Enrich2 Documentation

Release 1.2.0

Alan F Rubin

Sep 13, 2017

Contents

1 Getting started 3

1.1 Required packages . . . 3
1.2 Installation and example dataset . . . 3
1.3 Enrich2 executables . . . 4

2 Defining experiments 5

2.1 Experimental designs . . . 5
2.2 Elements . . . 6
2.3 SeqLibs . . . 7

3 Using the GUI 9

3.1 Configuring your analysis . . . 9
3.2 Saving and loading . . . 12
3.3 Context menus . . . 12
3.4 Analysis options . . . 13

4 SeqLib configuration details 15

4.1 General parameters . . . 15
4.2 Sequence file parameters . . . 16
4.3 Barcode parameters . . . 16
4.4 Variant parameters . . . 17
4.5 Identifier parameters . . . 17
4.6 Overlap parameters . . . 18

5 Output HDF5 files 19

5.1 Table organization . . . 19
5.2 List of tables by object type . . . 19

6 Automatically generated plots 23

6.1 Experiment plots . . . 23
6.2 Selection plots . . . 25
6.3 SeqLib plots . . . 31

7 Example notebooks 35

7.1 Selecting variants by input library count . . . 35
7.2 Selecting variants by number of unique barcodes . . . 38

8 Appendix: API documentation **43**
8.1 `storemanager` — Abstract class for Enrich2 data . . . 43
8.2 `seqlib` — Sequencing library file handling and element counting . . . 43
8.3 `selection` — Functional score calculation using SeqLib count data . . . 45
8.4 `condition` — Dummy class for GUI . . . 45
8.5 `experiment` — Aggregation of replicate selections . . . 45
8.6 Enrich2 plotting . . . 45
8.7 Utility functions . . . 45
8.8 Enrich2 entry points . . . 47

Python Module Index **49**

Enrich2 is a general software tool for processing, analyzing, and visualizing data from deep mutational scanning experiments.

The software is freely available from https://github.com/FowlerLab/Enrich2/ under the GPLv3 license.

For an example dataset, visit https://github.com/FowlerLab/Enrich2-Example/.

To cite Enrich2, please reference A statistical framework for analyzing deep mutational scanning data.

Enrich2 was written by Alan F Rubin http://orcid.org/0000-0003-1474-605X

Error: Important notice for users of Enrich2 v1.0 or v1.1

Enrich2 v1.2.0 corrects an error in the software that, for most datasets, resulted in the standard errors for combined scores being over-estimated. The counts, scores, and replicate-wise standard errors are unaffected.

If you have analyzed datasets that contain replicates with a previous version of Enrich2, the easiest way to get the correct standard error values is to delete the experiment HDF5 file (the file name ends with `'_exp.h5'`) and re-run the program. This will recalculate combined scores and standard errors without redoing other parts of the analysis.

CHAPTER 1

Getting started

Required packages

Enrich2 runs on Python 2.7 and has the following dependencies:

- NumPy version 1.10.4 or higher
- SciPy version 0.16.0 or higher
- pandas version 0.18 or 0.19
- PyTables version 3.2.0 or higher
- Statsmodels version 0.6.1 or higher
- matplotlib version 1.4.3 or higher

The configuration GUI requires Tkinter. Building a local copy of the documentation requires Sphinx.

Note: We recommend using a scientific Python distribution such as Anaconda or Enthought Canopy to install and manage dependencies.

Note: PyTables may not be installed when using the default settings for your distribution. If you encounter errors, check that the `tables` module is present.

Installation and example dataset

1. Make sure the *required packages* are installed.
    (a) To set up a new Anaconda environment for Enrich2, use the following environment file (`click to download`). Detailed instructions for setting up conda environments can be found in the conda documentation.

```
name: enrich2
dependencies:
  - python=2.7
  - numpy
  - scipy
  - pandas=0.19
  - pytables
  - statsmodels
  - matplotlib
```

2. Download Enrich2 from the GitHub repository and unzip it.
3. Using the terminal, navigate to the Enrich2 directory and run the setup script by typing `python setup.py install`

To download the example dataset, visit the Enrich2-Example GitHub repository. Running this preconfigured analysis will create several *Automatically generated plots*. The *Example notebooks* demonstrate how to explore the *Output HDF5 files*.

Enrich2 executables

The Enrich2 installer places two executable scripts into the user's path. Both executables run the same analysis, but through different interfaces.

