1 Basic Statistics

January 31, 2018

1 Table of Contents

1 Introduction
1.1 Data initialization
2 Isoforms Identified in all Genotypes
3 Differentially Expressed Genes per genotype
4 Pairwise shared transcriptomic phenotypes
4.1 SI Table 1

2 Introduction

In this notebook, I will go over the basic results from the RNA-seq in what is essentially a top-level view of the results. Nothing specific, mainly numbers, some histograms and that's it. First, I will load a number of useful libraries. Notable libraries to load are `genpy`, a module that contains useful graphing functions tailored specifically for this project and developed by us; `morgan` a module that specifies what a Morgan object and a McClintock object are, and `gvars`, which contains globally defined variables that we used in this project.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

import morgan as morgan
import genpy
import gvars
import pretty_table as pretty
import epistasis as epi

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc

rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
```

```
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%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")

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['legend.fontsize'] = 14
```

Next, I will specify my q-value cutoff. A typical value for RNA-seq datasets is q=0.1 for statistical significance. I will also initialize a `genvar.genvars` object, which contains all of the global variables used for this project.

```
In [2]: q = 0.1
        # this loads all the labels we need
        genvar = gvars.genvars()
```

2.1 Data initialization

Now, I will prepare to initialize a Morgan project. Morgan objects have a large number of attributes. I wrote the Morgan library, but over the past year it has become deprecated and less useful. We will load it here, but it's a bit messy. I am in the process of cleaning it up. When you initialize a Morgan object, you must pass at least a set of 4 strings. These strings are, in order, the column where the isoform names (unique) reside, the name of the column that holds the regression coefficient from sleuth; the name of the column that holds the TPM values passed by Kallisto and the name of the column that holds the q-values.

We can also add what I call a genmap. A genmap is a file that maps read files to genotypes. A genmap file has three columns: *'project_name'*, *'genotype'* and *'batch'* in that exact order. For this project, the genotypes are coded. In other words, they are letters, *'a'*, *'b'*, *'d'*,... and not specific genotypes. The reason for this is that we wanted to make sure that, at least during the initial phase of the project, I could not unduly bias the results by searching the literature and what not. Because the genotypes are coded, we need to specify which of the letters represent single mutants, and which letters represent double mutants. I also need to be able to figure out what the individual components of a double mutant are. Finally, we need to set the q-value threshold. If no q-value is specified, the threshold defaults to 0.1.

I will now initialize the object. I call it thomas. Then I will load in all the variables we will use; I will load in the genmap, and at last I will load in the datasets that contain the TPM and the Sleuth

β coefficients. After everything has been loaded, I will call `thomas.filter_data`, which drops all the rows that have a β coefficient equal to NaN

```
In [3]: # Specify the genotypes to refer to:
        single_mutants = ['b', 'c', 'd', 'e', 'g']

        # Specify which letters are double mutants and their genotype
        double_mutants = {'a' : 'bd', 'f':'bc'}

        # initialize the morgan.hunt object:
        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/'
        sleuth_loc = '../sleuth/kallisto/'
        thomas.add_tpm(kallisto_loc, '/kallisto/abundance.tsv', '')
        # load all the beta dataframes:
        for file in os.listdir("../sleuth/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)
                thomas.filter_data()

        # thomas.filter_data()
```

Finally, we will place all the data in a tidy dataframe, where each row is an observation.

```
In [4]: frames = []
        for key, df in thomas.beta.items():
            df['genotype'] = genvar.mapping[key]
            df['code'] = key
            df['sorter'] = genvar.sort_muts[key]
            df.sort_values('target_id', inplace=True)
            frames += [df]

        tidy = pd.concat(frames)
        tidy.dropna(subset=['ens_gene'], inplace=True)
```

```
        # Save to table
        tidy[['ens_gene', 'ext_gene', 'target_id', 'b', 'se_b',
              'qval', 'genotype', 'sorter',
              'code']].to_csv('../output/temp_files/DE_genes.csv', index=False)

        tidy.sort_values('sorter', inplace=True)
```

3 Isoforms Identified in all Genotypes

```
In [5]: total_genes_id = tidy.target_id.unique().shape[0]
        print("Total isoforms identified in total: {0}".format(total_genes_id))
```

```
Total isoforms identified in total: 19676
```

We identified 19,676 isoforms using 7 million reads. Not bad considering there are ~25,000 protein-coding isoforms in *C. elegans*. Each gene has just slightly over 1 isoform on average, so what this means is that we sampled almost 80% of the genome.

4 Differentially Expressed Genes per genotype

Next, let's figure out how many *genes* were differentially expressed in each mutant relative to the wild-type control.

```
In [6]: print('Genotype: DEG')
        for x in tidy.genotype.unique():
            # select the DE isoforms in the current genotype:
            sel = (tidy.qval < q) & (tidy.genotype == x)
            # extract the number of unique genes:
            s = tidy[sel].ens_gene.unique().shape[0]
            print("{0}: {1}".format(x, s))
```

```
Genotype: DEG
rhy-1: 3005
egl-9: 2549
vhl-1: 1275
hif-1: 1075
fog-2: 2840
egl-9;vhl-1: 3654
egl-9 hif-1: 744
```

From the above exploration, we can already conclude that: * *hif-1(lf)* has a transcriptomic phenotype * *hif-1;egl-9(lf)* has a transcriptomic phenotype * The *egl-9* phenotype is stronger than the *vhl-1* or the *hif-1* phenotypes.

We should be careful is saying whether *rhy-1*, *egl-9* and *egl-9;vhl-1(lf)* are different from each other, and the same goes for *hif-1(lf)*, *vhl-1(lf)* and *egl-9;hif-1(lf)* because we set our FDR threshold at 10%. Notice that *egl-9(lf)* and *rhy-1(lf)* are barely 300 genes separated from each other. A bit of wiggle from both, and they might be identical.

5 Pairwise shared transcriptomic phenotypes

5.1 SI Table 1

In order to be able to assess whether two genes are interacting, we must first determine that the mutants we are studying act upon the same phenotype. What defines a phenotype in transcriptomic space? We use an operational definition -- two genotypes share the same phenotype if they regulate more than a pre-specified(and admittedly subjective) number of genes in common between the two of them, agnostic of direction. In our paper, we call this the Shared Transcriptomic Phenotype (STP). Let's figure out to what extent the genes we have studied share the same phenotype.

We will measure the size of the STP using two distinct definitions. The first, percent shared isoforms, is defined as the number of isoforms in the STP divided by the number of differentially expressed isoforms in EITHER of the two mutants being compared. The second measurement, percent internalization, is defined as the number of isoforms in the STP divided by the number of differentially expressed isoforms in the mutant that has the smallest number of differentially expressed isoforms out of the two being compared.

```
In [8]: sig = (tidy.qval < q)
        string = 'pair,STP,% shared,% internalization'

        # print table header
        l = string.split(',')
        pretty.table_print(l, space=20)

        # print rest:
        for i, g1 in enumerate(tidy.genotype.unique()):
            for j, g2 in enumerate(tidy.genotype.unique()[i+1:]):
                tmp = tidy[sig]  # define a temporary dataframe with only DE genes in it

                # find DE genes in either genotype
                DE1 = tmp[tmp.genotype == g1]
                DE2 = tmp[tmp.genotype == g2]

                # find the overlap between the two genotypes:
                overlap = epi.find_overlap([g1, g2], df=tidy, col='genotype')
                n = len(overlap)  # number of DE isoforms in both genotypes
                genes_in_stp = tidy[tidy.target_id.isin(overlap)].ens_gene.unique()
                n_genes_stp = len(genes_in_stp)  # number of DE genes in both genotypes

                # find total number of DE transcripts in either genotype
                OR = ((tmp.genotype == g1) | (tmp.genotype == g2))
                ntot = tmp[OR].target_id.shape[0]

                # find which genotype has fewer DE transcripts
                n_intern = np.min([DE1.shape[0], DE2.shape[0]])

                # print
                string = "{0} & {1},{2},{3:.2g}%,{4:.2g}%".format(g1, g2, n_genes_stp, 100*n/nt
```

```
                l = string.split(',')
                pretty.table_print(l, space=20)
```

```
pair                    STP                 % shared            % internalization
rhy-1 & egl-9           1808                32%                 70%
rhy-1 & vhl-1           879                 20%                 69%
rhy-1 & hif-1           456                 11%                 42%
rhy-1 & fog-2           839                 14%                 29%
rhy-1 & egl-9;vhl-1     1730                26%                 57%
rhy-1 & egl-9 hif-1     484                 13%                 64%
egl-9 & vhl-1           872                 23%                 68%
egl-9 & hif-1           387                 10%                 36%
egl-9 & fog-2           782                 14%                 30%
egl-9 & egl-9;vhl-1     1872                30%                 73%
egl-9 & egl-9 hif-1     415                 12%                 54%
vhl-1 & hif-1           296                 12%                 27%
vhl-1 & fog-2           450                 11%                 35%
vhl-1 & egl-9;vhl-1     971                 19%                 76%
vhl-1 & egl-9 hif-1     323                 16%                 43%
hif-1 & fog-2           361                 8.8%                33%
hif-1 & egl-9;vhl-1     494                 10%                 46%
hif-1 & egl-9 hif-1     161                 8.9%                22%
fog-2 & egl-9;vhl-1     1069                16%                 37%
fog-2 & egl-9 hif-1     247                 6.6%                32%
egl-9;vhl-1 & egl-9 hif-1     535                               12%                                 70%
```

The number of genes that is shared between mutants of the same pathway ranges from ~100 genes all the way to ~1,300. However, the hypoxia mutants share between ~140 and ~700 genes in common with another mutant, the *fog-2(lf)* mutant that has never been reported to act in the hypoxia pathway. What are we to make of this? My own conclusion is that *fog-2* probably interacts with effectors downstream of the hypoxia pathway.

2 Predicting Interactions

January 31, 2018

1 Table of Contents

1 A first pass at genetic interactions
1.1 Loading the data
2 PCA
2.1 Figure 3
3 Visualizing STPs

In [ ]:

2 A first pass at genetic interactions

In this notebook, we focus on developing the idea that whole-organism RNA-seq contains sufficient information to predict interactions between genes, and we will make some graphs, namely a PCA graph, that motivates the idea that epistasis can be measured genome-wide. First, I will load a number of useful libraries. Notable libraries to load are `genpy`, a module that contains useful graphing functions tailored specifically for this project and developed by us; `morgan` a module that specifies what Morgan and McClintock objects are, and `gvars`, which contains globally defined variables that we used in this project.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

import genpy
import gvars
import morgan as morgan

# stats
import sklearn.decomposition
import statsmodels.api as stm

# network graphics
import networkx as nx
```

```
# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# mcmc
import pymc3 as pm

# Magic function to make matplotlib inline;
%matplotlib inline

# This enables SVG graphics inline.
%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")

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['legend.fontsize'] = 14
```

2.1 Loading the data

In the next cell, I will specify my q-value threshold and load the data. Finally, I will prepare to initialize a Morgan project. Morgan objects have a large number of attributes. I wrote the Morgan library, but over the past year it has become deprecated and less useful. We will load it here, but it's a bit messy. I am in the process of cleaning it up. So what does a Morgan object do? Well, when you initialize a Morgan object, you must pass at least a set of 4 strings. These strings are, in order, the column where the isoform names (unique) reside, the name of the column that holds the regression coefficient from sleuth; the name of the column that holds the TPM values passed by Kallisto and the name of the column that holds the q-values.

We can also add what I call a `genmap`. A genmap is a file that maps read files to genotypes. A genmap file has three columns: '`project_name`', '`genotype`' and '`batch`' in that exact order. For this project, the genotypes are coded. In other words, they are letters, '*a*', '*b*', '*d*', ... and not specific genotypes. The reason for this is that we wanted to make sure that, at least during the initial phase of the project, I could not unduly bias the results by searching the literature and what not. Because the genotypes are coded, we need to specify which of the letters represent single mutants, and which letters represent double mutants. I also need to be able to figure out what the individual components of a double mutant are. Finally, we need to set the q-value threshold. If no

q-value is specified, the threshold defaults to 0.1.

I will now initialize the object. I call it thomas.

```
In [2]: q = 0.1
        genvar = gvars.genvars()

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

Ok. Our Morgan object is up and running, but it doesn't have any data yet. So now, we need to specify where the object can look for the Sleuth outputs (`sleuth_loc`)and the Kallisto outputs (`kallisto_loc`). After we have specified these directories, we just let thomas loose in the directories. We will load the files into dictionaries: `{g1: df_beta1,..., gn: df_betan}`

```
In [3]: tidy_data = pd.read_csv('../output/temp_files/DE_genes.csv')
        tidy_data.dropna(subset=['ens_gene'], inplace=True)
        tidy_data.dropna(subset=['b'], inplace=True)
        tidy_data['fancy_genotype'] = tidy_data.code.map(genvar.fancy_mapping)
```

3 PCA

Now we will perform an exploratory procedure, PCA, to demonstrate that transcriptomic signatures from whole-organism RNA-seq have valuable information regarding genetic interactions. First, I will identify the set of genes that is differentially expressed in at least one genotype. Then, for each genotype I will find what β values have an associated q-value that is significant and which ones are not. Set all β values with $q > 0.1$ equal to 0. Finally, we will standardize each genotype so that the collection β values for each genotype has a mean of zero and standard deviation of 1.

```
In [4]: ID_in_all = []
        for tx in tidy_data[tidy_data.qval < q].target_id.unique():
            l = tidy_data[tidy_data.target_id == tx].shape[0]
            if l == len(tidy_data.code.unique()):
                ID_in_all += [tx]

In [5]: print('There are {0} isoforms DE\
               in at least one genotype'.format(len(ID_in_all)))

        grouped = tidy_data.groupby('code')
        bvals = np.array([])
        labels = []
        for code, group in grouped:
            # find names:
            names = group.target_id.isin(ID_in_all)
            # extract (b, q) for each gene
            bs = group[names].b.values
            qs = group[names].qval.values
```

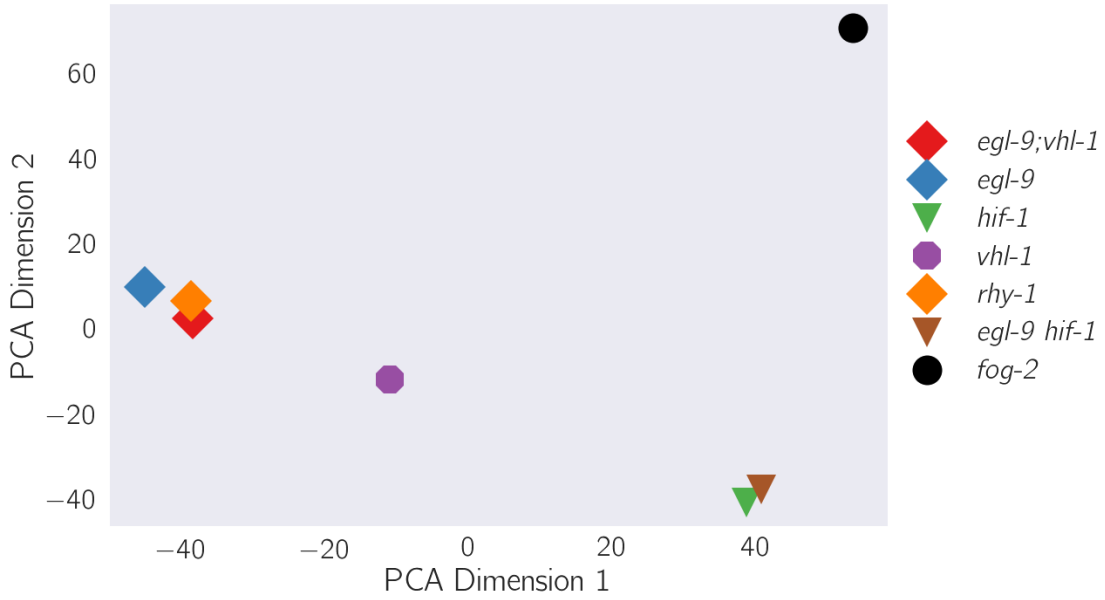
```
            # find sig genes:
            inds = np.where(qs > q)
            # set non-sig b values to 0
            bs[inds] = 0
            #  standardize bs
            bs = (bs - bs.mean())/(bs.std())

            # place in array
            if len(bvals) == 0:
                bvals = bs
            else:
                bvals = np.vstack((bvals, bs))
            # make a label array
            labels +=  [code]
```

```
There are 7609 isoforms DE       in at least one genotype
```

3.1 Figure 3

Next, initialize the PCA object, specifying that we want to project the data onto two axes. Finally, we plot.

```
In [6]: # initialize the PCA object and fit to the b-values
        sklearn_pca = sklearn.decomposition.PCA(n_components=2).fit(bvals)
        coords = sklearn_pca.fit(bvals).transform(bvals)

        colors = ['#e41a1c', '#377eb8', '#4daf4a',
                  '#984ea3', '#ff7f00', '#a65628', 'k']
        shapes = ['D', 'D', 'v', '8', 'D', 'v', 'o']

        # go through each pair of points and plot them:
        for i, array in enumerate(coords):
            l = genvar.fancy_mapping[labels[i]]
            plt.plot(array[0], array[1], shapes[i], color=colors[i],
                     label=l, ms=17)

        # plot prettify:
        plt.legend(loc=(1, 0.25), fontsize=16)
        plt.xlabel('PCA Dimension 1')
        plt.ylabel('PCA Dimension 2')
        plt.savefig('../output/PCA_genotypes.svg', bbox_inches='tight')
```



We can see that the diamonds all cluster together and triangles cluster together. The triangles are HIF-1^{-} genotypes, whereas the diamonds (and purple octagon) are HIF-1^{+} genotypes. The *fog-2* mutant is far away from genes in this pathway. The closeness of the *egl-9;hif-1(lf)* mutant to the *hif-1* double mutant suggests to me that epistasis can be measured genome-wide.

4 Visualizing STPs

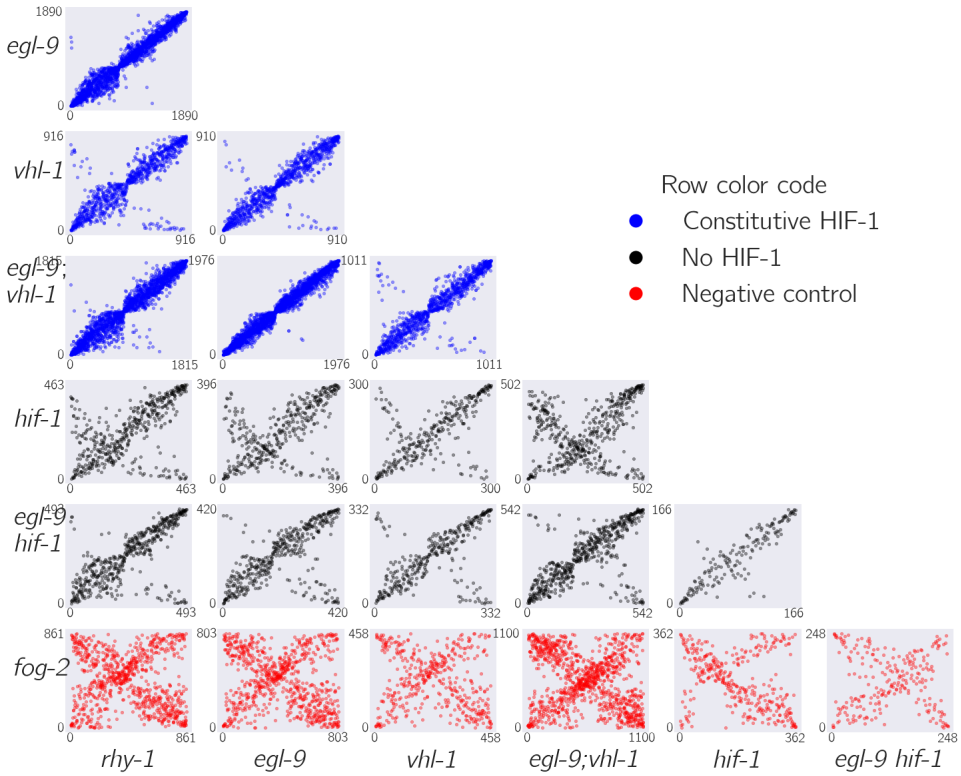
```
In [7]: # the genotypes to compare
        def find_overlap(genotypes, df, col='code', q=q):
            sig = tidy_data[(tidy_data[col].isin(letters)) & (tidy_data.qval < q)]
            grouped = sig.groupby('target_id')
            genes = []

            # find the intersection between the two.
            for target, group in grouped:
                # make sure the group contains all desired genotypes
                all_in = (len(group[col].unique()) == 2)
                if all_in:
                    genes += [target]
            return genes

        # extract a temporary dataframe with all the desired genes
        letters = ['e', 'b']  # rhy-1 and egl-9
        genes = find_overlap(letters, tidy_data)
        temp = tidy_data[tidy_data.target_id.isin(genes)]

        # split the dataframes and find the rank of each gene
```

```
        ovx = genpy.find_rank(temp[temp.code == letters[0]])
        ovy = genpy.find_rank(temp[temp.code == letters[1]])
```

In [8]:
```
        # modify the sorting order just once to prettify the upcoming figure
        sorter = gvars.genvars()
        sorter.sort_muts['g'] = 10  # set to some large value so fog-2 is sorted last
        sorter.sort_muts['c'] = 7  # set so hif-1 and egl-hif show up together
        sorter.sort_muts['f'] = 8  # set so hif-1 and egl-hif show up together
```

In [9]:
```
        mpl.rcParams['xtick.labelsize'] = 11
        mpl.rcParams['ytick.labelsize'] = 11
        rc = {'axes.labelsize': 20}
        sns.set_context('notebook', rc=rc)

        nplots = len(tidy_data.code.unique())
        fig, ax = plt.subplots(nrows=nplots-1, ncols=nplots, figsize=(14,10))

        tidy_data['sort'] = tidy_data.code.map(sorter.sort_muts)
        tidy_data.sort_values('sort', inplace=True)

        for col, a in enumerate(tidy_data.fancy_genotype.unique()):
            for j, b in enumerate(tidy_data.fancy_genotype.unique()[col+1:]):
                row = col + j
                letters = [a, b]
                genes = find_overlap(letters, tidy_data, col='fancy_genotype')
                if len(genes) == 0:
                    raise ValueError('list is empty')
                temp = tidy_data[tidy_data.target_id.isin(genes)]

                # split the dataframes and find the rank of each gene
                ovx = genpy.find_rank(temp[temp.fancy_genotype == letters[0]])
                ovy = genpy.find_rank(temp[temp.fancy_genotype == letters[1]])

                # plot
                if b == '\\emph{egl-9 hif-1}' or b == '\\emph{hif-1}':
                    ax[row, col].scatter(ovx.r, ovy.r, s=5, alpha=0.4, color='k')
                elif b != '\\emph{fog-2}':
                    ax[row, col].scatter(ovx.r, ovy.r, s=5, alpha=0.4, color='blue')
                else:
                    ax[row, col].scatter(ovx.r, ovy.r, s=5, alpha=0.4, color='red')

                # set row labels
                ylabel = b.replace(' ', '}\n\emph{').replace(';', '};\n\emph{')
                ax[row, 0].set_ylabel(ylabel, rotation=0, labelpad=22,
                                      horizontalalignment='left')

                # remove tick labels for cleanliness
                ax[row, col].xaxis.set_ticks([0, ovx.r.max()])
```

```
        ax[row, col].yaxis.set_ticks([0, ovx.r.max()])
        ax[row, col].tick_params(axis='both', which='major', pad=.2)

    # set column labels
    ax[nplots-2, col].set_xlabel(a)

# remove upper triangle plots
for row in range(nplots+1):
    for col in range(row, nplots):
        if col > row:
            ax[row, col].axis('off')

# add a legend:
texts = ["Constitutive HIF-1", "No HIF-1",
         'Negative control']
types = ['bo', 'ko', 'ro']
patches = [plt.plot([], [], types[i], ms=10, ls="",
           label="{:s}".format(texts[i]))[0] for i in range(len(types))]

# adjust legend size:
mpl.rcParams['legend.fontsize'] = 20

# draw legend
legend = plt.legend(handles=patches, loc=(-3, 4), ncol=1,
                    numpoints=1, title='Row color code')
plt.setp(legend.get_title(), fontsize='20')

# plt.savefig('../output/rank_plots/triangle_plot.svg', bbox_inches='tight')

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
```


egl-9
vhl-1
egl-9; vhl-1
hif-1
egl-9 hif-1
fog-2
rhy-1
egl-9
vhl-1
egl-9;vhl-1
hif-1
egl-9 hif-1
Row color code
Constitutive HIF-1
No HIF-1
Negative control

In [ ]:

3 Enrichment Analysis of Hypoxia Pathway Data

January 31, 2018

1 Table of Contents

1 Defining the hypoxia response
2 Enrichment Analysis of the Global HIF-1 response

In this notebook, we will isolate the hypoxia response (defined as the set of genes that fulfill the genetic equalities *egl-9 = egl9;vhl-1* and *hif-1 = egl-9 hif-1*), and we will perform enrichment analysis on the hypoxia response. We will also perform enrichment analyses on each mutant transcriptomes, to try to understand how different each transcriptome actually is.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

# TEA and morgan
import tissue_enrichment_analysis as tea
import morgan as morgan
import gvars
import epistasis as epi

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif',
              'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%matplotlib inline

# This enables SVG graphics inline.
%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")

        mpl.rcParams['xtick.labelsize'] = 16
        mpl.rcParams['ytick.labelsize'] = 16
        mpl.rcParams['legend.fontsize'] = 14
```

```
In [2]: q = 0.1
        # this loads all the labels we need
        genvar = gvars.genvars()

        tissue_df = tea.fetch_dictionary()
        phenotype_df = tea.fetch_dictionary('phenotype')
        go_df = tea.fetch_dictionary('go')
        respiratory_complexes = pd.read_excel('../input/respiratory_complexes.xlsx')
```

