get_yticklabels()):
                     ax.get_yticklabels()[i].set_path_effects(effect)
```

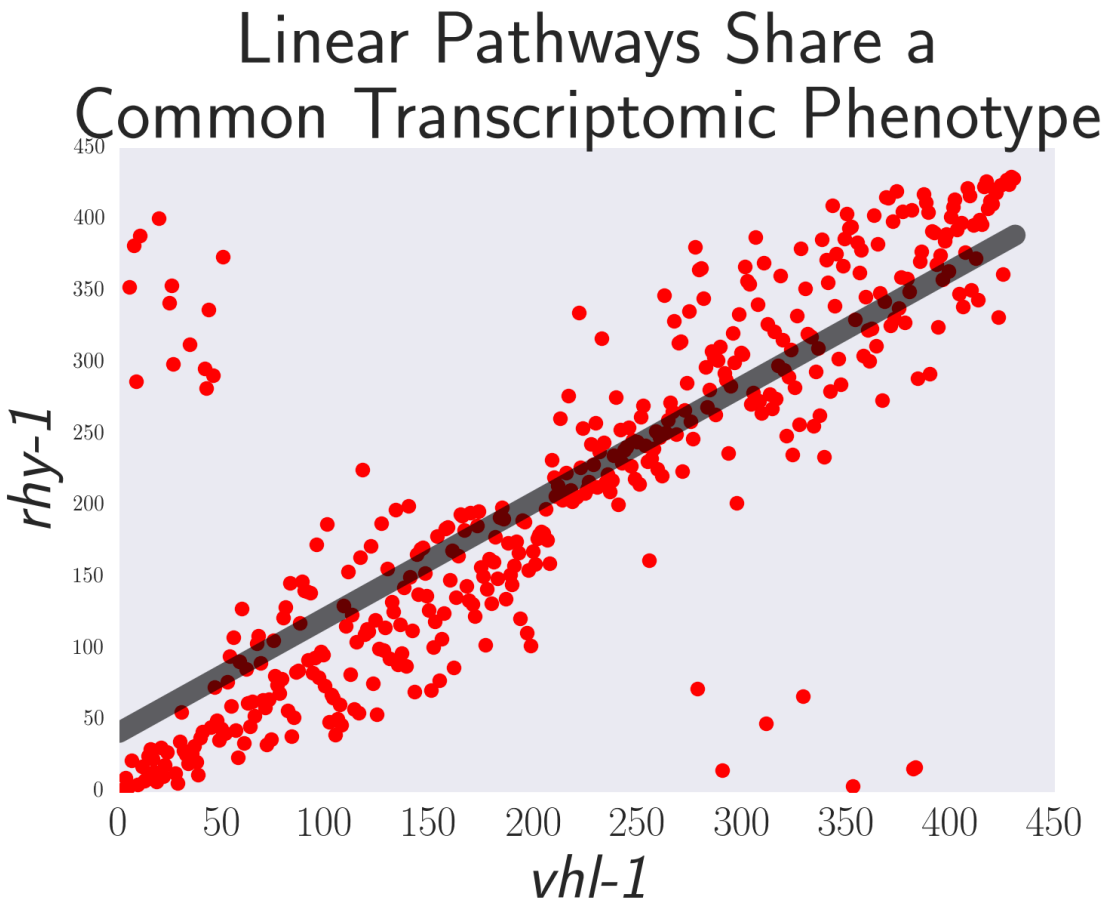
7.1 Spearman Regression Method

In order to perform the Spearman regression, this is what I will do:

- Extract the pertinent datasets for the two genotypes I want to convert.
- Find the genes that are altered between both conditions, regardless of direction
- After finding this overlap, rank the genes by their β coefficient in each dataset
- Perform Least Squares on the ranked data, which is the Spearman Regression
- Plot the results

```
In [7]: # lambda index function:
        lind = lambda x: (x.qval < 0.1)

        # As an example, let me show you what a
        # good spearman correlation looks like:
        # name of the column that contains the
        # isoform names:
        genes = 'target_id'

        # the genotypes to compare
        letters = ['d', 'e']

        # extract the dataframes from the morgan.hunt object
        x = thomas.beta_filtered[letters[0]]
        y = thomas.beta_filtered[letters[1]]

        # boolean logic to find the stat. sig. diff.
        # genes that appear in both x and y
        ovx = x[lind(x)]
        ovy = y[lind(y) & y[genes].isin(ovx[genes])].copy()
        ovx = x[lind(x) & x[genes].isin(ovy[genes])].copy()

        # a function to rank order the data
        def find_rank(df):
            """A function to find the rank values of a variable."""
            # make a copy of the dataframe, then sort it inplace
            d = df.copy()
            d.sort_values('b', inplace=True)
            # make a rank vector and append it to the sorted dataframe
            rank = np.linspace(0, len(d)-1, len(d))
            d['r'] = rank
            # sort by isoform name again and return the modified df
            d.sort_values('target_id', inplace=True)
            return d

        # apply said function
        ovx = find_rank(ovx)
        ovy = find_rank(ovy)
```

```
# calculate a linear regression on ranked data,
# which is equivalent to Spearman Ranked Regression
slope, intercept, r_value, p_value, std_err = sts.linregress(ovx.r,ovy.r)

# We make two vectors in order to draw the best fit line on a plot
X = np.linspace(0, len(ovx.r))
Y = slope*X + intercept

# Plot the genes that are significantly altered in both X and Y
plt.plot(ovx.r, ovy.r, 'ro',
         alpha=1, label='Overlapped Diff. Exp. Genes')

# Plot the best fit line
plt.plot(X, Y, 'k-', alpha=0.6,
         lw=10, label= 'Best Fit')

# prettify the plot
plt.title('Linear Pathways Share a\nCommon Transcriptomic Phenotype',
          fontsize=ft).\
            set_path_effects([path_effects.Normal()])
plt.xlabel(genotype_mapping[letters[0]],
           fontsize=30).set_path_effects([path_effects.Normal()])
plt.ylabel(genotype_mapping[letters[1]],
           fontsize=30).set_path_effects([path_effects.Normal()])
plt.xticks(fontsize=20)
plt.savefig('../output/spearmanr_b_and_c.pdf')
```



Spearman R predicts a trend, but the best fit we found doesn't agree with the one I would predict with my eye. We can probably do better if we use a Bayesian regression that minimizes least squares but using a Student-T distribution, NOT a Gaussian.

7.2 Bayesian Robust Regression

In order to perform the Spearman regression, this is what I will do:

- Extract the pertinent datasets for the two genotypes I want to convert.
- Find the genes that are altered between both conditions, regardless of direction
- After finding this overlap, rank the genes by their β coefficient in each dataset
- Perform Least Squares on the ranked data, but use a Student-T instead of a Gaussian prior.
- The regression in this case is sampled using a full Monte Carlo Simulation.
- Identify outliers to this regression, and run a second regression on outliers to see if there are complex regulatory relationships between these genes.
- Plot the results

In other words, given ranked data, I will find the line that is likeliest to explain the data by finding:

$$P(D|\mu,\sigma) \propto \prod_i \text{StudentT}(D_i(x) - \mu(x), \sigma) \tag{2}$$

The Student-T distribution has considerably heavier tails than a Gaussian distribution, so it will not consider the evidence provided by outliers as informative as a Gaussian would.

In order to perform this simulation, we will use the **pymc3** package to specify the model. This model was inspired and successfully deployed thanks to Thomas Wiecki; specifically thanks to this blog entry by him. Thanks Tom!

7.3 Figure 4. Complex Regulation Generates Detectable Patterns in Transcriptomes

The code that we will run below was used to generate the bottom panel of figure 4 in our paper. The figure as is output here was only subjected to very minor aesthetic modifications post-generation (such as moving the title a little above where it appears here).

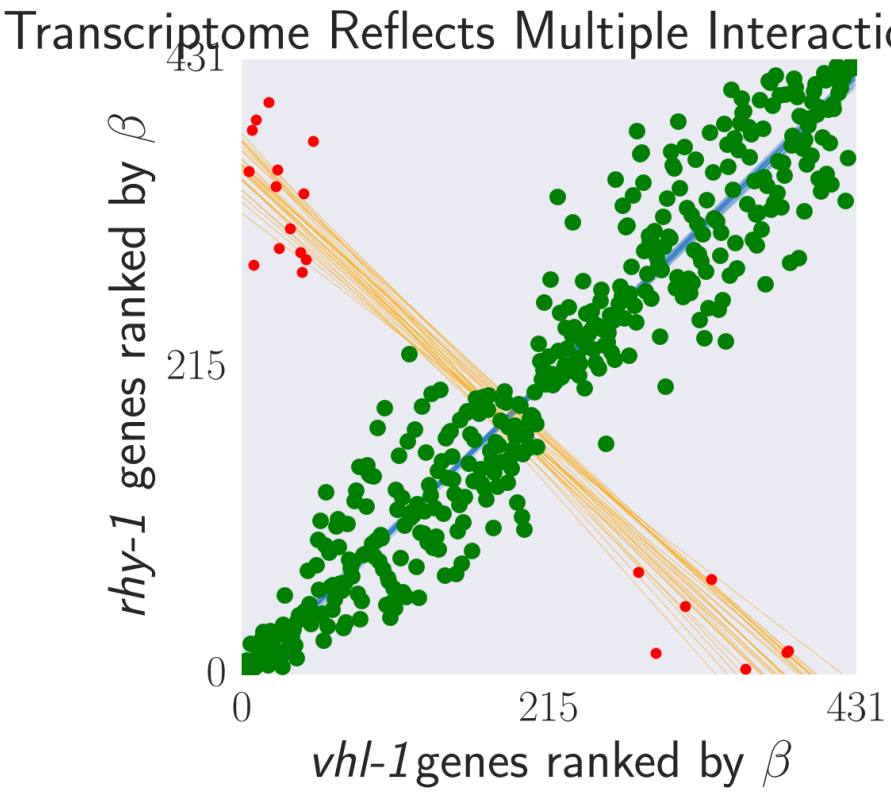
```
In [8]: def robust_regress(data):
            """A robust regression using a StudentT distribution instead of a Gauss
            with pm.Model() as model_robust:
                # set the model. pymc is nice because
                # it will automatically choose
                # appropriate priors for us once we
                # specify our likelihood is StudentT
                family = pm.glm.families.StudentT()
                # specify we want a generalized linear
                # model with a Student T distribution
                pm.glm.glm('y ~ x', data, family=family)
                # find the MAP as a good starting point
                start = pm.find_MAP()
                # do the simulation and return the results
                step = pm.NUTS(scaling=start)
                trace_robust = pm.sample(2000, step, progressbar=True)
                return trace_robust
```

```
In [9]: # Take the data from the Spearman example and put it
        # into a dictionary to feed into the robust regression
        data = dict(x=ovx.r, y=ovy.r)

        x = np.linspace(ovx.r.min(), ovx.r.max())

        # perform the simulation
        trace_robust = robust_regress(data)

        # draw a figure
        plt.figure(figsize=(5, 5))

        # some statistics.
        # normalize everything so that all points are centered around 0
        # by taking the y-coordinates and subtracting
```

```
# the value of the model at the point we calculated
intercept = trace_robust.Intercept.mean()
slope = trace_robust.x.mean()
distribution = ovy.r - intercept - ovx.r*slope
# find the mean and stdev of the distribution
# (even though mean should be 0 now)
mean = distribution.mean()
std = distribution.std()

# find inliers and outliers (see text description below)
def find_inliers(distribution, mean, trace):
    """A function to identify inliers and outliers in a distribution"""
    # find the outliers:
    sel = np.abs(distribution - mean)/(trace_robust.x.std()
                + trace_robust.Intercept.std()+ std) < 1.5

    # get the outliers and inliers
    distribution_inliers = distribution[sel]
    distribution_outliers = distribution[~sel]

    # get the gene names of the outliers
    inverse = distribution_outliers + \
                          intercept + ovx.r*slope
    outliers = ovy[ovy.r.isin(inverse)].target_id

    return distribution_inliers, distribution_outliers, outliers

# call the function
results = find_inliers(distribution, mean, trace_robust)
distribution_inliers, distribution_outliers, outliers = results
# run a secondary regression on the outliers
data2 = dict(x=ovx[ovx.target_id.isin(outliers)].r,
             y=ovy[ovy.target_id.isin(outliers)].r)

# run the second trace
trace_robust2 = robust_regress(data2)

# get the y-coordinate of the inliers and outliers
yri = distribution_inliers + intercept + ovx.r*slope
yro = distribution_outliers + intercept + ovx.r*slope

# plot the regression lines
label = 'posterior predictive regression lines'
pm.glm.plot_posterior_predictive(trace_robust, eval=x,
                                 label=label,
                                 color='#357EC7')
pm.glm.plot_posterior_predictive(trace_robust2, eval=x,
                                 label=label,
```

```
                                        color='#FFA500')

        # plot the data. Inliers are plotted as
        #large green dots, outliers as small red dots
        plt.plot(ovx.r, yri, 'go', ms = 7.5, alpha=0.7)
        plt.plot(ovx[yro > 0].r, yro[yro > 0], 'ro', ms = 5)

        # prettify plot
        plt.xlim(0, len(ovx))
        plt.ylim(0, len(ovy))
        plt.yticks([0, np.floor(len(ovx)/2), len(ovx)], fontsize=20)
        plt.xticks([0, np.floor(len(ovx)/2), len(ovx)], fontsize=20)
        pathify('Transcriptome Reflects Multiple Interaction Modes',
                genotype_mapping[letters[0]] + r'genes ranked by $\beta$',
                genotype_mapping[letters[1]] + r' genes ranked by $\beta$',
                labelsize=24)
        comp = letters[0] + letters[1]
        plt.savefig('../output/multiple_modes_{0}.pdf'.format(comp)
```

```
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.5 secApplied l
 [-----------------100%-----------------] 2000 of 2000 complete in 2.0 sec
```



There, much better! Penalizing outliers allows us to estimate the primary regression slope really well (blue lines). Here, each blue line is the result of a simulation, and we can see that they are all clustered together. Having estimated the first regression, we can proceed to identify outliers.

Now, there are multiple ways to identify outliers, and this is the point where I am going to cheat just a little bit. If we pretend that the model is gaussian (it isn't), then we can identify outliers (fairly aggressively) via the following method. Thus, if

$$z(i) = \frac{D_i(x) - \mu(x)}{\sigma_\mu + \sigma_{\text{Intercept}} + \sigma_{\text{Data}}} > 1 \tag{3}$$

then $z(i)$ is an outlier.

After identifying these outliers, I can then pool them and run a second regression on them (orange lines). These lines are also quite clustered, and the slope is practically 1! Wow!

7.4 Figure 2. Positive Regulatory Relationships Can Be Identified By Transcriptomic Correlation

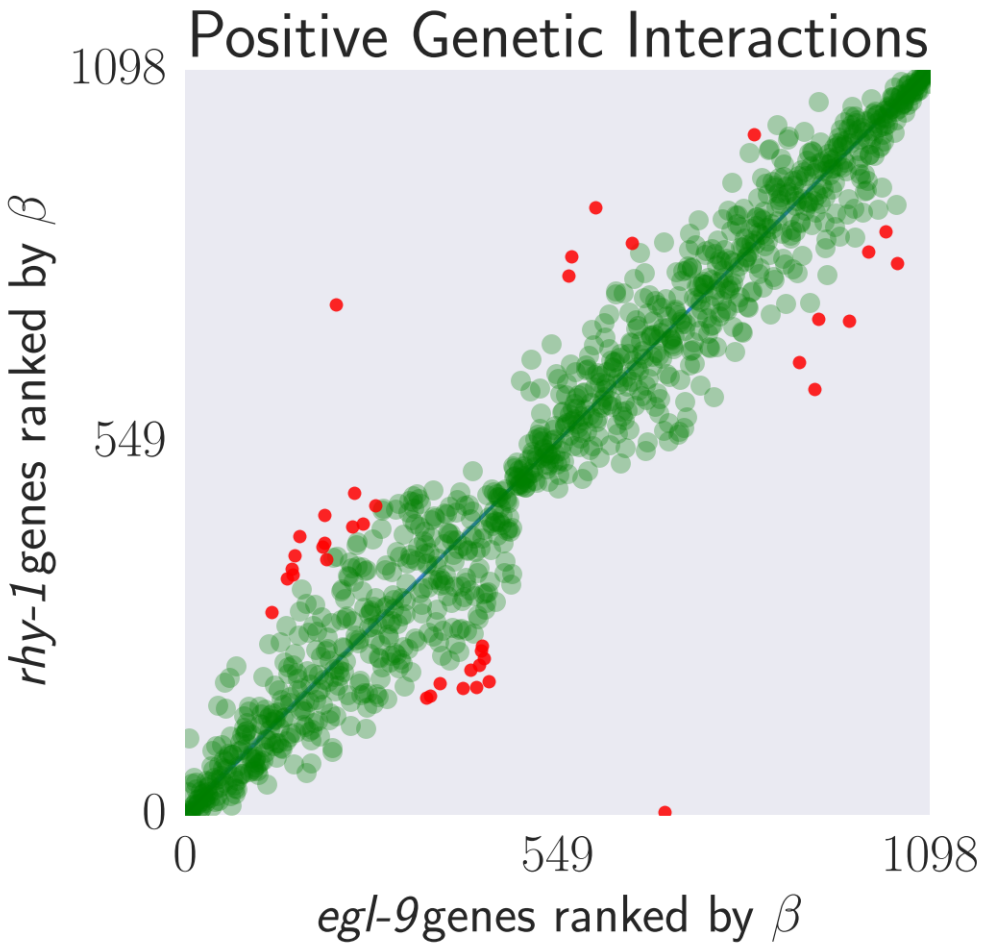
In the above example, we saw that *rhy-1* and *hif-1* share a complex regulatory relationship as exhibited by the cross-pattern. However, an important point is whether or not this cross pattern appears in all datasets. While we cannot generalize yet (this kind of analysis is very new) we should definitely make sure that not all of our genes are exhibiting this cross pattern. Below, I show some a pair of genes that don't exhibit a cross pattern.