- `enrich_gui` launches the Enrich2 graphical user interface. This is the recommended way to create a configuration file for Enrich2. See *Using the GUI* for a step-by-step guide.
- `enrich_cmd` launches the program from the command line. This is recommended for users performing analyses on a remote server who have already created configuration files. For a detailed list of command line options, type `enrich_cmd --help`

CHAPTER 2

Defining experiments

Experimental designs

Enrich2 represents deep mutational scanning experimental designs as a tree of objects. The hierarchy of object types is defined below:

- Experiment

  The root object for most experimental designs. Parent of at least one experimental condition.

- Condition

  A single experimental condition. Parent of at least one replicate selection performed under the condition.

- Selection

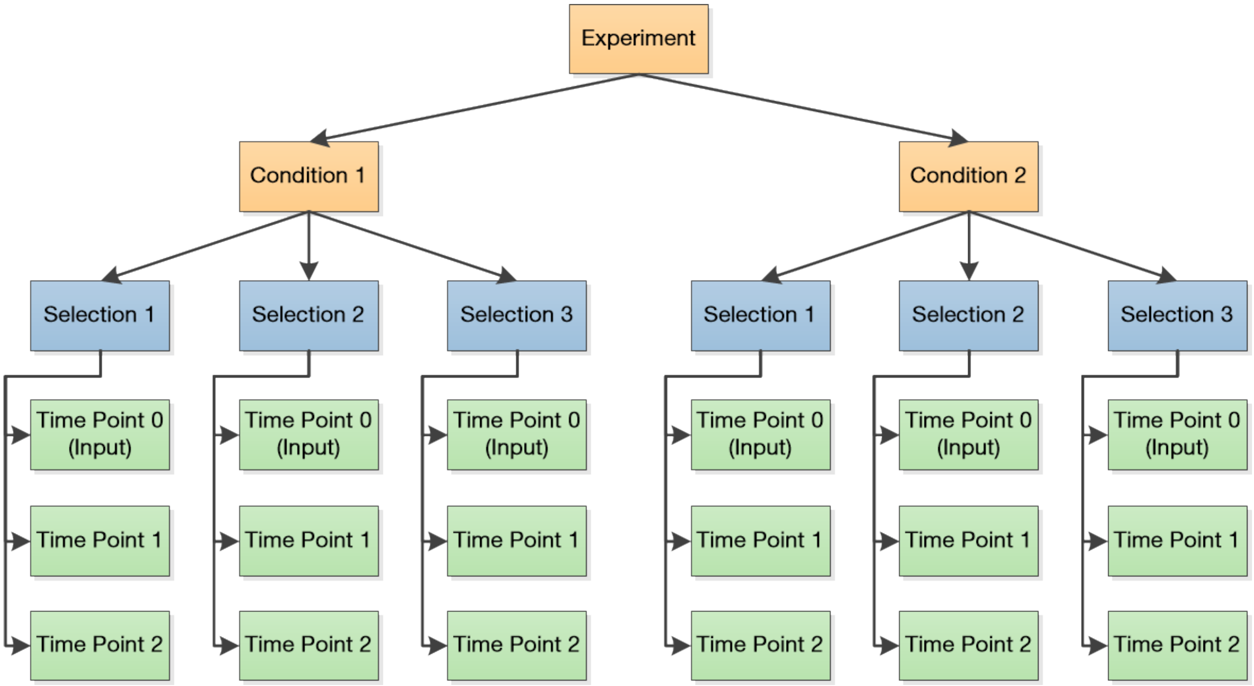
  A single deep mutational scanning replicate. Parent of at least two sequencing libraries, one or more for each time point/round/bin of the selection.

- Sequencing library (SeqLib)

  FASTQ output or count data from a deep mutational scanning time point/round/bin. Has no children.

Each experimental design has a single root object, which can be an Experiment, Selection, or SeqLib. With the exception of Conditions, each experimental design object has its own HDF5 file containing its data.

Note: Conditions do not have their own HDF5 file. If there is only one condition, use an Experiment as the root.



The above diagram illustrates an experimental design with two conditions, each with three replicates sampled at three time points (including the input).

Elements

Enrich2 counts elements to quantify their enrichment or depletion in a complex population. The four element types are defined below:

- Barcode

  A short DNA barcode sequence often used for tagging variants. Stored as the barcode DNA sequence. Barcodes are counted directly from sequencing data.