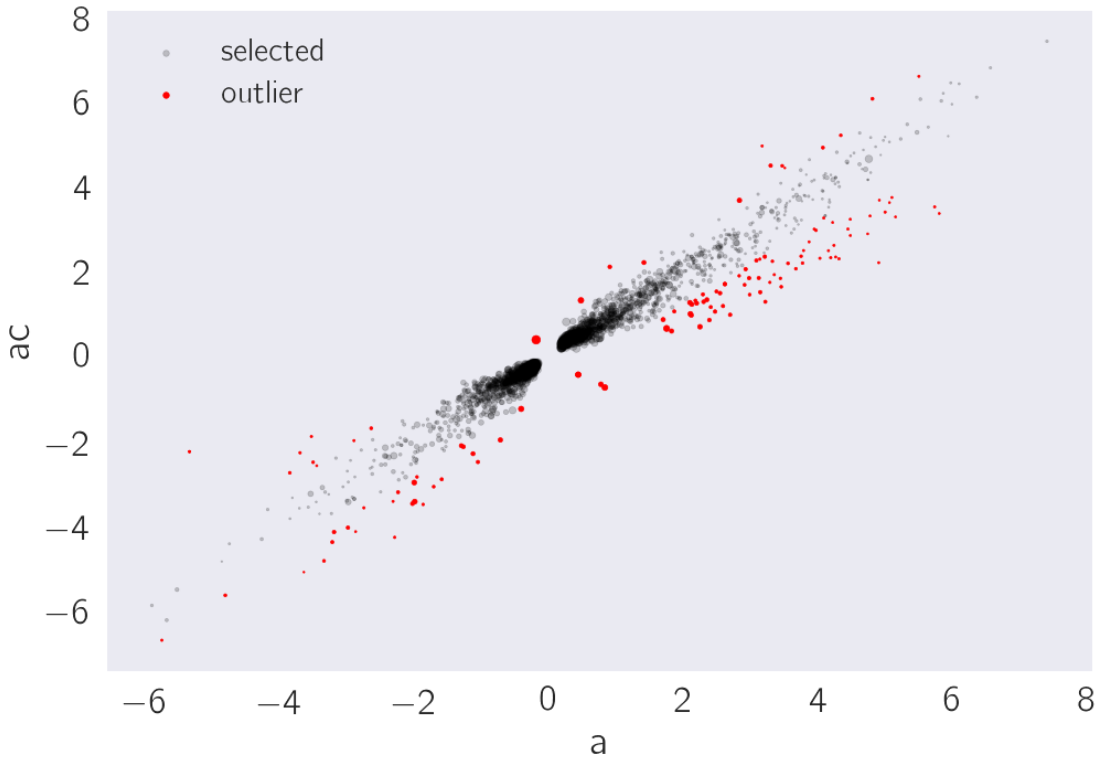
```
In [3]: tidy = pd.read_csv('../output/temp_files/DE_genes.csv')
        tidy.sort_values('target_id', inplace=True)
        tidy.dropna(subset=['ens_gene'], inplace=True)
```

2 Defining the hypoxia response

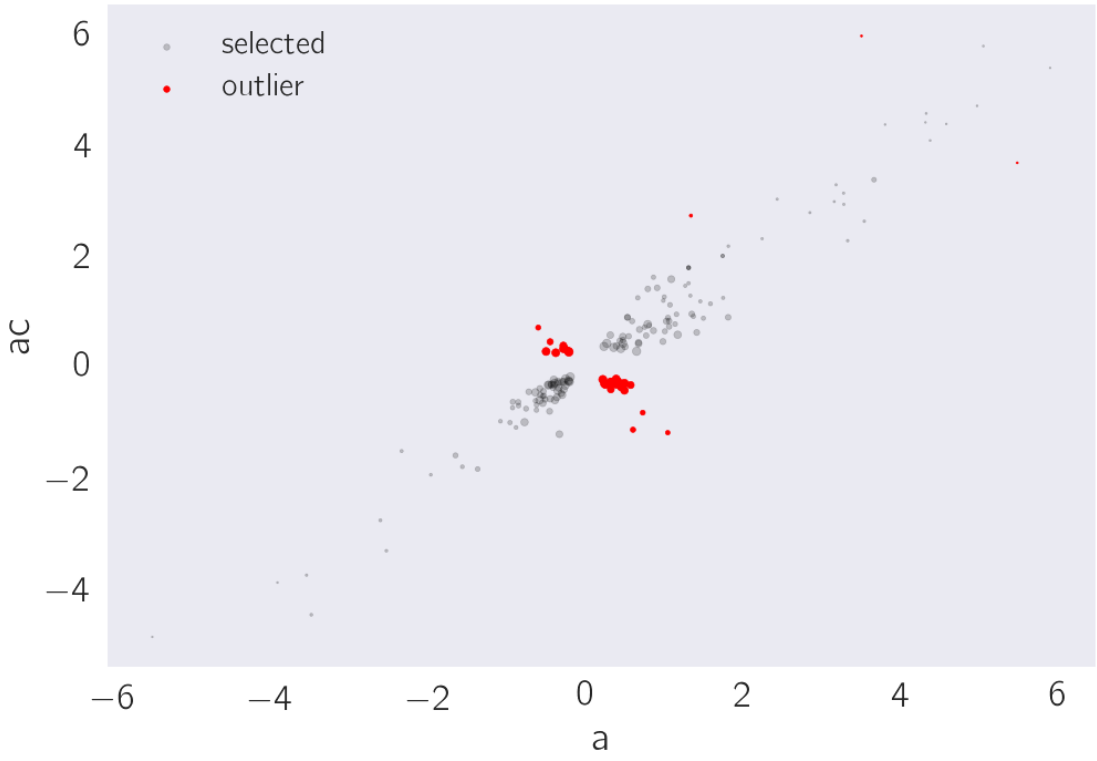
The hypoxia response can be defined in genetic terms as those genes that obey the two epistasis relationships, *egl-9 = egl-9;vhl-1* and *hif-1 = egl-9 hif-1*.

```
In [4]: def test_equality(equal_genotypes, third_genotype, df, col='code', q=0.1, n_std=2):
            """
            A function to test epistasis equality.

            For a set of genotypes, `a`, `b`, and `ab`, suppose that we want to find those
            genes that obey the rule `a`=`ab`. To identify genes with this expression
            pattern, we first calculate the epistasis coefficient for transcripts within
            the STP(`a`, `ab`). Then, we find those transcripts that are <2sigma
            deviations away from the line of best fit.

            Params:
            equal_genotypes: the two genotypes that we want to set equal to each other
            third_genotype: the third genotype to be considered (needed to calculate
            epistasis coeff.).
            df - dataframe to use. Must contain `target_id` column
            col - column that encodes the genotypes
            q - q-value to be used
            n_std - number of standard deviations to use as cutoff
```

```
        Output:
        A list of target_ids
        """
        a, ac = equal_genotypes
        c = third_genotype

        # make sure the dataframe only contains the desired genotypes
        all_genotypes = [a, ac, c]

        df = df[df[col].isin(all_genotypes)]

        overlap = epi.find_overlap(equal_genotypes, df, col=col, q=q)

        df = df[df.target_id.isin(overlap)]
        a_df = df[df[col] == a].copy()
        c_df = df[df[col] == c]
        ac_df = df[df[col] == ac]

        # the code below works only if the variance is invariant to expected value
        normed_deltas = (ac_df.b.values - a_df.b.values)
        normed_deltas = normed_deltas/np.std(normed_deltas)

        # first condition guarantees we're not too far from the line y=x
        # second condition guarantees we are not on the line y=-x
        inside = (np.abs(normed_deltas) < n_std) & (ac_df.b.values*a_df.b.values > 0)

        # print a diagnostic plot:
        plt.scatter(a_df[inside].b, ac_df[inside].b, s=1/ac_df[inside].se_b,
                    color='black', alpha=.2, label='selected')
        plt.scatter(a_df[~inside].b, ac_df[~inside].b, s=1/ac_df[~inside].se_b,
                    color='red', alpha=1, label='outlier')
        plt.xlabel('a')
        plt.ylabel('ac')
        plt.legend()

        # return list of target ids that meet criteria
        return a_df[inside].target_id.values


In [5]: # find the genes that obey egl-9 = egl-9;vhl-1
        filtered_egl = test_equality(['b', 'a'], 'd', tidy, n_std=2)
```



In [6]:
```
# find the genes that obey hif-1 = egl-9 hif-1
filtered_hif = test_equality(['c', 'f'], 'b', tidy, q=.1, n_std=2)
```



In [7]:
```
# find those genes that are not DE in either hif-1 or egl-9 hif-1
not_DE_hif = (tidy.code.isin(['c', 'f'])) & (tidy.qval > q)
# a neat trick:
not_DE = epi.find_overlap(['c', 'f'], tidy[not_DE_hif], q=1)
```

In [8]:
```
# genes DE in hif-1 and egl-9, and obey both equations:
equal_and_DE = ((tidy.target_id.isin(filtered_egl)) &
                (tidy.target_id.isin(filtered_hif)))
# genes that are DE in egl-9, but not in hif-1 genotypes
# and also obey both equations:
equal_no_hif = ((tidy.target_id.isin(filtered_egl)) &
                (tidy.target_id.isin(not_DE)))

# get the lists of both, then concatenate them for a
# hypoxia response: most of the genes will come from
# the equal_no_hif condition
de_both = tidy[(equal_and_DE)].target_id.unique()
de_one = tidy[(equal_no_hif)].target_id.unique()
overlap = list(set(np.append(de_both, de_one)))

# find the hypoxia response
hyp_response = tidy[tidy.target_id.isin(overlap)].copy()
```

```
In [9]: # annotate whether they are candidates for direct or
        # indirect regulation.
        def annotate(x):
            if x > 0:
                return 'candidate for direct regulation'
            else:
                return 'candidate for indirect regulation'

        # annotate
        hyp_response['regulation'] = hyp_response.b.apply(annotate)
```

```
In [10]: # save to file
         cols = ['genotype','target_id', 'ens_gene', 'ext_gene', 'b',
                 'se_b', 'qval', 'regulation', 'code']
         hyp_response[cols].to_csv('../output/temp_files/hypoxia_response.csv',
                                   index=False)

         m = 'There are {0} genes in the predicted hypoxia response'
         print(m.format(len(hyp_response.ens_gene.unique())))
```

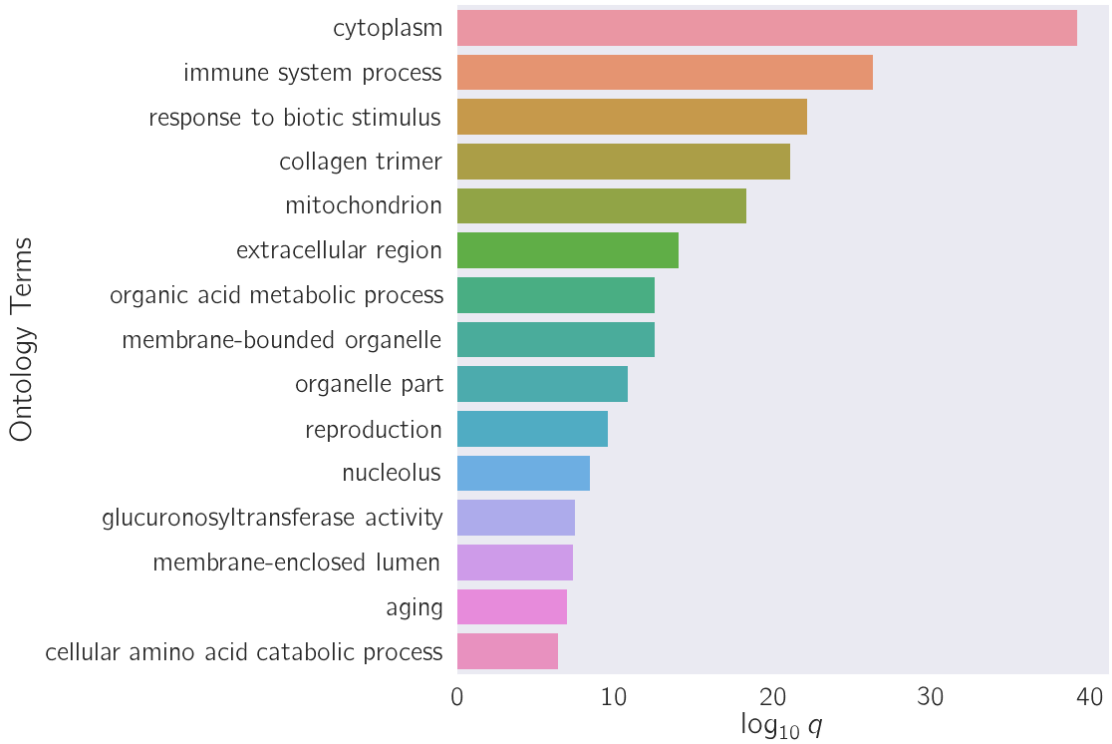
```
There are 1258 genes in the predicted hypoxia response
```

3 Enrichment Analysis of the Global HIF-1 response

Now that we have found the hypoxia response, we can perform tissue, phenotype and gene ontology enrichment analysis on this gene battery. Note that we don't show all possibilities. When a particular analysis is not present, it is because the enrichment results were empty.

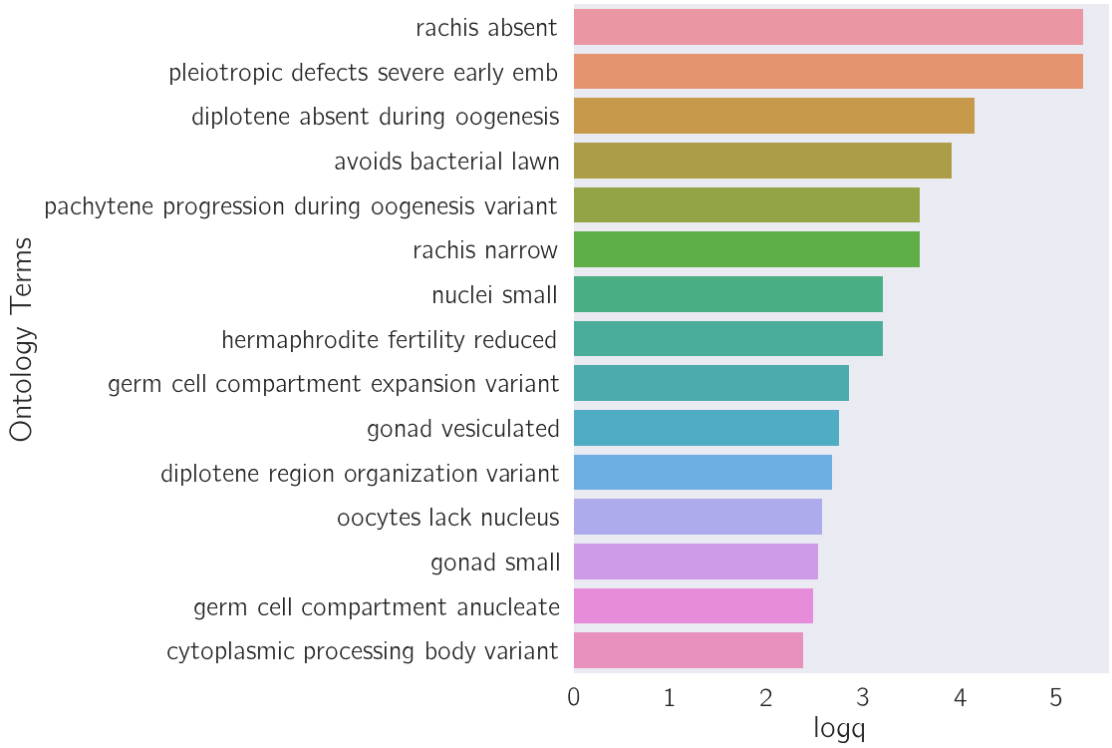
```
In [11]: teaH = tea.enrichment_analysis(hyp_response.ens_gene.unique(),
                                        tissue_df, show=False)
         peaH = tea.enrichment_analysis(hyp_response.ens_gene.unique(),
                                        phenotype_df, show=False)
         geaH = tea.enrichment_analysis(hyp_response.ens_gene.unique(),
                                        go_df, show=False)

         for df in [teaH, peaH, geaH]:
             df['logq'] = -df['Q value'].apply(np.log10)
```

```
In [12]: ax = tea.plot_enrichment_results(geaH, analysis='go', y='logq')
         plt.xlabel('$\log_{10}{q}$')
         plt.savefig('../output/supp_figures/supplementary_figure_3.pdf')
```

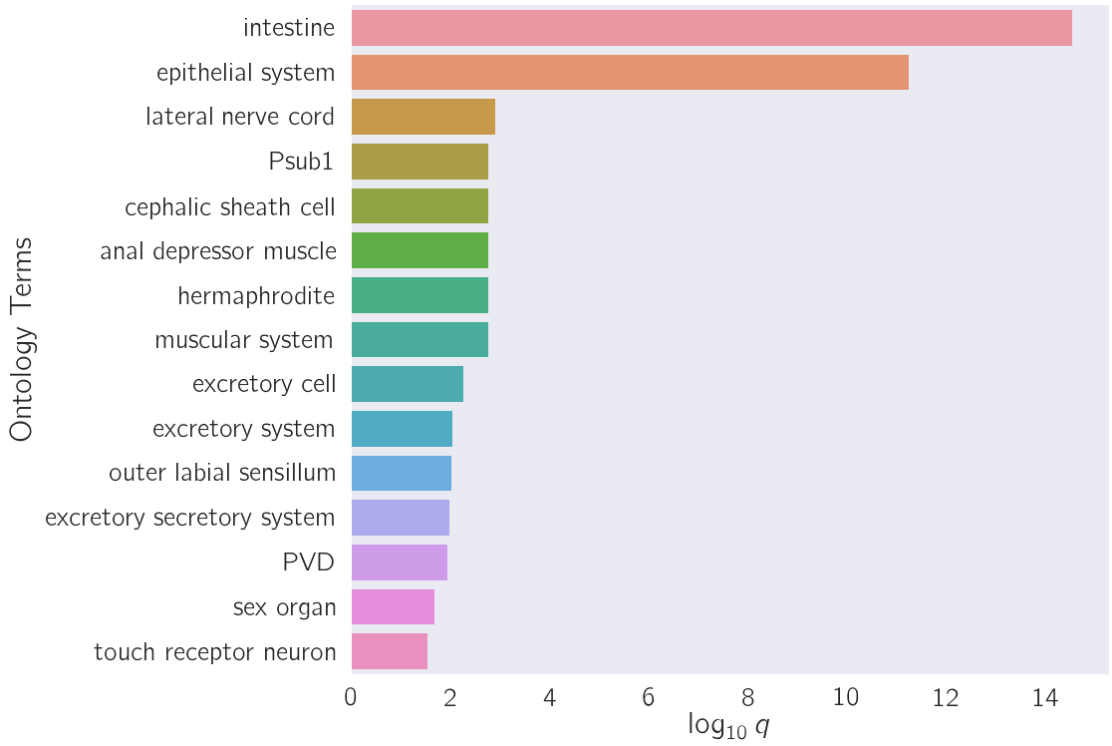



```
In [13]: tea.plot_enrichment_results(peaH, analysis='phenotype', y='logq')
```

```
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x10e823470>
```




```
In [14]: ax = tea.plot_enrichment_results(teaH, analysis='tissue', y='logq')
         plt.xlabel('$\log_{10}{q}$')
         plt.savefig('../output/supp_figures/supplementary_figure_4.pdf', bbox_inches='tight')
```


intestine
epithelial system
lateral nerve cord
Psub1
cephalic sheath cell
anal depressor muscle
hermaphrodite
muscular system
excretory cell
excretory system
outer labial sensillum
excretory secretory system
PVD
sex organ
touch receptor neuron
Ontology Terms
0
2
4
6
8
10
12
14
log10 q

In [ ]:

4 Understanding the decoupled transcriptomes

January 31, 2018

1 Table of Contents

1 Finding HIF-1 direct target candidates
2 vhl-1 dependent, hif-1-independent, genes
2.1 Plot vhl-1-dependent, hif-1-independent genes

In this notebook, I will identify gene targets that are specifically regulated by each *egl-9*, *vhl-1*, and *hif-1*. I define a specific regulatory node to mean the node that is the nearest regulatory node to these targets out of the subset of genes we have mutants for. As usual, we first load up all the libraries

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

# morgan
import tissue_enrichment_analysis as tea
import epistasis as epi
import genpy
import gvars

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%matplotlib inline

# This enables SVG graphics inline.
%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")

        mpl.rcParams['xtick.labelsize'] = 16
        mpl.rcParams['ytick.labelsize'] = 16
        mpl.rcParams['legend.fontsize'] = 14
```

In [2]:
```
        q = 0.1
        genvar = gvars.genvars()
        tissue_df = tea.fetch_dictionary()
        phenotype_df = pd.read_csv('../input/phenotype_ontology.csv')
        go_df = pd.read_csv('../input/go_dictionary.csv')
```

In [3]:
```
        tidy_data = pd.read_csv('../output/temp_files/DE_genes.csv')
        tidy_data.sort_values('target_id', inplace=True)
        tidy_data.dropna(subset=['ens_gene'], inplace=True)
        tidy_data['fancy genotype'] = tidy_data.code.map(genvar.fancy_mapping)
        tidy_data = tidy_data[tidy_data.genotype != 'fog-2']
        tidy_data.head()
```

Out[3]:

	ens_gene	ext_gene	target_id	b	se_b	qval	\
0	WBGene00007064	2RSSE.1	2RSSE.1a	1.121038	0.586487	0.216276	
19676	WBGene00007064	2RSSE.1	2RSSE.1a	0.524134	0.586487	0.887525	
118056	WBGene00007064	2RSSE.1	2RSSE.1a	0.519789	0.586487	0.791051	
98380	WBGene00007064	2RSSE.1	2RSSE.1a	0.934036	0.586487	0.409735	
59028	WBGene00007064	2RSSE.1	2RSSE.1a	0.809959	0.586487	0.496563	

	genotype	sorter	code	fancy genotype
0	egl-9;vhl-1	6	a	\emph{egl-9;vhl-1}
19676	egl-9 hif-1	7	f	\emph{egl-9 hif-1}
118056	hif-1	4	c	\emph{hif-1}
98380	egl-9	2	b	\emph{egl-9}
59028	rhy-1	1	e	\emph{rhy-1}

2 Finding HIF-1 direct target candidates

We are interested in identifying gene targets of HIF-1. In order to do this, I will decouple my data into two parts: * a `positive` dataframe, which contains all genes with β values greater than 0 * a `negative` dataframe, which contains all genes with β values less than 0

I will also define a function called `collate`. This function takes in a list or a numpy array and returns a boolean indicator of what genes are in a specified dataframe. It's a lot shorter to define this function than it is to write the one-liner over and over again.

In [4]:
```
        def collate(x):
            """For a vector `x`, find what elements in x are contained in
```

```
            tidy_data.target_id."""
        return tidy_data.target_id.isin(x)
```

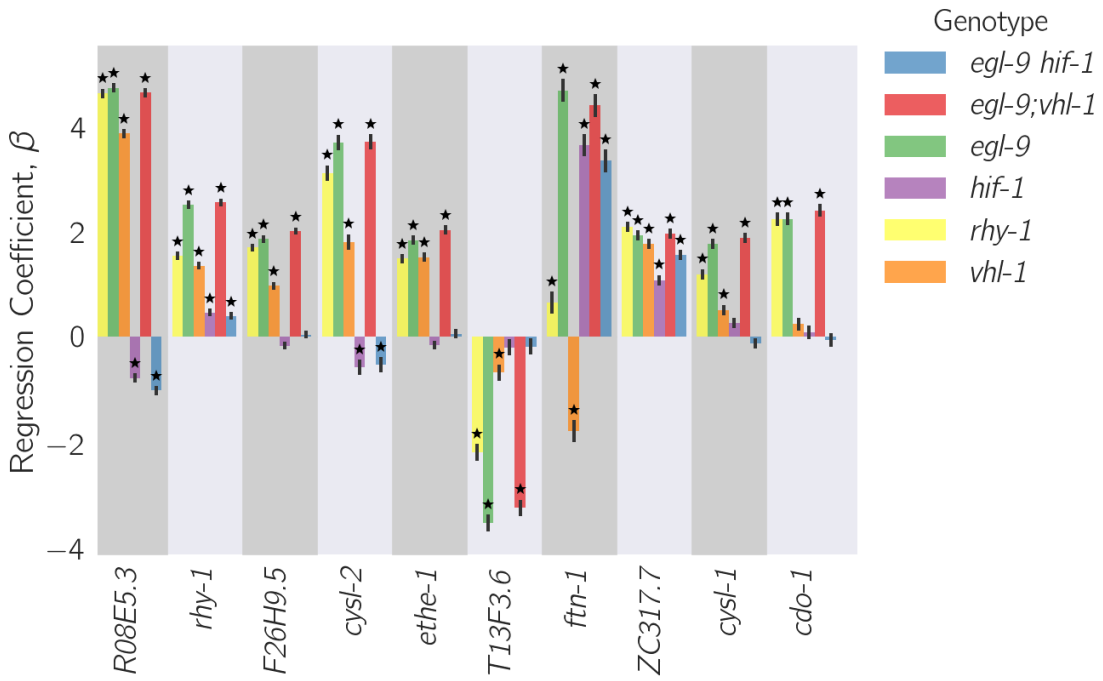
```
In [5]: hif_genes = pd.read_csv('../output/temp_files/hypoxia_response.csv')

        n = len(hif_genes[hif_genes.b > 0].ens_gene.unique())
        message = 'There are {0} unique genes that' +\
                  ' are candidates for HIF-1 direct binding'
        print(message.format(n))
```