```
In [10]: genes = 'target_id'

         letters = ['b', 'e']

         # datasets
         x = thomas.beta_filtered[letters[0]]
         y = thomas.beta_filtered[letters[1]]

         # overlap
         ovx = x[lind(x)]
         ovy = y[lind(y) & y[genes].isin(ovx[genes])].copy()
         ovx = x[lind(x) & x[genes].isin(ovy[genes])].copy()

         # find rank
         ovx = find_rank(ovx)
         ovy = find_rank(ovy)

         # Take the data and place it
         # into a dictionary to feed into the robust regression
         data = dict(x=ovx.r, y=ovy.r)

         x = np.linspace(ovx.r.min(), ovx.r.max())

         # perform the simulation
         trace_robust = robust_regress(data)
```

```
# draw a figure
plt.figure(figsize=(5, 5))

# some statistics.
# normalize everything so that all points are centered around 0
# by taking the y-coordinates and subtracting
# the value of the model at the point we calculated
intercept = trace_robust.Intercept.mean()
slope = trace_robust.x.mean()
distribution = ovy.r - intercept - ovx.r*slope

# find the mean and stdev of the distribution
# (even though mean should be 0 now)
mean = distribution.mean()
std = distribution.std()

# find inliers
results = find_inliers(distribution, mean, trace_robust)
distribution_inliers, distribution_outliers, outliers = results

# y-coordinate of outliers
yri = distribution_inliers + intercept + ovx.r*slope
yro = distribution_outliers + intercept + ovx.r*slope

# plot the regression lines
label = 'posterior predictive regression lines'
pm.glm.plot_posterior_predictive(trace_robust, eval=x,
                                 label=label,
                                 color='#357EC7')
pm.glm.plot_posterior_predictive(trace_robust2, eval=x,
                                 label=label,
                                 color='#FFA500')

# plot the data. Inliers are plotted as
#large green dots, outliers as small red dots
plt.plot(ovx.r, yri, 'go', ms = 7.5, alpha=0.7)
plt.plot(ovx[yro > 0].r, yro[yro > 0], 'ro', ms = 5)

# prettify plot
plt.xlim(0, len(ovx))
plt.ylim(0, len(ovy))
plt.yticks([0, np.floor(len(ovx)/2), len(ovx)], fontsize=20)
plt.xticks([0, np.floor(len(ovx)/2), len(ovx)], fontsize=20)
pathify('Transcriptome Reflects Multiple Interaction Modes',
        genotype_mapping[letters[0]] + r'genes ranked by $\beta$',
        genotype_mapping[letters[1]] + r' genes ranked by $\beta$',
        labelsize=24)
```

```
        comp = letters[0] + letters[1]
        plt.savefig('../output/multiple_modes_{0}.pdf'.format(comp)
```

```
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 2.2 sec
```



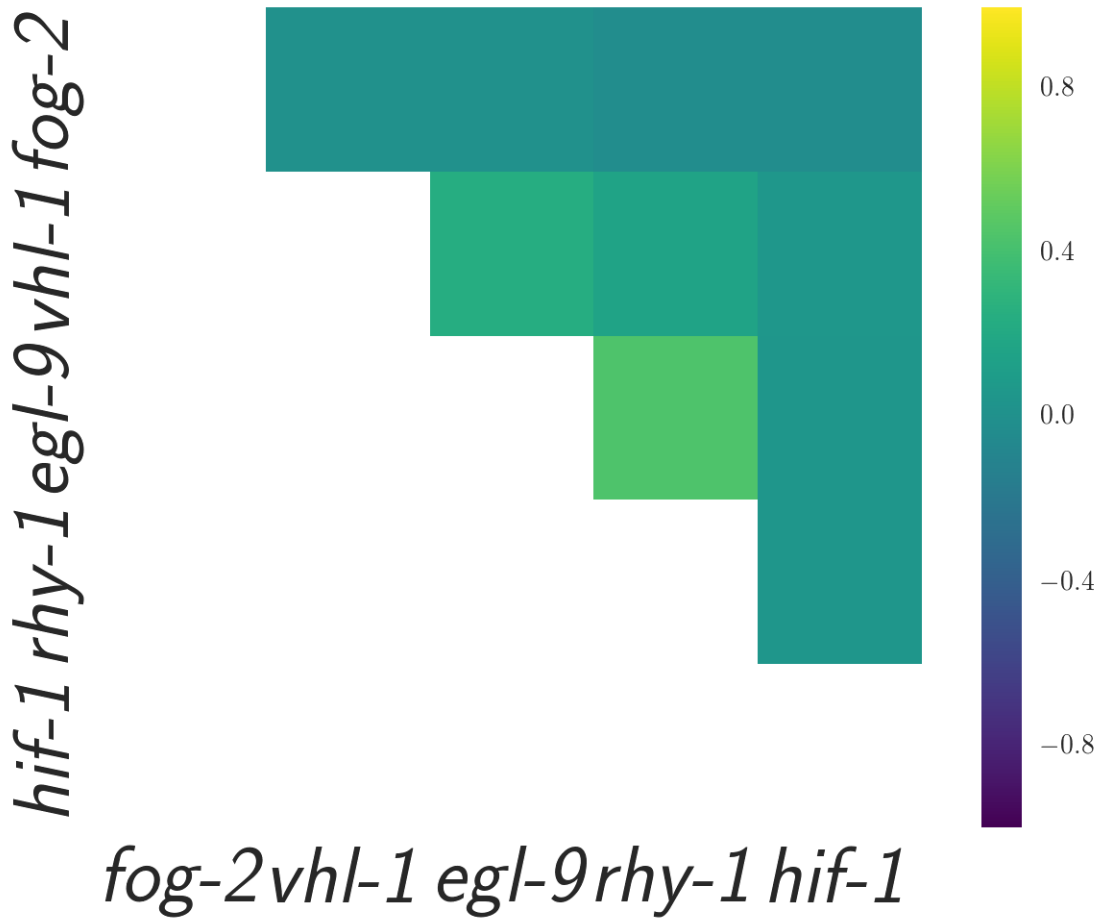
8 Pairwise Analysis of All Genes Using Spearman Correlation

Given what we found above, we know that the spearman analysis is good to use to get a rough fast idea of how our data looks. Let's go ahead and run the analysis for all pairwise comparisons that can be made using this dataset.

Spearman correlations and hypergeometric tests can be performed using the class morgan.brenner.

Once I calculate the quantities of interest, I will plot them on a heatmap. However, reader beware. Do NOT **EVER** use jet. The red and blue heatmaps? Awful. They have very serious pathologies that have been well described, and they generate patterns via optical illusions. Instead, we will use the *viridis* colormap, which was used by scientists in LIGO.

In [11]:
```
# Define a 'genes' variable that will be used
# for labelling every plot from here on out:
genes = [genotype_mapping[x] for x in thomas.single_mutants]

# Define a plotting function to plot only a triangular heat map
def tri_plot(matrix, xlabels, ylabels=[]):
    """Given a matrix, draw a triangle plot."""
    # Minimum and maximum for colormap
    vmin= matrix.min().min()
    vmax= np.max(matrix).max()

    # if user didn't specify xlabels, assume ylabels
    # are the same as xlabels
    if len(ylabels) == 0:
        ylabels = xlabels

    # make the lower triangle of the matrix,
    # since we are only dealing with
    # symmetric matrices. Also, remove the diagonal
    mask = np.zeros_like(matrix)
    mask[np.tril_indices_from(mask)] = True

    # draw and adjust xtick size
    with sns.axes_style("white"):
        ax = sns.heatmap(matrix, xticklabels=xlabels
                         yticklabels=ylabels, cmap='viridis',
                         mask=mask, square=True, vmin=vmin,
                         vmax=vmax)
    plt.xticks(fontsize=30)
    plt.yticks(fontsize=30)

# Perform Correlation Analysis
sydney = morgan.brenner('spearman', thomas)
# Plot
tri_plot(sydney.rho.as_matrix(columns=thomas.single_mutants),
         genes)
```



So maybe *egl-9* and *rhy-1* interact? Let's wait for the full Bayesian treatment.

9 Hypergeometric Analysis:

Another method to query whether there are interactions between genes is perform an urn test of enrichment. We ask: What is the probability that the DE genes in a gene B are the same as those in A? This probability should intuitively be 1 if A and B have exactly the same DE genes; it should be 0 (or close to) if they share no genes in common.

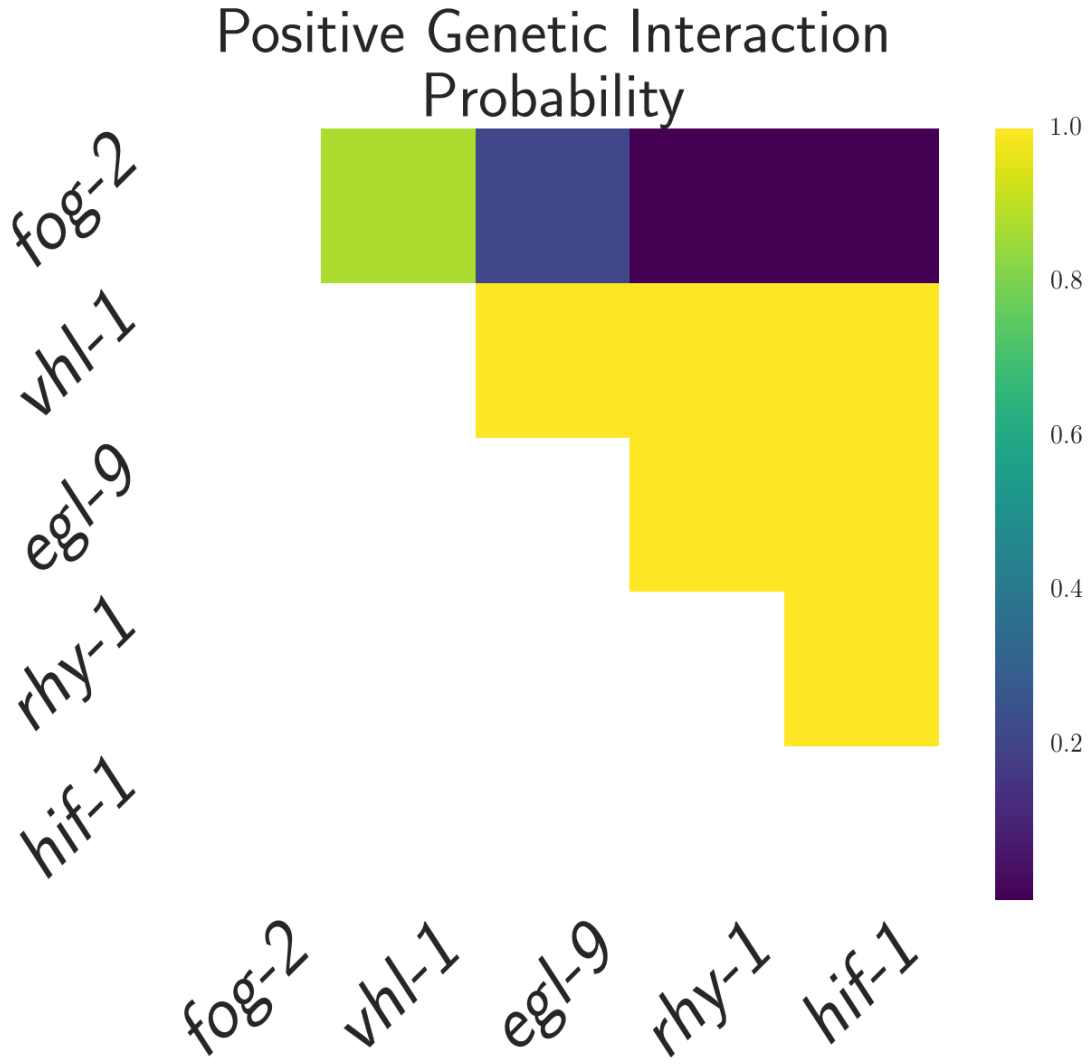
I approached this problem by assuming that the urn to be drawn from is exactly gene A in the above example. We define one of the two genes in the comparison to be the 'urn' (specifically, I always select the one which has more DE genes). Then, the question can be rephrased as follows: What is the probability that the DE genes in B were drawn from A? In other words, we want to calculate:

$$P(A \text{ and } B \text{ are in a pathway}|B, A) \;\propto\; P(A|B) = \text{Hypergeom}(A, B,\; A \cap B,\; \text{genome}) \qquad (4)$$

In reality this approach has an incredibly low threshold for activation. Biological networks are not random, and therefore my suspicion is that this test is biased towards positive results. This

would be the case particularly if networks are dense (everything interacts with everything). Part of the problem with the hypergeometric is that it saturates very rapidly, so the dynamic range is low. In effect, the answers become largely binary (....great for an SVM?). However, this means that we can't use the hypergeometric probabilities to gauge functional distance (these probabilities cannot be weighted intuitively).