- Variant

  A DNA-level variant of the wild type sequence, which can be coding or non-coding. Stored as an HGVS string describing the nucleotide and any amino acid differences from the wild type sequence. Variants can be counted either directly from sequencing data or as the sum of counts for linked barcodes as defined by a barcode-variant map.

- Synonymous

  A protein-level variant of the wild type sequence. Stored as an HGVS string describing the amino acid differences from the wild type sequence. Synonymous elements are counted as the sum of counts for variant elements with the same amino acid sequence. Variant elements with the wild type amino acid sequence but a non-wild type DNA sequence are assigned to a special variant.

- Identifier

  An arbitrary label (such as a target gene name) for barcode assignment. Stored as the label string. Identifiers are counted as the sum of counts for associated barcodes as defined by a barcode-identifier map or specified as counts.

SeqLibs

Enrich2 implements five types of SeqLib, each supporting different element types and/or methods of sequencing deep mutational scanning populations.

Note: Synonymous elements are only present if the wild type sequence is protein coding.

- Barcoded Variant

  Contains barcode, variant, and synonymous elements. Each DNA variant in the experiment is linked to one or more DNA barcode sequences. A barcode-variant map describes which barcodes map to each variant. The FASTQ file contains only barcode sequences.

- Barcoded Identifier

  Contains barcode and identifier elements. Each identifier in the experiment is associated with one or more DNA barcode sequences. A barcode-identifier map describes which barcodes map to each identifier. The FASTQ file contains only barcode sequences.

- Overlap

  Contains variant and synonymous elements. DNA variants are sequenced directly using overlapping paired-end reads. Requires FASTQ files for both forward and reverse reads.

- Basic

  Contains variant and synonymous elements. DNA variants are sequenced directly using single-end reads.

- Barcodes Only

  Contains barcode elements. The FASTQ file contains only barcode sequences.

- Identifiers Only

  Contains identifier elements. No FASTQ file is processed, so the counts must be provided by the user.

For more information, see *SeqLib configuration details*.

CHAPTER 3

Using the GUI

The graphical user interface makes it easy to specify an experimental design that Enrich2 can understand. For more information about how these are organized, see *Experimental designs*.

Configuring your analysis

The Enrich2 installer places the graphical user interface (GUI) entry point in your path. Type `enrich_gui` from the command line to launch the program.

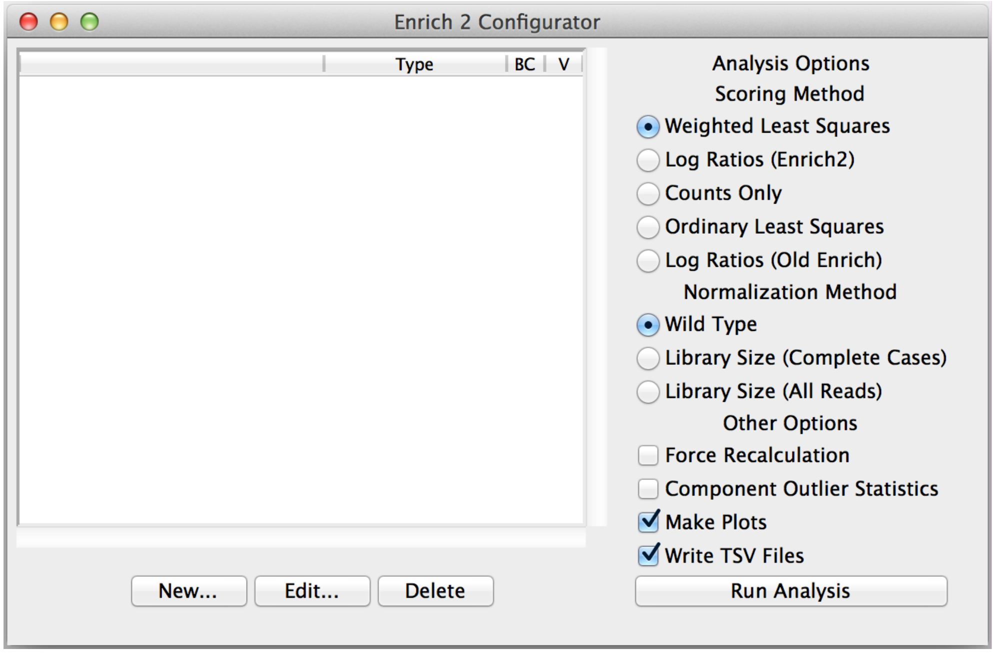
Error: Mac OS X users running the Enrich2 GUI in a virtualenv may encounter the following error:

```
2016-10-10 12:34:56.789 python[12345:12345678] -[NSApplication _setup:]: 
↪unrecognized selector sent to instance 0x12345abcd
```