```
There are 1173 unique genes that are candidates for HIF-1 direct binding
```

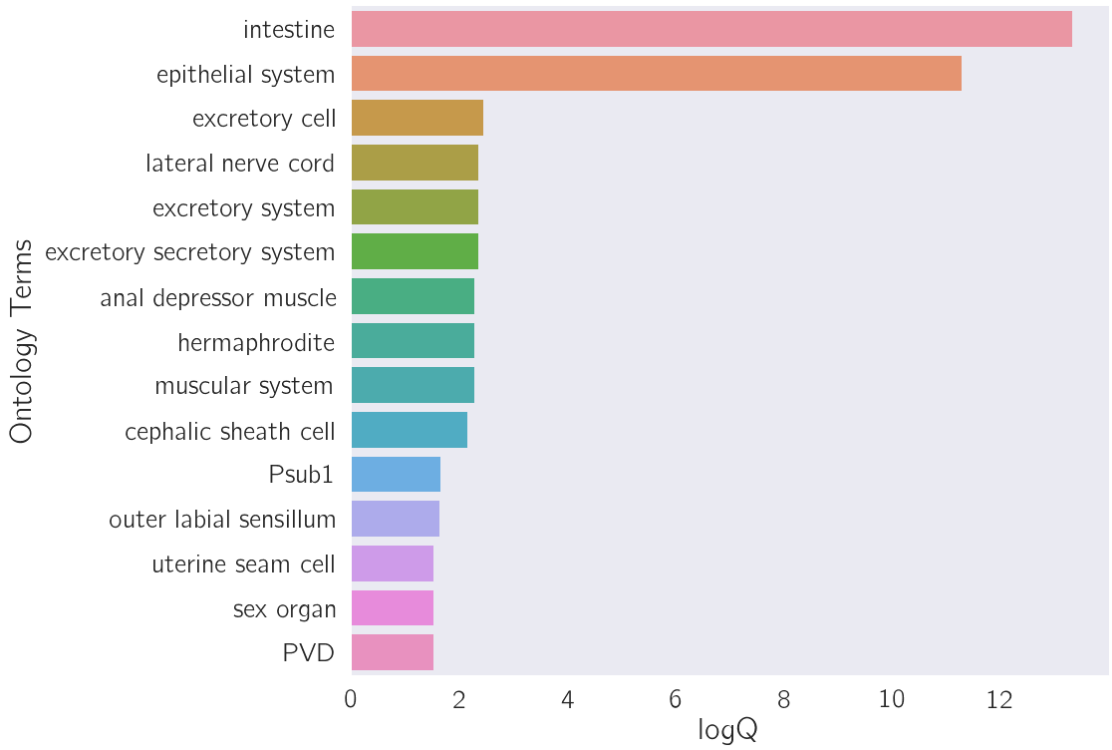
As a safety check, let's make a qPCR like plot to visualize our genes, and let's make sure they have the behavior we want:

```
In [6]: ids = hif_genes[hif_genes.b > 0].target_id
        hypoxia_direct_targets = tidy_data[tidy_data.target_id.isin(ids)]
```

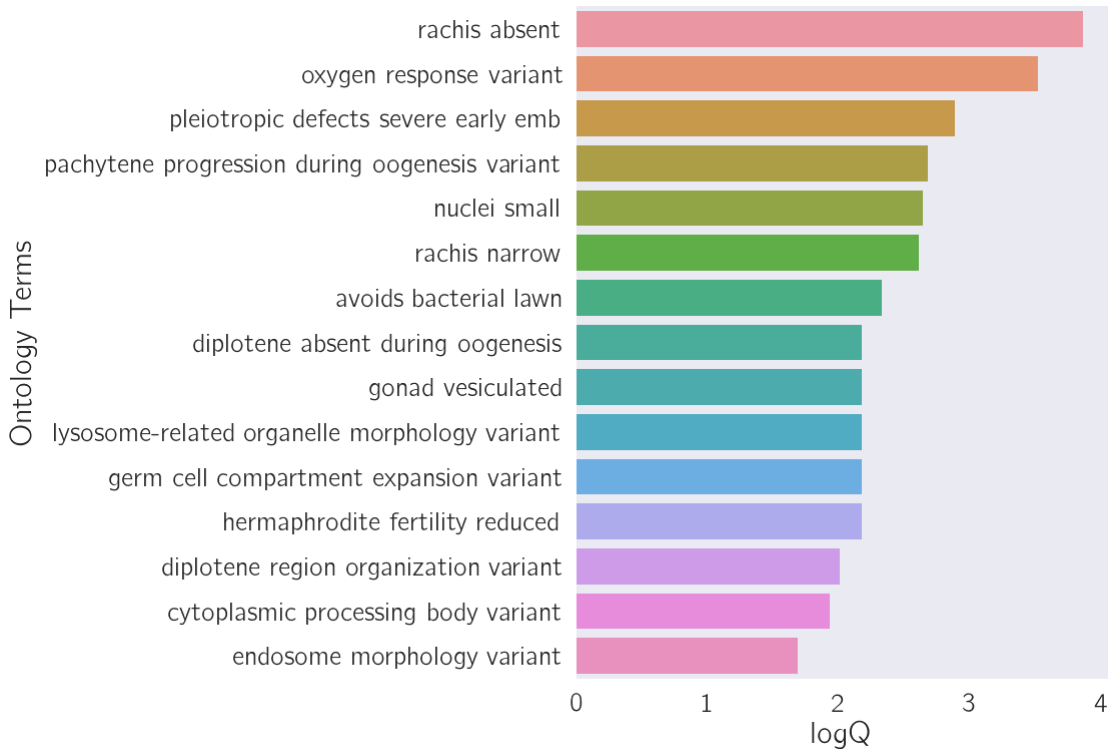
```
In [7]: names = hypoxia_direct_targets.sort_values('qval').target_id.unique()[0:10]

        name_sort = {}
        for i, name in enumerate(names):
            name_sort[name] = i+1

        plot_df = tidy_data[tidy_data.target_id.isin(names)].copy()
        plot_df['order'] = plot_df.target_id.map(name_sort)
        plot_df.sort_values('order', inplace=True)
        plot_df.reset_index(inplace=True)

        genpy.qPCR_plot(plot_df, genvar.plot_order, genvar.plot_color,
                        clustering='fancy genotype', plotting_group='target_id',
                        rotation=90)
```



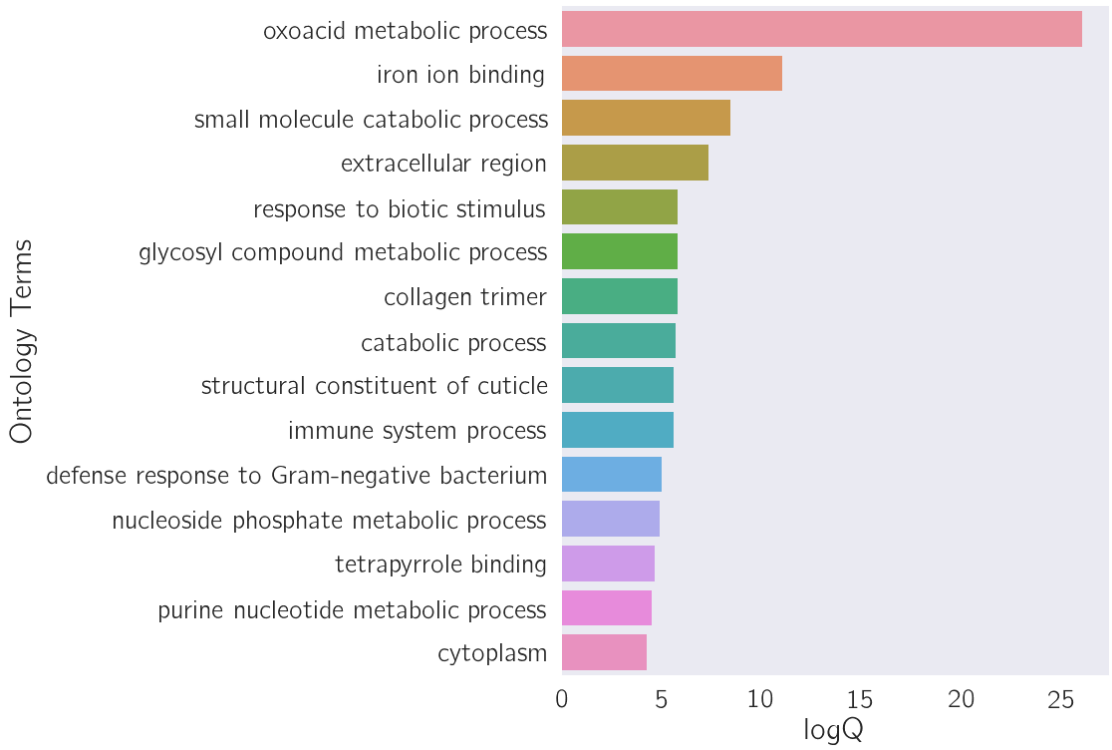

```
In [8]: res = tea.enrichment_analysis(hypoxia_direct_targets.ens_gene.unique(),
                                      tissue_df, show=False)
        res['logQ'] = -res['Q value'].apply(np.log10)
        tea.plot_enrichment_results(res, analysis='tissue', y='logQ')

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x10a628278>
```




```
In [9]: res = tea.enrichment_analysis(hypoxia_direct_targets.ens_gene.unique(),
                                     phenotype_df, show=False)
        res['logQ'] = -res['Q value'].apply(np.log10)
        tea.plot_enrichment_results(res, analysis='phenotype', y='logQ')

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x109a6f2e8>
```




```
In [10]: res = tea.enrichment_analysis(hypoxia_direct_targets.ens_gene.unique(),
                                       go_df, show=False)
         res['logQ'] = -res['Q value'].apply(np.log10)
         tea.plot_enrichment_results(res, analysis='go', y='logQ')

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x10a6c1978>
```



3 *vhl-1* dependent, *hif-1*-independent, genes

We can gate our settings to observe only *vhl-1*-dependent genes, by selecting only those genes that were present in the *vhl-1* and *egl-9;vhl-1* genotypes.

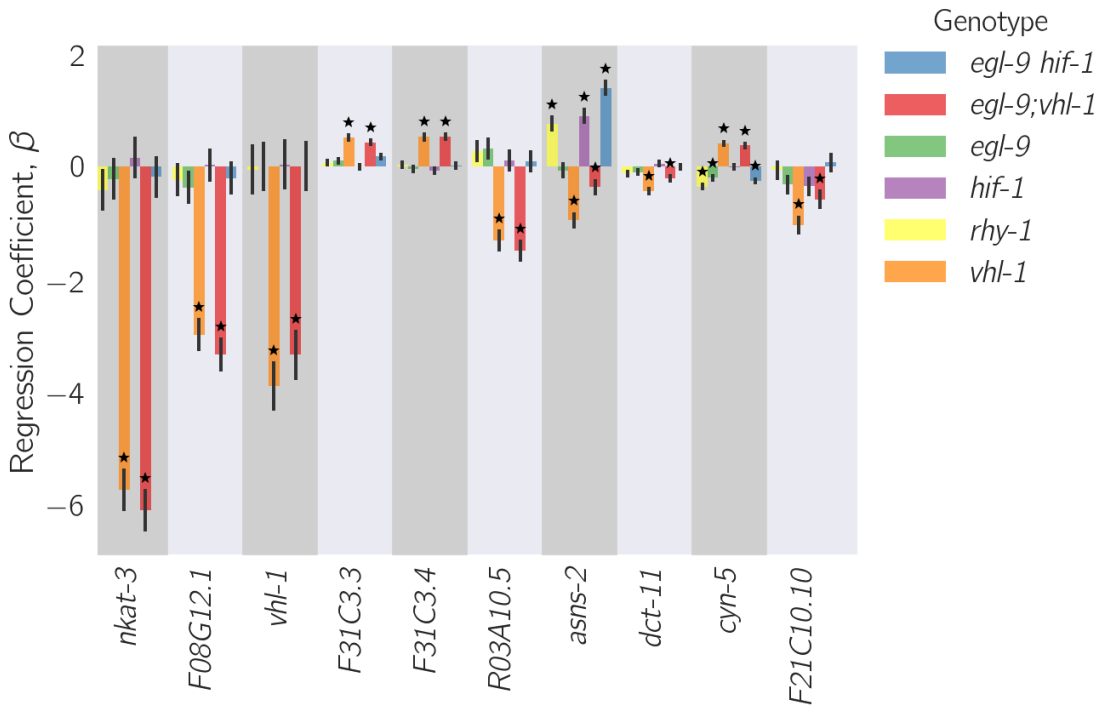
```
In [11]: positive = tidy_data[(tidy_data.qval < q) & (tidy_data.b > 0)]
         negative = tidy_data[(tidy_data.qval < q) & (tidy_data.b < 0)]

         # find the genes that overlap between vhl1 and egl-9vhl-1 and change in
         # same directiom
         vhl_pos = epi.find_overlap(['d', 'a'], positive)
         vhl_neg = epi.find_overlap(['d', 'a'], negative)
         vhl = list(set(vhl_pos + vhl_neg))

         # find genes that change in the same direction in vhl(-) and
         # vhl(+ datasets)
         same_vhl = []
         for genotype in ['b', 'e', 'f', 'c']:
             same_vhl += epi.find_overlap(['d', 'a', genotype], positive)
             same_vhl += epi.find_overlap(['d', 'a', genotype], negative)

         # put it all together:
```

```
ind = (collate(vhl)) & (~collate(same_vhl))
vhl_regulated = tidy_data[ind & (tidy_data.code == 'd')]

n = len(vhl_regulated.ens_gene.unique())
message = 'There are {0} genes that appear to be ' + \
          'regulated in a hif-1-independent, vhl-1-dependent manner.'
print(message.format(n))
```

```
There are 72 genes that appear to be regulated in a hif-1-independent, vhl-1-dependent manner.
```

3.1 Plot *vhl-1*-dependent, *hif-1*-independent genes

In [12]:
```
# begin plotting
names = vhl_regulated.sort_values('qval').target_id.unique()[0:10]
name_sort = {}
for i, name in enumerate(names):
    name_sort[name] = i+1

plot_df = tidy_data[tidy_data.target_id.isin(names)].copy()
plot_df['order'] = plot_df.target_id.map(name_sort)
plot_df.sort_values('order', inplace=True)
plot_df.reset_index(inplace=True)

genpy.qPCR_plot(plot_df, genvar.plot_order, genvar.plot_color,
                clustering='fancy genotype', plotting_group='target_id',
                rotation=90)

# save to file
cols = ['ext_gene', 'ens_gene', 'target_id', 'b', 'qval']
vhl_regulated[cols].to_csv('../output/temp_files/vhl_1_regulated_genes.csv')
```



No enrichment was observed for these genes.

5 Quality check of the RNA-seq data

January 31, 2018

1 Table of Contents

1 Quality control
1.1 Plot showing normal nhr-57 expression patterns in hypoxia mutants
2 Quality Control on the hypoxia response and the hif-1 direct target predictions

In this notebook, we present some basic sanity checks that our RNA-seq worked and that the data is picking up on the right signals. It's a fairly short notebook.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

# morgan
import morgan as morgan
import gvars
import genpy

# stats
from scipy import stats as sts

# 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)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%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(style='dark', context='notebook', font='sans-serif')

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['legend.fontsize'] = 14
```

```
In [2]: # import the code <--> genotype mapping and other useful variables
        genvar = gvars.genvars()

        tf_df = pd.read_csv('../input/tf_list.csv')
        hypoxia_gold = pd.read_csv('../input/hypoxia_gold_standard.csv', sep=',')
        hypoxia_response = pd.read_csv('../output/temp_files/hypoxia_response.csv')
```

```
In [3]: # Specify the genotypes to refer to:
        single_mutants = ['b', 'c', 'd', 'e', 'g']
        double_mutants = {'a' : 'bd', 'f':'bc'}
```

```
In [4]: tidy = pd.read_csv('../output/temp_files/DE_genes.csv')
        tidy.sort_values('target_id', inplace=True)
        tidy.dropna(subset=['ens_gene'], inplace=True)
        # drop the fog-2 dataset
        tidy = tidy[tidy.code != 'g']
        tidy['fancy genotype'] = tidy.code.map(genvar.fancy_mapping)
```

2 Quality control

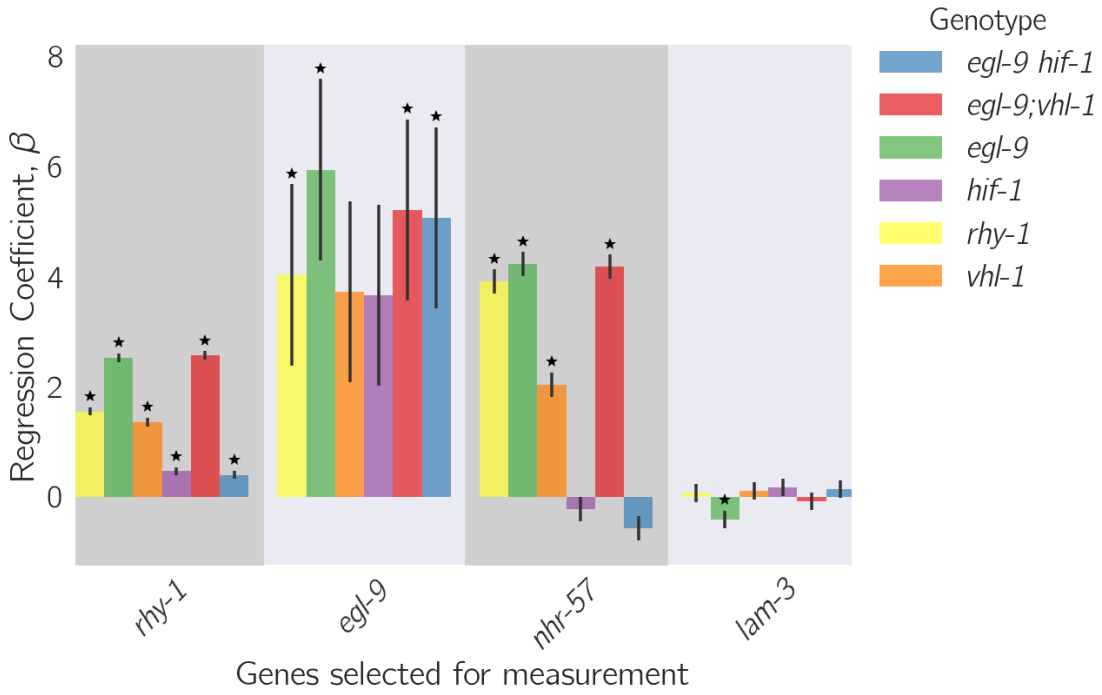
egl-9, *rhy-1* and *nhr-57* are known to be HIF-1 responsive. Let's see if our RNA-seq experiment can recapitulate these known interactions. For ease of viewing, we will plot these results as bar-charts, as if they were qPCR results. To do this, we must select what genes we will use for our quality check. I would like to take a look at *nhr-57*, since this gene is known to be incredibly up-regulated during hypoxia. If N2 worms became hypoxic during treatment for a period long enough to induce transcriptional changes, then *nhr-57* should appear to be significantly down-regulated in the *hif-1* and *egl-9 hif-1* genotypes.

```
In [5]: x = ['WBGene00012324', 'F22E12.4a.1',
             'WBGene00003647', 'WBGene00002248']

        find_x = ((tidy.ens_gene.isin(x)) | (tidy.target_id.isin(x)))
        plot_df = tidy[find_x].copy()
        x_sort = {'WBGene00012324': 1, 'WBGene00001178': 2,
```

```
        'WBGene00003647': 3, 'WBGene00002248': 4}

plot_df['order'] = plot_df.ens_gene.map(x_sort)
plot_df.sort_values('order', inplace=True)
plot_df.reset_index(inplace=True)
```

2.1 Plot showing normal *nhr-57* expression patterns in hypoxia mutants

```
In [6]: genpy.qPCR_plot(plot_df, genvar.plot_order, genvar.plot_color,
                        clustering='fancy genotype', plotting_group='ens_gene', rotation=45)
        plt.xlabel(r'Genes selected for measurement', fontsize=20)
        save = '../output/supp_figures/supplementary_figure_1.svg'
        plt.savefig(save, bbox_inches='tight')
```

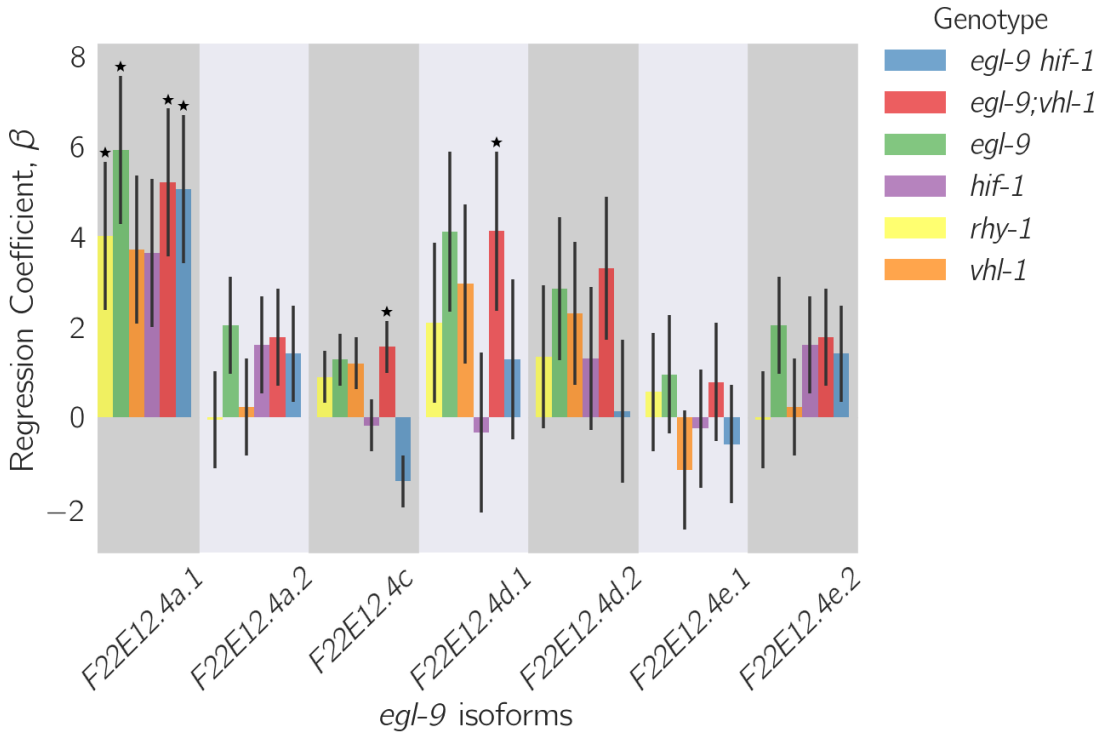


It looks like we are able to recapitulate most of the known interactions between these reporters and HIF-1 levels. There are no contradicting results, although the *egl-9* levels don't all quite reach statistical significance. For completeness, below I show ALL the *egl-9* isoforms.

```
In [7]: x = ['WBGene00001178']

        find_x = tidy.ens_gene.isin(x)
        plot_df = tidy[find_x].copy()

        x_sort = {}
        for i, target in enumerate(plot_df.target_id.unique()):
```

```
        x_sort[target] = i + 1

    plot_df['order'] = plot_df.target_id.map(x_sort)
    plot_df.sort_values('order', inplace=True)
    plot_df.reset_index(inplace=True)

    genpy.qPCR_plot(plot_df, genvar.plot_order, genvar.plot_color,
                    clustering='fancy genotype', plotting_group='target_id', rotation=45)
    plt.xlabel(r'\emph{egl-9} isoforms', fontsize=20)
```

```
Out[7]: <matplotlib.text.Text at 0x1164887b8>
```



3 Quality Control on the hypoxia response and the hif-1 direct target predictions

That's one way to check the quality of our RNA-seq. Another way is to look for what genes are D.E. in our hypoxia dataset. We will test the most conservative guess for the hypoxia response, and the predicted hypoxia targets using a hypergeometric test.

```
In [8]: q = 0.1
        def test_significance(df, gold=hypoxia_gold):
            ind = df.ens_gene.isin(hypoxia_gold.WBIDS)
```

```
    found = df[ind].ens_gene.unique()
    sig = len(df.ens_gene.unique())  # number of genes that we picked
    ntotal = len(tidy.ens_gene.unique())  # total genes measured
    pval = sts.hypergeom.sf(len(found), ntotal,
                            len(hypoxia_gold), sig)
    if pval < 10**-3:
        print('This result is statistically significant' +\
              ' with a p-value of {0:.2g} using a\n hypergeometric test. '.format(pval)
              'You found {0} gold standard genes!'.format(len(found)))
    else:
        print(pval)
```

Hypoxia response (conservative guess):

```
In [9]: test_significance(hypoxia_response)
```

```
This result is statistically significant with a p-value of 7.6e-06 using a
 hypergeometric test. You found 9 gold standard genes!
```

Both datasets are enriched for known hypoxic response genes!

6 Quantifying Epistasis

January 31, 2018

1 Table of Contents

1 Introduction
2 Transcriptome-wide epistasis: A definition
3 Introduction to epistasis plots
3.1 egl-9 is epistatic to vhl-1
3.2 Figure 5B
3.3 Figure 5C
4 Odds ratios
4.1 Writing the theoretical models
4.2 Writing the free slope model
4.3 Writing the Odds Ratio function
4.4 Odds ratio for the epistasis between egl-9 and vhl-1
5 Measuring suppressive epistasis
5.1 hif-1 suppresses egl-9
6 Transitivity in transcriptomes
6.1 Predicting epistasis between egl-9 and vhl-1 using the rhy-1 transcriptome
6.2 Predicting epistasis between egl-9 and hif-1 using the rhy-1 transcriptome

```
In [ ]:
```

2 Introduction

In this notebook, we develop the notion of 'genome-wide epistasis'. Genome-wide epistasis is a generalization of the methods used to measure epistasis between genotypes using qPCR. Why genome-wide epistasis can even begin to appear seems a bit mysterious, and we briefly touch on this philosophical aspect at the end of the notebook.