```
In [12]: # Perform a test for positive interaction, put them in a matrix
         mat = sydney.hyper_plus.as_matrix(columns=thomas.single_mutants)
         genes = [genotype_mapping[x] for x in thomas.single_mutants]

         # plot the results
         tri_plot(mat, genes)
         pathify('Positive Genetic Interaction\nProbability', '', '')
         plt.xticks(rotation=45)
         plt.yticks(rotation=45)

Out[12]: (array([ 0.5,  1.5,  2.5,  3.5,  4.5]), <a list of 5 Text yticklabel objec
```

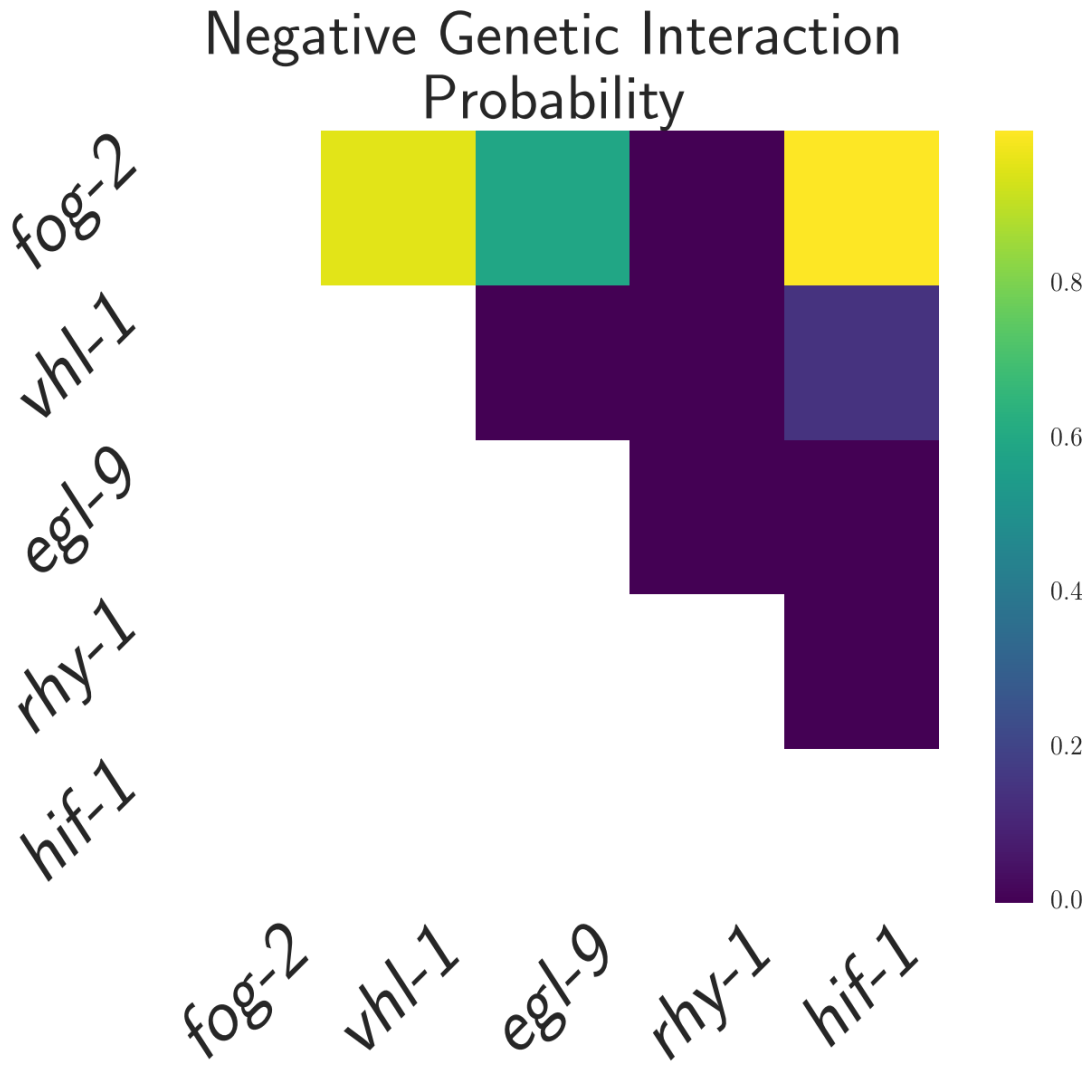


fog-2 has a weak probability of interacting with any genes in this pathway. It would be most likely that *vhl-1* and *fog-2* share a positive interaction according to this diagram. Moreover, notice that all the hypoxia genes are predicted to interact; however, the saturation makes it hard to gauge exactly what kind of interaction is occurring.

9.0.1 Probability of Negative Regulatory Interaction

Next, we can repeat our query, but we modify it slightly to ask about the possibility of a negative regulatory interaction:

```
In [13]: # perform the query, place results in a matrix and plot:
         mat = sydney.hyper_minus.as_matrix(columns=thomas.single_mutants)
         genes = [genotype_mapping[x] for x in thomas.single_mutants]
         tri_plot(mat, genes)
         pathify('Negative Genetic Interaction\nProbability', '', '')
         plt.xticks(rotation=45)
         plt.yticks(rotation=45)
         plt.savefig('../output/probability_of_inhibition_single_mutants.pdf',
                     bbox_inches='tight')
```




```
In [14]: for key, value in sydney.overlap_ids_minus.items():
             print(genotype_mapping[key[0]], genotype_mapping[key[1]],
                   len(value))
```

```
\emph{hif-1} \emph{vhl-1} 18
\emph{rhy-1} \emph{vhl-1} 23
\emph{egl-9} \emph{vhl-1} 15
\emph{vhl-1} \emph{rhy-1} 23
\emph{fog-2} \emph{hif-1} 113
\emph{rhy-1} \emph{egl-9} 3
\emph{hif-1} \emph{egl-9} 39
\emph{fog-2} \emph{vhl-1} 112
\emph{egl-9} \emph{fog-2} 248
\emph{vhl-1} \emph{fog-2} 112
```

```
\emph{vhl-1} \emph{egl-9} 15
\emph{rhy-1} \emph{fog-2} 255
\emph{hif-1} \emph{fog-2} 113
\emph{fog-2} \emph{rhy-1} 255
\emph{egl-9} \emph{hif-1} 39
\emph{fog-2} \emph{egl-9} 248
\emph{rhy-1} \emph{hif-1} 38
\emph{vhl-1} \emph{hif-1} 18
\emph{egl-9} \emph{rhy-1} 3
\emph{hif-1} \emph{rhy-1} 38
```

We see that this time *fog-2* has a real probability of negative interaction with the hypoxia pathway. Interesting.

Aside from that, there is a small probability (20%) that *hif-1* and *egl-9* are interacting in a negative manner. Incidentally, that is the **correct** mode of interaction, according to the canonical literature.

10 Analysis Using Robust Regression

What I do here is try to identify trends. Namely, for two genotypes X and Y, I fit a line using a Bayesian robust regression (see methods) thru the rank-ordered regression coefficients that are statistically significantly different from 0 and that are present in both X and Y. Next, I use this Bayesian framework to identify any and all outliers to the regression.

In order to test for alternative modes of interaction in this dataset, I take the outliers and I run the same regression again *on the outliers*. If this second interaction has an opposite sign to the first, then we predict that there are two modes of interaction, subject to the following caveats:

- Both the primary and secondary regressions yield strong correlations (>0.7)
- The first regression results in many outliers

Finally, because of the approach I used, I can also predict which genes are under what mode of regulation. Super cool!

```
In [15]: barbara = morgan.mcclintock('bayesian', thomas, True)
```

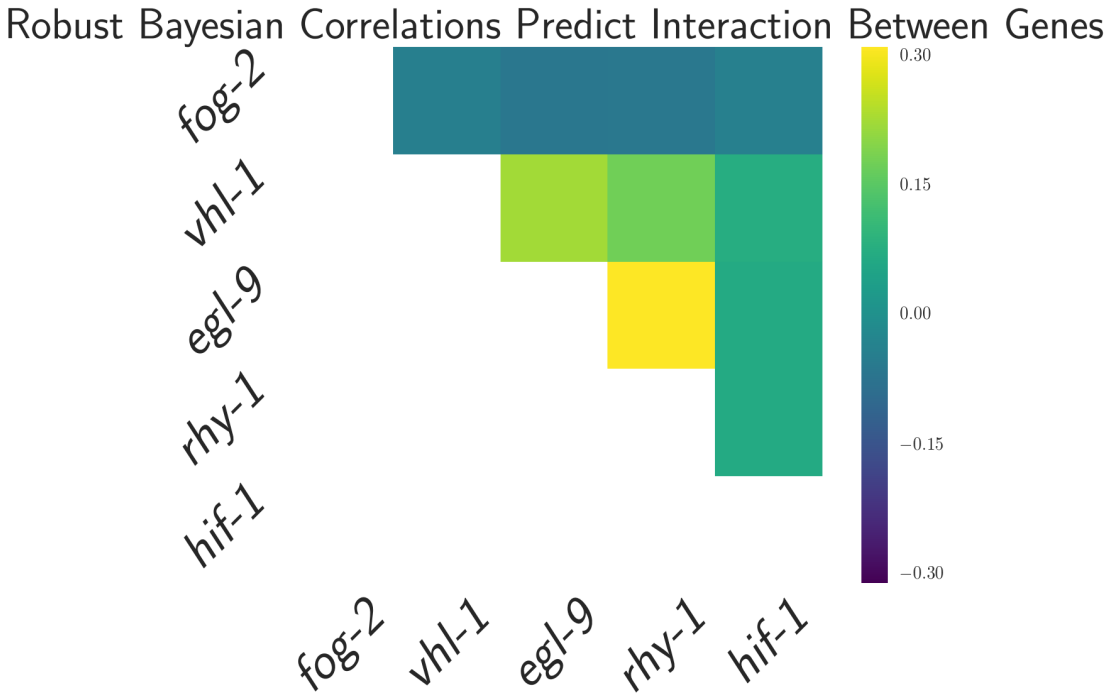
```
starting comparison of g, d
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.5 sec
starting comparison of g, b
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.9 sec
starting comparison of g, e
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 2.0 sec
starting comparison of g, c
Applied log-transform to lam and added transformed lam_log_ to model.
```

```
 [----------------100%----------------] 2000 of 2000 complete in 1.7 sec
starting comparison of d, b
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.6 sec
starting comparison of d, e
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.8 sec
starting comparison of d, c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 sec
starting comparison of b, e
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.9 sec
starting comparison of b, c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 sec
starting comparison of e, c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.5 secg g
g d
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 secg b
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 secg e
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 secg c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.5 secd d
d b
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 secd e
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.5 secd c
b b
b e
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.5 secb c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.3 sece e
e c
Applied log-transform to lam and added transformed lam_log_ to model.
 [----------------100%----------------] 2000 of 2000 complete in 1.5 secc c
```

The simulation has finished, and now I should make the heat map showing the primary correlations.

10.1 Figure 3. Pairwise regression values between all single mutants

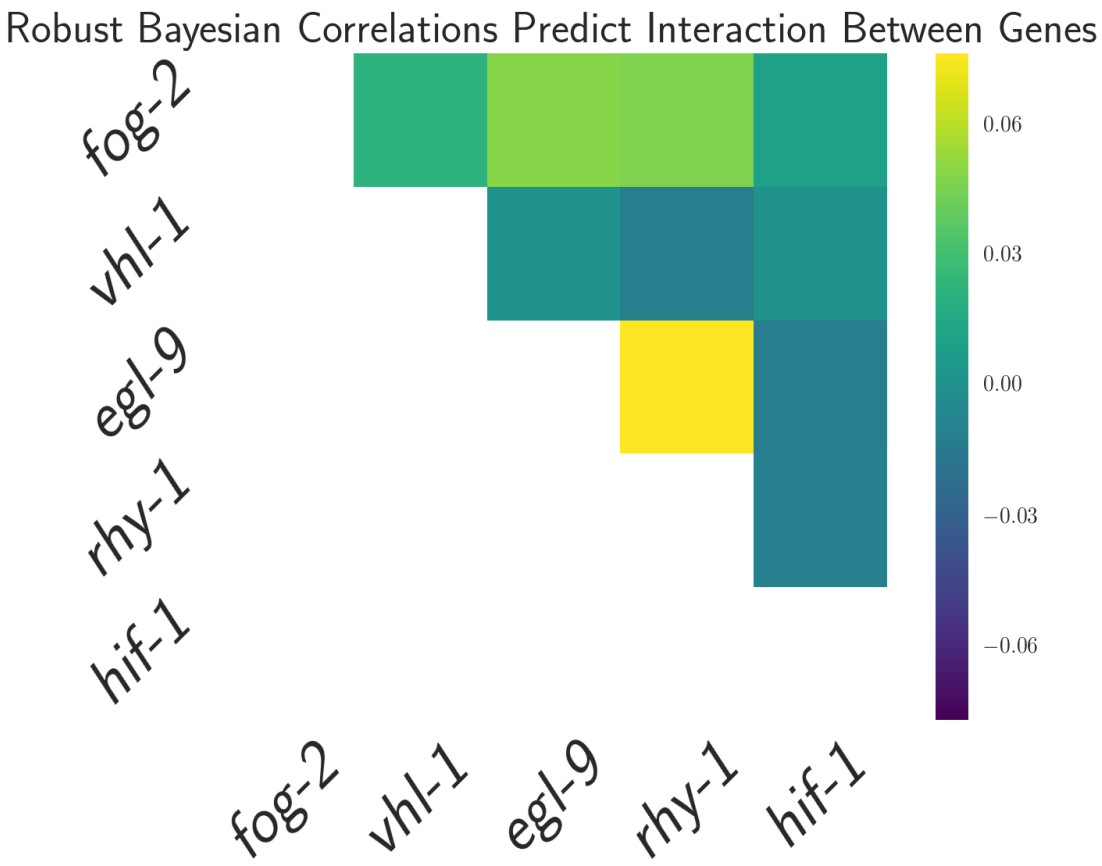
```
In [16]: mat = barbara.robust_slope.as_matrix(columns=thomas.single_mutants)
         tri_plot(mat, genes)
         plt.xticks(rotation=45)
         plt.yticks(rotation=45)
         pathify('Robust Correlations Predict Regulatory Interactions',
                 '', '')
         plt.savefig('../output/bayes_primary_single_mutants.pdf',
                     bbox_inches='tight')
```



Notice the correlation values. They are very small, less than 0.30 in magnitude. This is fine though: While you weren't looking, I weighted the correlation by the fraction of DE genes that two mutant transcriptomes share divided by the total number of DE genes in either transcriptome. Since this fraction is usually $<< 1$, our correlations have all shifted down, but that's OK! Remember the hypergeometric test suggested that every single on of the hypoxia interactions is non-random (for positive interactions) so we can freely interpret these correlation coefficients.

We can also plot the secondary correlations and see what comes out of them

```
In [17]: mat = barbara.secondary_slope.as_matrix(columns=thomas.single_mutants)
         tri_plot(mat, genes)
         plt.xticks(rotation=45)
         plt.yticks(rotation=45)
         plt.title('Secondary Regulatory Interactions', fontsize=20)

Out[17]: <matplotlib.text.Text at 0x1854a40b8>
```



11 Extracting functional relationships between genes

Another way to plot the results from the regression above is to make scatterplots. For that, it's best to work with tidy dataframes. What I will do next is the following:

Take the square correlation matrix I calculated above. Place it in a tidy dataframe. Concatenate this dataframe to include both primary and secondary relationships. Add a column that contains an indicator variable specifying whether the primary and the secondary regressions have different sign. If they don't have different signs, then add the correlations together. The reasoning for this is that:

$$w_1\rho + w_2\rho = (w_1 + w_2)\rho \tag{5}$$

In other words, we expect the correlation coefficient to be the same between the primary and secondary correlation and performing this is heuristically the same as adding the weights together. I know there's a better way to do this, and I am actively working on implementing this is in a proper manner. In the future, the weights will be added, and the secondary regression wil be discarded.

Finally, I order the pairs of genes to reflect decreasing functional distance and plot.

11.1 Decorrelation due to branching pathways generates monotonically decreasing plots

If we consider a pathway, A –> B –> C, then we can imagine the following thought process, granted that at each point of the pathway there are branches (i.e., A controls more genes than just B, but B is entirely controlled by A; etc. . .): * The mutant A^- should lose the transcriptomes associated with all proteins, since it is equivalent to $A^-B^-C^-$ * The mutant B^- is equivalent to losing only B^-C^-, since the off-pathway transcriptome associated with A is intact * The mutant C^- is equivalent to losing only C^-

Therefore, if the mutants for A and B ought to share a fraction equal to:

$$\frac{A \cap B}{A \cup B} = \frac{|B^-| + |C^-|}{|A^-| + |B^-| + |C^-|}, \tag{6}$$

Where $|x|$ signifies 'the number of elements in X'. Moreover, here we consider the set of genes associated with loss of a gene to be only those genes that are differentially expressed relative to wild-type, NOT the entire genome.

And the mutants for A and C should share a fraction equal to:

$$\frac{A \cap C}{A \cup C} = \frac{|C^-|}{|A^-| + |B^-| + |C^-|}. \tag{7}$$

Therefore, it follows that the fraction of genes that A and B share should be larger than the fraction that A and C share.

Moreover, if we relax the assumption that A, B and C should be entirely under control of this chain, we can imagine that the correlation between a LOF allele of A and a mutant B or C will necessarily be < 1. We can combine both assumptions by generating a weighted correlation coefficient:

$$\hat{\rho_{i,j}} = \frac{|i \cap j|}{|i \cup j|}\rho_{i,j} \tag{8}$$

```
In [37]: def tidy_df(df, corr='corr', morgan_obj=thomas):
             """
             A function that returns a tidied up dataframe.