This is caused by an interaction between Tkinter and the matplotlib backend. To fix the issue, edit (or create) the "~/.matplotlib/matplotlibrc" file and add the line:

```
backend: TkAgg
```

Note: Once you have created your configuration file, you can also run the program in command line mode. Type `enrich_cmd --help` for usage and a list of command line options.



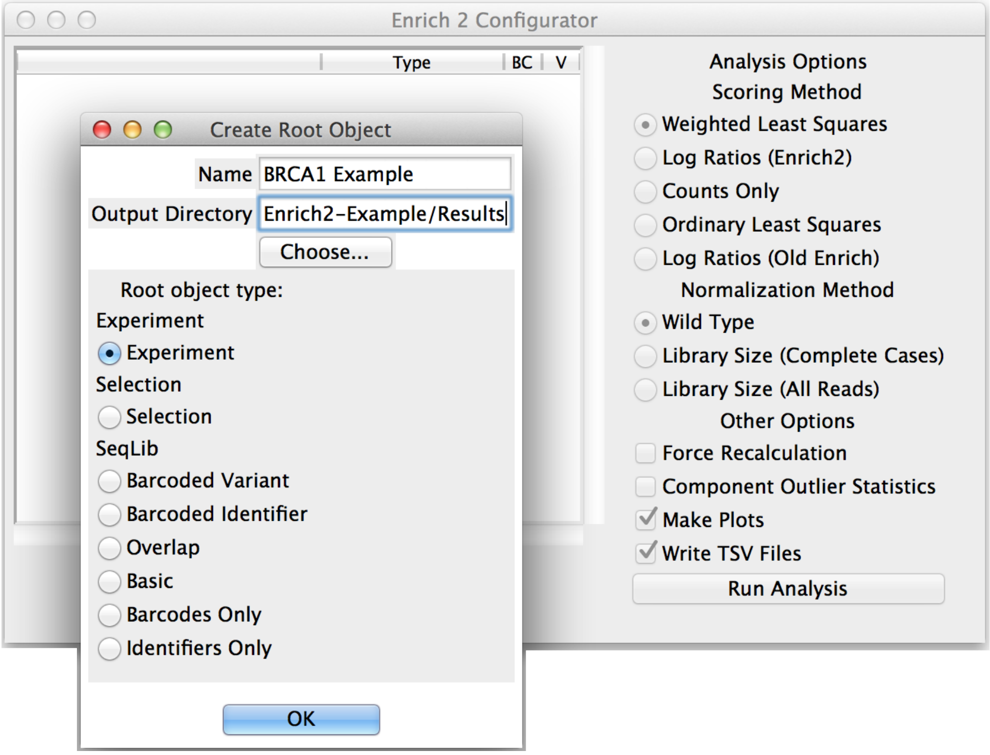
Analysis Options

Scoring Method
● Weighted Least Squares
○ Log Ratios (Enrich2)
○ Counts Only
○ Ordinary Least Squares
○ Log Ratios (Old Enrich)

Normalization Method
● Wild Type
○ Library Size (Complete Cases)
○ Library Size (All Reads)

Other Options
☐ Force Recalculation
☐ Component Outlier Statistics
☑ Make Plots
☑ Write TSV Files

Click "New..." to create the root object.



Name: BRCA1 Example
Output Directory: Enrich2-Example/Results

Root object type:
Experiment
● Experiment
Selection
○ Selection
SeqLib
○ Barcoded Variant
○ Barcoded Identifier
○ Overlap
○ Basic
○ Barcodes Only
○ Identifiers Only

Analysis Options

Scoring Method
● Weighted Least Squares
○ Log Ratios (Enrich2)
○ Counts Only
○ Ordinary Least Squares
○ Log Ratios (Old Enrich)

Normalization Method
● Wild Type
○ Library Size (Complete Cases)
○ Library Size (All Reads)