3 Transcriptome-wide epistasis: A definition

Epistasis is defined by Huang and Sternberg (2006) as one allele masking another allele's phenotype. In other words, if an allele X has a phenotype Ph_1, and an allele Y (at a different locus) has a different phenotype Ph_2, we can say that X and Y are epistatic if the double homozygote has a phenotype that is equal to either Ph_1 or Ph_2. Epistasis is also known as non-additivity, and it is the basis of the definition of genetic interactions. Of course, stating that two genes are epistatic to each other is subject to a large number of qualifiers. A particularly important qualifier is that

the phenotypes under study must have a reasonable dynamic range—they must not be too strong or too subtle, or non-additivity could occur simply as a result of a compressed range. Another important consideration is that the alleles used to study a genetic interaction must be complete loss of function alleles for the phenotype under consideration. If they are not, trouble can arise from making inferences that are just too strong.

The null hypothesis when observing two mutants of different genes is that they do not interact. Therefore, when the double mutant is made, the result must be that the two phenotypes add. We reasoned that, ideally, this should also be the case for vectorial phenotypes. This enabled us to make a prediction about what a double mutant would look like. Given two alleles X and Y that code for different genes (i.e. that complement), the double mutant X^-Y^- should have expression levels equal to:

$$\beta_{XY,Predicted\ Additive,i} = \beta_{X,i} + \beta_{Y,i},$$

where $\beta_{G,i}$ is the regression coefficient (from Sleuth) for genotype G and isoform i, and $\beta_{XY,Predicted\ Additive,i}$ is the predicted expression of isoform i in a double mutant of X and Y under an additive model. Since we have data for double and single mutants, we reasoned that we should be able to plot the predicted expression, $\beta_{XY,Pred,i}$, against deviations from the predicted expression $\Delta_i = \beta_{XY,i} - \beta_{XY,Pred}$. Given these two numbers (the predicted additive effect and the deviation from predicted), we can generate an epistasis plot, where the X-axis reflects the predicted expression level of the double mutant assuming an additive model, and the Y-axis defines the deviation from predicted. For additive mutants, we expect to see that the genes fall along the line $\Delta_i = 0$ with some noise ϵ_i.

Having defined our null hypothesis, it is now possible to explore what other results could be expected. Suppose that X and Y act along a single, activating pathway of the form $X \rightarrow Y \rightarrow Ph$ or $Y \rightarrow X \rightarrow Ph$. In that case, both genes should: 1. Act on the same phenotype 2. Have the same magnitude of effect.

We can predict the additive effect of an additive interaction when both genes have the same effect on a phenotype, it should be $2\beta_{X,i} = 2\beta_{Y,i} = 2\beta_i$. We can also reason about what the phenotype of the mutant should be. If the two genes are acting along a single pathway, breaking the pathway twice should have the same effect as breaking it once. Therefore, it must be the case that $\beta_{XY,Pred,i} = \beta_i$. Next, we can calculate that the idealized deviation from the additive value should be $\Delta_i = \beta_i - 2\beta_i = -\beta_i$. Putting it all together, we then would expect the coordinates for each isoform to be:

$$(2\beta_i, -\beta_i),$$

which suggests that when two genes interact positively through a single unbranched pathway, an epistasis plot should show points that fall along the line $y = -0.5x$.

What about a model where $X - -|Y$? For this case, I will invoke a limit argument. Suppose that, under "usual laboratory conditions" (whatever those are!), X is `ON` and it is often present in large quantities in the cell. Suppose further, that X is the strongest possible (non-competitive) inhibitor of Y. Then it follows that under usual conditions, Y must be `OFF`. Therefore, a null mutant of Y should look transcriptomically wild-type or very close to it. The predicted expression of a double mutant should therefore be $\beta_{X,i} + \beta_{Y,i} \sim \beta_{X,i}$. We can reason about the actual expression level of a double mutant as follows: If X inhibits Y, then removing X causes a large increase in the protein levels of Y. However, removing Y from the X^- animal means that protein levels of Y return to wild-type. This is an effect known as suppression. Suppression means that the allele that is downstream of the inhibitor defines what the phenotype will be. Therefore, the expression phenotype of this double mutant, $\beta_{XY,i} = \beta_{Y,i}$. With this number in hand, we can now calculate

$\Delta_i = \beta_{Y,i} - \beta_{Y,i} - \beta_{X,i} = -\beta_{X,i}$. From this, it follows that the points will fall near the coordinates,

$$(\beta_{X,i}, -\beta_{X,i}).$$

In other words, the points will fall along the line $y = -x$.

At this point, we have covered most of the notable simple cases. Only two remain. Suppose that for two mutants under study, the double mutant expresses the phenotype of one of the single mutants. This means that $X^-Y^- = X^-$. What slope should we observe? Well, clearly we can predict the additive (x-axis) coordinate: $\beta_{X,i} + \beta_{Y,i}$. What about the deviation from additive? Well, if the double mutant looks like the mutant X^-, then it follows that the expression should also match. In other words, $\beta_{XY,i} = \beta_X$. From this, we can predict the coordinates of each point on the epistasis plot to be:

$$(\beta_{X,i} + \beta_{Y,i}, -\beta_{Y,i}).$$

What does this mean? Well, if $\beta_{X,i}$ was completely uncorrelated from $\beta_{Y,i}$, we might be tempted to say that this should still fall along the line of $y = -x$, perhaps with more noise than the case of suppression. However, this is not the case! $\beta_{X,i}$ and $\beta_{Y,i}$ are covariant! Nothing remains but to make a line of best fit. The closer this α is to -0.5, the closer the two genes are to interacting exclusively in a linear manner; the closer the slope is to -1, the closer these genes are to being in the limit of strong suppression. Anything in between? Well, the in-between is also interpretable.

How can we know that the points will fall on a straight line? Well. Let us consider a branched pathway, where $X \rightarrow Y \rightarrow Ph$, but $X \rightarrow Ph$ is also true (i.e., X acts on Ph in Y-dependent and independent manners). How do we know these will form a line? Well, suppose that the effect of X on Ph is complete. Then this means that $XY = X$. If X interacts with Ph in a simple manner (i.e., suppose X activates a transcription factor that mediates Ph), then we can make the following statement: Y accounts for a fraction f of the interaction of X on Ph.

Given the above statement is true, then it follows that

$$\beta_{Y,i} = f\beta_{X,i}.$$

Then we can now predict the additive effect of the double mutant:

$$\beta_{XY,AddPred,i} = (1+f)\beta_{X,i}.$$

However, because we know that $XY = X$, we know that the expression of the double mutant will match the expression of X. Therefore, the expected deviation of the double mutant should be

$$\Delta_i = -f\beta_{X,i},$$

and the data will fall along the coordinates $((1+f)\beta_{X,i}, -f\beta_{X,i})$. Therefore, the points will fall along the line:

$$y = -\frac{f}{1+f}x$$

Notice that f can only range from [0, 1], which restricts the range of slopes from [0, -0.5].

```
In [1]: # important stuff:
        import os
        import pandas as pd
        import numpy as np
        import statsmodels.tools.numdiff as smnd
```

```
import scipy

# TEA and morgan
import morgan as morgan
import epistasis as epi
import gvars

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

from scipy.stats import gaussian_kde

# Magic function to make matplotlib inline;
%matplotlib inline

# This enables SVG graphics inline.
%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")

mpl.rcParams['xtick.labelsize'] = 16
mpl.rcParams['ytick.labelsize'] = 16
mpl.rcParams['legend.fontsize'] = 14
```

In [2]:
```
q=0.1
genvar = gvars.genvars()

# Specify the genotypes to refer to:
single_mutants = ['b', 'c', 'd', 'e', 'g']
double_mutants = {'a' : 'bd', 'f':'bc'}
```

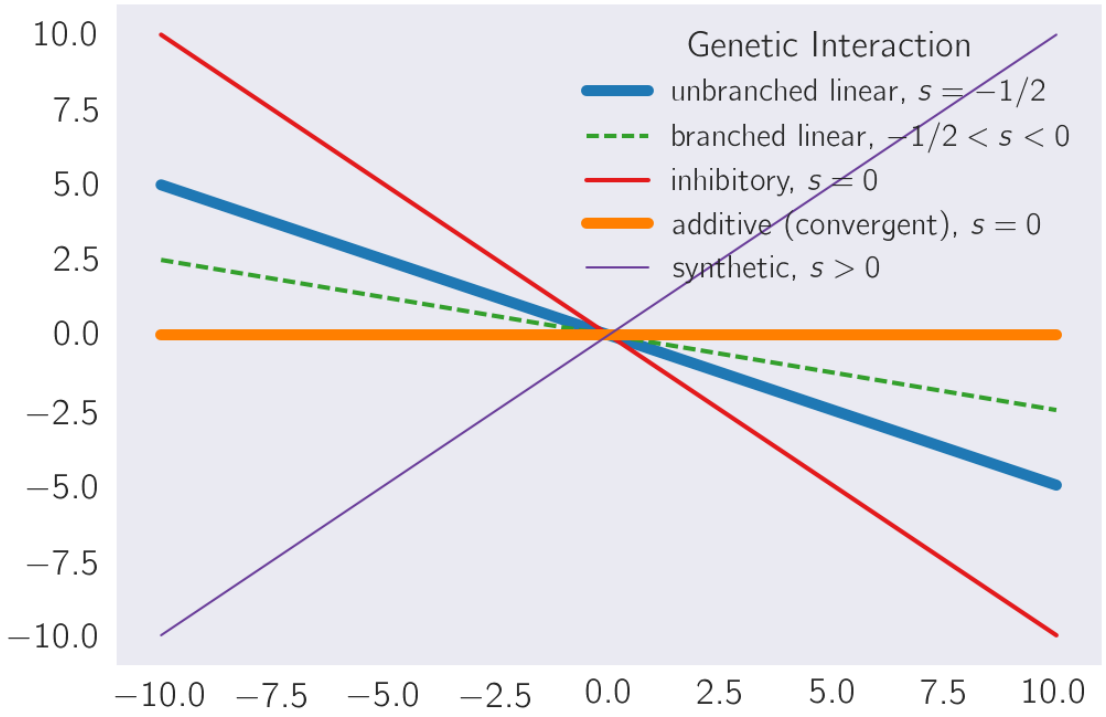
In [3]:
```
tidy_data = pd.read_csv('../output/temp_files/DE_genes.csv')
tidy_data.sort_values('target_id', inplace=True)
tidy_data.dropna(subset=['ens_gene'], inplace=True)
```

In [4]:
```
tidy_data.head()
```

Out[4]:
```
              ens_gene ext_gene target_id         b      se_b      qval  \
0       WBGene00007064  2RSSE.1  2RSSE.1a  1.121038  0.586487  0.216276
19676   WBGene00007064  2RSSE.1  2RSSE.1a  0.524134  0.586487  0.887525
118056  WBGene00007064  2RSSE.1  2RSSE.1a  0.519789  0.586487  0.791051
39352   WBGene00007064  2RSSE.1  2RSSE.1a  0.150147  0.829418  1.000000
98380   WBGene00007064  2RSSE.1  2RSSE.1a  0.934036  0.586487  0.409735

           genotype  sorter code
0       egl-9;vhl-1       6    a
19676   egl-9 hif-1       7    f
118056        hif-1       4    c
39352         fog-2       5    g
98380         egl-9       2    b
```

Before we begin, let's make a schematic diagram of what the slopes should look like:

In [5]:
```
X = np.linspace(-10, 10)
Y = -1/2*X

plt.plot(X, -1/2*X, ls='-', color= '#1f78b4', lw=5,
         label='unbranched linear, $s=-1/2$')
plt.plot(X, -1/4*X, ls='--', color= '#33a02c',
         label='branched linear, $-1/2 < s < 0$')
plt.plot(X, -X, ls='-', lw=2, color= '#e31a1c',
         label='inhibitory, $s = 0$')
plt.plot(X, 0*X, 'k-', lw=5, color= '#ff7f00',
         label='additive (convergent), $s = 0$')
plt.plot(X, X, '-', lw=1,color= '#6a3d9a',
         label='synthetic, $s > 0$')

lgd = plt.legend()
lgd.set_title('Genetic Interaction',
              prop=(mpl.font_manager.FontProperties(size=16)))
plt.savefig('../output/epistasis_plot_show.svg',
            bbox_inches='tight')
```



4 Introduction to epistasis plots

Having worked out the theory, we can now make the epistasis plot given our data. Let's plot this for *egl-9* and *vhl-1*.

4.1 *egl-9* is epistatic to *vhl-1*

As a first step, I will define what genotypes I am working with. In this case, we want to work with the *egl-9*, *vhl-1* and *egl-9;vhl-1* genotypes.

```
In [6]: letter1 = 'b'
        letter2 = 'd'
        double = genvar.double_mapping[letter1 + letter2]
```

The procedure to follow now is as follows:

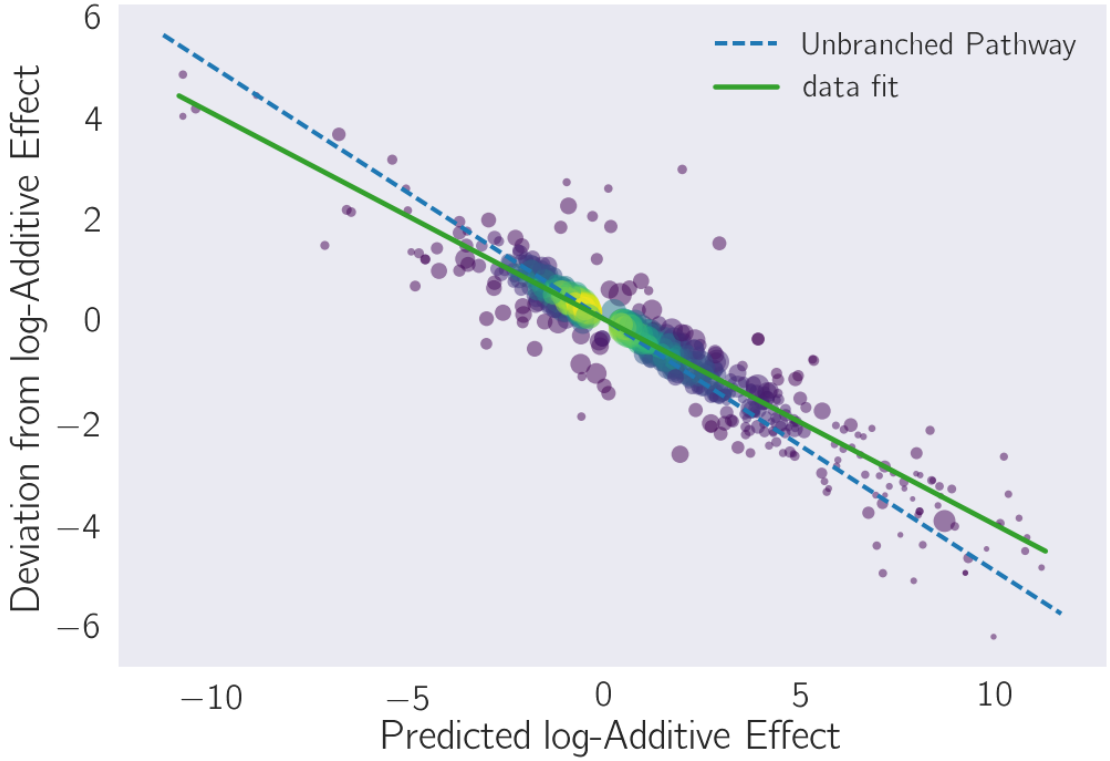
1. Find the set of genes that are differentially expressed (direction agnostic) between the three genotytpes. Call that set D
2. For the set D, make a prediction of what the double mutant looks like by adding the single mutants (additive null model). Calculate the y-axis by taking the difference between the observed coefficient and the expected.
3. Calculate error bars—remember, variances add.
4. Find the line of best fit using Orthogonal Distance Regression with `scipy.odr`.
5. Plot.

I have implemented this procedure in the function `epi.epistasis_plot`, and I call it below. It returns a set of four things: * x - a dataframe containing the identities, beta and q-values of the first letter that was passed to the function (in this case, the *egl-9* genes) * y - same, but for the second genotype (*vhl-1*) * xy - same, but for the double mutant * ax - the plot axis

4.2 Figure 5B

```
In [7]: x = epi.find_overlap([letter1, letter2, double], tidy_data)
```

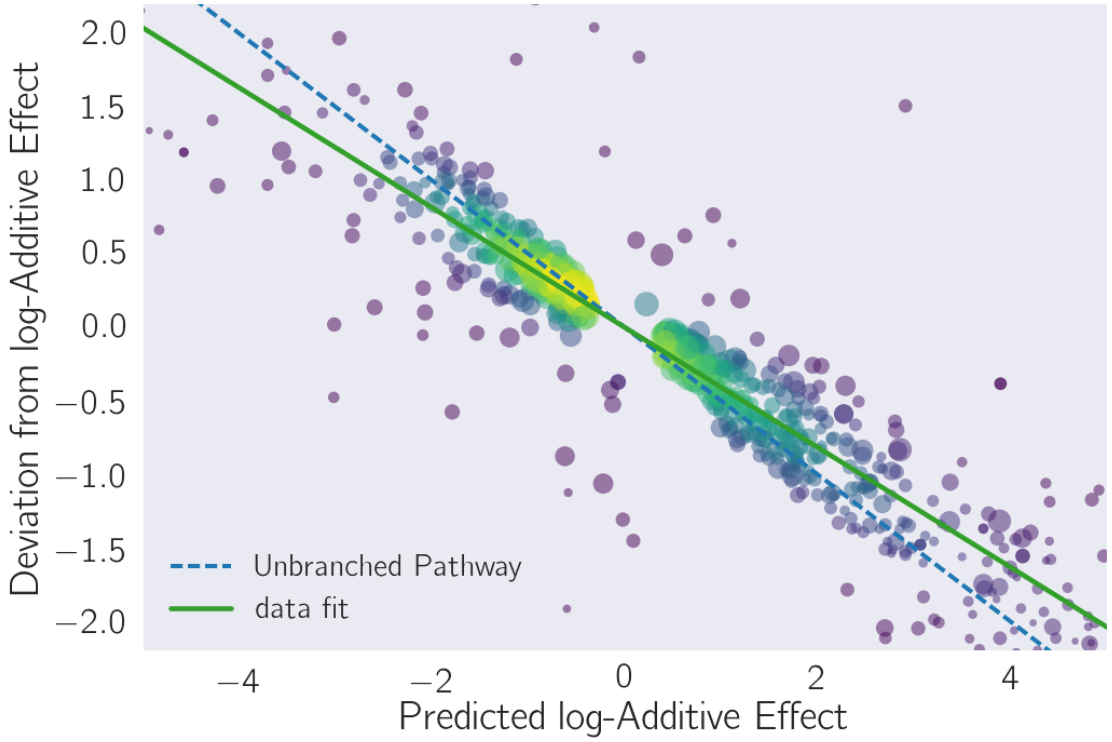
```
In [8]: x, y, xy = epi.find_STP([letter1, letter2], double, tidy_data)
        epi.ODR([x, y], xy, 'actual')
        x, y, xy, ax = epi.make_epiplot([letter1, letter2], double, tidy_data)
        plt.savefig('../output/epistasis{0}{1}.svg'.format(genvar.mapping[letter1],
                                                           genvar.mapping[letter2]),
                    bbox_inches='tight')
```



The points all fall along a line!!! Yes! We could even look at it in a little more detail, see how the scatter looks like if we zoom in.

```
In [9]: _ = epi.make_epiplot([letter1, letter2], double, tidy_data)
        plt.xlim(-5, 5)
        plt.ylim(-2.2, 2.2)
```

```
Out[9]: (-2.2, 2.2)
```



Let's figure out what the calculated slope of best fit is

```
In [10]: # calculate slope from the data:
         actual = epi.ODR([x,y], xy, epistasis='actual')
         actual.pprint()
```

```
Beta: [-0.40766836]
Beta Std Error: [ 0.00590136]
Beta Covariance: [[  2.84372564e-05]]
Residual Variance: 1.2246619461047352
Inverse Condition #: 1.0
Reason(s) for Halting:
  Sum of squares convergence
```

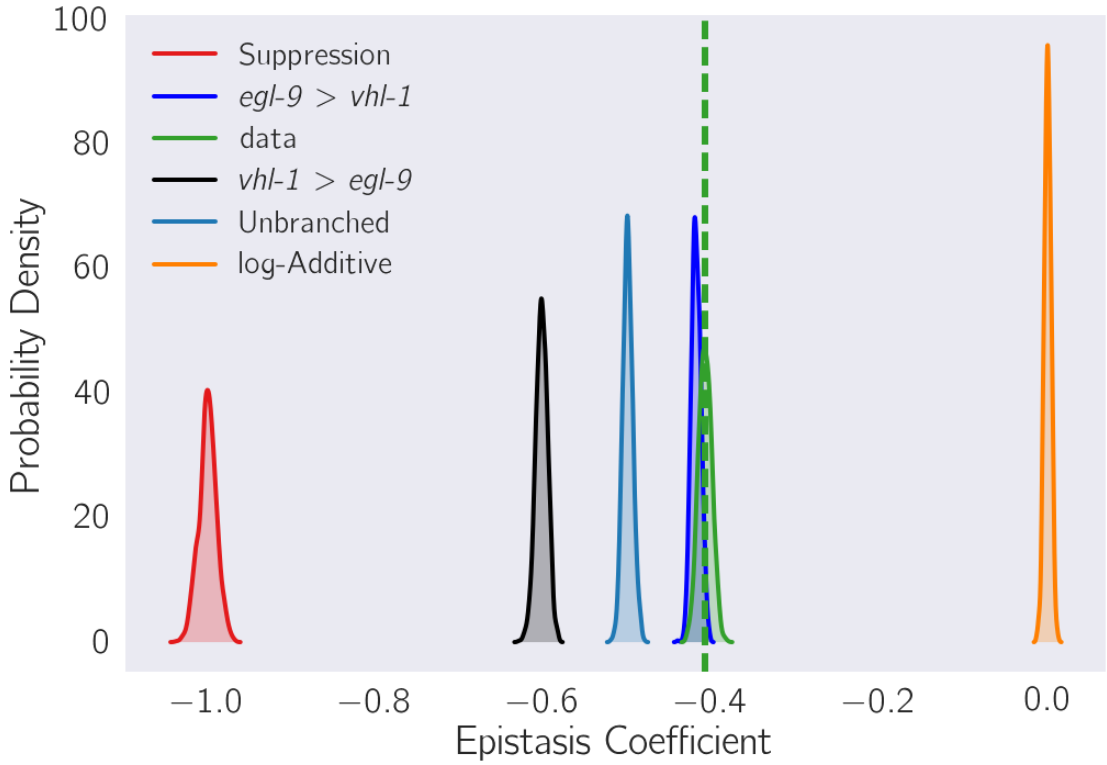
Ok. It's a line, even though it does have scatter. Fortunately, the largest points are pretty close to the line of best fit, which has a slope of -0.41 ± 0.006. Now, what we need to do is perform all of the simulations for the epistasis possibilities. We will also bootstrap the observed distribution just to make sure the slope distribution is what we think it is.

```
In [11]: s = epi.calculate_all_bootstraps(letter1, letter2, double, tidy_data, nsim=1000)
```

4.3 Figure 5C

```
In [12]: ax = epi.plot_bootstraps(letter1, letter2, s, cumulative=False, shade=True)
         plt.xlabel('Epistasis Coefficient')
```

```
plt.ylabel('Probability Density')
plt.savefig('../output/kde-epistasis{0}{1}.svg'.format(genvar.mapping[letter1],
                                                       genvar.mapping[letter2]),
            bbox_inches='tight')
```




```
In [13]: np.median(s['actual'])
```

```
Out[13]: -0.40820229363003913
```

Alright! The predicted epistatic curve fits the data perfectly!! Woohoo!!! And the unbranched curve doesn't even overlap with the other ones. We could tentatively say that it looks like we are dealing with a branched pathway of some sort. From part 1 above, we had concluded that the relationship between the slope and the fraction of the effect mediated through the 'main' pathway was:

$$\alpha = \frac{f}{1+f}.$$

We can invert this equation to solve for f, which yields, $f = \alpha/(1-\alpha)$. Plugging in, we find that $f = 0.42/.58 = 0.72$. 72% of the inhibition of HIF-1 by EGL-9 is through the VHL-1-dependent degradation pathway. The other 28% is presumably coming from the SWAN-1-dependent pathway. In order to truly have any confidence in this result, we should have a different way to check. Let's implement a Bayesian Odds Ratio test and see whether we can choose a model this way.

5 Odds ratios

We will perform pairwise comparison between a free model with variable slope and the five theoretical models we tested. First, we need to define the Bayesian function we must optimize. It will be:

$$P(D\,|\alpha, M_1, I) \propto \prod_{(x_i,y_i,w_i)\in D} \exp\,(\frac{(y_i - \alpha\cdot x_i)^2}{w_i})\cdot(1+\beta^2)^{-3/2},$$

where (x_i, y_i) are the coordinates of the point D_i, and w_i is the standard error of y_i. For the theoretic models, we will find the probability,

$$P(D\,|M_i, I) \propto \prod_{(x_i,y_i,w_i)\in D} \exp\,(\frac{(y_i - y_{pred,i})^2}{w_i}),$$

where $y_{pred,i}$ is the predicted y-value under model i. Finally, we will approximate the odds ratio by using a Laplace approximation of the functions where the probability is maximized. Briefly, model selection is performed by evaluating the quotient:

$$O_{1i} = \frac{P(M_1\,|I)}{P(M_i\,|I}\frac{P(D\,|M_1, I)}{P(D\,|M_i, I)}$$

The first term in the odds ratio is impossible to evaluate. We cannot know the probability of one model versus another. Qualitatively, we might say that certain models are more likely (for example, tried and true physical models are more likely than brand new recently invented models that come out of nowhere), but we cannot easily assign a number to them. Arbitratrily, we will assign the simpler models slight support, because genetics has been around for a long time. So, we will say the first term is equal to $\exp -2$ in favour of the theoretical models M_j. What is the second term?