             Dataframe provided must be the result of morgan.robust_regression()
             or morgan.robust_regression_secondary()

             df - dataframe to tidy up
             corr - a string indicating whether to use 'corr' or 'outliers'

             outputs:
             df - a tidied dataframe with columns 'corr_wit', 'variable',
                 'fraction' and 'pair'
             """
             # make a copy of the df
             df = df.copy()
             # append a column called corr_with
             if 'corr_with' not in df:
```

```
            df['corr_with'] = morgan_obj.single_mutants
        # melt it so that each row has a single correlation
        df = pd.melt(df, id_vars='corr_with')
        # drop any observations where the correlated letters are the same
        df = df[df.corr_with != df.variable]

        def calculate_fraction(x, fraction='corr'):
            """Fraction of genes that participate in a given interaction."""
            if (x.corr_with, x.variable) in barbara.correlated_genes.keys():
                dd = barbara.correlated_genes[(x.corr_with, x.variable)]
                outliers = len(dd['outliers'])
                corr = len(dd['corr'])
                total = outliers + corr
                if fraction == 'corr':
                    return corr/total
                else:
                    return outliers/total
            else:
                return np.nan

        # calculate the fraction of genes participating in any interaction
        df['fraction'] = df.apply(calculate_fraction, args=(corr,), axis=1)
        # generate a new variable 'pair' that is
        # the sum of the correlated genotypes (i.e. 'a', 'b' --> 'ab')
        df['pair'] =  df.variable + df.corr_with
        # return the damned thing:
        return df

    def different(x, d):
        """
        Returns an indicator variable if the primary regression
        is different in sign from the secondary.
        """
        # extract the pair in question:
        p = x.pair
        # search for the primary interaction in the dataframe
        primary = d[(d.pair == p) &
                    (d.regression == 'primary')].value.values[0]
        # search for the secondary
        secondary = d[(d.pair == p) &
                      (d.regression == 'secondary')].value.values[0]

        # if the interactions are 0, return 0
        if primary == 0 or secondary == 0:
            return 0
        # if they have the same sign, return -1
        elif (primary*secondary > 0):
            return -1
```

```
    # otherwise return 1
    else:
        return 1

def special_add(x):
    """
    If the primary and secondary have the same sign,
    returns the addition of both.
    """
    # if the current row is a secondary row
    # and the primary and secondary rows are the same
    # then return np.nan since we will want to ignore
    # the secondary correlation
    # if they are different in sign, return the current value
    if x.regression == 'secondary':
        if x.different == -1:
            return np.nan
        else:
            return x.value

    # if the regression is primary,
    # then add the values if the correlations have the same sign
    # otherwise just return the current value:
    check = d[(d.regression=='secondary') & \
         (d.pair == x.pair)].different.values
    if check == -1:
        to_add = d[(d.regression=='secondary') &
                   (d.pair == x.pair)].value.values[0]
        return x.value + to_add
    else:
        return x.value

# tidy up the dataframe w/bayesian primary interactions:
d_pos = tidy_df(barbara.robust_slope)
# add a label specifying these are the primary regressions
d_pos['regression'] = 'primary'
# tidy up the secondary interactions
d_minus = tidy_df(barbara.secondary_slope, corr='outliers')
# add a label specifying these are the secondary regressions
d_minus['regression'] = 'secondary'

# concatenate the dataframes
frames = [d_pos, d_minus]
d = pd.concat(frames)

# identify whether primary and secondary
# interactions have different signs
d['different'] = d.apply(different, args=(d,), axis=1)
```

```
        # drop any fractions that are NAN
        d.dropna(subset=['fraction'], inplace=True)
        # calculate corrected coefficients
        d['corrected'] = d.apply(special_add, axis=1)
        # drop any NAN corrected columns
        d.dropna(subset=['corrected'], inplace=True)

        # sort the pairs according to functional distance
        d['sort_pairs'] = d.pair.map(sort_pairs)
        d.sort('sort_pairs', inplace=True)

        # add the labels for plotting:
        d['genes'] = d.pair.map(decode_pairs)
```

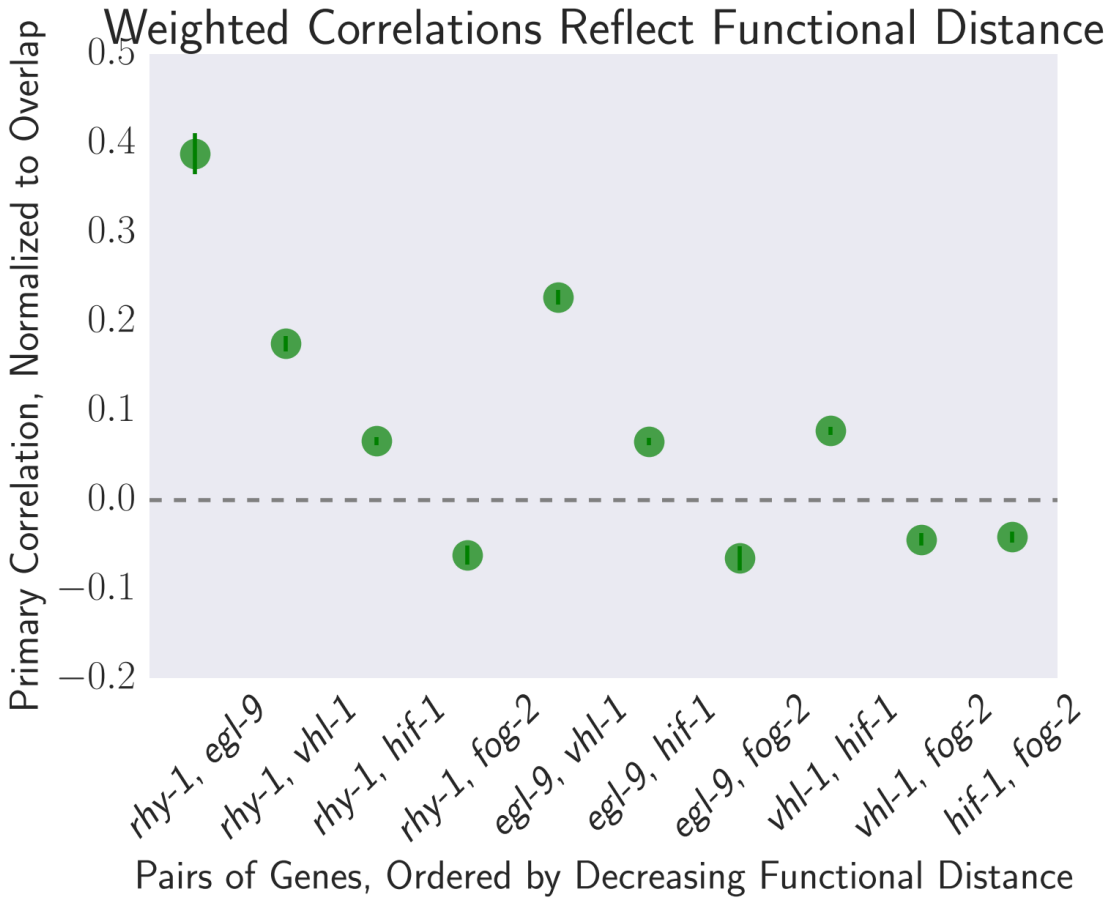
```
In [38]: # extract the standard error for each correlation
         e_plus = tidy_df(barbara.errors_primary)
         e_plus['sort_pairs'] = e_plus.pair.map(sort_pairs)
         e_plus['genes'] = e_plus.pair.map(decode_pairs)
         e_plus.sort('sort_pairs', inplace=True)
         e_plus.dropna(inplace=True)

         e_minus = tidy_df(barbara.errors_secondary)
         e_minus['sort_pairs'] = e_minus.pair.map(sort_pairs)
         e_minus['genes'] = e_minus.pair.map(decode_pairs)
         e_minus.sort('sort_pairs', inplace=True)
         e_minus.dropna(inplace=True)
```

11.2 Figure 4. Weighted Correlations Reflect Functional Distance

Now we can plot the weighted correlations, making sure that they are ordered by functional distance.

```
In [39]: # generate a stripplot with all the
         sns.stripplot(x='genes', y='corrected',
                       data=d[d.regression=='primary'], size=15,
                       color='g', alpha=0.7)

         # add errorbars:
         for x, xlabel in zip(plt.gca().get_xticks(),
                              plt.gca().get_xticklabels()):
             temp = d[d.regression=='primary']
             f = temp.genes == xlabel.get_text()
             f2 = e_plus.genes == xlabel.get_text()
             plt.gca().errorbar(np.ones_like(temp[f].corrected.values)*x,
                                temp[f].corrected.values,
                                yerr=e_plus[f2].value.values,
                                ls='none', color='g')
```

```
# prettify:
plt.xticks(rotation=45, fontsize=20)
# plt.yticks([-0.1, 0, 0.5], fontsize=20)
plt.yticks(fontsize=20)
plt.axhline(0, lw=2, ls='--', color='gray')
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
pathify('Weighted Correlations Reflect Functional Distance',
        'Pairs of Genes, Ordered by Decreasing Functional Distance',
       'Primary Correlation, Normalized to Overlap')
plt.savefig('../output/weighted_corr_decreases_w_distance.pdf')
```



11.3 *hif-1* has negative interactions with *rhy-1*, and *egl-9*

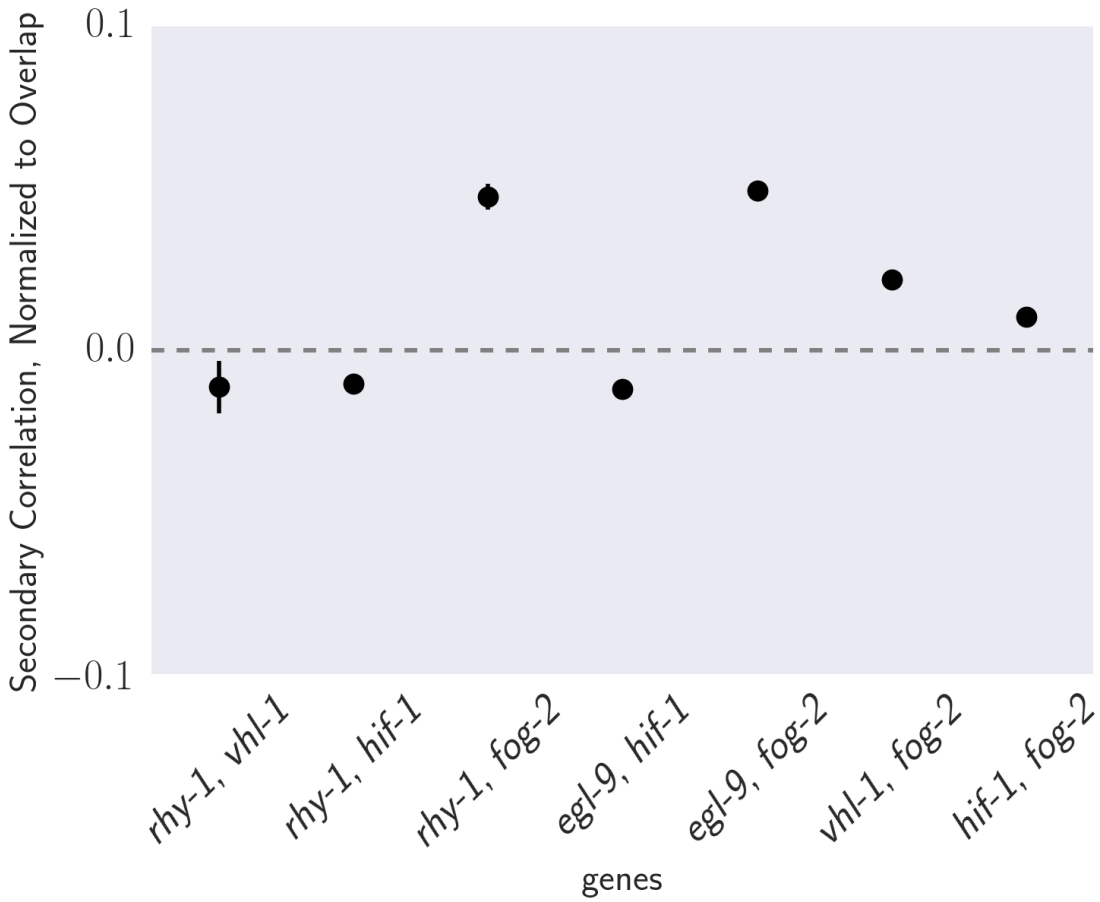
If we look at the secondary, what picture emerges? The most striking picture is that there are two modes of interaction between *rhy-1* and *vhl-1*, *hif-1* and *fog-2*. In every case, the secondary interaction is considerably less than 10% of the total overlap. I will make the claim that these interactions are stronger and more important than we may believe at first sight, but I won't push it. We will wait until we have more datasets to show this.

Regardless, let's think about what this entails. In theory, *rhy-1*, *egl-9* and *vhl-1* should have exclusively positive relationships. However, we see that these genes share at least some other secondary interactions.

In particular, I find the fact that *rhy-1* and *egl-9* both have negative secondary interactions interesting. In theory, we would have expected this to be the primary mode of interaction (negative), but the data did not bear it out. The fact that these genes have two modes of interaction with *hif-1* is therefore (strongly) suggestive of a cyclic pathway, or other types of homeostasis. Moreover, the presence of this cycle promises to be explicatory: If *hif-1* has a cycle with rhy-1 that constitutes a positive feedback loop (the primary observed mode of interaction), then *vhl-1* should have a positive primary relationship with *rhy-1*, in which case we wouldn't observe a negative interaction. However, if *hif-1* has a negative feedback loop with *rhy-1*, then *vhl-1* should have a negative secondary relationship with *rhy-1*, which we can observe.

Therefore, a parsimonious model that can explain these interactions is one that suggests that *hif-1* is interacting in a negative feedback loop with *rhy-1* in a *vhl-1* independent manner. This should lead to both positive and negative transcriptomic interactions between *hif-1* and *rhy-1* and also between *rhy-1* and *vhl-1* and *egl-9* and *hif-1*.

```
In [41]: sns.stripplot(x='genes', y='corrected',
                       data=d[(d.regression=='secondary') &
                              (d.different == 1)],
                       size=10, color='k')

         # add errorbars:
         for x, xlabel in zip(plt.gca().get_xticks(),
                              plt.gca().get_xticklabels()):
             temp = d[d.regression=='secondary']
             f = temp.genes == xlabel.get_text()
             f2 = e_minus.genes == xlabel.get_text()
             plt.gca().errorbar(np.ones_like(temp[f].corrected.values)*x,
                                temp[f].corrected.values,
                                yerr=e_minus[f2].value.values,
                                ls='none', color='k')


         plt.axhline(0, ls='--', color='0.5')
         plt.xticks(rotation=45, fontsize=20)
         plt.yticks([-0.1, 0, 0.1], fontsize=20)
         plt.axhline(0, lw=2, ls='--', color='gray')
         plt.ylabel('Secondary Correlation, Normalized to Overlap')

Out[41]: <matplotlib.text.Text at 0x17482db00>
```

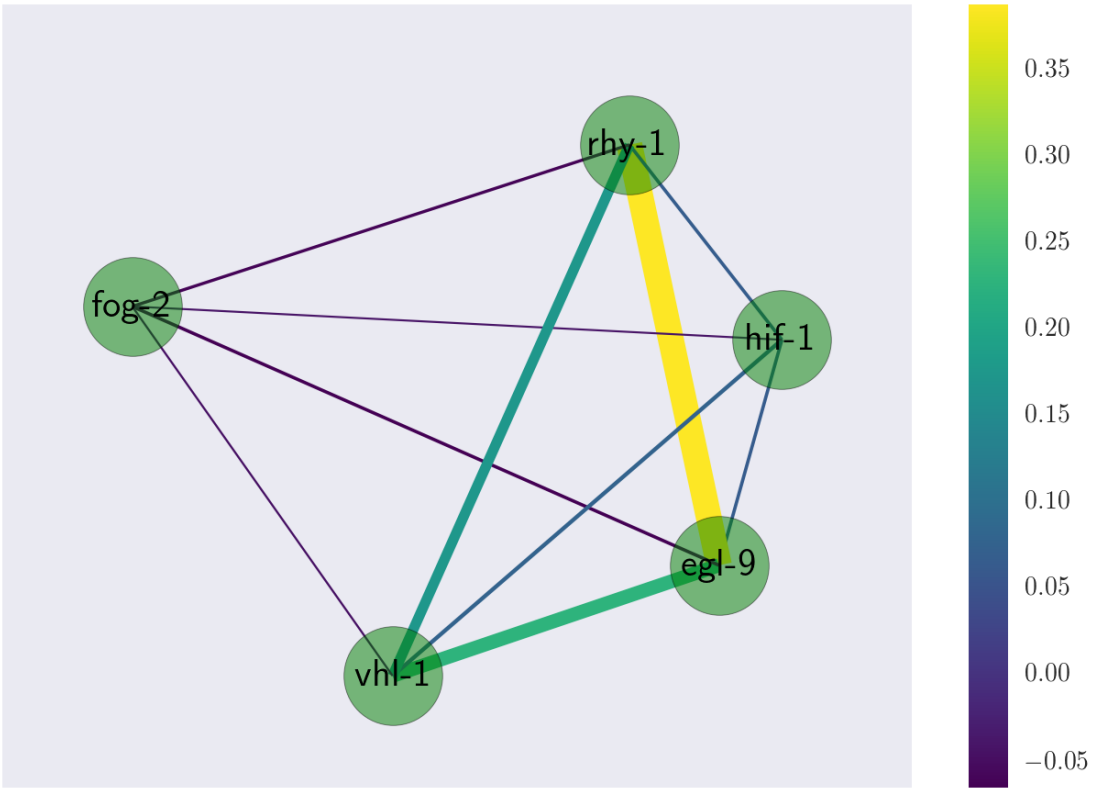


12 Figure 3 (Bottom) Correlation Graph

Now that I have estimated the primary correlation accurately, we can draw a network using these. This code will generate the bottom part of figure 3.