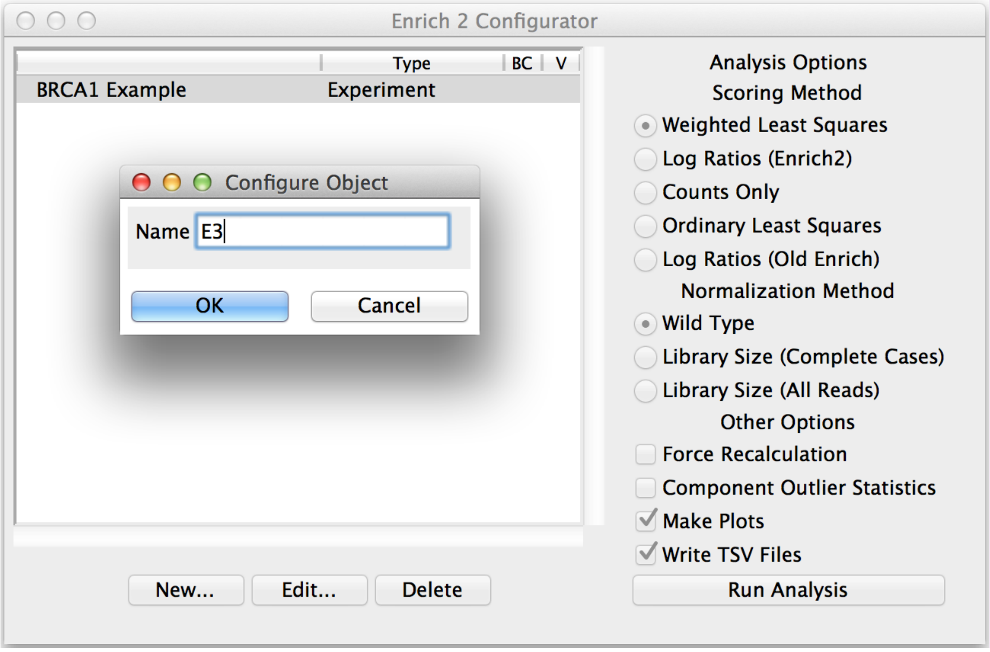
Other Options
☐ Force Recalculation
☐ Component Outlier Statistics
☑ Make Plots
☑ Write TSV Files

Enter a short but descriptive object name that will not conflict with other objects in the analysis.

Choose the output directory for the HDF5, plot, and tab-separated files generated by the analysis.

Select the appropriate object type: Experiment, Selection if there are no replicates, or SeqLib if you only want to count a single sequencing library.

If you created a Selection or Experiment root object, select it and click "New..." to add a child object.



	Type
BRCA1 Example	Experiment

Configure Object

Name E3

Analysis Options

Scoring Method

● Weighted Least Squares
○ Log Ratios (Enrich2)
○ Counts Only
○ Ordinary Least Squares
○ Log Ratios (Old Enrich)

Normalization Method

● Wild Type
○ Library Size (Complete Cases)
○ Library Size (All Reads)

Other Options

☐ Force Recalculation
☐ Component Outlier Statistics
☑ Make Plots
☑ Write TSV Files