Let's remember that the model we specified above is in terms of $P(D\,|M_1, \alpha, I)$. We can get rid of α by marginalizing:

$$P(D\,|M_1) = \int d\alpha\, P(D\,|\alpha, M_1, I).$$

We can use a laplacian approximation on this integral to obtain:

$$P(D\,|M_1) \sim P(D\,|\alpha^*, M_1, I)\cdot P(\alpha^*|M_1, I)\sqrt{2\pi}\sigma_1,$$

where α^* is the *Maximum A Posteriori* (MAP) estimate of α, and σ_1 is the covariance of the Gaussian approximation of the posterior around the point α^*. Therefore, we can now calculate the approximate odds ratio:

$$O_{1i} = \exp\,(-2)\frac{P(D\,|\alpha^*, M_1, I)\cdot P(\alpha^*|M_1, I)\sqrt{2\pi}\sigma_1}{P(D\,|M_i, I)}.$$

Let's code all of this up!

5.1 Writing the theoretical models

In [14]:
```
# bayes model fitting:
def log_likelihood_fixed(w, x, y, model, alpha=None):
    """Likelihood probability for the theoretical models of epistasis"""
    sigma = w
    epistasis = ['actual', 'xy=x', 'xy=y', 'xy=x=y', 'xy=x+y',
                 'suppress']

    # errors:
    if model not in epistasis:
        raise ValueError('model is not allowed')

    if (model is 'xy=x') or (model is 'xy=y'):
        if alpha is None:
            raise ValueError('alpha cannot be none for epistasis\
                              models `xy=x` or `xy=y`')

    # pick your model
    if model == 'xy=x+y':
        y_model = 0

    elif model == 'xy=x=y':
        y_model = -1/2*x

    elif model == 'suppress':
        y_model = -x

    elif (model == 'xy=x') or (model == 'xy=y'):
        y_model = alpha[model]*x

    # return the probability function
    return -0.5 * np.sum(np.log(2 * np.pi * sigma ** 2) +\
                         (y - y_model) ** 2 / sigma ** 2)

def log_posterior_fixed(w, x, y, model, alpha=None):
    """The posterior probability of the theoretical models"""
    return log_likelihood_fixed(w, x, y, model, alpha)
```

5.2 Writing the free slope model

In [15]:
```
def log_prior(theta):
    """Pareto prior, which makes the lines be evenly sampled
       between (-1,1) and plus\minus [1, infinity]."""
    return -1.5 * np.log(1 + theta ** 2)

def log_likelihood(theta, x, y, w):
    """Calculates the weighted chi-square for the free model"""
```

```
    sigma = w
    y_model = theta * x
    return -0.5 * np.sum(np.log(2 * np.pi * sigma ** 2) + \
                         (y - y_model) ** 2 / sigma ** 2)

def log_posterior(theta, x, y, w):
    """The complete logarithm of the posterior"""
    return log_prior(theta) + log_likelihood(theta, x, y, w)

def neg_log_prob_free(theta, x, y, w):
    """Negative log of the posterior."""
    return -log_posterior(theta, x, y, w)
```

5.3 Writing the Odds Ratio function

Procedure to follow:

1. Find the MAP for the probability function of the free model. It should agree closely but not exactly with the result from `scipy.ODR` because we are using a slightly different method.
2. Calculate the variance of the logarithm of the posterior as $(dP\ /\ d\alpha)^{-}1$
3. Calculate $P(D\,|M_i, I)$ for each theoretical model M_i
4. Calculate the Odds Ratio and print the results.

```
In [16]: def model_selection(X, Y, wdev, alpha, **kwargs):
             """
             Finds MAP for the free model, then does OR calculation for free
             model versus theoretical predictions.

             Params:
             -------
             X - The x-coordinates of the points to be used
             Y - y-coordinates
             wdev - the error in the y-coordinates
             alpha - slope for XY=X and XY=Y models. Must be a dictionary of
                     the form {'XY=X': a number, 'XY=Y': a number}
             guess - starting guess for the MAP approximation (we will use the
                     output from scipy.ODR)

             Outputs:
             Prints the OR for the models.
             """
             guess = kwargs.pop('guess', -0.5)

             # calculate probability of free model:
             res = scipy.optimize.minimize(neg_log_prob_free, guess,
                                           args=(X, Y, wdev), method='Powell')

             # Compute error bars
```

```
second_derivative = scipy.misc.derivative(log_posterior, res.x,
                                          dx=1.0, n=2, args=(X, Y, wdev), order=3)
cov_free = -1/second_derivative
alpha_free = np.float64(res.x)
log_free = log_posterior(alpha_free, X, Y, wdev)

# log goodness of fit for fixed models
eps = ['xy=x', 'xy=y', 'xy=x=y', 'xy=x+y',
       'suppress']
good_fits = {}

for epistasis in eps:
    log_MAP = log_posterior_fixed(wdev, X, Y, model=epistasis,
                                  alpha=alpha)

    good_fits[epistasis] = log_free - log_MAP


    # occam factor - only the free model has a penalty
    log_occam_factor =(-np.log(2 * np.pi) + np.log(cov_free)
                       - 0) / 2

    # give more standing to simpler models. but just a little bit!
    lg = log_free - log_MAP + log_occam_factor - 2
    print('{0} Odds Ratio: {1:2g}'.format(epistasis, np.exp(lg)))

std = np.float64(np.sqrt(cov_free))
m = 'the value used for the observed fit was {0:.3g} +/- {1:.3g}'
print(m.format(alpha_free, std))
```

5.4 Odds ratio for the epistasis between *egl-9* and *vhl-1*

Now that we have written our odds ratio functions, we should test it. Now, one thing to bear in mind is that we have written the theoretical models in such a way that they are extremely conservative. This means that ANY systematic deviation from them will rapidly lead to their rejection in favor of the slightly more complex (but theoretically less pleasing) free-slope model. As a result, we need to be careful how we interpret the Odds ratio. Here are some guidelines:

- Reject theoretical models when there is strong support for them. This means reject when $OR > 10^3$
- When in need of an interpretation, selecting the model with the best support is also valid. Rejecting a model just means we need to keep in mind the epistasis is not exactly what we expected... but when push comes to shove we have to pick a conclusion. Select the conclusion with the most evidence, i.e., the lowest odds ratio.
- Use your gut. If something isn't right, study it more. Let the data speak until you can resolve the controversy.

```
In [17]: alpha_eglvhl = {'xy=x': s['xy=x'].mean(),
                        'xy=y': s['xy=y'].mean()
```

```
        }

    # Calculate X-coordinates
    X = x.b.values + y.b.values
    # Calculate Y-coordinates
    Y = xy.b.values - X
    # Calculate the corrected standard error for the Y-axis
    wdev = np.sqrt(x.se_b.values**2 + y.se_b.values**2 +\
                   xy.se_b.values**2)
    # do the model selection
    model_selection(X, Y, wdev, alpha=alpha_eglvhl,
                    guess=actual.beta[0])
```

```
xy=x Odds Ratio: 0.136722
xy=y Odds Ratio: inf
xy=x=y Odds Ratio: 2.994e+80
xy=x+y Odds Ratio: inf
suppress Odds Ratio: inf
the value used for the observed fit was -0.4 +/- 0.00506
```

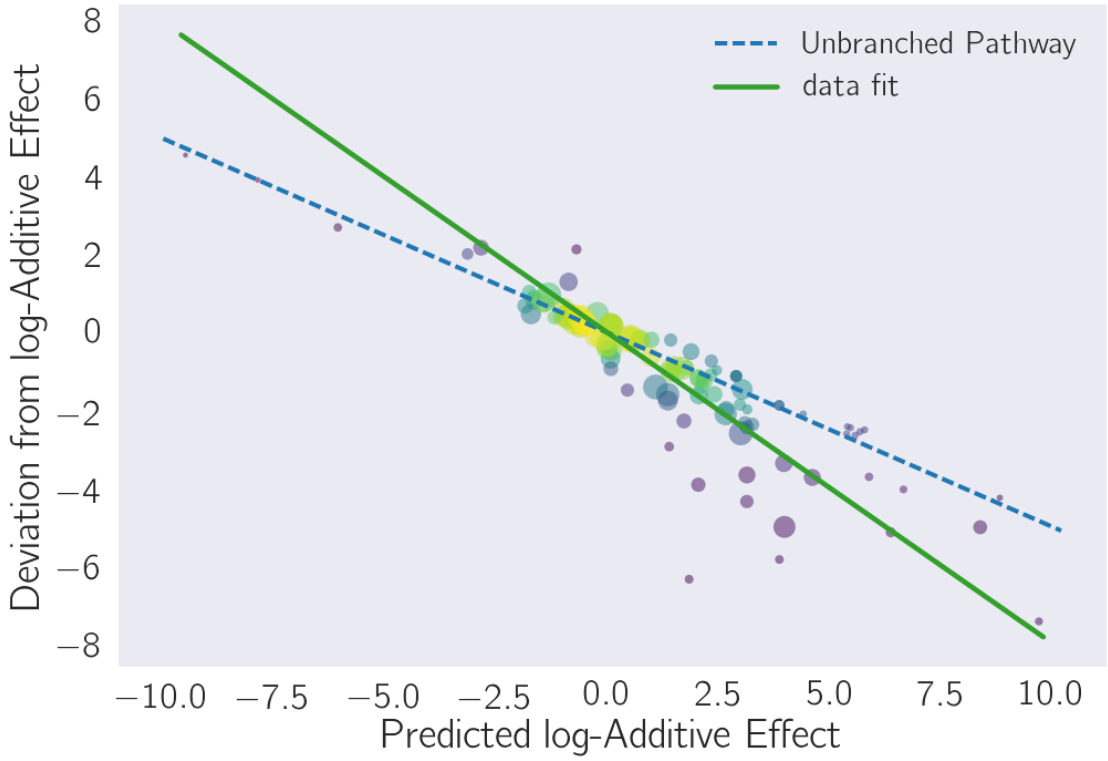
Wow. We can see that all of the models are basically rejected in favor of the free-slope model. Except one. We fail to reject the model $XY = X$. In this case, X is *egl-9*, and Y is *vhl-1*. This means that the parameter-free prediction that *egl-9* is epistatic over *vhl-1* is a preferred model over the free-slope model. Genetics. Works.

6 Measuring suppressive epistasis

6.1 hif-1 suppresses egl-9

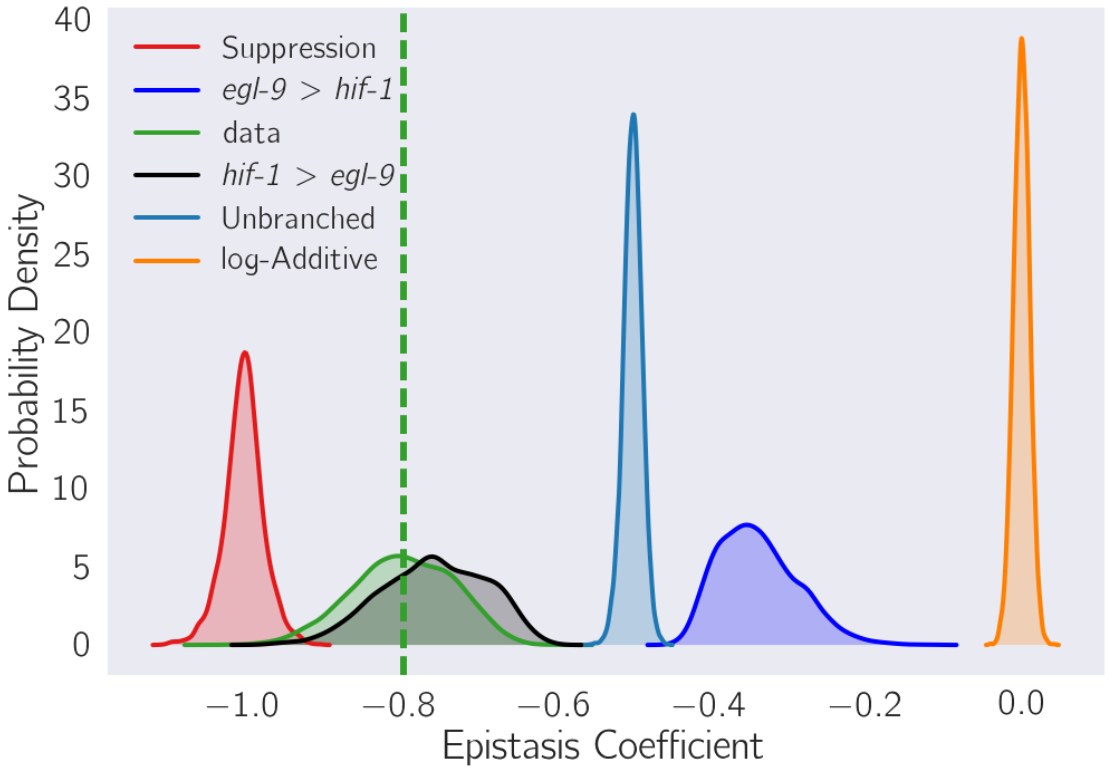
This is self explanatory, but let's repeat the analysis above for *egl-9* and *hif-1*.

```
In [18]: letter1 = 'b'
         letter2 = 'c'
         double = genvar.double_mapping[letter1 + letter2]
         x, y, xy, ax = epi.make_epiplot([letter1, letter2], double, tidy_data)
         plt.savefig('../output/epistasis{0}{1}.svg'.format(genvar.mapping[letter1],
                                                            genvar.mapping[letter2]),
                     bbox_inches='tight')
```



We can immediately notice a couple of things. First, there are a LOT less points here (around 50) as opposed to the previous plot (around 330). Secondly, they form a line that has a slope $< -\frac{1}{2}$. Both of these are characteristic of a gene that is under strong suppression. Let's see what our models will predict when we simulate them.

```
In [19]: s = epi.calculate_all_bootstraps(letter1, letter2, double,
                                           tidy_data, nsim=5000)

In [20]: ax = epi.plot_bootstraps(letter1, letter2, s,
                                  cumulative=False, shade=True)
         plt.xlabel('Epistasis Coefficient')
         plt.ylabel('Probability Density')
         plt.savefig('../output/supp_figures/supplementary_figure_2.svg'.format(genvar.mapping
                                                                                genvar.mapping
                     bbox_inches='tight')
```



We notice a couple of things from this graph. First, all of the predictions are considerably wider. This is the result of the considerably smaller number of points in this dataset. The observed fit distribution overlaps significantly with a model where *hif-1* is epistatic over *egl-9* (black curve), but also with the model of complete suppression. Let's take a look at the OR before we can decide what's going on.

```
In [21]: alpha_eglhif = {'xy=x': s['xy=x'].mean(),
                         'xy=y': s['xy=y'].mean()
                         }

         actual = epi.ODR([x,y], xy, epistasis='actual')
         X = x.b.values + y.b.values
         Y = xy.b.values - X
         wdev = np.sqrt(x.se_b.values**2 + y.se_b.values**2 + xy.se_b.values**2)
         model_selection(X, Y, wdev, alpha=alpha_eglhif, guess=actual.beta[0])
```

```
xy=x Odds Ratio: 1.27982e+251
xy=y Odds Ratio: 0.000356019
xy=x=y Odds Ratio: 4.66488e+93
xy=x+y Odds Ratio: inf
suppress Odds Ratio: 1.86065e+79
the value used for the observed fit was -0.76 +/- 0.0123
```

In this case, we reject all of the models. If we really wanted to select a model, we would say that $XY = Y$ is the one that maximizes the probability of observing the data. The second most likely model is the complete suppression model. Well, this matches intuition. In this case, I am not offended by our inability to select an OR. We had very few data points.

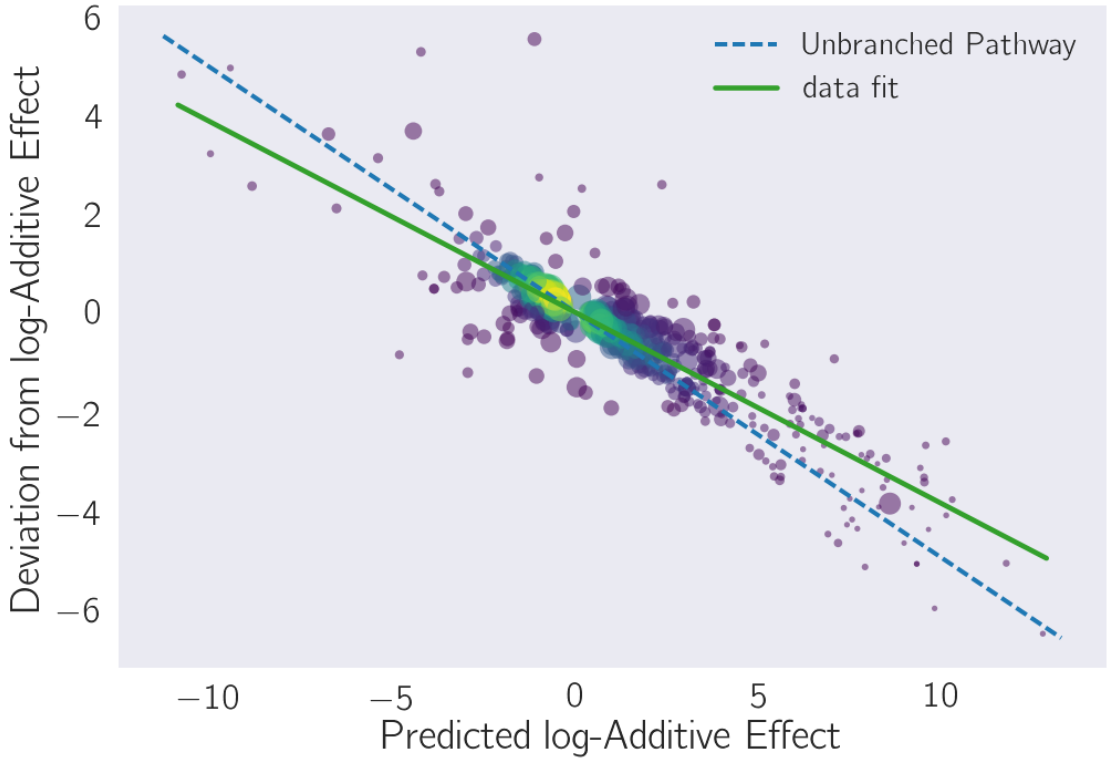
7 Transitivity in transcriptomes

In theory, if two genes are acting through a linear pathway, then both genes should have identical transcriptomes. If they are truly equal, we should be able to substitute one transcriptome for another for any computation we are performing. It follows that we should be able to substitute transcriptomes to predict and/or measure epistasis between two genes if we have a third gene that is related via a linear pathway.

7.1 Predicting epistasis between *egl-9* and *vhl-1* using the *rhy-1* transcriptome

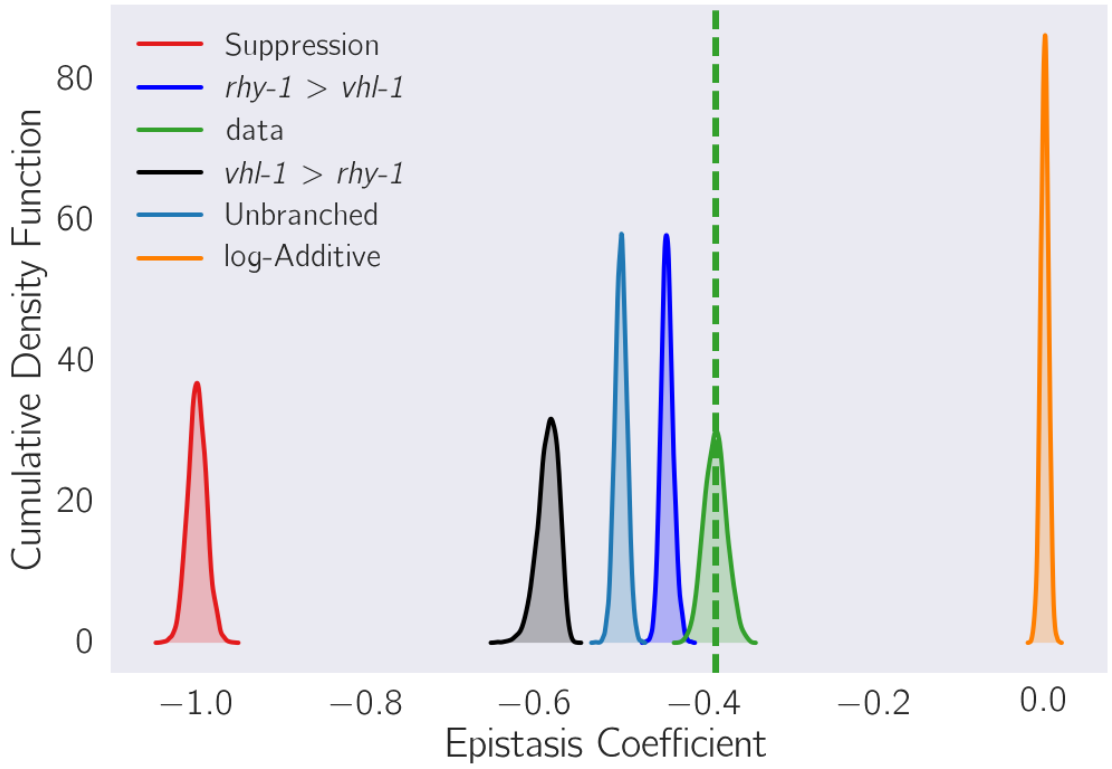
Recall that *rhy-1* genetically activates *egl-9*. If transcriptomes are transitive, then we could use the *rhy-1* transcriptome to predict the epistasis coefficient between *egl-9* and *vhl-1*. We could also use it to "measure" the transcriptome-wide coefficient by substituting *rhy-1* instead of the *egl-9* mutant.

```
In [22]: letter1 = 'e'
         letter2 = 'd'
         double = genvar.double_mapping['b' + letter2]
         x, y, xy, ax = epi.make_epiplot([letter1, letter2], double, tidy_data)
```



```
In [23]: s = epi.calculate_all_bootstraps(letter1, letter2, double,
                                          tidy_data, nsim=5000)
```

```
In [24]: ax = epi.plot_bootstraps(letter1, letter2, s, cumulative=False, shade=True)
         plt.xlabel('Epistasis Coefficient')
         plt.ylabel('Cumulative Density Function')
         plt.savefig('../output/kde-epistasis{0}{1}.svg'.format(genvar.mapping[letter1],
                                                                genvar.mapping[letter2]),
                     bbox_inches='tight')
```




```
In [25]: alpha = {'xy=x': s['xy=x'].mean(),
                  'xy=y': s['xy=y'].mean()
                 }

         actual = epi.ODR([x,y], xy, epistasis='actual')
         m = 'The observed slope from the ODR regression was {0:.2g}'
         print(m.format(actual.beta[0]))
         X = x.b.values + y.b.values
         Y = xy.b.values - X
         wdev = np.sqrt(x.se_b.values**2 + y.se_b.values**2 + xy.se_b.values**2)
         model_selection(X, Y, wdev, alpha=alpha, guess=actual.beta[0])
```

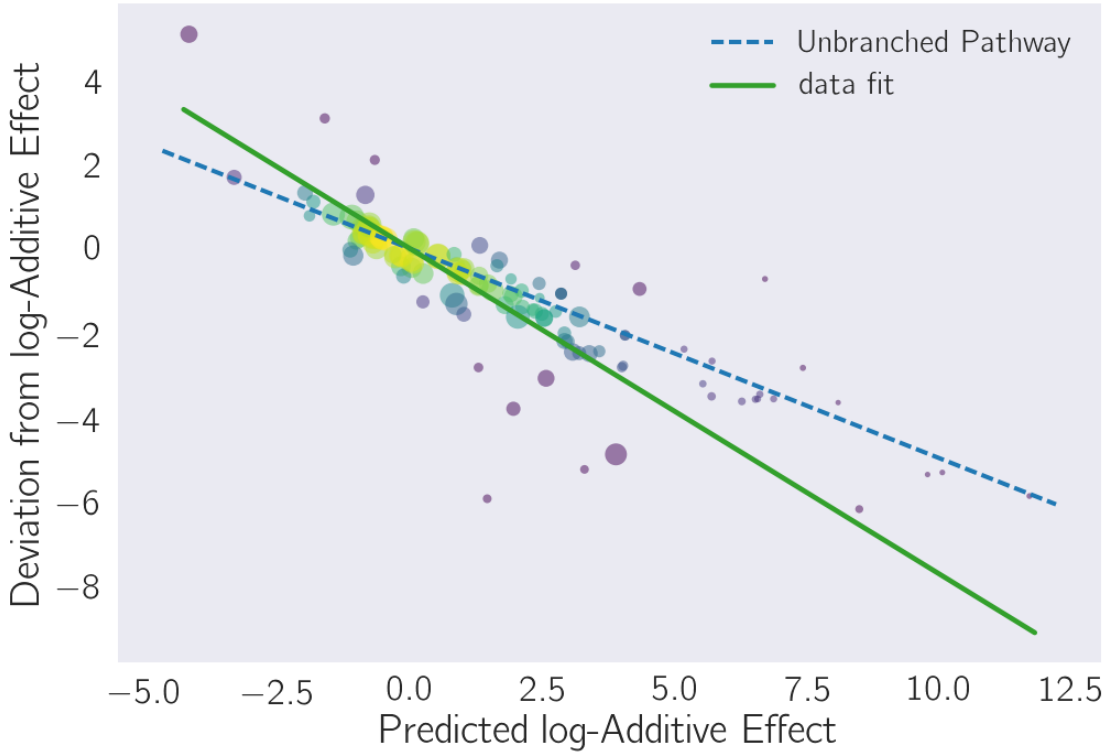
```
The observed slope from the ODR regression was -0.39
xy=x Odds Ratio: 5.38647e+29
```

```
xy=y Odds Ratio: 4.5473e+302
xy=x=y Odds Ratio: 5.47657e+102
xy=x+y Odds Ratio: inf
suppress Odds Ratio: inf
the value used for the observed fit was -0.376 +/- 0.00559
```

If we use ODR to predict the epistasis coefficient, we would "measure" an epistasis value of -0.38, which agrees with what we obtained with the *egl-9* mutant. However, unlike with the *egl-9* mutant data, the odds ratio test fails to accept any theoretical model. Clearly, there are deviations that occur between *rhy-1* and *egl-9*. Maybe knocking out *rhy-1* does not fully inactivate *egl-9*. Indeed, this is a good hypothesis. We see that the epistasis models, $XY = X$ and $XY = Y$, both begin to overlap with the unbranched model. This could suggest that knocking out *rhy-1* inhibits the VHL-1-dependent inhibition of HIF-1 by EGL-9, but not the remaining inhibition.