```
In [42]: # initialize an empty graph:
         G=nx.Graph()

         # create a dictionary of labels to translate
         mutant_dict = {'b': 'egl-9',
                        'c': 'hif-1',
                        'e': 'rhy-1',
                        'd': 'vhl-1',
                        'g': 'fog-2'
                       }

         # add edges between nodes
         # go through each mutant:
```

```
for i, key in enumerate(thomas.single_mutants):
    # and through each mutant:
    for j, key2 in enumerate(thomas.single_mutants):
        # extract the correlation coefficient
        r = d[(d.variable == key) & (d.corr_with == key2) &
              (d.regression == 'primary')].corrected.values
        # only add an edge if a correlation coefficient exists:
        if r:
            # add the edge
            G.add_edge(mutant_dict[key],
                       mutant_dict[key2], weight=r[0])

# parameterize the edge and width and color of the graphs
# extract the edges:
elarge=[(u,v) for (u,v,d) in G.edges(data=True)]
# set the width -- I will use a slightly supra-linear function to
# make the widths easier to discern to human eyes:
width=[30*np.abs(d['weight'])**1.15
       for (u,v,d) in G.edges(data=True)]
# extract the weights (keep these exact!)
weights=[d['weight'] for (u,v,d) in G.edges(data=True)]

# paint the canvas:
fig, ax = plt.subplots()
pos=nx.spring_layout(G)  # positions for all nodes

# draw the nodes:
nx.draw_networkx_nodes(G, pos, node_size=1500,
                       node_color='g', alpha=.5)

# draw the edges:
edges = nx.draw_networkx_edges(G, pos, edgelist=elarge,
                               width=width, edge_color=weights,
                               edge_cmap=plt.cm.viridis)

# add the labels:
nx.draw_networkx_labels(G, pos, font_size=15,
                        font_family='sans-serif')

# add a colorbar:
fig.colorbar(edges)

# de-tick and save
plt.xticks([])
plt.yticks([])
plt.savefig("../output/weighted_graph.pdf") # save as png
plt.show() # display
```



13 Double Mutant Analysis

With double mutants, the analysis gets slightly more complicated. Now we're getting into full pathways!

A first approach is to inspect the double mutants by Spearman correlation analysis to the single mutants that make them up. A quick visualization will show us any epistasis and the extent of it:

```
In [ ]: alfred = morgan.sturtevant('epistasis analysis')
        alfred.epistasis_analysis(thomas)
        alfred.epistasis_secondary(thomas, True)
```

```
In [ ]: alfred.epistasis['unweighted'] = alfred.epistasis.correlation/alfred.epista

        alfred.epistasis['double mutant'] = alfred.epistasis.double_mutant.map(geno
        alfred.epistasis['corr with'] = alfred.epistasis.corr_with.map(genotype_map

        sns.stripplot(x='double mutant', y='unweighted',
                      hue='corr with', data=alfred.epistasis,
                      size=15, jitter=True)
        plt.xticks(fontsize=20)
        plt.axhline(0, ls='--', color='0.5')
        plt.yticks([-0.2, 0.5, 1], fontsize=20)
```

```
In [ ]: sns.stripplot(x='double mutant', y='correlation',
                    hue='corr with', data=alfred.epistasis,
                    size=15, jitter=True)
        plt.xticks(fontsize=20)
        plt.axhline(0, ls='--', color='0.5')
        plt.yticks([-0.1, 0.2, 0.4], fontsize=20)

In [ ]: alfred.epistasis_secondary['unweighted'] = alfred.epistasis_secondary.corre
        alfred.epistasis_secondary['fractional corr'] = alfred.epistasis_secondary.

        alfred.epistasis_secondary['double mutant'] = alfred.epistasis_secondary.do
        alfred.epistasis_secondary['corr with'] = alfred.epistasis_secondary.corr_w

        sns.stripplot(x='double mutant', y='unweighted',
                    hue='corr with', data=alfred.epistasis_secondary,
                    size=15, jitter=True)
        plt.xticks(fontsize=20)
        # plt.ylim(-1.1, 1.1)
        # plt.axhline(0, ls='--', color='0.5')
        # plt.yticks([-1, 0, 1], fontsize=20)
```

14 Table 1. Epistasis Can Be Quantified Via Linear Regression Models

Previously we have observed that correlations between double and single mutants might be informative. However, raw correlations alone are not the most informative in this situation. The reason is that the robust comparison (ranked genes) between single mutants and double mutants is not likely to be highly variable between single and double mutants under an additive model or even certain epistatic models.

However, it is illustrative to look at a double mutant under an additive model. Consider that a double mutant ought to have extensive overlap with both single mutants that make it up. If the individual genes are interacting, then it makes sense that the double mutant should exhibit a perturbation proportional to the kind of relationship these genes share. Specifically, the genes that are involved in *both* genes will be the ones that will exhibit the expected perturbation.

Therefore, the plan of attack should be as follows: * Identify the genes that appear in each single mutant and the double mutant * Run a linear regression on the regression coefficients between the single and the double mutant (these are two correlations).

If the mutants are additive, as would be expected if they both interact via a branched third party (i.e. *A* -> X; *B* -> X; X -> Output), then the double mutant should show worsening (a linear regression with slope > 1). If they are interacting via a linear pathway, then the double mutant should have a linear slope with value of unity. If they share an inhibitory relationship, then the double mutant ought to have a slope < 1.

In the next steps, I will calculate these linear regressions. However, I will plot something slightly different. I will plot Δ versus the single mutant, where $\Delta = \beta_{\text{Double}} - \beta_{\text{Single}}$, and run the regression on the resulting values. The slopes that I will obtain will be trivially related to the ones I have described above by a scalar.

First, I will define a few functions, then plot the results.

```
In [43]: def quadrature(x):
```

```
    """Variance of sums is sum of the variances."""
    def square(x):
        return x**2
    return np.sqrt(np.sum(x.apply(square)))

def identify(pair, df0, df1, double):
    """
    identify the genes that are altered
    in all of these mutant dataframes

    Params:
    pair =
    """
    # predictions, single mutants
    df_pred0 = df0.copy()
    df_pred1 = df1.copy()

    # identify single mutants DE genes
    df_pred0 = df_pred0[df_pred0.qval < thomas.q]
    df_pred1 = df_pred1[df_pred1.qval < thomas.q]

    # find the predicted genes
    ind =
    pred_genes = df_pred0[ind].target_id

    # get the double mutant
    df_test = thomas.beta[double]

    # find the predicted genes that show up in the double,
    # these constitute the final overlap
    df_test = df_test[df_test.target_id.isin(pred_genes) &
                      (df_test.qval< 0.1)].copy()

    # select only the overlapping genes
    df_pred0 = df_pred0[df_pred0.target_id.isin(df_test.target_id)]
    df_pred1 = df_pred1[df_pred1.target_id.isin(df_test.target_id)]

    # drop anything that is NA... hopefully that means dropping nothing...
    df_pred0 = df_pred0.dropna(subset=['b'])
    df_pred1 = df_pred1.dropna(subset=['b'])
    df_test = df_test.dropna(subset=['b'])

    # return the damned thing
    return df_pred0, df_pred1, df_test

def plot_delta(prediction, test, fmt, label):
    """
    Given a prediction, calculates the delta between
```

```
    the prediction and the test, and plots the result.
    """
    # variance of the sum is sum of the variances:
    yerr = np.sqrt(test.se_b.values**2 +
                   prediction.se_b.values**2)
    # calculate the delta
    delta = test.b.values - prediction.b.values
    # plot the errorbar
    plt.errorbar(prediction.b, delta,
                 xerr=prediction.se_b,
                 yerr=yerr, fmt=fmt,
                 alpha=0.2, ms=4,
                 label=label)
    plt.legend()


def weighted_regression_delta(prediction, test):
    """
    Given a prediction and a test,
    find the weighted least squares regression between them.
    """
    # fit a WLS
    wls = sm.WLS(test.b.values - prediction.b.values,
                 prediction.b.values,
                 weights=1./prediction.se_b.values**2)
    res_wls = wls.fit()
    return res_wls


def plot_epistasis_regression(prediction0, prediction1,
                              double, slope0, slope1):
    """Plot the WLS line."""
    # find the xmin and xmax:
    xmin = np.min([prediction0.b.min(),
                   prediction1.b.min()])
    xmax = np.max([prediction0.b.max(),
                   prediction1.b.max()])

    # make a linear array for x
    x = np.linspace(xmin - 0.1, xmax + 0.1, 1000)

    # make the models
    y0 = x*slope0
    y1 = x*slope1

    # plot the models
    plt.plot(x, y0, 'g-', lw=2)
    plt.plot(x, y1, 'b-', lw=2)
```

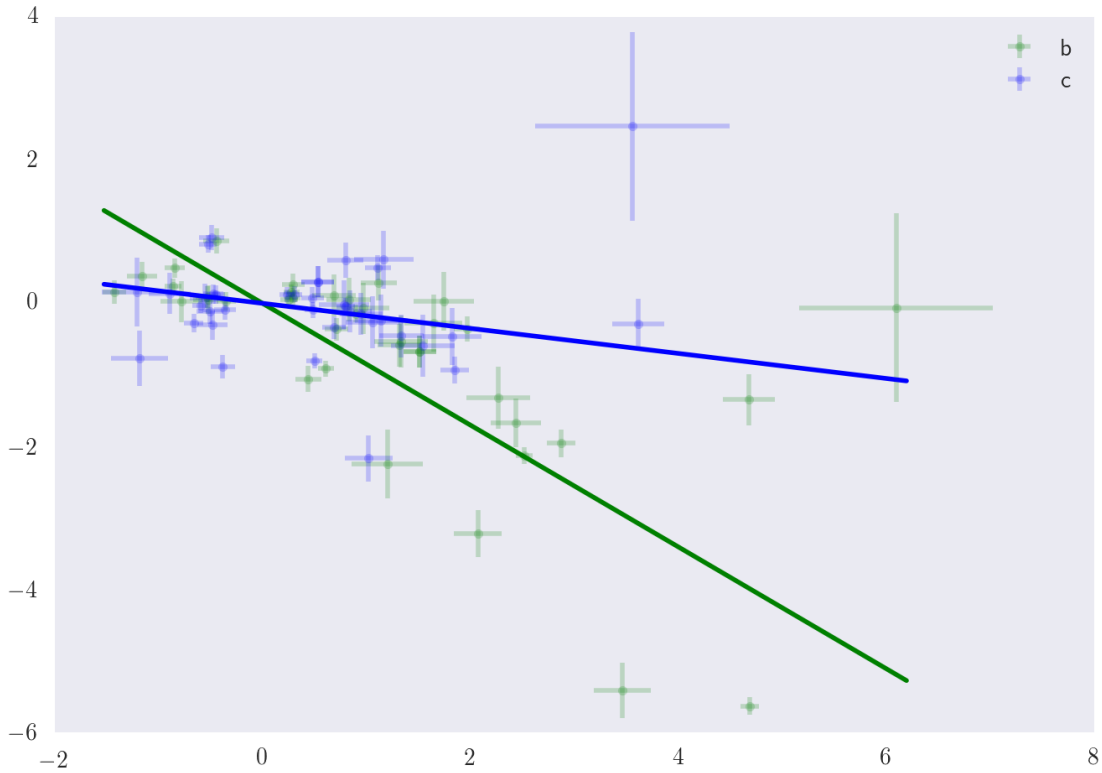
```
        def print_WLS_summary(pair, slopes, res0, res1):
            """Print a string with the WLS results."""
            string = """mutant {0} slope: {1:.2g} +/- {2:.2g}, pvalue={3:.2g}"""
            print(string.format(pair[0], slopes[0],
                                res0.bse[0], res0.pvalues[0]))
            print(string.format(pair[1], slopes[1],
                                res1.bse[0], res1.pvalues[0]))
```

14.1 Double Mutant *hif-1; egl-9* WLS epistatic analysis

```
In [48]: double = 'f'
         pair = thomas.double_muts[double]

         results = identify(pair, thomas.beta_filtered[pair[0]],
                            thomas.beta_filtered[pair[1]], double)
         df_pred0, df_pred1, df_test = results

         # plot the prediction versus the delta:
         plot_delta(df_pred0, df_test, 'go', pair[0])
         plot_delta(df_pred1, df_test, 'bo', pair[1])

         # perform the WLS
         res_wls0 = weighted_regression_delta(df_pred0, df_test)
         res_wls1 = weighted_regression_delta(df_pred1, df_test)

         # get the slopes out:
         slope0 = res_wls0.params[0]
         slope1 = res_wls1.params[0]

         # print a summary of the results
         print_WLS_summary(pair, [slope0, slope1],
                           res_wls0, res_wls1)

         # plot
         plot_epistasis_regression(df_pred0, df_pred1,
                                   df_test, slope0, slope1)

         # print out basic stats about the 'fold change'
         # which should agree with WLS
         def print_b_summary(pair, p1, p2, test):
             adj_p1 = np.sqrt(quadrature(p1.se_b)/len(p1)**2
             adj_p2 = np.sqrt(quadrature(p2.se_b)/len(p2)**2
             adj_test = np.sqrt(quadrature(test.se_b)/len(test)**2
             string = """
         {0}: mean b = {3:.2g} +/- {6:.2g}
         {1}: mean b = {4:.2g} +/- {7:.2g}
```

```
        {2}: mean b = {5:.2g} +/- {8:.2g}
        """
            print(string.format(pair[0], pair[1], pair,
                  p1.b.abs().mean(),
                  p2.b.abs().mean(),
                  test.b.abs().mean(),
                  (p1.b.abs().std() + adj_p1))/np.sqrt(len(p1)),
                  (p2.b.abs().std() + adj_p2)/np.sqrt(len(p2)),
                  (test.b.abs().std() + adj_test)/np.sqrt(len(test)),
                  )
            )


        print_b_summary(pair, df_pred0, df_pred1, df_test)
```

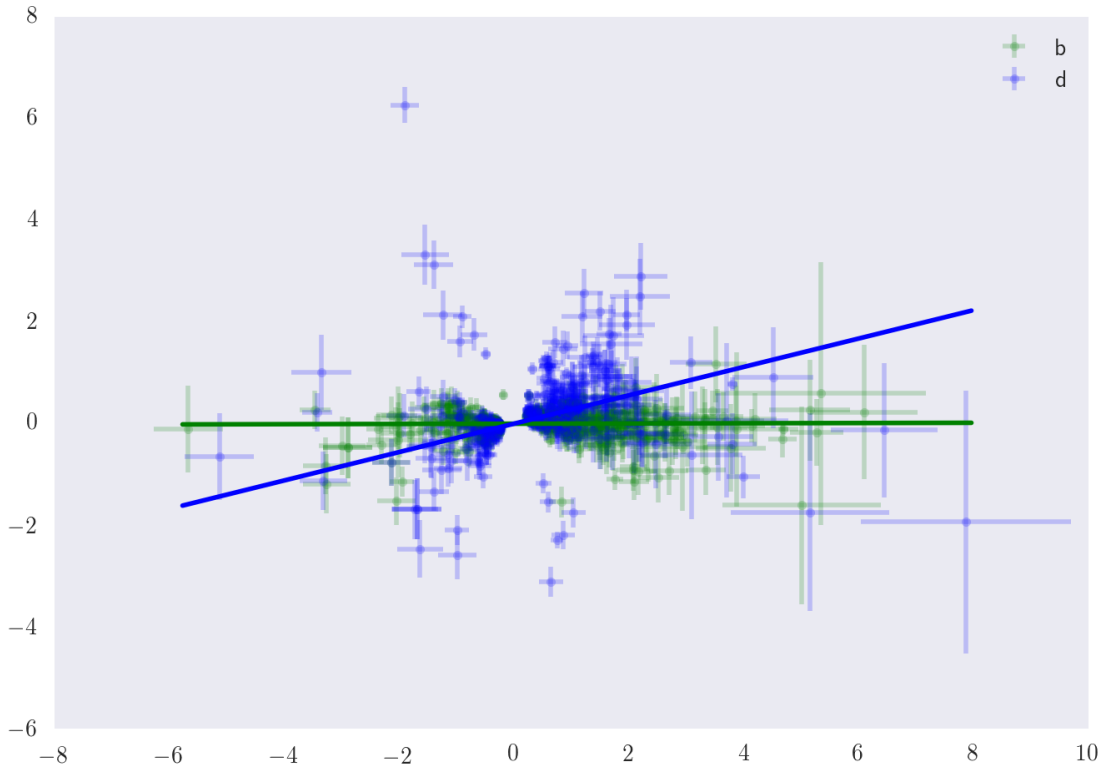
```
mutant b slope: -0.85 +/- 0.074, pvalue=1e-13
mutant c slope: -0.18 +/- 0.1, pvalue=0.097

b: mean b = 1.5 +/- 0.22
c: mean b = 0.95 +/- 0.13
bc: mean b = 1 +/- 0.17
```



14.2 Double Mutant *egl-9; vhl-1* WLS epistatic analysis