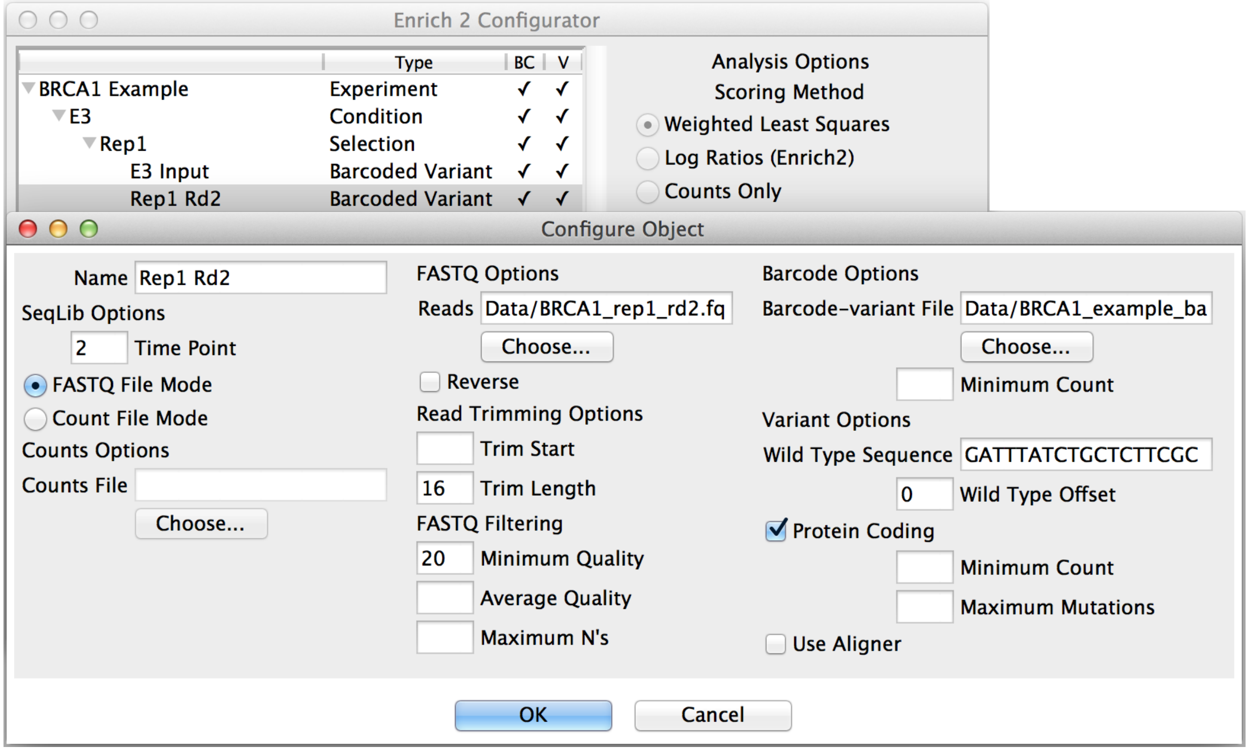
Conditions and Selections do not have any parameters beyond their names.

Continue adding child objects until the entire experimental design is represented. When creating a new SeqLib, choose the appropriate type depending on how the experiment was performed (see *SeqLibs*).

Note: To avoid re-counting the reads when multiple Selections share the same input library, use the same object name for the input library in each Selection.

Most parameters are specified in SeqLib objects, such as the wild type sequence, filtering options, and the location of the sequencing files or counts files (see *SeqLib configuration details*).

Note: Time points can have multiple sequencing libraries, which are added together before scores are calculated.



	Type	BC	V
BRCA1 Example	Experiment	✓	✓
E3	Condition	✓	✓
Rep1	Selection	✓	✓
E3 Input	Barcoded Variant	✓	✓
Rep1 Rd2	Barcoded Variant	✓	✓

Analysis Options
Scoring Method
● Weighted Least Squares
○ Log Ratios (Enrich2)
○ Counts Only

Name: Rep1 Rd2
SeqLib Options
Time Point: 2
● FASTQ File Mode
○ Count File Mode
Counts Options
Counts File:

FASTQ Options
Reads: Data/BRCA1_rep1_rd2.fq
☐ Reverse
Read Trimming Options
Trim Start:
Trim Length: 16
FASTQ Filtering
Minimum Quality: 20
Average Quality:
Maximum N's:

Barcode Options
Barcode-variant File: Data/BRCA1_example_ba
Minimum Count:
Variant Options
Wild Type Sequence: GATTTATCTGCTCTTCGC
Wild Type Offset: 0
☑ Protein Coding
Minimum Count:
Maximum Mutations:
☐ Use Aligner

Clicking "New..." with a SeqLib object selected will add a sibling SeqLib to the Selection that shares the same FASTQ filtering and other options.