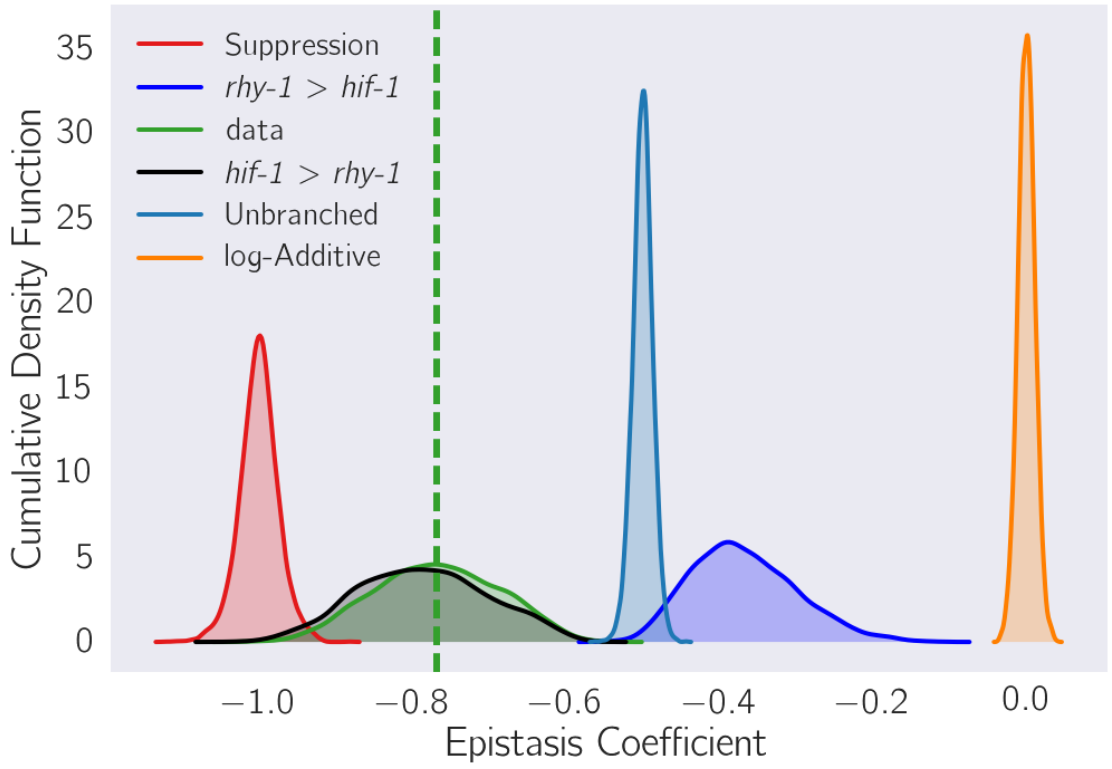
7.2 Predicting epistasis between *egl-9* and *hif-1* using the *rhy-1* transcriptome

```
In [26]: letter1 = 'e'
         letter2 = 'c'
         double = genvar.double_mapping['b' + letter2]
         x, y, xy, ax = epi.make_epiplot([letter1, letter2],
                                         double, tidy_data)
```



```
In [27]: s = epi.calculate_all_bootstraps(letter1, letter2, double,
                                          tidy_data, nsim=5000)
```

```
In [28]: ax = epi.plot_bootstraps(letter1, letter2, s, cumulative=False, shade=True)
         plt.xlabel('Epistasis Coefficient')
         plt.ylabel('Cumulative Density Function')
         plt.savefig('../output/kde-epistasis{0}{1}.svg'.format(genvar.mapping[letter1],
                                                                genvar.mapping[letter2]),
                     bbox_inches='tight')
```




```
In [29]: alpha = {'xy=x': s['xy=x'].mean(),
                  'xy=y': s['xy=y'].mean()
                 }

         actual = epi.ODR([x,y], xy, epistasis='actual')
         X = x.b.values + y.b.values
         Y = xy.b.values - X
         wdev = np.sqrt(x.se_b.values**2 + y.se_b.values**2 + xy.se_b.values**2)
         model_selection(X, Y, wdev, alpha=alpha, guess=actual.beta[0])
```

```
xy=x Odds Ratio: 6.7891e+151
xy=y Odds Ratio: 11.7818
xy=x=y Odds Ratio: 2.11803e+58
```

```
xy=x+y Odds Ratio: inf
suppress Odds Ratio: 4.23636e+88
the value used for the observed fit was -0.725 +/- 0.0133
```

7 Hydroxylated Hif-1

January 31, 2018

1 Table of Contents

1 Genes that display non-canonical epistasis:
2 Plotting genes that display non-canonical changes:
2.1 Figure 7A
2.2 7B

In this notebook, I will identify genes that do not conform to the canonical epistasis relationships expected for the hypoxia pathway in *C. elegans*.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

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

# 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)
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%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

mpl.rcParams['xtick.labelsize'] = 18
mpl.rcParams['ytick.labelsize'] = 18
mpl.rcParams['legend.fontsize'] = 14

genvar = gvars.genvars()
q=0.1

tidy_data = pd.read_csv('../output/temp_files/DE_genes.csv')
tidy_data.sort_values('target_id', inplace=True)
tidy_data.dropna(subset=['ens_gene'], inplace=True)
tidy_data = tidy_data[tidy_data.genotype != 'fog-2']
tidy_data['fancy genotype'] = tidy_data.code.map(genvar.fancy_mapping)
```

2 Genes that display non-canonical epistasis:

To identify genes that display non-canonical epistasis, I will fuse some columns to the dataframe containing the *rhy-1* transcriptome. Using these columns, we will find genes that have inverse expression changes between *vhl-1(lf)* mutants and *egl-9(lf)* or *rhy-1(lf)* mutants.

In [2]:
```
# Specify the genotypes to refer to:
single_mutants = ['b', 'c', 'd', 'e', 'g']

# Specify which letters are double mutants and their genotype
double_mutants = {'a' : 'bd', 'f':'bc'}

# initialize the morgan.hunt object:
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/'
        sleuth_loc = '../sleuth/kallisto/'
        thomas.add_tpm(kallisto_loc, '/kallisto/abundance.tsv', '')
        # load all the beta dataframes:
        for file in os.listdir("../sleuth/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)
                thomas.filter_data()

        # place all
        df1 = thomas.beta['e'].copy()
        df2 = thomas.beta['b']
        df3 = thomas.beta['d']

        df1['b_b'] = df2.b
        df1['b_d'] = df3.b

        df1['q_b'] = df2.qval
        df1['q_d'] = df3.qval
```

In [3]:
```
        # use least strict conditions:
        lowestrhy = (df1.b*df1.b_d < 0)    # egl anti vhl

        lowestsigrhy = ((df1.qval < q) &   # egl sig
                        (df1.q_d < q))     # vhl sig


        lowestegl = (df1.b_b*df1.b_d < 0)  # egl anti vhl


        lowestsigegl = ((df1.q_b < q) &    # egl sig
                        (df1.q_d < q))     # vhl sig
```

Now that we have coded up the conditions, let's see what we get!

In [4]:
```
        df1.sort_values('qval', ascending=True)

        hifoh = df1[
                (lowestegl & lowestsigegl) |
                (lowestrhy & lowestsigrhy)].target_id.unique()

        print('{0} candidates found for HIF-1-OH regulation'.format(len(hifoh)))
        df1[(lowestegl & lowestsigegl) |
            (lowestrhy & lowestsigrhy)].to_csv('../output/temp_files/hifoh_candidates.csv', inc
```

```
56 candidates found for HIF-1-OH regulation
```

```
In [5]: hypoxia = pd.read_csv('../output/temp_files/hypoxia_response.csv')
```

```
In [6]: len(hypoxia[hypoxia.target_id.isin(hifoh)].ens_gene.unique())
```

```
Out[6]: 14
```

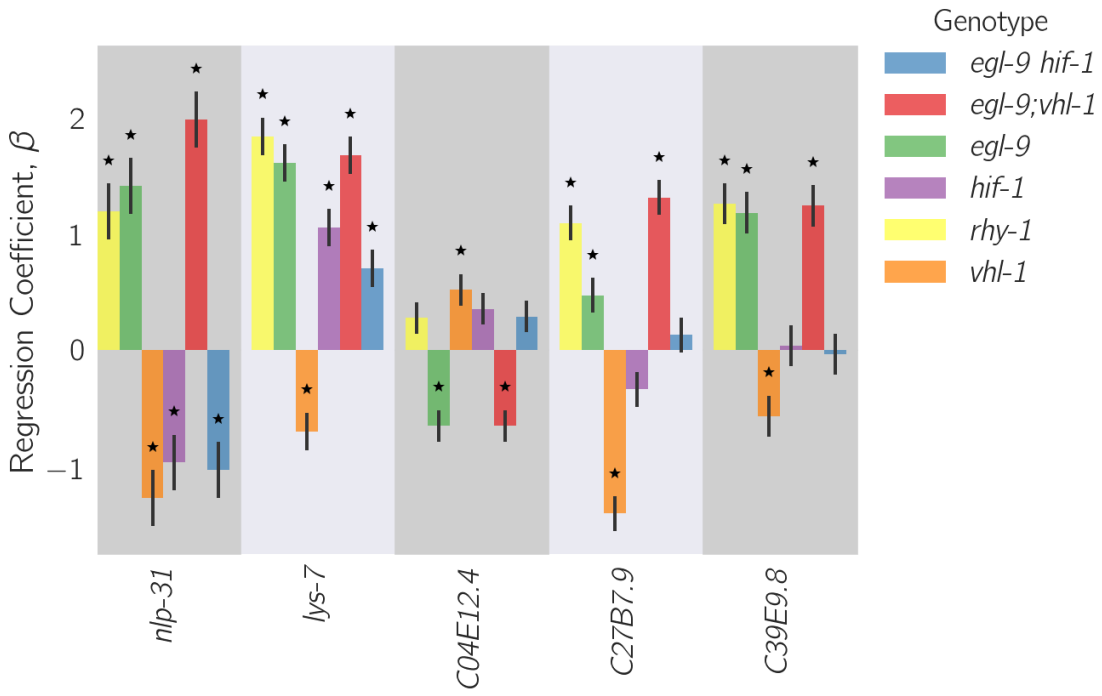
3 Plotting genes that display non-canonical changes:

So far, all I have done is find the genes that have different expression between *vhl-1* and *egl-9*. It would be very interesting if genes that have these different behaviors still conform to the same epistatic rules (*egl-9* = *egl-9;vhl-1* and *hif-1* = *egl-9 hif-1*). We can make a qPCR plot to see if that is the case:

```
In [7]: tidy = tidy_data[tidy_data.target_id.isin(hifoh)].copy()

        x_sort = {}
        for i, xi in enumerate(tidy.target_id.unique()):
            x_sort[xi] = i + 1


        tidy['order'] = tidy.target_id.map(x_sort)
        tidy.sort_values('order', inplace=True)
        tidy.reset_index(inplace=True)
```

Initially, let's just look at the first five genes in our set:

```
In [8]: ind = tidy.target_id.isin(tidy.target_id.unique()[0:5])
        genpy.qPCR_plot(tidy[ind], genvar.plot_order, genvar.plot_color,
                        clustering='fancy genotype', plotting_group='target_id', rotation=90)
```



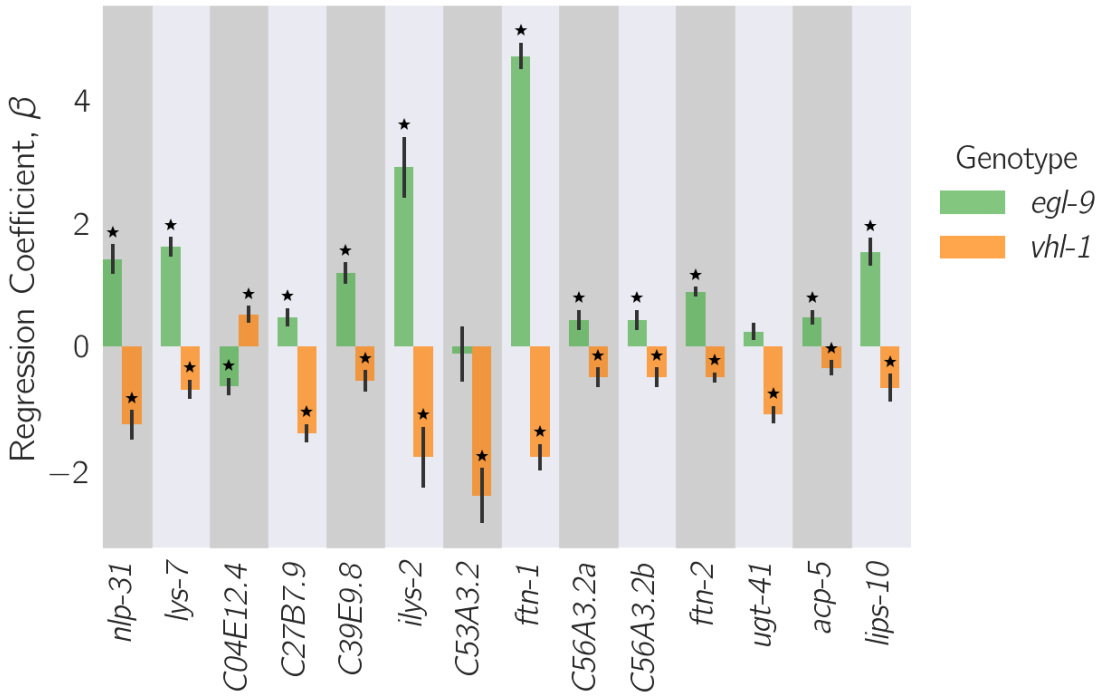
Wow! All of them obey the epistatic rules! This is cool.

3.1 Figure 7A

Next, i will generate figure 7A and 7B in the paper.

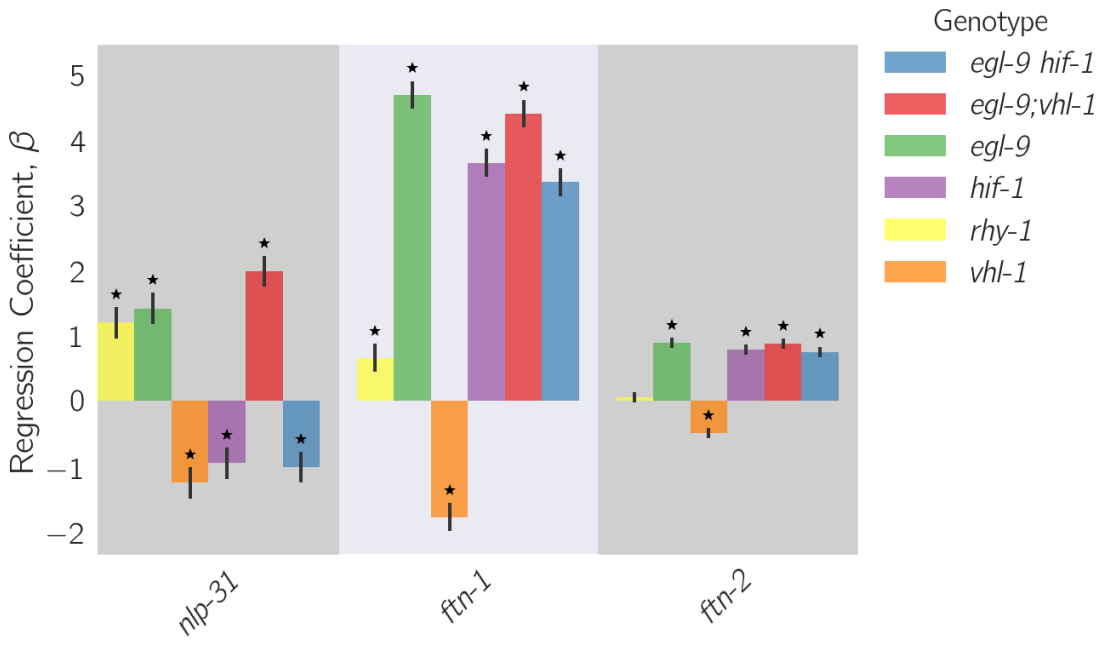
```
In [9]: new_order = {r'\emph{egl-9}': 0,
                     r'\emph{vhl-1}': 1,
                    }
```

```
In [10]: genpy.qPCR_plot(tidy[(tidy.target_id.isin(hifoh[0:15])) & (tidy.code.isin(['b', 'd']))
                         new_order, genvar.plot_color, clustering='fancy genotype',
                         plotting_group='target_id', rotation=90)

         plt.savefig('../output/vhl1_noncanonical.svg')
```



3.2 7B

```
In [11]: to_focus_on = ['nlp-31', 'ftn-1', 'ftn-2']

         gene_order = {'nlp-31': 1, 'ftn-1': 2, 'ftn-2': 3}

         temp = tidy[(tidy.ext_gene.isin(to_focus_on))].copy()
         del temp['order']
         temp['order'] = temp.ext_gene.map(gene_order)

         genpy.qPCR_plot(temp, genvar.plot_order, genvar.plot_color,
                         clustering='fancy genotype', plotting_group='ext_gene')
         plt.savefig('../output/hif1oh_qPCR.svg', bbox_inches='tight')
```


Genotype
egl-9 hif-1
egl-9;vhl-1
egl-9
hif-1
rhy-1
vhl-1
Regression Coefficient, β
5
4
3
2
1
0
−1
−2
nlp-31
ftn-1
ftn-2

In [ ]:

8 Pathway Overview of the pan-hif-1 transcriptome

January 31, 2018

1 Table of Contents

1 Defining a gene compactifier for easy printing
2 Effects of HIF-1 on mitochondrial proteins
3 HIF-1 effects on the ribosome
4 Bioenergetics of HIF-1
5 Effects of HIF-1 on the Proteasome and Mediator
5.1 Effect of HIF-1 on proteins involved in 'protein catabolic process'
6 Proteins annotated as involved in protein folding
7 Immune Involvement

An important question that I also wanted to address was the cell-wide effects of HIF-1. Although the hypoxia response by itself is informative as to what HIF-1 actually turns on and off in *C. elegans*, enrichment analyses are not the only way to get information out of transcriptomes.

Another way to get information about these effects is to change what biological units we are studying. In this paper, we have focused a lot on single genes. However, we could also ask what pathways, or what entities, are represented in our dataset.

The way I will look at pathways is by identifying the genes that are in a 'pathway' or biological process of interest. I will extract the genes within this process that are differentially expressed in each mutant. Then, I will look at how the pathway changes *overall*. If a pathway is being down-regulated in a given set of mutants, we would expect that all of the genes that are D.E. in this pathway would show up as down-regulated. However, we no longer require that ALL of the genes in this pathway be D.E. in our dataset.

When a pathway is mainly changing in one direction, with the exception of a single gene that is changing in the opposite direction, I only consider that gene to be informative if and only if it was represented in 2 samples or more. Why? Because false positives exist, but we also need to take into consideration that pathways are human constructs that are likely to be incomplete. Branching may be ocurring, and there could be specific reasons for why a single node changes in opposite direction to the rest of the pathway.

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

# morgan
import morgan as morgan
import tissue_enrichment_analysis as tea
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# 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'}

import genpy
import seqplotter
import gvars
import epistasis as epi

q = 0.1
genvar = gvars.genvars()
```

I will load the respiratory complexes and central dogma complexes, which I obtained from a manual curation of Wormbase using WormMine

```
In [2]: respiratory_complexes = pd.read_excel('../input/respiratory_complexes.xlsx')
        central_dogma = pd.read_excel('../input/central_dogma.xlsx')
```

```
In [3]: tissue_df = tea.fetch_dictionary()
        phenotype_df = pd.read_csv('../input/phenotype_ontology.csv')
        go_df = pd.read_csv('../input/go_dictionary.csv')
```

```
In [4]: melted_tissue = pd.melt(tissue_df, id_vars='wbid',
                                 var_name='term', value_name='expressed')
        melted_tissue = melted_tissue[melted_tissue.expressed == 1]

        melted_phenotype = pd.melt(phenotype_df, id_vars='wbid',
                                   var_name='term', value_name='expressed')
        melted_phenotype = melted_phenotype[melted_phenotype.expressed == 1]

        melted_go = pd.melt(go_df, id_vars='wbid',
                            var_name='term', value_name='expressed')
        melted_go = melted_go[melted_go.expressed == 1]
```

```
In [5]: tidy_data = pd.read_csv('../output/temp_files/DE_genes.csv')
        tidy_data.sort_values('target_id', inplace=True)
        tidy_data.dropna(subset=['ens_gene'], inplace=True)
        # tidy_data.sort_values('sort_order', inplace=True)

        # drop the fog-2 data:
        tidy_data = tidy_data[tidy_data.genotype != 'fog-2']
        # tidy_data = tidy_data[tidy_data.qval < q]  # keep only sig data.
```

2 Defining a gene compactifier for easy printing

Before we start, I will define a function, called `gene_compactifier` which will make visualization of gene representation much easier. How does it work?

Given a gene list, it: 1. Finds all the genes that have the same WORM family name. In other words, find all the *unc* genes, all the *rpl* genes. 2. If there's more than one gene in a given family, print the number of genes that have that family name. 3. Print a list of all the suffixes.

So if a gene list contains *unc-119*, *unc-15* and *unc-1*, the program will output:

```
Gene "Family", Number Found unc,  3 ['1', '15', '119']
```

Moreover, if a gene list contains *unc-119*, *unc-119* and *unc-119* (the same gene repeated *n* times), the program will output:

```
Gene "Family", Number Found unc,  3 ['119', '119', '119']
```

This makes it quite easy to visualize what genes within a pathway are represented in all of the mutants (coverage), as well as how many times each gene is represented in the dataset (coverage).

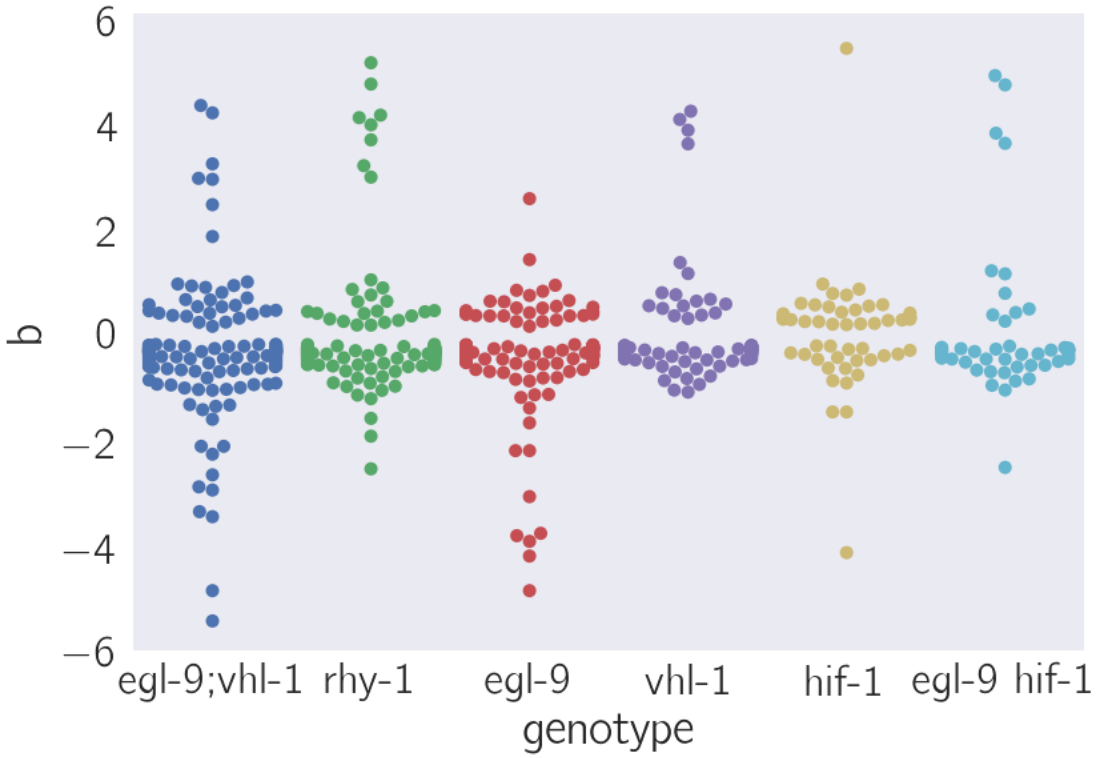
```
In [6]: def gene_compactifier(ext_gene):
            """Given a list of ext_gene names, compactify them and print"""
            d = {}
            ext_gene = sorted(ext_gene)
            for gene in ext_gene:
                ind = gene.find('-')
                if ind > 1:
                    name = gene[:ind]
                    number = gene[ind+1:]
                else:
                    name = gene
                    number = ''

                if name in d.keys():
                    d[name] += [number]
                else:
                    d[name] = [number]

            print('Gene "Family", Number Found')
            for name, numbers in d.items():
                if len(numbers) > 1:
                    print(name + ', ', len(numbers), sorted(numbers))
                else:
                    if len(numbers[0]) > 0:
                        print(name + '-' + numbers[0])
                    else:
                        print(name)
```

3 Effects of HIF-1 on mitochondrial proteins

First, let me extract all the genes that are overrepresented in mitochondria. The way I do this is via a function call `plot_by_term` which, given a string, a dataframe to search, and the kind of ontology that the string should be found in, plots for each genotype the perturbation values of the

significantly altered genes and returns the axis that contains that plot, as well as the list of genes that are annotated with the desired string.

```
In [7]: ax, mito = seqplotter.plot_by_term('mitochondrion', df=tidy_data,
                                           kind='go', swarm=True)
```



Visually, it looks like maybe 1/3 of the mitochondrial genes go up, and the rest go down. What genes are most represented in this pathway? How many points consistently show up in mutants that have a constitutive HIF-1 response? Let's find out.