```
In [49]: double = 'a'
         pair = thomas.double_muts[double]

         results = identify(pair, thomas.beta_filtered[pair[0]],
                            thomas.beta_filtered[pair[1]], double)

         df_pred0, df_pred1, df_test = results
         # plot the prediction versus the delta:
         plot_delta(df_pred0, df_test,
                    'go', pair[0])
         plot_delta(df_pred1, df_test,
                    'bo', pair[1])

         # perform the WLS
         res_wls0 = weighted_regression_delta(df_pred0, df_test)
         res_wls1 = weighted_regression_delta(df_pred1, df_test)

         # get the slopes out:
         slope0 = res_wls0.params[0]
         slope1 = res_wls1.params[0]

         # print a summary of the results
         print_WLS_summary(pair, [slope0, slope1],
                           res_wls0, res_wls1)

         # plot
         plot_epistasis_regression(df_pred0, df_pred1,
                                   df_test, slope0, slope1)

         # print out basic stats about the 'fold change',
         # which should agree with WLS
         print_b_summary(pair, df_pred0, df_pred1, df_test)
```

```
mutant b slope: 0.0023 +/- 0.0098, pvalue=0.81
mutant d slope: 0.28 +/- 0.033, pvalue=3.7e-16

b: mean b = 1.2 +/- 0.05
d: mean b = 0.9 +/- 0.041
bd: mean b = 1.2 +/- 0.05
```



15 Double Mutants exhibit more than additive perturbations

```
In [ ]: for key, value in thomas.double_muts.items():
            x = thomas.beta_filtered[key]
            y = thomas.beta_filtered[value[0]]
            z = thomas.beta_filtered[value[1]]

            x = x[x.qval < thomas.q]
            y = y[y.qval < thomas.q]
            z = z[z.qval < thomas.q]

            yANDz = len(y[y.target_id.isin(z)])
            yORz = len(y) + len(z)
            expected = yORz - yANDz

            pred1 = x[(x.target_id.isin(z.target_id))]
            pred2 = x[x.target_id.isin(y.target_id)]

            pred = len(list(set(pred1.target_id.tolist()
                                + pred2.target_id.tolist())))
```

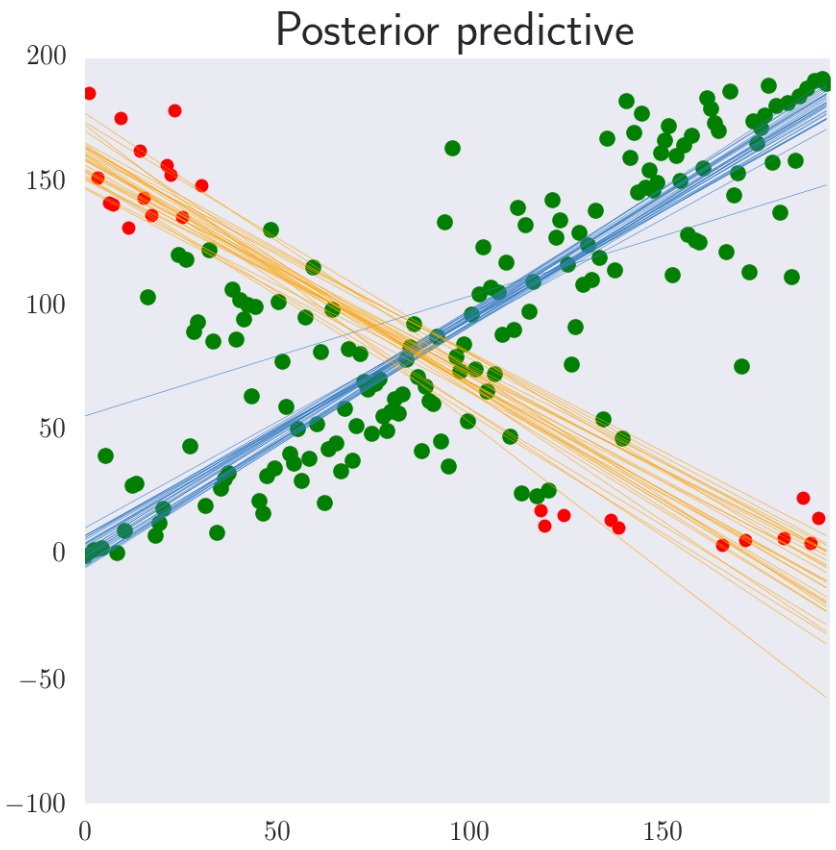
```
    print('Expected: ', expected)
    print('Observed: ', len(x))
    print('Predicted: ', pred)
    print('Observed/Expected: {0:.2g}'.format(len(x)/expected))
    print('Predicted/Expected: {0:.2g}'.format(pred/expected))
    print('Surprise factor: {0:.2g}'.format(pred/len(x)))
```

16 Double Mutants Exhibit Complex Interactions

```
In [50]: letters = ['a', 'c']

         x = thomas.beta_filtered[letters[0]]
         y = thomas.beta_filtered[letters[1]]

         ovx = x[lind(x)]
         ovy = y[lind(y) &
                 y.target_id.isin(ovx.target_id)].copy()
         ovx = x[lind(x) &
                 x.target_id.isin(ovy.target_id)].copy()

         ovx = find_rank(ovx)
         ovy = find_rank(ovy)

         data = dict(x=ovx.r, y=ovy.r)

         x = np.linspace(ovx.r.min(), ovx.r.max())

         trace_robust = robust_regress(data)
         plt.figure(figsize=(5, 5))

         intercept = trace_robust.Intercept.mean()
         slope = trace_robust.x.mean()

         distribution =ovy.r - intercept - ovx.r*slope
         mean = distribution.mean()
         std = distribution.std()

         # find the inliers and outliers
         _ = find_inliers(distribution, mean, trace_robust)
         distribution_inliers, distribution_outliers, outliers = _

         data2 = dict(x=ovx[ovx.target_id.isin(outliers)].r,
                      y=ovy[ovy.target_id.isin(outliers)].r)

         trace_robust2 = robust_regress(data2)
         intercept = trace_robust2.Intercept.mean()
         slope = trace_robust2.x.mean()
```

```
        yri = distribution_inliers + intercept + ovx.r*slope
        yro = distribution_outliers + intercept + ovx.r*slope

        plt.plot(ovx.r, yri, 'go', ms = 6)
        plt.plot(ovx.r, yro, 'ro', ms = 5)
        label = 'posterior predictive regression lines'
        pm.glm.plot_posterior_predictive(trace_robust, eval=x,
                                         label=label,
                                         color='#357EC7')
        pm.glm.plot_posterior_predictive(trace_robust2, eval=x,
                                         label=label,
                                         color='#FFA500')
        plt.xlim(0, len(ovx))
        # plt.legend()

        if np.abs(slope) > 0.6 and np.abs(slope2) > 0.6:
            t1 = slope/np.abs(slope)
            t2 = slope2/np.abs(slope2)
            if t1 == -t2:
                print('Complex Regulation at Work')
```

```
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.7 secApplied l
 [-----------------100%-----------------] 2000 of 2000 complete in 1.3 secComplex R
```



17 Quality Control

Load up the hypoxia gold standard Carmie gave me. These genes are expected to go up in everything except in Hif-1.

By this point, I have solved the network, I guessed which gene was Hif-1 and I have been told the identities of the genes.

C = Hif-1
B = egl-9
D = vhl-1
E = rhy-1

nhr-57 is an important gene for hypoxia. WBID is WBGene00003647

```
In [52]: # Figure out how many hits we get and if the pvalue is significant!
         for sm in thomas.single_mutants:
             df = thomas.beta_filtered[sm]
             ind = (genpy.find(df, hypoxia_gold.WBIDS,
                               col='ens_gene'))
             found = df[ind & (df.qval < 0.1)]
             sig = df[df.qval < 0.1]
```

```
            pval = sts.hypergeom.sf(len(found),
                                    len(df),
                                    len(hypoxia_gold),
                                    len(sig))
            print('genotype: ', sm)
            print('found: ', len(found),
                  '  Mean b: {0:.2g}'.format(found.b.mean()))
            print('pval: {0:.2g}'.format(pval))


            print('Maximum change was:')
            print(found[found.b == found.b.max()].ext_gene.values[0])
            print(np.exp(found[found.b == found.b.max()].b.values)[0],
                  ' fold change')
            if 'WBGene00003647' in found.ens_gene.values:
                ind = found.ens_gene == 'WBGene00003647'
                nhr57 = np.exp(found[ind].b.values[0])
                print('nhr-57 is in', sm,
                      'and its fold change was {0:.2g}'.format(nhr57))

            print('----------')
```

```
genotype:  g
found:  4   Mean b: -1.4
pval: 0.22
Maximum change was:
nhr-57
0.534489095703  fold change
nhr-57 is in g and its fold change was 0.53
----------
genotype:  d
found:  7   Mean b: 2.2
pval: 2.7e-06
Maximum change was:
dod-3
44.7574873949  fold change
nhr-57 is in d and its fold change was 7.1
----------
genotype:  b
found:  9   Mean b: 2.6
pval: 5.7e-05
Maximum change was:
oac-54
66.7297848473  fold change
nhr-57 is in b and its fold change was 63
----------
genotype:  e
found:  10   Mean b: 2.4
```

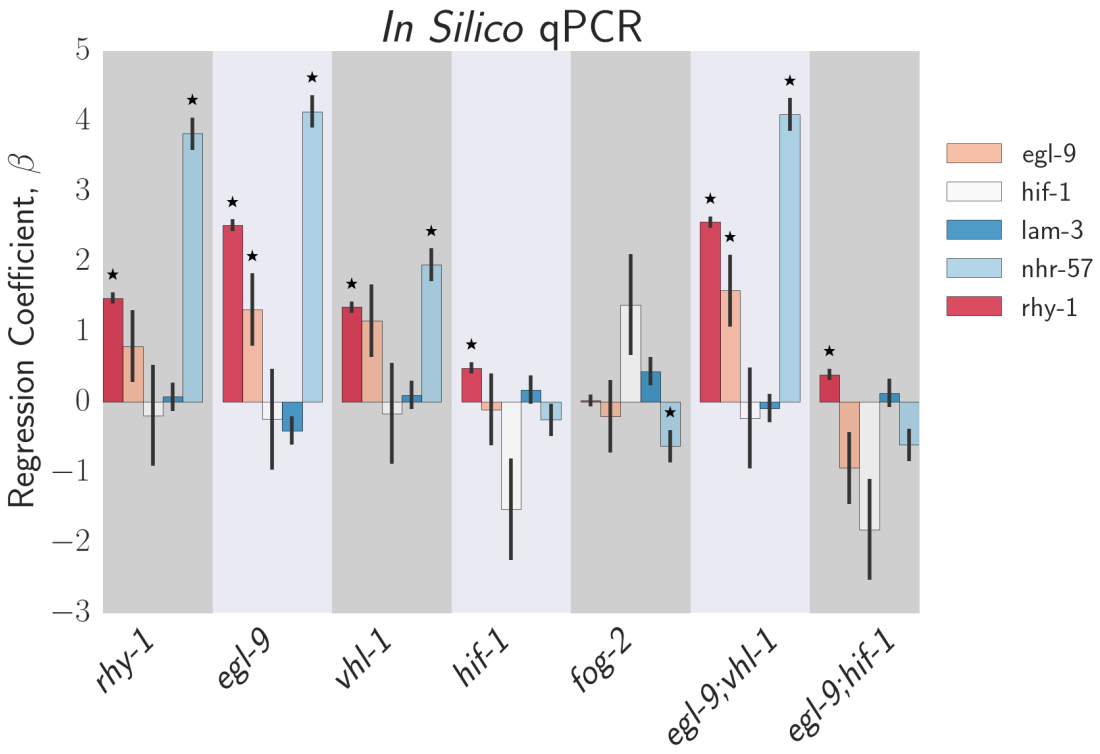
```
pval: 4.7e-05
Maximum change was:
nhr-57
45.6749505822  fold change
nhr-57 is in e and its fold change was 46
----------
genotype:  c
found:  1   Mean b: 0.49
pval: 0.19
Maximum change was:
rhy-1
1.62443595728  fold change
----------
```

18 An *in silico* qPCR experiment:

```
In [53]: sorter = {'a': 6,
                   'f': 7,
                   'b': 2,
                   'c': 4,
                   'd':3,
                   'e': 1,
                   'g': 5
                  }

         x = ['WBGene00001851',
              'WBGene00012324',
              'WBGene00001178',
              'WBGene00006922',
              'WBGene00003647',
              'WBGene00002248'
             ]

         # run the experiment!
         def qPCR_prep(morgan, genelist):
             g = []
             data = np.array([])
             i = 0
             for genotype, df in thomas.beta.items():
                 for j, xi in enumerate(genelist):
                     geno = genotype_mapping[genotype]
                     cols = ['ens_gene', 'ext_gene','b', 'se_b', 'qval']
                     y = df[(df.ens_gene == xi)][cols].values
                     if len(y) == 0:
                         continue
                     # hif has two isoforms, so take F38A6.3c
```

```
                    if y.shape[0] > 1:
                        y = df[(df.target_id == 'F38A6.3c')][cols].values
                    if len(data) == 0:
                        data = y
                    else:
                        data = np.vstack((data, y))
                    g += [genotype]
                i += 1

            d = pd.DataFrame(data, columns=cols)
            d['code'] = g
            d['genotype'] = d.code.map(genotype_mapping)
            d['order'] = d.code.map(sorter)
            d.sort_values('order', inplace=True)
            d.reset_index(inplace=True)
            return d

        d = qPCR_prep(thomas, x)

In [54]: # a qPCR barplot
        temp = d
        index = np.linspace(0, temp.genotype.unique().shape[0]-1,
                            temp.genotype.unique().shape[0])
        alpha = 0.7
        error_config = {'ecolor': '0.2'}

        plotting = {'rhy-1': 0,
                    'egl-9': 1,
                    'hif-1': 2,
                    }

        color = {'rhy-1': "#ca0020",
                 'egl-9': '#f4a582',
                 'hif-1': '#f7f7f7',
                 'nhr-57':'#92c5de',
                 'lam-3': '#0571b0'
                }

        grouped = temp.groupby('ext_gene')

        bar_width = 1/(len(grouped)+1)

        for name, group in grouped:
            if name not in plotting.keys():
                where = max(plotting.keys(),
                            key=lambda k: plotting[k])
                val = plotting[where]
                plotting[name] = val + 1
```

```
    add = plotting[name]*bar_width
    if name in color.keys():
        barlist = plt.bar(index + add, group.b.values,
                          bar_width, alpha=alpha,
                          yerr=group.se_b,
                          error_kw=error_config, label=name,
                          color=color[name])
    else:
        barlist = plt.bar(index + add, group.b.values,
                          bar_width, alpha=alpha,
                          yerr=group.se_b,
                          error_kw=error_config, label=name)
    sig = group.qval < 0.1
    k = group[sig].order -1
    plt.plot(k + add + bar_width/2,
             group[sig].b.values + group[sig].se_b.values + 0.25,
             r'*', color='k')

grouped2 = temp.groupby('genotype')
k = 0
col = '#CFCFCF'
for name, group in grouped2:
    if k % 2 == 0:
        xmin = k - bar_width*0.5
        xmax = k + bar_width*(len(grouped) + 0.5)
        ymin, ymax = plt.gca().get_ylim()
        plt.fill_between([xmin, xmax], ymax, color=col)
        plt.fill_between([xmin, xmax], ymin, color=col)
    k += 1

plt.xlim(0, plt.gca().get_xlim()[1] - bar_width)
plt.tick_params(axis='y',
                which='major', labelsize=18)
plt.xticks(index + bar_width,
           temp.genotype.unique(), rotation=45, fontsize=20)
pathify(r'\emph{In Silico} qPCR', '',
        r'Regression Coefficient, $\beta$', )
plt.legend(loc=(1.02, 0.5),fontsize=15)
plt.savefig('../output/pathwaygenes_qPCR.pdf',
            bbox_inches='tight')
```



19 Searching for a TF that is activated by both egl-9 and hif-1

And possibly regulates rhy-1 as a result

```
In [55]: # Figure out how many hits we get and if the pvalue is significant!
         tfs = {}
         for sm in thomas.single_mutants:
             df = thomas.beta_filtered[sm]
             ind = (genpy.find(df, tf_df.target_id,
                               col='target_id'))
             found = df[ind & (df.qval < 0.1)]
             sig = df[df.qval < 0.1]
             pval = sts.hypergeom.sf(len(found),
                                     len(df),
                                     len(hypoxia_gold),
                                     len(sig))
             print('genotype: ', sm)
             print('found: ', len(found),
                   '   Mean b: {0:.2g}'.format(found.b.mean()))
             print('pval: {0:.2g}'.format(pval))
             tfs[sm] = found.copy()

             print('----------')
```

```
genotype:  g
found:  44    Mean b: 0.15
pval: 0
----------
genotype:  d
found:  10    Mean b: 0.64
pval: 5.4e-10
----------
genotype:  b
found:  21    Mean b: 0.6
pval: 0
----------
genotype:  e
found:  31    Mean b: 0.49
pval: 0
----------
genotype:  c
found:  15    Mean b: 0.19
pval: 2.3e-19
----------
```

```
In [56]: def add_b(x):
             vals = tfs['c'][tfs['c'].target_id == x]
             if len(vals):
                 return vals.b.values[0]
             else:
                 return np.nan

         temp = tfs['b']
         temp['b_c'] = temp.target_id.apply(add_b)
```

```
In [57]: temp[(temp.b_c < 0) &
              (temp.b < 0)][['ext_gene','b_c','b', 'qval']]
```

Out[57]:

	ext_gene	b_c	b	qval
5522	mxl-3	-0.89506	-0.781183	0.002857

```
In [58]: # mxl-3 should be down in the egl, hif double:
         temp = thomas.beta_filtered['f']
         temp[(temp.ext_gene == 'mxl-3')][['ext_gene', 'b', 'qval']]
```

Out[58]:

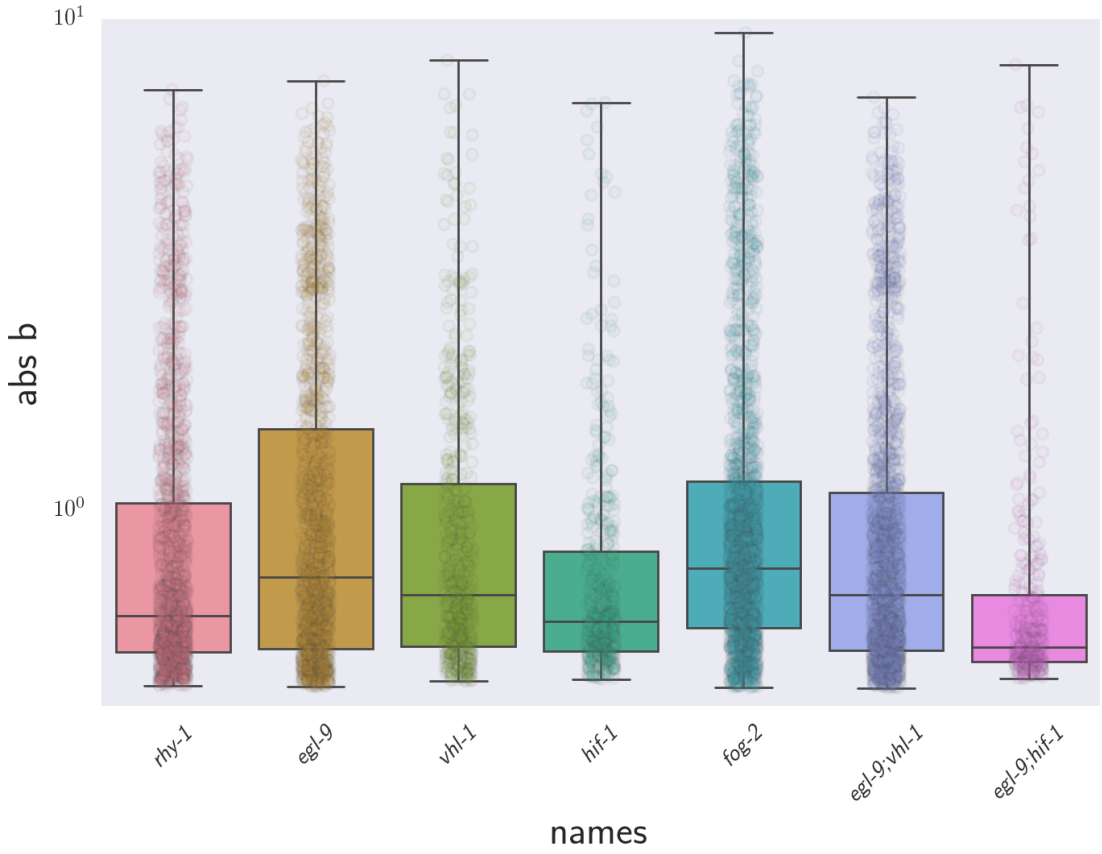
	ext_gene	b	qval
5522	mxl-3	-0.757522	0.020871

```
In [59]: temp = thomas.beta_filtered['a']
         temp[(temp.ext_gene == 'mxl-3')][['ext_gene', 'b', 'qval']]
```

Out[59]:

	ext_gene	b	qval
5522	mxl-3	-0.319002	0.404185

```
In [60]: temp = thomas.beta_filtered['e']
         temp[(temp.ext_gene == 'mxl-3')][['ext_gene', 'b', 'qval']]

Out[60]:      ext_gene         b      qval
         5522    mxl-3 -0.343135  0.388758

In [61]: temp = thomas.beta_filtered['d']
         temp[(temp.ext_gene == 'mxl-3')][['ext_gene', 'b', 'qval']]

Out[61]:      ext_gene         b      qval
         5522    mxl-3 -0.383277  0.465043

In [62]: bs = np.array([])
         for key in thomas.beta.keys():
             df = thomas.beta_filtered[key]
             if len(bs) == 0:
                 bs = df.b.values
                 qs = df.qval.values
             else:
                 bs = np.vstack((bs, df.b.values))
                 qs = np.vstack((qs, df.qval.values))

In [63]: b_df = pd.DataFrame(np.transpose(bs),
                             columns = thomas.beta.keys())
         q_df = pd.DataFrame(np.transpose(qs),
                             columns = thomas.beta.keys())
         b_df = pd.melt(b_df, var_name='genotype',
                        value_name='b')
         q_df = pd.melt(q_df, var_name='genotype',
                        value_name='q')
         b_df['abs b'] = b_df.b.abs()
         b_df['sorter'] = b_df.genotype.map(sorter)
         q_df['sorter'] = q_df.genotype.map(sorter)
         b_df.sort_values('sorter', inplace=True)
         q_df.sort_values('sorter', inplace=True)
         b_df['names'] = b_df.genotype.map(genotype_mapping)

In [64]: sns.boxplot(x="names", y="abs b",
                     data=b_df[q_df.q < 0.1],
                     whis=np.inf, linewidth=1)
         sns.stripplot('names', 'abs b',
                       data=b_df[q_df.q < 0.1],
                       linewidth=1, jitter=True,
                       alpha=0.05)
         plt.xticks(rotation=45)
         plt.ylim(10**-1, 10)
         plt.yscale('symlog')
```




```
In [65]: for genotype in b_df.genotype.unique():
             print(genotype,
                   b_df[(b_df.genotype == genotype) &
                        (q_df.q < 10**-1)].shape[0])
```

```
e 1924
b 1591
d 630
c 504
g 2098
a 2129
f 368
```

20 Finding direct targets of *hif-1*, *vhl-1*, *egl-9* and *rhy-1*

20.1 Hydroxylated *hif-1* direct targets

Knocking out *hif-1* should decrease levels of hydroxylated and non-hydroxylated *hif-1*
Knocking out *vhl-1* should increase levels of both forms.

Knocking out *egl-9* or *rhy-1* should decrease the hydroxylated form and increase the non-hydroxylated form.

Knocking out *egl-9;hif-1* should decrease levels of hydroxylated and non-hydroxylated *hif-1*

Therefore: Overlap *hif-1*, *egl-9* and *rhy-1* mutants with coexpression and find only targets that go DOWN. Next, overlap *vhl-1* with anti-expression. These are the hydroxylated targets of *hif-1* and possibly some other stuff

```
In [66]: df1 = thomas.beta['f'].copy()
         df2 = thomas.beta['c']
         df3 = thomas.beta['e']
         df4 = thomas.beta['b']
         df5 = thomas.beta['a']
         df6 = thomas.beta['d']

         df1['b_c'] = df2.b
         df1['b_e'] = df3.b
         df1['b_b'] = df4.b
         df1['b_a'] = df5.b
         df1['b_d'] = df6.b

         df1['se_b_c'] = df2.b
         df1['se_b_e'] = df3.b
         df1['se_b_b'] = df4.b
         df1['se_b_a'] = df5.b
         df1['se_b_d'] = df6.b

         df1['q_c'] = df2.qval
         df1['q_e'] = df3.qval
         df1['q_b'] = df4.qval
         df1['q_a'] = df5.qval
         df1['q_d'] = df6.qval

         ind = (df1.qval < 0.1) & (df1.q_c < 0.1) & (df1.q_e < 0.1) &\
               (df1.q_b < 0.1) & (df1.q_a < 0.1) & (df1.q_d < 0.1)

         ind2 = (df1.b < 0) & (df1.b_c < 0) & \
                (df1.b_e < 0) & (df1.b_b < 0) & (df1.b_a < 0)
         ind3 = (df1.b_d > 0)

         df1[ind & ind2 & ind3][['ext_gene', 'b_d', 'q_d']]

Out[66]:       ext_gene       b_d           q_d
         9850   F20D6.11  0.589394  3.392273e-03
         11479     cat-4  0.751071  1.325726e-08
```

We find two genes. . It's possible that we just couldn't measure the hydroxylated *hif-1* targets properly because A less stringent filter would be to just anti-overlap the *vhl-1* with the (*egl-9* and *rhy-1*) mutants and see what comes up. This means we are no longer measuring *hif-1* directly,

rather, we are inferring it and hoping that the noise from other stuff that these genes have in common gets drowned out.

```
In [67]: # significant in all relevant depts.
         ind = (df1.q_e < 0.1) & (df1.q_b < 0.1) &\
               (df1.q_a < 0.1) & (df1.q_d < 0.1)
         # loss of egl causes hif-OH to go down
         ind2 = (df1.b_e < 0) & (df1.b_b < 0) & (df1.b_a < 0)
         ind3 = (df1.b_d > 0) # vhl causes hif-OH to go UP
         hydroxylated_hif = df1[ind & ind2 & ind3]
         cols = ['ext_gene', 'b_d', 'q_d', 'b_e']
         hydroxylated_hif[cols].sort_values('q_d').head()
```

```
Out[67]:       ext_gene       b_d           q_d       b_e
         11479     cat-4  0.751071  1.325726e-08 -0.763864
         15045  F55G11.2  0.854800  1.708696e-03 -0.587116
         9850   F20D6.11  0.589394  3.392273e-03 -0.554782
         15953     asp-8  0.646281  6.687030e-02 -1.769170
```

Nope! It looks that even our less stringen filter yields quite the small list of targets!

20.2 Non-hydroxylated *hif-1* targets

Let's find the genes associated with non-hydroxylated hypoxia factor next.

Knocking out *hif-1* should decrease levels of hydroxylated and non-hydroxylated *hif-1*

Knocking out *vhl-1* should increase levels of both forms.

Knocking out *egl-9* or *rhy-1* should decrease the hydroxylated form and increase the non-hydroxylated form.

Knocking out *egl-9;hif-1* should decrease levels of hydroxylated and non-hydroxylated *hif-1*

Therefore: Overlap *vhl-1*, *egl-9* and *rhy-1* mutants with coexpression and find only targets that go UP. Next, overlap *hif-1* with anti-expression. These are targets of non-hydroxylated *hif-1* and possibly some other stuff

```
In [68]: ind = ((df1.qval < 0.1) & (df1.q_c < 0.1) &
                (df1.q_e < 0.1) & (df1.q_b < 0.1) &
                (df1.q_a < 0.1) & (df1.q_d < 0.1))
         ind2 = ((df1.b_e > 0) & (df1.b_b > 0) &
                 (df1.b_a > 0) & (df1.b_d > 0))
         ind3 = (df1.b < 0) & (df1.b_c < 0)

         df1[ind & ind2 & ind3][['ext_gene', 'b_e', 'q_e']].sort_values('q_e')
```

```
Out[68]:       ext_gene       b_e           q_e
         19626  R08E5.3  4.577666  0.000000e+00
         31611    nit-1  3.369282  4.598481e-32
```

Again, we observe only very few genes. Let's try to remove the *hif-1;egl-9* double mutant to increase our sample size:

```
In [69]: ind = ((df1.q_c < 0.1) & (df1.q_e < 0.1) &
                (df1.q_b < 0.1) & (df1.q_a < 0.1) &
                (df1.q_d < 0.1))
         ind2 = ((df1.b_e > 0) & (df1.b_b > 0) &
                 (df1.b_a > 0) & (df1.b_d > 0))
         ind3 = (df1.b_c < 0)
         df1[ind & ind2 & ind3][['ext_gene', 'b_e', 'q_e']].sort_values('q_e')
```

Out[69]:

	ext_gene	b_e	q_e
19626	R08E5.3	4.577666	0.000000e+00
26372	Y37A1B.5	2.590956	1.489199e-70
31611	nit-1	3.369282	4.598481e-32
20101	R12C12.5	0.488469	3.239911e-06
12288	hsp-12.3	1.473647	2.173847e-05

We can definitely say that these are the best candidates for direct control by *hif-1*. However, the list could probably still be larger.

Let's weaken the conditions a little bit. Now, we will only require that whatever our candidate genes are, they should not be **significantly upregulated in response to loss of hif-1**.

```
In [70]: ind =  ((df1.q_e < 0.1) & (df1.q_b < 0.1) &
                 (df1.q_a < 0.1) & (df1.q_d < 0.1))
         ind2 = ((df1.b_e > 0) & (df1.b_b > 0) &
                 (df1.b_a > 0) & (df1.b_d > 0))
         ind3 = (~((df1.q_c < 0.1) & (df1.b_c > 0)) &
                 ~((df1.qval < 0.1) & (df1.b > 0)))
         hypoxia_direct_targets = df1[ind & ind2 & ind3]
         print(hypoxia_direct_targets.shape[0])
         hypoxia_direct_targets[['ext_gene', 'b_e', 'q_e']].sort_values('q_e').head
```

```
133
```

Out[70]:

	ext_gene	b_e	q_e
19626	R08E5.3	4.577666	0.000000e+00
4509	C31C9.2	2.469312	1.688359e-99
10774	F26H9.5	1.599513	9.709034e-91
26372	Y37A1B.5	2.590956	1.489199e-70
21073	nhr-57	3.821550	1.190906e-57

Now we're talking! We can definitely say that these are the best candidates for direct control by *hif-1*.

If we are willing to let go of the difference between hydroxylated and non-hydroxylated *hif-1*, we could remove the *hif-1* filters entirely. However, I think these 167 genes are probably enough for now. Let's take a moment now to verify that this result matches what is known in the literature:

```
In [71]: ids = hypoxia_direct_targets.ens_gene.unique()
         _ = tea.enrichment_analysis(ids, tissue_df, show=True)
         _ = tea.enrichment_analysis(ids, phenotype_df, show=True)
```

```
Executing script

Analysis returned no enriched tissues.
Executing script

                                                 Tissue  Expected  Observed  \
0  neuron development variant WBPhenotype:0000816  0.374431         4

   Enrichment Fold Change   P value   Q value
0                10.68287  0.000033  0.007724
```

20.3 Quality control on identified genes:

Before we call these *hif-1* targets, we should make sure that at least some known targets are contained in this set. In order to do this, we have curated a list of 20 or so genes that have been published before as *hif-1* targets.

That being said though, we need to be aware of 1 major issue with this dataset.

hif-1 and *rhy-1* form an incredibly tight loop. There is a LOT of feedback between *hif-1* and *rhy-1*. Given the kind of logic we are using, we are probably excluding a number of targets as a result. In fact, the logic we have used to develop this list excludes *rhy-1* itself, a known *hif-1* target! I could do better, but not without a lot more lines of code, and it just doesn't seem reasonable to do this.