Next, I find out what genes that are annotated with the term 'mitochondria' go up across genotypes with a constitutive HIF-1 response:

```
In [8]: common  = epi.find_overlap(['e', 'b', 'd', 'a'], tidy_data)
        trial = tidy_data[(tidy_data.ens_gene.isin(mito)) &
                          (tidy_data.target_id.isin(common)) &
                          (tidy_data.b > 0)].ext_gene
        gene_compactifier(trial)

Gene "Family", Number Found
F02A9.4
acl,  6 ['6', '6', '6', '6', '6', '6']
mdh-2
Y53G8AL.2
phb,  2 ['1', '2']
```

```
pcca,  5 ['1', '1', '1', '1', '1']
Y39E4A.3
F20D6.11
ZK669.4
wah-1
F53F4.10
fum,  6 ['1', '1', '1', '1', '1', '1']
nuo-1
B0272.3
sucl-1
timm-23
got-2.1
oxa-1
tomm-40
atp-5
sdha,  5 ['1', '1', '1', '1', '1']
mai,  5 ['1', '1', '1', '1', '1']
```

What about the genes that go DOWN in all genotypes with a constitutive HIF-1 response?

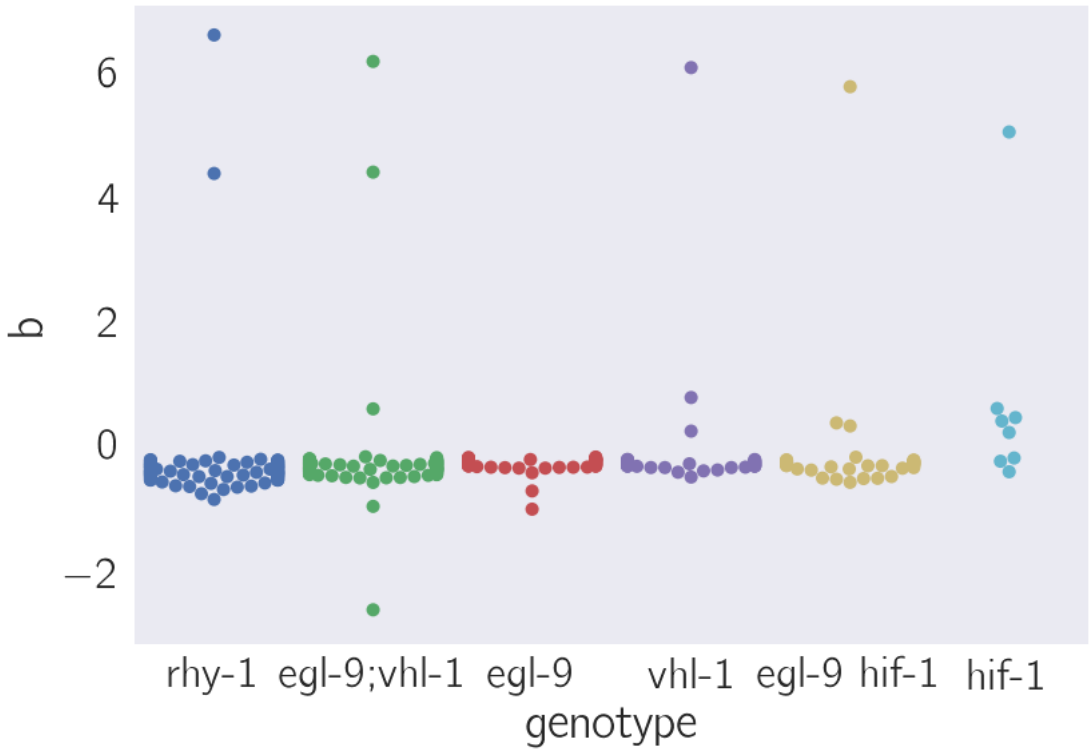
```
In [9]: trial = tidy_data[(tidy_data.ens_gene.isin(mito)) &
                          (tidy_data.target_id.isin(common)) &
                          (tidy_data.b < 0)].ext_gene
        gene_compactifier(trial)
```

```
Gene "Family", Number Found
F45H10.3,  6 ['', '', '', '', '', '']
C05C10.3,  6 ['', '', '', '', '', '']
ZK1320.9,  6 ['', '', '', '', '', '']
sco,  6 ['1', '1', '1', '1', '1', '1']
mdh,  5 ['2', '2', '2', '2', '2']
Y53G8AL.2,  5 ['', '', '', '', '']
acdh,  6 ['1', '1', '1', '1', '1', '1']
Y39E4A.3,  5 ['', '', '', '', '']
wah,  5 ['1', '1', '1', '1', '1']
Y48A6B.3,  6 ['', '', '', '', '', '']
Y38F1A.6,  6 ['', '', '', '', '', '']
T27E9.2,  6 ['', '', '', '', '', '']
sucg,  6 ['1', '1', '1', '1', '1', '1']
phb,  10 ['1', '1', '1', '1', '1', '2', '2', '2', '2', '2']
pdhb,  6 ['1', '1', '1', '1', '1', '1']
sdha-1
cyc,  6 ['1', '1', '1', '1', '1', '1']
acaa,  6 ['2', '2', '2', '2', '2', '2']
F02A9.4,  11 ['', '', '', '', '', '', '', '', '', '', '']
T02G5.7,  6 ['', '', '', '', '', '']
mrps,  6 ['6', '6', '6', '6', '6', '6']
```

```
F54D5.12,  6 ['', '', '', '', '', '']
mtss,  6 ['1', '1', '1', '1', '1', '1']
Y54F10AM.5,  6 ['', '', '', '', '', '']
pcca-1
C14B9.10,  6 ['', '', '', '', '', '']
F20D6.11,  5 ['', '', '', '', '']
ZK669.4,  5 ['', '', '', '', '']
F56B3.11,  6 ['', '', '', '', '', '']
nduf,  6 ['7', '7', '7', '7', '7', '7']
F53F4.10,  5 ['', '', '', '', '']
sucl,  5 ['1', '1', '1', '1', '1']
R07E5.13,  6 ['', '', '', '', '', '']
B0272.3,  5 ['', '', '', '', '']
nuo,  11 ['1', '1', '1', '1', '1', '6', '6', '6', '6', '6', '6']
timm,  5 ['23', '23', '23', '23', '23']
got,  5 ['2.1', '2.1', '2.1', '2.1', '2.1']
oxa,  5 ['1', '1', '1', '1', '1']
tomm,  11 ['22', '22', '22', '22', '22', '22', '40', '40', '40', '40', '40']
atp,  5 ['5', '5', '5', '5', '5']
C25H3.9,  6 ['', '', '', '', '', '']
hsp,  6 ['60', '60', '60', '60', '60', '60']
mrpl,  24 ['2', '2', '2', '2', '2', '2', '47', '47', '47', '47', '47', '47', '47', '47',
mai-1
F09F7.4,  6 ['', '', '', '', '', '']
```

4 HIF-1 effects on the ribosome

```
In [10]: term = 'structural constituent of ribosome GO:0003735'
         ax, ribosome = seqplotter.plot_by_term(term, df=tidy_data, kind='go')
```




```
In [11]: trial = tidy_data[(tidy_data.ens_gene.isin(ribosome)) & (tidy_data.qval < q)].ext_gene
         gene_compactifier(trial)
```

```
Gene "Family", Number Found
dap-3
F54D7.6
rps,  28 ['1', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '
ubq-2
mrps,  16 ['12', '14', '15', '16', '17', '18B', '18C', '21', '22', '23', '24', '25', '34', '6'
C37A2.7
ubl-1
rpl,  41 ['1', '10', '11.1', '11.2', '12', '13', '14', '15', '16', '17', '18', '19', '2', '20'
W01D2.1
F54D7.7
T07A9.14
rla,  3 ['0', '1', '2']
mrpl,  18 ['10', '11', '12', '13', '15', '16', '17', '19', '2', '23', '24', '32', '34', '41',
Y37E3.8
```

5 Bioenergetics of HIF-1

What about the effects of HIF-1 on the Electron Transport Chain? Or the TCA cycle?

To explore this, I will make a new dataframe, that contains only the genes in the ETC. I will also annotate each gene with the complex it belongs to, and then I will add a column called sort_order so I can sort the dataframe at my pleasure.

```
In [12]: resp = tidy_data[tidy_data.ens_gene.isin(respiratory_complexes.ens_gene)
                          & (~tidy_data.code.isin(['f', 'c']))].copy()

         f = lambda x: respiratory_complexes[respiratory_complexes.ens_gene == x].complex.value
         resp['complex'] = resp.ens_gene.map(f)

         f = lambda x: respiratory_complexes[respiratory_complexes.ens_gene == x].sort_order.va
         resp['sort_order'] = resp.ens_gene.map(f)
         resp.sort_values('sort_order', inplace=True)
         resp = resp[resp.complex != 'Ubiquinone Biosynthesis']
```

This is what the dataframe looks like:

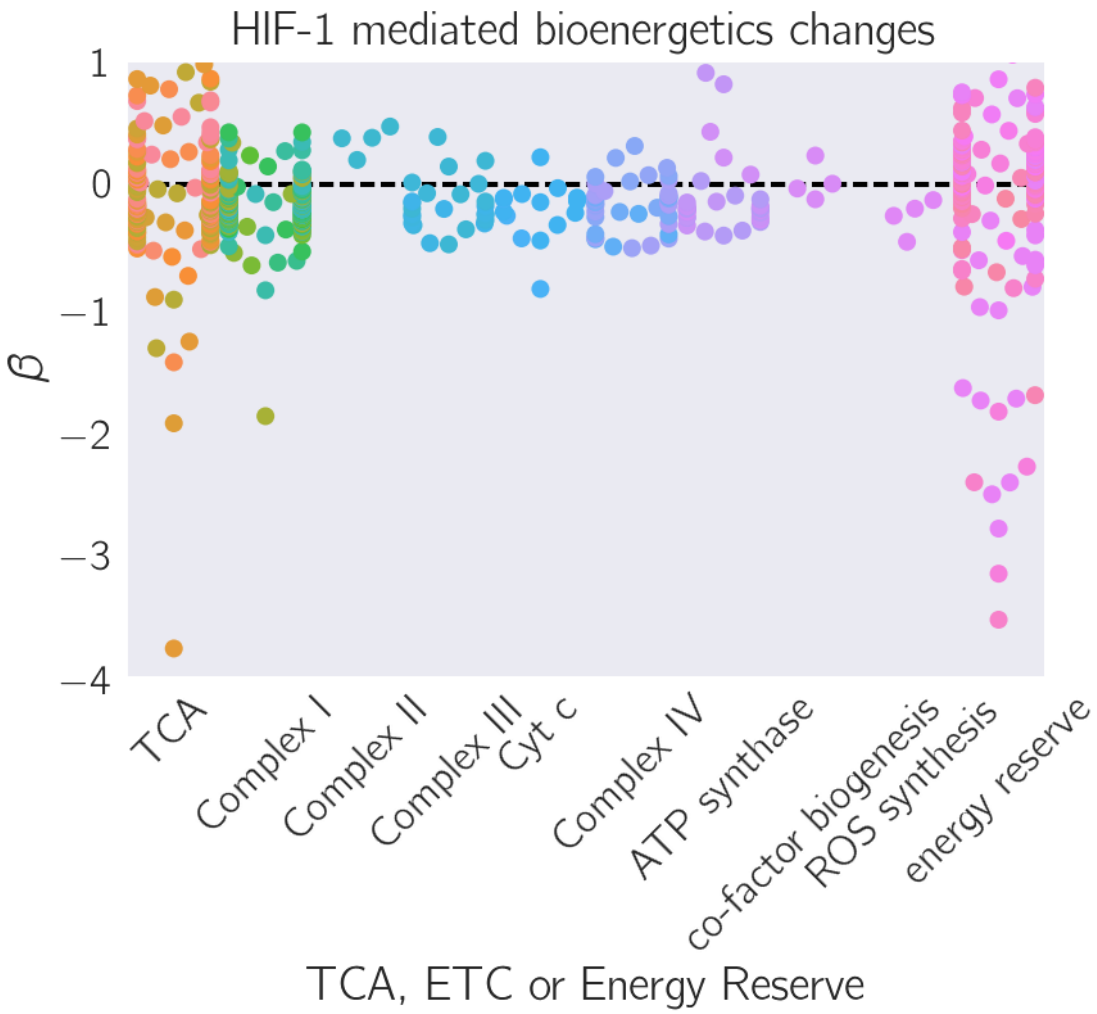
```
In [13]: resp[['ext_gene', 'genotype', 'complex', 'sort_order']].head()
```

Out[13]:

	ext_gene	genotype	complex	sort_order
88496	fum-1	vhl-1	TCA	0
124985	sdhd-1	egl-9;vhl-1	TCA	0
85633	sdhd-1	vhl-1	TCA	0
26605	sdhd-1	rhy-1	TCA	0
124986	sdhd-1	egl-9;vhl-1	TCA	0

Let's plot the dataframe, see what comes out. We would expect all genes in the ETC and TCA to go down:

```
In [14]: fig, ax = plt.subplots()
         ax = sns.swarmplot(x='complex', y='b', hue='ens_gene', data=resp, size=7)
         plt.xticks(rotation=45)
         ax.legend_.remove()
         plt.title('HIF-1 mediated bioenergetics changes')
         plt.ylabel(r'\beta')
         plt.xlabel('TCA, ETC or Energy Reserve')
         ax.hlines(0, xmin=-2, xmax=10, lw=2, linestyle='--')
         plt.ylim(-4, 1)
         plt.savefig('../output/mito_function.pdf')
```



Well, we can definitely see that not all genes in the ETC and TCA go down. Let's figure out what genes go UP in each cycle/complex.

```
In [15]: gene_compactifier(resp[(resp.complex == 'TCA') & (resp.b > 0)].ext_gene)
```

```
Gene "Family", Number Found
fum,  13 ['1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1']
sdhd,  6 ['1', '1', '1', '1', '1', '1']
sdhb,  3 ['1', '1', '1']
mdh,  8 ['1', '1', '1', '1', '1', '1', '1', '1']
ZK836.2,  6 ['', '', '', '', '', '']
ogdh,  7 ['1', '1', '1', '1', '1', '1', '1']
idhb,  3 ['1', '1', '1']
idhg,  4 ['1', '1', '1', '1']
sdha,  4 ['2', '2', '2', '2']
idh,  12 ['1', '1', '1', '1', '1', '1', '1', '1', '2', '2', '2', '2']
sucl,  4 ['2', '2', '2', '2']
```

```
aco,  8 ['2', '2', '2', '2', '2', '2', '2', '2']
```

```
In [16]: gene_compactifier(resp[(resp.complex == 'Complex I') & (resp.b > 0)].ext_gene)
```

```
Gene "Family", Number Found
F44G4.2
C16A3.5,  4 ['', '', '', '']
lpd,  2 ['5', '5']
nuo,  14 ['2', '2', '2', '2', '3', '3', '3', '3', '3', '3', '3', '3', '3', '3']
C25H3.9,  2 ['', '']
nduo,  4 ['3', '4', '5', '5']
ndfl,  3 ['4', '4', '4']
Y51H1A.3,  5 ['', '', '', '', '']
```

```
In [17]: gene_compactifier(resp[(resp.complex == 'Complex II') & (resp.b > 0)].ext_gene)
```

```
Gene "Family", Number Found
sdha,  4 ['1', '1', '1', '1']
```

```
In [18]: gene_compactifier(resp[(resp.complex == 'energy reserve') & (resp.b > 0)].ext_gene)
```

```
Gene "Family", Number Found
Y67D8A.2,  15 ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
agl,  4 ['1', '1', '1', '1']
T22F3.3,  16 ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
T04A8.7,  6 ['', '', '', '', '', '']
gyg,  8 ['1', '1', '1', '1', '1', '1', '1', '1']
gsy-1
CC8.2,  8 ['', '', '', '', '', '', '', '']
aagr,  9 ['1', '1', '1', '1', '3', '3', '3', '3', '3']
Y50D7A.3,  2 ['', '']
oga,  4 ['1', '1', '1', '1']
H18N23.2,  5 ['', '', '', '', '']
ogt,  7 ['1', '1', '1', '1', '1', '1', '1']
```

Notice that for complex I, the genes *nuo-2* and *nduo-4* are up-regulated. But those exact same genes are also down-regulated (see below). Therefore, there is insufficient information to conclude whether these genes are going up, or down as a result of HIF-1. However, for other genes, namely *fum-1* and *sdha-1* we can conclude that those are significantly and consistently up-regulated in mutants that have a constitutive HIF-1 mutant.

```
In [19]: gene_compactifier(resp[(resp.complex == 'TCA') & (resp.b < 0)].ext_gene)
```

```
Gene "Family", Number Found
sucg,  4 ['1', '1', '1', '1']
```

```
fum,  3 ['1', '1', '1']
sdhb,  5 ['1', '1', '1', '1', '1']
suca,  4 ['1', '1', '1', '1']
sdhd,  2 ['1', '1']
mdh,  8 ['1', '1', '1', '1', '2', '2', '2', '2']
ZK836.2,  6 ['', '', '', '', '', '']
dlst,  4 ['1', '1', '1', '1']
ogdh,  5 ['1', '1', '1', '1', '1']
idha,  4 ['1', '1', '1', '1']
idhb,  5 ['1', '1', '1', '1', '1']
sdha,  4 ['2', '2', '2', '2']
icl,  4 ['1', '1', '1', '1']
idhg,  8 ['1', '1', '1', '1', '2', '2', '2', '2']
idh,  12 ['1', '1', '1', '1', '1', '1', '1', '1', '2', '2', '2', '2']
sucl,  8 ['1', '1', '1', '1', '2', '2', '2', '2']
cts,  4 ['1', '1', '1', '1']
aco,  12 ['1', '1', '1', '1', '2', '2', '2', '2', '2', '2', '2', '2']
```

In [20]:
```
gene_compactifier(resp[(resp.complex == 'Complex I') & (resp.b < 0)].ext_gene)
```

```
Gene "Family", Number Found
djr,  4 ['1.1', '1.1', '1.1', '1.1']
F53F4.10,  4 ['', '', '', '']
C18E9.4,  4 ['', '', '', '']
F45H10.3,  4 ['', '', '', '']
nduo,  16 ['1', '1', '1', '1', '2', '2', '2', '2', '3', '3', '3', '4', '4', '4', '5', '5']
C16A3.5,  4 ['', '', '', '']
nuo,  30 ['1', '1', '1', '1', '2', '2', '2', '2', '3', '3', '3', '3', '3', '3', '3', '3', '3',
Y69A2AR.3,  4 ['', '', '', '']
T20H4.5,  4 ['', '', '', '']
nduf,  20 ['5', '5', '5', '5', '6', '6', '6', '6', '6', '6', '6', '6', '7', '7', '7', '7', '7'
Y53G8AL.2,  4 ['', '', '', '']
F44G4.2,  3 ['', '', '']
Y54F10AM.5,  4 ['', '', '', '']
lpd,  6 ['5', '5', '5', '5', '5', '5']
gas,  4 ['1', '1', '1', '1']
C25H3.9,  6 ['', '', '', '', '', '']
Y63D3A.7,  4 ['', '', '', '']
C33A12.1,  4 ['', '', '', '']
F59C6.5,  4 ['', '', '', '']
Y51H1A.3,  7 ['', '', '', '', '', '', '']
ndfl-4
```

In [21]:
```
gene_compactifier(resp[(resp.complex == 'energy reserve') & (resp.b < 0)].ext_gene)
```

```
Gene "Family", Number Found
Y67D8A.2,  17 ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
```

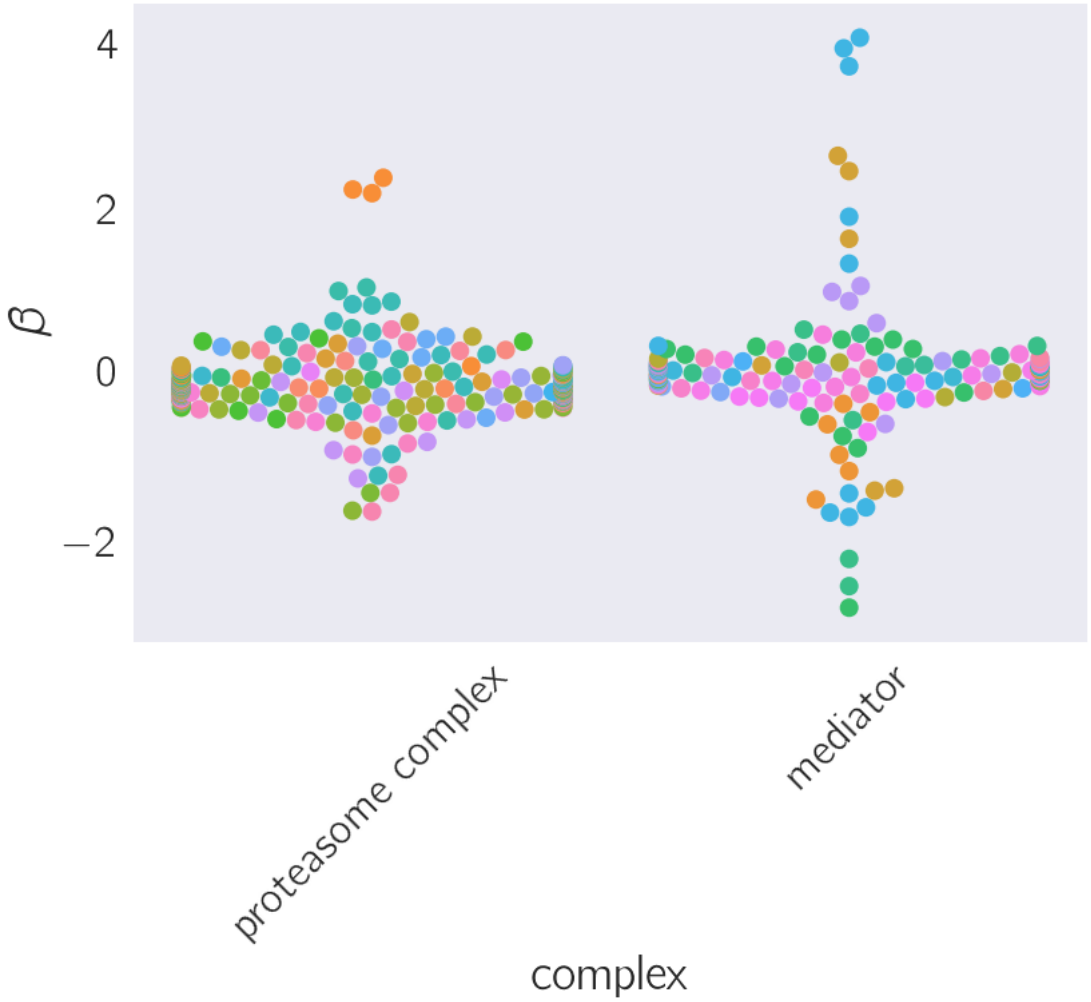
```
R05F9.6,  4 ['', '', '', '']
C14B9.8,  4 ['', '', '', '']
aagr,  19 ['1', '1', '1', '1', '1', '1', '1', '1', '2', '2', '2', '2', '3', '3', '3', '3', '3'
Y50D7A.3,  2 ['', '']
oga,  4 ['1', '1', '1', '1']
T04A8.7,  2 ['', '']
H18N23.2,  3 ['', '', '']
gsy,  3 ['1', '1', '1']
ogt-1
```

6 Effects of HIF-1 on the Proteasome and Mediator

In [22]:
```
prot = tidy_data[tidy_data.ens_gene.isin(central_dogma.ens_gene)].copy()
prot['complex'] = prot.ens_gene.map(lambda x: central_dogma[central_dogma.ens_gene ==
```

In [23]:
```
fig, ax = plt.subplots()
ax = sns.swarmplot(x='complex', y='b', hue='ens_gene', data=prot, size=7)
plt.xticks(rotation=45)
ax.legend_.remove()
# plt.title('HIF-1 mediated changes in ETC expression')
plt.ylabel(r'\beta')
# plt.xlabel('Electron Transport Chain Complexes')
```