```
In [72]: ind = hypoxia_direct_targets.ens_gene.isin(hypoxia_gold.WBIDS)
         found = hypoxia_direct_targets[ind].ens_gene.unique()
         sig = len(hypoxia_direct_targets)
         pval = sts.hypergeom.sf(len(found),
                                 len(thomas.beta_filtered['a']),
                                 len(hypoxia_gold), sig)

         if pval < 10**-3:
             print('This result is statistically significant \
         with a p-value of {0:.2g} using a hypergeometric test.\
         You found {1} gold standard genes!'.format(pval, len(found)))

This result is statistically significant with a p-value of 4.2e-08 using a hyperged
```

Very nice! We are actually sampling from the hif-1 pool! Fantastic!

20.4 Identifying *rhy-1* targets

Next, let's identify rhy-1 associated genes. We will also insist that rhy-1 genes NOT be associated with hif-1 (they really shouldn't be, the logic between these two sets is mutually exclusive, I think, but it's best to make sure; let's hit stuff with a hammer):

```
In [73]: ind =  ((df1.q_e < 0.1) & (df1.q_b < 0.1) &
                 (df1.q_a < 0.1) & (df1.q_c < 0.1))
```

```
        ind2 = ((df1.b_e*df1.b_b > 0) & (df1.b_e*df1.b_a > 0) &
                (df1.b_c*df1.b_e > 0) & (df1.b_c*df1.b_b > 0))
        ind3 = (~df1.target_id.isin(hypoxia_direct_targets.target_id))
        rhy1_targets = df1[ind & ind2 & ind3]
        print(rhy1_targets.shape)
        rhy1_targets[['ext_gene', 'b_e', 'q_e']].sort_values('q_e').head()
```

```
(74, 30)
```

Out[73]:

	ext_gene	b_e	q_e
24825	rhy-1	1.482076	9.672920e-71
31240	ZC317.7	2.144840	9.204531e-63
3949	C26B9.3	2.686908	2.936705e-40
20836	T04A11.1	2.014769	5.957883e-35
20843	T04A11.4	2.014769	5.957883e-35

In [74]:
```
rhy1_targets[(rhy1_targets.b_e <= rhy1_targets.b_e.min() + 1.5)][['ext_ger
```

Out[74]:

	ext_gene	b_e
1082	C01G10.8	-0.505152
1581	C05C10.3	-0.378379
6216	C47G2.4	0.290649
6981	dnj-7	-0.620841
6992	acdh-1	-0.476948
8936	ldh-1	0.308663
9850	F20D6.11	-0.554782
11192	unc-45	-0.416793
11479	cat-4	-0.763864
11703	pfn-2	0.278972
12331	dnj-12	-0.529155
12385	F39G3.3	-0.446410
13120	NaN	-0.348950
14513	cct-3	-0.397977
14659	F54D5.12	-0.626779
16687	ran-1	-0.325141
18340	tra-3	-0.307254
20316	pfd-5	-0.517453
20809	lis-1	-0.341856
21101	cct-1	-0.371549
21106	dnj-19	-0.403246
21205	ech-6	-0.575509
21867	cct-7	-0.420124
22384	T15B7.1	-0.876063
22391	lgc-54	-1.190153
26108	Y23H5B.5	-0.380572
26660	Y38F1A.6	-0.573424
29202	arl-8	-0.263515

20.5 Identifying *egl-9* targets

OK! We found 'em. Let's find the *egl-9* related genes next.

```
In [75]: ind =  ((df1.q_e < 0.1) & (df1.q_b < 0.1) &
                 (df1.q_a < 0.1) & (df1.q_d > 0.1) &
                 (df1.q_c > 0.1))
         ind2 = ((df1.b_e*df1.b_b > 0) &
                 (df1.b_e*df1.b_a > 0) &
                 (df1.b_b*df1.b_a > 0))
         #         & (df1.b_c*df1.b_e < 0) & (df1.b_c*df1.b_b < 0)\
         #         & (df1.b_b*df1.b_d > 0) & (df1.b_e*df1.b_d > 0)
         ind3 = True
         # remember ids contains the hypoxia_direct_targets
         ind4 = (~df1.target_id.isin(ids))
         egl_targets = df1[ind & ind2 & ind3  & ind4]
         print(egl_targets.ens_gene.unique().shape[0])
         egl_targets[['ext_gene', 'b_b', 'q_b']].sort_values('q_b').head(10)
```

```
464
```

```
Out[75]:        ext_gene       b_b           q_b
         3558    C18H9.6 -3.021398  1.269703e-72
         17056    nas-33  3.102550  4.313509e-38
         25104  W10G11.3 -2.330393  8.491660e-34
         9265       far-3  4.401688  3.042296e-33
         15295      cdo-1  2.171999  6.303173e-32
         18213    nas-11  1.521469  7.037006e-32
         18219  K11G12.5  0.937802  9.660511e-32
         22260   T13F3.6 -3.477438  3.973725e-28
         1197       tyr-1  1.606652  3.979676e-28
         17862  cyp-35A3 -2.002178  4.214261e-23
```

```
In [76]: egl_ids = egl_targets.ens_gene.unique()
         _ = tea.enrichment_analysis(egl_ids,
                                     tissue_df, show=True)
         _ = tea.enrichment_analysis(egl_ids,
                                     phenotype_df, show=False)
         _[0].head(10)
```

```
Executing script

                              Tissue   Expected  Observed  \
0  anal depressor muscle WBbt:0004292  13.798814        30

   Enrichment Fold Change   P value   Q value
0                  2.1741  0.000011  0.002962
Executing script
```

Out[76]:

	Tissue	Expected	Observed
8	neuron morphology variant WBPhenotype:0000905	12.309859	56
11	vulval cell induction reduced WBPhenotype:0000219	29.154930	87
42	serotonin response variant WBPhenotype:0001232	7.690141	44
5	mid larval arrest WBPhenotype:0001019	12.985915	56
9	intestinal cell development variant WBPhenotyp...	4.225352	31
54	early larval lethal WBPhenotype:0000057	18.112676	61
28	RNA expression variant WBPhenotype:0000113	20.591549	61
10	coiling variant WBPhenotype:0002297	60.619718	111
36	embryonic cell morphology variant WBPhenotype:...	39.774648	84
55	transgene expression reduced WBPhenotype:0001278	58.394366	107

	Enrichment Fold Change	P value	Q value
8	4.549199	2.342003e-23	5.433448e-21
11	2.984058	5.636866e-23	6.538764e-21
42	5.721612	1.234956e-22	9.550330e-21
5	4.312364	3.505868e-22	2.033404e-20
9	7.336667	9.227203e-20	4.281422e-18
54	3.367807	1.724677e-18	6.668751e-17
28	2.962380	9.503627e-16	3.149773e-14
10	1.831087	7.114180e-13	2.063112e-11
36	2.111898	9.361201e-13	2.413110e-11
55	1.832369	2.407076e-12	5.584415e-11

20.6 Identifying *vhl-1* targets

Next, let's find the vhl-related genes.

In [77]:
```
ind =  (df1.q_a < 0.1) & (df1.q_d < 0.1)
ind2 = (df1.b_d > 0) & (df1.b_a > 0)
ind4 = ((df1.q_e > 0.1) & (df1.q_b > 0.1) &
        (df1.q_b > 0.1))

vhl_targets = df1[ind & ind2 & ind4]
print(vhl_targets.shape[0])
vhl_targets[['ext_gene', 'b_d', 'q_d', 'b_b', 'b_e']].sort_values('q_d').h
```

```
30
```

Out[77]:

	ext_gene	b_d	q_d	b_b	b_e
11227	F31C3.4	0.556422	1.653782e-08	-0.039171	0.069805
11226	F31C3.3	0.509236	5.936751e-07	0.108664	0.062222
24756	hgo-1	0.500935	3.375891e-06	0.099012	-0.092270
6937	mboa-3	0.506001	3.954259e-03	0.164309	-0.127737
11228	psf-2	0.592778	4.516616e-03	0.224087	0.094825

```
5991   C46C2.2  0.455548  4.771579e-03  0.255569 -0.016089
11606  F33H2.6  0.390757  6.374993e-03 -0.127979 -0.199501
31155   duxl-1  0.382592  6.407984e-03  0.136819  0.063909
11229  F31C3.6  0.576217  7.482708e-03  0.005797  0.059090
1132     cam-1  0.393836  1.444158e-02  0.201850  0.268705
```

```
In [78]: vhl_ids = vhl_targets.ens_gene.unique()
         _ = tea.enrichment_analysis(vhl_ids,
                                     tissue_df, show=True)
         _ = tea.enrichment_analysis(vhl_ids,
                                     phenotype_df, show=False)
         _[0].head(10)
```

```
Executing script

             Tissue  Expected  Observed  Enrichment Fold Change   P value  \
0  ADE WBbt:0005415  0.084783         2               23.589744  0.000069

    Q value
0  0.017912
Executing script
```

Out[78]:

	Tissue	Expected	Observed	Enrichment Fold Change	P value	Q value
0	early larval arrest WBPhenotype:0000055	0.107692	2	18.571429	0.000149	0.034665

20.7 Identifying New Biology - understanding the role of *rhy-1* and *egl-9* in the *hif-1* dependent response

```
In [79]: df1.head()
```

Out[79]:

	index	Unnamed: 0	target_id	pval	qval	b	se_b
1	11089	11090	2RSSE.1a	0.645621	1.000000	0.325045	0.706845
13	5648	5649	AC3.10	0.284737	0.941480	-0.422590	0.395040
17	225	226	AC3.2	0.000395	0.032654	-0.390064	0.110081
20	11093	11094	AC3.5a.1	0.692645	1.000000	0.271445	0.686737
21	2238	2239	AC3.5a.2	0.077470	0.645159	-2.343688	1.327448

	mean_obs	var_obs	tech_var	...	se_b_c	se_b_e	se_b_b
1	2.553185	1.452565	0.481532	...	-0.014713	0.778026	0.112480
13	3.090031	2.466209	0.056133	...	0.235891	0.057975	-0.267943
17	5.625880	0.054498	0.004335	...	0.005548	0.082966	-0.223640
20	5.133816	7.740471	0.686008	...	0.232033	0.153005	-0.164163
21	4.484407	2.582413	1.790455	...	-1.880010	-0.670477	-0.473572

```
      se_b_a    se_b_d       q_c       q_e       q_b       q_a       q_d
1   0.713097  0.090601  1.000000  0.697366  0.994068  0.683970  1.000000
13 -0.458696 -0.008479  0.942580  0.998112  0.872448  0.608039  1.000000
17 -0.057873  0.065041  1.000000  0.844141  0.259769  0.887128  0.988684
20 -0.004266 -0.078161  0.989014  0.993105  0.983249  0.999848  1.000000
21 -0.290185 -0.418795  0.683965  0.929625  0.962053  0.974710  1.000000

[5 rows x 30 columns]
```

In [91]:
```
ind = ((df1.q_c < 0.1) & (df1.q_e < 0.1) &
       (df1.q_b < 0.1) & (df1.q_a < 0.1) &
        (df1.q_d < 0.1))
y = df1[ind]
y.shape
```

Out[91]: (55, 30)

In [80]:
```
y_bs = df1[ind][['b_e', 'b_b', 'b_d', 'b_c', 'b', 'b_a']]

all_down = y_bs[y_bs<0].dropna().index
all_down.shape
```

Out[80]: (103,)

In [86]:
```
cols = ['ext_gene', 'ens_gene',
        'b_e', 'b_b', 'b_d', 'b_c', 'b_a',
        'q_e', 'q_b', 'q_d', 'q_c', 'q_a']
sel = df1.index.isin(y_bs[y_bs<0].index)
all_down = df1[sel][cols].sort_values('q_a')
all_down.to_csv('../output/all_down.csv', index=False)
```

In [87]:
```
all_up = y_bs[y_bs>0].dropna().index
print(all_up.shape)
all_up = df1[df1.index.isin(all_up)][cols].sort_values('q_a')
all_up.to_csv('../output/all_up.csv', index=False)
```

```
(127,)
```

In [92]:
```
# What pops up on hover?
tooltips = [('ext_gene', '@ext_gene'),
            ('egl_qval', '@q_b')]

# Make the hover tool
hover = bokeh.models.HoverTool(tooltips=tooltips, names=['circles'])

# Create figure
p = bokeh.plotting.figure(plot_width=650,
                          plot_height=450)
```

```
p.xgrid.grid_line_color = 'white'
p.ygrid.grid_line_color = 'white'
p.xaxis.axis_label = 'egl-9'
p.yaxis.axis_label = r'Delta'

# Add the hover tool
p.add_tools(hover)

# Define colors in a dictionary to access them with
# the key from the pandas groupby funciton.
source1 = bokeh.models.ColumnDataSource(y)


transformed_q = np.sqrt(-y.q_c.apply(np.log))
cols = [
    "#%02x%02x%02x" % (int(r), int(g), int(b))
    for r, g, b, _ in
    255*mpl.cm.viridis(mpl.colors.Normalize()(transformed_q))
]


# Specify data source
p.circle(x='b_b', y=y.b_a -y.b_b, size=7,
         alpha=0.4, source=source1, color=cols, name='circles')
p.circle(x='b_b', y=y.b_a -y.b_b, size=7,
         alpha=1, fill_color=None, color='black', source=source1)

# create the coordinates for the errorbars
errx_xs = []
errx_ys = []

for x, z, xerr in zip(y.b_b, y.b_a - y.b_b, y.se_b_b):
    errx_xs.append((x - xerr, x + xerr))
    errx_ys.append((z, z))

erry_xs = []
erry_ys = []
for x, z, yerr in zip(y.b_b, y.b_a - y.b_b,
                      np.sqrt(y.se_b_a**2 + y.se_b_b**2)):
    erry_xs.append((x, x))
    erry_ys.append((z - yerr, z + yerr))

# plot them
p.multi_line(errx_xs, errx_ys,color=cols, alpha=0.4)
p.multi_line(erry_xs, erry_ys, color=cols, alpha=0.4)

p.line([-10, 10], [0, 0], line_width=2, color='black')
```

```
        p.background_fill_color = "#DFDFE5"
        p.background_fill_alpha = 0.5
        html = file_html(p, CDN, "my plot")
        HTML(html)
```

```
Out[92]: <IPython.core.display.HTML object>
```

```
In [ ]: from bokeh.plotting import figure
        from bokeh.resources import CDN
        from bokeh.embed import file_html
        # Display graphics in this notebook
        bokeh.io.output_notebook()
```

```
In [93]: # What pops up on hover?
         tooltips = [('ext_gene', '@ext_gene'),
                     ('hif_qval', '@q_c')]

         # Make the hover tool
         hover = bokeh.models.HoverTool(tooltips=tooltips, names=['circles'])

         # Create figure
         title = 'egl-9 is suppressed by the egl-9;hif-1 double mutant'
         p = bokeh.plotting.figure(title=title, plot_width=650,
                                   plot_height=450)


         p.xgrid.grid_line_color = 'white'
         p.ygrid.grid_line_color = 'white'
         p.xaxis.axis_label = 'egl-9'
         p.yaxis.axis_label = r'Delta'

         # Add the hover tool
         p.add_tools(hover)

         # Define colors in a dictionary to access them with
         # the key from the pandas groupby funciton.
         source1 = bokeh.models.ColumnDataSource(y)

         transformed_q = np.sqrt(-y.q_c.apply(np.log))
         cols = [
             "#%02x%02x%02x" % (int(r), int(g), int(b))
             for r, g, b, _ in
             255*mpl.cm.viridis(mpl.colors.Normalize()(transformed_q))
         ]


         # Specify data source
         p.circle(x='b_b', y=y.b -y.b_b, size=7,
```

```
                alpha=0.4, source=source1,
                color=cols, name='circles')
        p.circle(x='b_b', y=y.b -y.b_b, size=7,
                alpha=1, fill_color=None,
                color='black', source=source1)

        # create the coordinates for the errorbars
        errx_xs = []
        errx_ys = []

        for x, z, xerr in zip(y.b_b, y.b - y.b_b, y.se_b_b):
            errx_xs.append((x - xerr, x + xerr))
            errx_ys.append((z, z))

        erry_xs = []
        erry_ys = []
        for x, z, yerr in zip(y.b_b, y.b - y.b_b,
                              np.sqrt(y.se_b**2 + y.se_b_b**2)):
            erry_xs.append((x, x))
            erry_ys.append((z - yerr, z + yerr))

        # plot them
        p.multi_line(errx_xs, errx_ys, color=cols, alpha=0.4)
        p.multi_line(erry_xs, erry_ys, color=cols, alpha=0.4)

        p.line([-10, 10], [0, 0], line_width=2, color='black')
        p.line([-10, 10], [10, -10], line_width=1,
               line_dash=(4, 4), color='red')
        # p.circle(x='avg_b', y='logq', size=7,
        #          alpha=0.2, source=source2, color='red')
        p.background_fill_color = "#DFDFE5"
        p.background_fill_alpha = 0.5
        html = file_html(p, CDN, "my plot")
        HTML(html)
```

Out[93]: <IPython.core.display.HTML object>