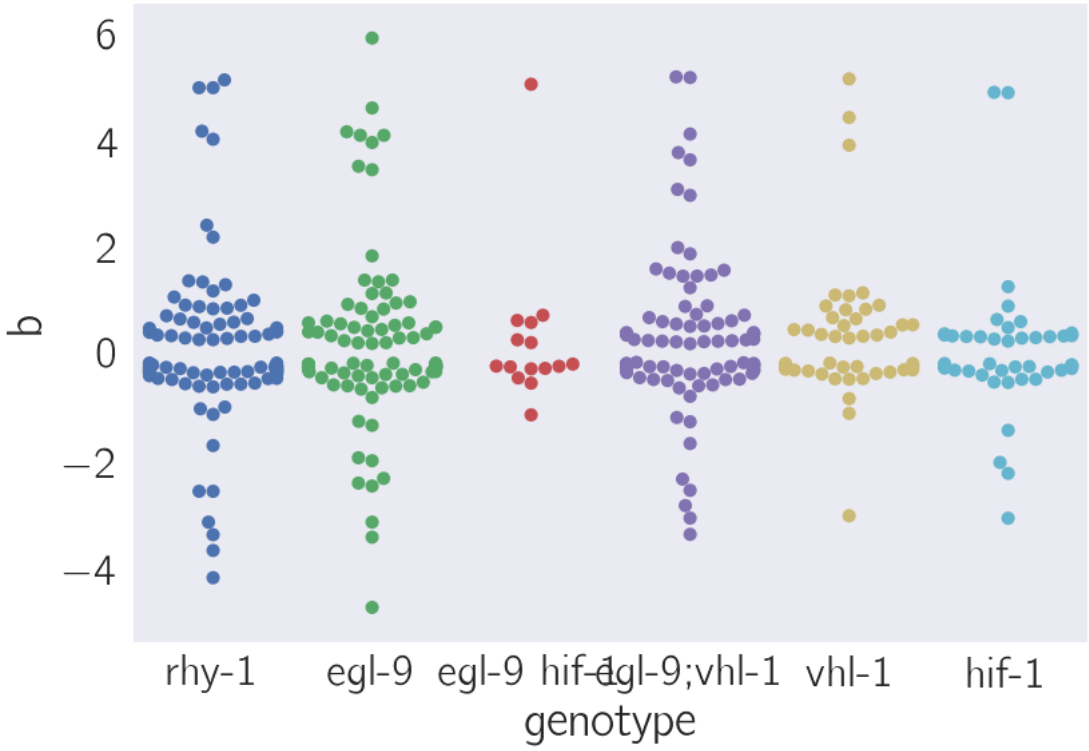
Out[23]:
```
<matplotlib.text.Text at 0x10bb912b0>
```



6.1 Effect of HIF-1 on proteins involved in 'protein catabolic process'

This GO term includes proteins that are involved in protein degradation, including the proteasome, a variety of ubiquitin-related enzymes and proteases

```
In [24]: ax, negregproteolysis = seqplotter.plot_by_term('protein catabolic process GO:0030163
                                                         df=tidy_data, kind='go')
```




```
In [25]: temp = tidy_data[(tidy_data.ens_gene.isin(negregproteolysis)) &
                          (tidy_data.target_id.isin(common)) &
                          (tidy_data.b > 0)
                         ].ext_gene.unique()
         gene_compactifier(temp)
```

```
Gene "Family", Number Found
asp,  3 ['14', '5', '8']
ctsa-2
Y119C1B.5
cpr,  3 ['1', '3', '6']
rpt-6
uev-3
K10C2.1
zyx-1
cpz-1
mans,  2 ['3', '4']
F57F5.1
aex-5
```

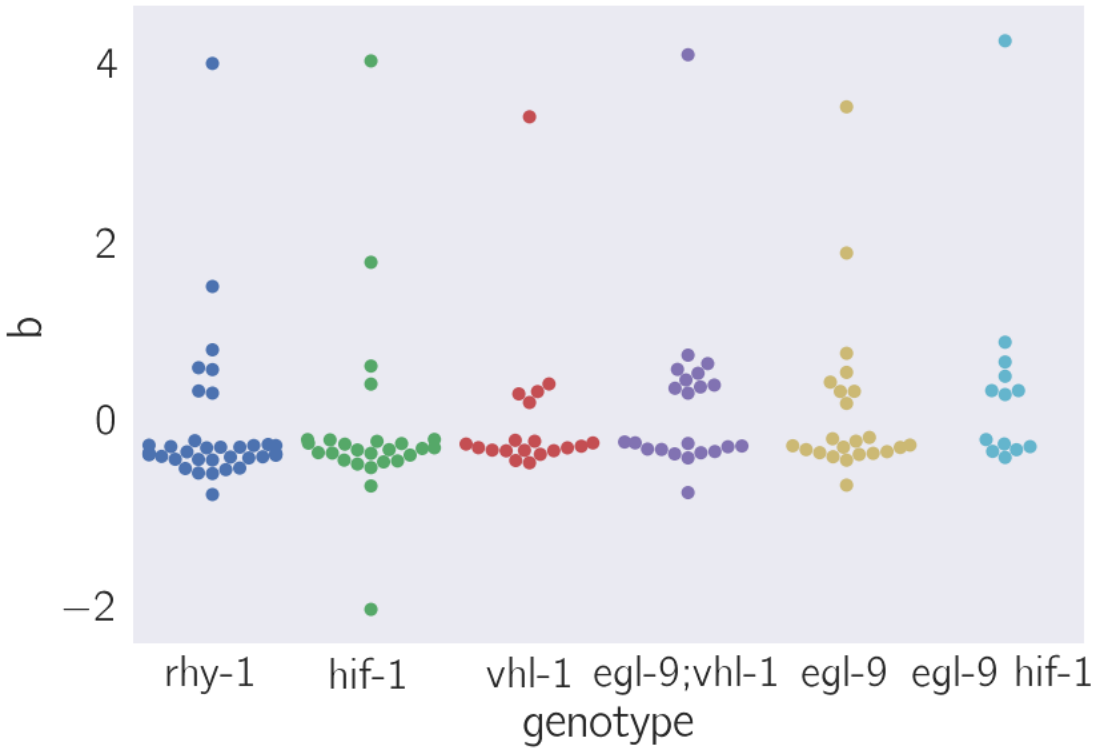
```
In [26]: temp = tidy_data[(tidy_data.ens_gene.isin(negregproteolysis)) &
                          (tidy_data.target_id.isin(common)) &
```

```
                          (tidy_data.b < 0)
                         ].ext_gene.unique()
        gene_compactifier(temp)
```

```
Gene "Family", Number Found
asp,  2 ['14', '8']
pas-1
unc-60
ubq-2
ctsa-2
Y119C1B.5
cpr,  2 ['1', '6']
rpt-6
uev-3
ubh-3
pbs-4
rpn-3
cpz-1
mans-3
ubc-20
F57F5.1
Y66D12A.9
cpl-1
```

7 Proteins annotated as involved in protein folding

```
In [27]: ax, folding = seqplotter.plot_by_term('protein folding', df=tidy_data, kind='go')
```




```
In [28]: temp = tidy_data[(tidy_data.ens_gene.isin(folding)) & (tidy_data.b > 0)].ext_gene.unic
         gene_compactifier(temp)
```

```
Gene "Family", Number Found
pdi-6
dnj,  5 ['10', '12', '15', '20', '27']
C06A6.5
fkb,  6 ['1', '3', '4', '5', '7', '8']
C03H12.1
C34C12.8
F47B7.2
dpy-11
emc,  2 ['1', '3']
C30H7.2
ZC250.5
hsp,  3 ['6', '60', '75']
M04D5.1
enpl-1
uggt,  2 ['1', '2']
crt-1
C14B9.2
ZK973.11
K07E8.6
```

```
W01B11.6
tbcc-1
pfd,  3 ['1', '3', '6']
unc-23
sig-7
catp-6
cyn,  10 ['10', '12', '13', '15', '2', '4', '5', '6', '8', '9']
cdc-37
Y17G9B.4
F53A3.7
cnx-1
ooc-5
cct,  4 ['4', '5', '6', '8']
Y71F9AL.11
daf-21
F42G8.7
Y22D7AL.10
T10H10.2
ero-1
trx,  2 ['2', '4']
R05D3.9
F35G2.1
Y49E10.4
```

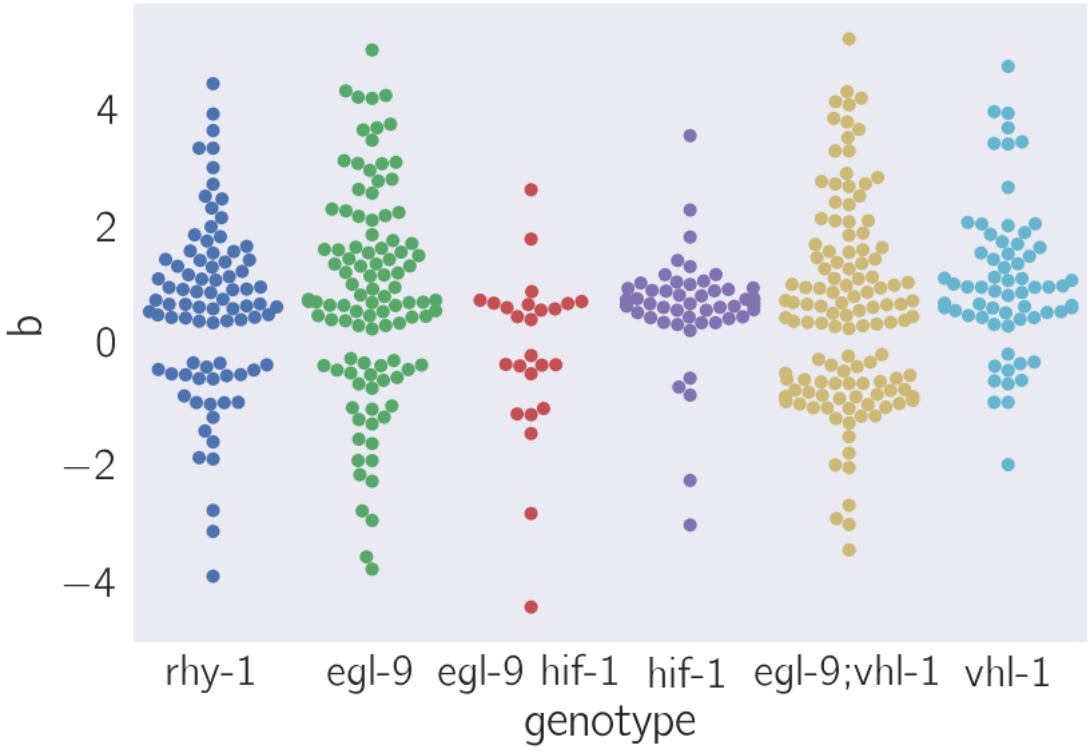
```
In [29]: temp = tidy_data[(tidy_data.ens_gene.isin(folding)) & (tidy_data.b < 0)].ext_gene.unic
         gene_compactifier(temp)
```

```
Gene "Family", Number Found
pdi-6
C06A6.5
C03H12.1
C34C12.8
dpy-11
emc,  3 ['1', '3', '6']
C30H7.2
enpl-1
C05G5.3
K07E8.6
fkb,  6 ['1', '2', '3', '5', '6', '7']
pfd,  6 ['1', '2', '3', '4', '5', '6']
unc-23
catp-6
tbcc-1
txl-1
tbcd-1
F53A3.7
tbca-1
```

```
cct,  7 ['1', '3', '4', '5', '6', '7', '8']
Y71F9AL.11
daf-21
bag-1
Y22D7AL.10
trx,  2 ['2', '4']
dnj,  7 ['10', '12', '13', '15', '19', '20', '27']
nud-1
F47B7.2
ero-1
ZC250.5
crt-1
C14B9.2
ZK973.11
W01B11.6
Y55F3AR.2
sig-7
cdc-37
uggt,  2 ['1', '2']
Y17G9B.4
ooc-5
cyn,  15 ['1', '10', '11', '12', '13', '15', '16', '2', '3', '4', '5', '6', '7', '8', '9']
F42G8.7
hsp,  3 ['6', '60', '75']
cnx-1
R05D3.9
F35G2.1
```

8 Immune Involvement

```
In [30]: ax, immune = seqplotter.plot_by_term('immune system process', df=tidy_data, kind='go'
```




```
In [31]: temp = tidy_data[(tidy_data.ens_gene.isin(immune)) & (tidy_data.target_id.isin(common)
                          (tidy_data.b > 0)].ext_gene.unique()
         gene_compactifier(temp)
```

```
Gene "Family", Number Found
asp-14
T24B8.5
aqp-10
C17H12.8
lec-11
C25D7.5
F35E12.9
clec,  4 ['210', '66', '70', '72']
lys,  2 ['2', '7']
fat-3
cyp-35A5
C49C3.9
tag-244
F55G11.8
nhr-57
dod,  2 ['22', '24']
cpr-3
Y41D4B.17
```

```
F01D5.1
F01D5.5
dct-17
his-10
C34H4.2
gst-7
F55G11.2
F53A9.6
K08D8.4
```

```
In [32]: temp = tidy_data[(tidy_data.ens_gene.isin(immune)) & (tidy_data.target_id.isin(common)
                          (tidy_data.b < 0)].ext_gene.unique()
         gene_compactifier(temp)
```

```
Gene "Family", Number Found
F55G11.8
acdh-1
asp-14
lys-7
aqp-10
clec,  2 ['210', '72']
F55G11.2
dod-24
nhr-57
cyp-35A5
```

9 Decorrelation Within Pathways

January 31, 2018

1 Table of Contents

1 Figure 7

In this notebook, I show that decorrelation could help order a pathway. The approach I will take is as follows:

- Calculate primary pairwise correlations between each mutant transcriptome
- Weight all correlations by the number of isoforms that are DE in both transcriptomes, divided by the total number of isoforms in either transcriptome.
- Plot

In [1]:
```
# important stuff:
import os
import pandas as pd
import numpy as np

import morgan as morgan
import genpy
import gvars

# Graphics
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rc
rc('text', usetex=True)
rc('text', usetex=True)
rc('text.latex', preamble=r'\usepackage{cmbright}')
rc('font', **{'family': 'sans-serif', 'sans-serif': ['Helvetica']})

# Magic function to make matplotlib inline;
%matplotlib inline

# This enables SVG graphics inline.
%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")

        mpl.rcParams['xtick.labelsize'] = 16
        mpl.rcParams['ytick.labelsize'] = 16
        mpl.rcParams['legend.fontsize'] = 14
```

In [2]:
```
genvar = gvars.genvars()
```

In [4]:
```
# Specify the genotypes to refer to:
single_mutants = ['b', 'c', 'd', 'e', 'g']

# Specify which genotypes are double mutants
double_mutants = {'a' : 'bd', 'f':'bc'}

# initialize the morgan.hunt object:
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
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/kallisto/'
for file in os.listdir("../sleuth/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)
        thomas.filter_data()
```

```
In [5]: barbara = morgan.mcclintock('bayesian', thomas, True)
```

```
starting comparison of d, 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, 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 d, 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 d, g
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.9 sec
starting comparison of c, e
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 2.1 sec
starting comparison of c, b
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.8 sec
starting comparison of c, g
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.7 sec
starting comparison of e, b
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 2.6 sec
starting comparison of e, g
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 2.1 sec
starting comparison of b, g
Applied log-transform to lam and added transformed lam_log_ to model.
 [-----------------100%-----------------] 2000 of 2000 complete in 1.9 secd d
d c
Applied log-transform to lam and added transformed lam_log_ to model.
d e
Applied log-transform to lam and added transformed lam_log_ to model.
d b
Applied log-transform to lam and added transformed lam_log_ to model.
d g
```

```
Applied log-transform to lam and added transformed lam_log_ to model.
c c
c e
Applied log-transform to lam and added transformed lam_log_ to model.
c b
Applied log-transform to lam and added transformed lam_log_ to model.
c g
Applied log-transform to lam and added transformed lam_log_ to model.
e e
e b
Applied log-transform to lam and added transformed lam_log_ to model.
e g
Applied log-transform to lam and added transformed lam_log_ to model.
b b
b g
Applied log-transform to lam and added transformed lam_log_ to model.
g g
```

Next, I define some functions that will help me clean up the matrix I just generated with the above command and place it into a tidy dataframe.

```
In [6]: 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
        df['pair'] =  df.variable + df.corr_with
        # return the damned thing:
        return df
```

```
In [7]: 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
```

```
In [8]: 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 dataframes:

```
In [9]: # tidy up the dataframe w/bayesian primary interactions:
        d_pos = tidy_df(barbara.robust_slope)
        d_pos['regression'] = 'primary'
        # tidy up the secondary interactions
        d_minus = tidy_df(barbara.secondary_slope, corr='outliers')
        d_minus['regression'] = 'secondary'

        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(genvar.sort_pairs)
        d.sort('sort_pairs', inplace=True)
```

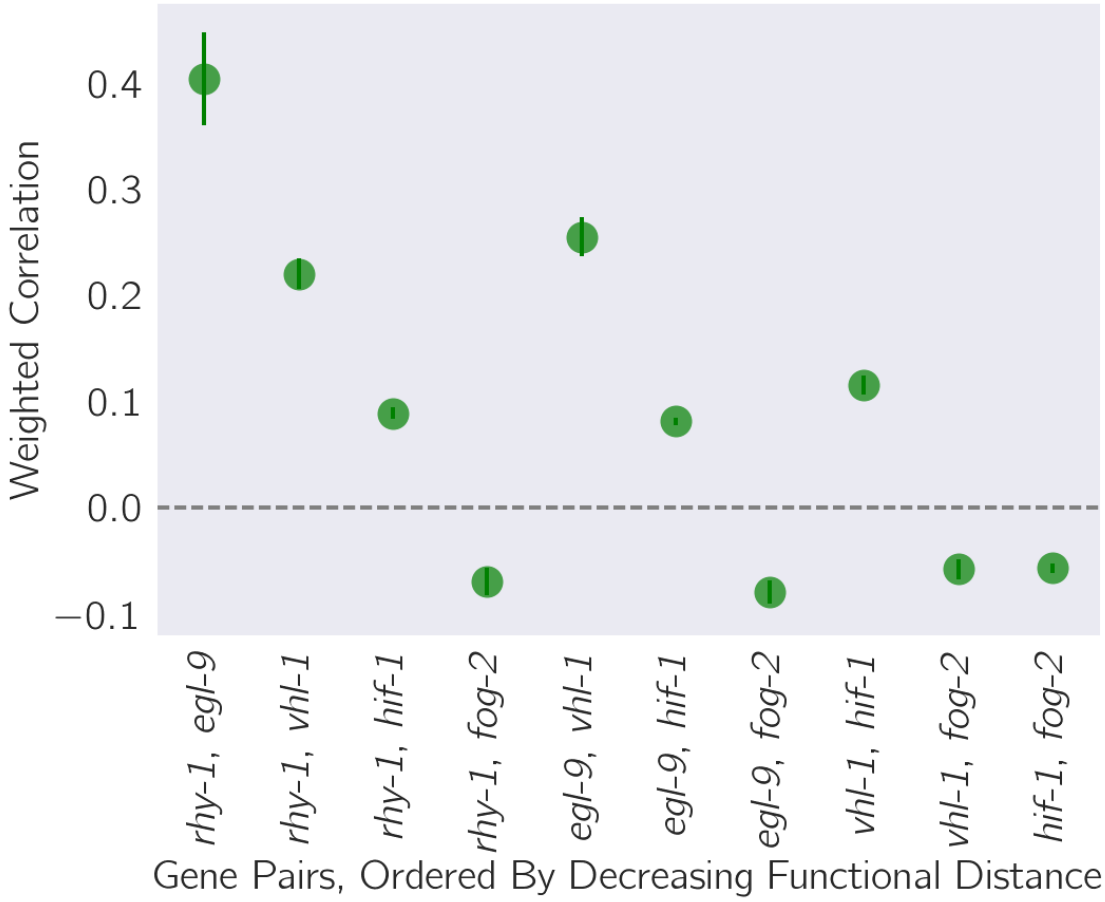
```
        # add the labels for plotting:
        d['genes'] = d.pair.map(genvar.decode_pairs)
```

```
In [10]: # extract the standard error for each correlation
         e_plus = tidy_df(barbara.errors_primary)

         # add a sort pairs column
         e_plus['sort_pairs'] = e_plus.pair.map(genvar.sort_pairs)
         # decode the gene pairs
         e_plus['genes'] = e_plus.pair.map(genvar.decode_pairs)
         # sort
         e_plus.sort('sort_pairs', inplace=True)
         # drop nonnumeric values
         e_plus.dropna(inplace=True)

         # repeat for secondary errors
         e_minus = tidy_df(barbara.errors_secondary)
         e_minus['sort_pairs'] = e_minus.pair.map(genvar.sort_pairs)
         e_minus['genes'] = e_minus.pair.map(genvar.decode_pairs)
         e_minus.sort('sort_pairs', inplace=True)
         e_minus.dropna(inplace=True)
```

2 Figure 7

```
In [11]: # 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 each xtick and xticklabel
         for x, xlabel in zip(plt.gca().get_xticks(),
                              plt.gca().get_xticklabels()):
             # get the data
             temp = d[d.regression=='primary']
             # get the gene ID
             f = temp.genes == xlabel.get_text()
             # get the error bar gene ID
             f2 = e_plus.genes == xlabel.get_text()
             # plot the errorbar
             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=90, 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)
plt.xlabel('Gene Pairs, Ordered By Decreasing Functional Distance', fontsize=20)
plt.ylabel('Weighted Correlation', fontsize=20)

# save
plt.savefig('../output/weighted_corr_decreases_w_distance.svg')
```

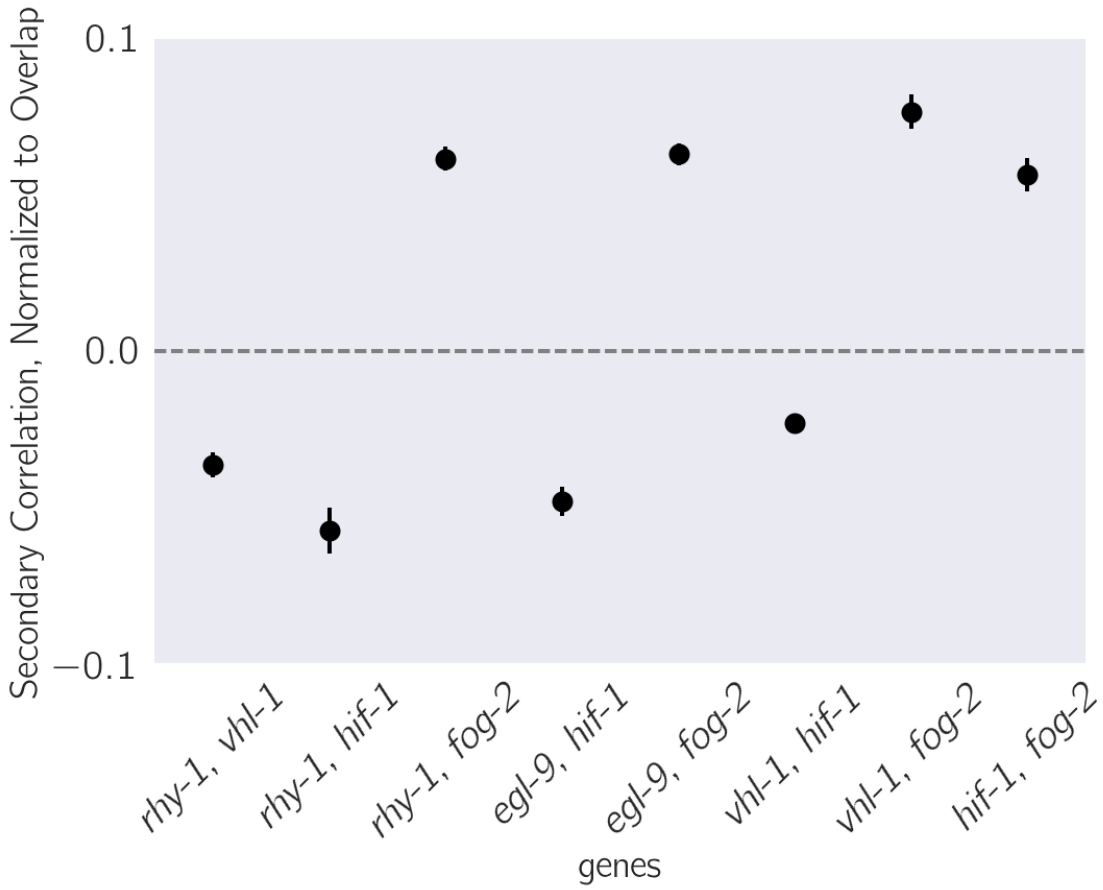


Secondary correlations do not seem to have this property. That may be a result of the low number of genes (we should have sequenced deeper) or a result of other things that may be occurring. I don't really know.

In [12]:
```
# plot secondary interactions
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')

    # prettify
    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[12]: <matplotlib.text.Text at 0x13737d978>



In [ ]:

Generate Supplementary File

January 31, 2018

1 Table of Contents

```
In [1]: import pandas as pd

In [2]: quants = pd.read_csv('../output/temp_files/DE_genes.csv')
        hypoxia = pd.read_csv('../output/temp_files/hypoxia_response.csv')
        hifoh = pd.read_csv('../output/temp_files/hifoh_candidates.csv')
        vhl = pd.read_csv('../output/temp_files/vhl_1_regulated_genes.csv')

In [3]: hypoxia = hypoxia.target_id.unique()
        hifoh = hifoh.target_id.unique()
        vhl = vhl.target_id.unique()

In [4]: pathway = {}
        for gid in hypoxia:
            pathway[gid] = 'hypoxia'
        for gid in vhl:
            pathway[gid] = 'vhl'

        hOH = {}
        for gid in hifoh:
            hOH[gid] = 'non-canonical'

In [5]: quants['pathway'] = quants.target_id.map(pathway)

In [6]: quants['non_canonical_epistasis'] = quants.target_id.map(hOH)

In [7]: quants.head()
```

Out[7]:

	ens_gene	ext_gene	target_id	b	se_b	qval	genotype	\
0	WBGene00007064	2RSSE.1	2RSSE.1a	0.150147	0.829418	1.000000	fog-2	
1	WBGene00007065	pot-3	3R5.1a	0.063856	1.909284	1.000000	fog-2	
2	WBGene00007065	pot-3	3R5.1b	0.274498	1.268484	1.000000	fog-2	
3	WBGene00004964	spe-10	AC3.10	0.197351	0.453000	0.998032	fog-2	
4	WBGene00007070	ugt-49	AC3.2	-0.340556	0.140666	0.100833	fog-2	

	sorter	code	pathway	non_canonical_epistasis
0	5	g	NaN	NaN

```
1       5    g     NaN                    NaN
2       5    g     NaN                    NaN
3       5    g     NaN                    NaN
4       5    g     NaN                    NaN
```

In [8]:
```
quants = quants[['ens_gene', 'ext_gene', 'target_id', 'b', 'se_b', 'qval', 'genotype',
quants = quants[quants.genotype != 'fog-2']
quants.to_csv('../output/supplementary_tables/supplementary_file_1.csv', index=False)
```