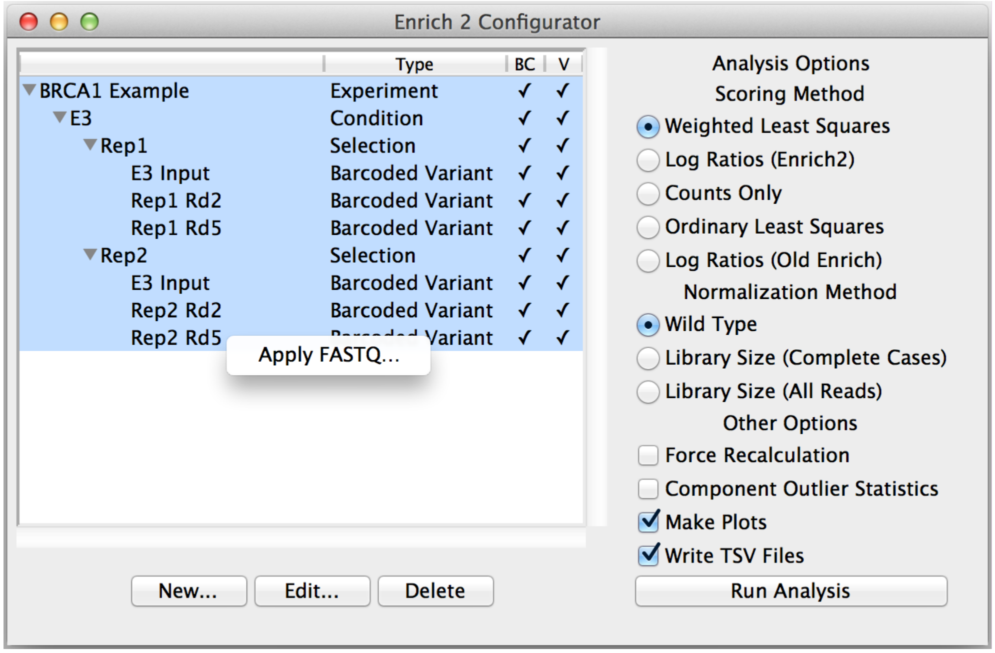
Saving and loading

After you have configured the analysis, you can save a configuration file by selecting "Save" or "Save As..." from the File menu. You can also use the File menu to load an existing configuration file by selecting "Open."

Note: If you encounter an error when loading a configuration file, try using a validator such as JSONLint to identify any issues.

Context menus

Right-clicking on an object will open a context menu with additional actions not covered by the New/Edit/Delete buttons.



	Type	BC	V
BRCA1 Example	Experiment	✓	✓
E3	Condition	✓	✓
Rep1	Selection	✓	✓
E3 Input	Barcoded Variant	✓	✓
Rep1 Rd2	Barcoded Variant	✓	✓
Rep1 Rd5	Barcoded Variant	✓	✓
Rep2	Selection	✓	✓
E3 Input	Barcoded Variant	✓	✓
Rep2 Rd2	Barcoded Variant	✓	✓
Rep2 Rd5	Barcoded Variant	✓	✓

Analysis Options

Scoring Method

● Weighted Least Squares
○ Log Ratios (Enrich2)
○ Counts Only
○ Ordinary Least Squares
○ Log Ratios (Old Enrich)

Normalization Method

● Wild Type
○ Library Size (Complete Cases)
○ Library Size (All Reads)

Other Options

☐ Force Recalculation
☐ Component Outlier Statistics
☑ Make Plots
☑ Write TSV Files

- Apply FASTQ...

  Copy the FASTQ filtering options from the chosen SeqLib to every highlighted SeqLib of the same type.

Analysis options

These choices are not saved in the configuration file and should be reviewed before running each analysis. For further information about the scoring and normalization methods below, see the Enrich2 manuscript.

Scoring method

- Weighted Least Squares

  Recommended for selections with at least three time points (including the input).

- Log Ratios (Enrich2)

  Recommended for selections with two time points (input and selected). For selections with more than two time points, the last time point is used as the selected time point. Intermediate time points not used.

- Counts Only

  No element scores are calculated. The output contains only element counts.

- Ordinary Least Squares

  Provided for comparison and legacy support.

- Log Ratios (Old Enrich)

  Provided for comparison and legacy support. This method is a re-implementation of the previously published Enrich software. Standard errors are not calculated. For selections with more than two time points, the last time point is used as the selected time point. Intermediate time points not used.

Normalization method

- Wild Type

  Recommended if your selection has a wild type sequence. Normalizes counts by the wild type count as described in the Enrich2 manuscript. For designs with identifiers instead of variants, the special wild type identifier "_wt" can be used.

- Library Size (Complete Cases)

  Normalizes counts by the library size. Only elements present in all time points within a selection contribute to the library size.

- Library Size (All Reads)

  Normalizes counts by the library size. All elements contribute to the library size.

Other options

- Force Recalculation

  Discards all data that are not raw counts before performing the analysis. See *Table organization* for more about raw counts.

- Component Outlier Statistics

  Tests whether the score of each barcode differs significantly from that of its assigned variant or identifier. Performs an analogous calculation for variant and synonymous scores.

Warning: Testing for outliers is experimental and very computationally inefficient.

- Make Plots

  Creates plots for this analysis.

- Write TSV Files

  Outputs tab-separated files for this analysis.

Once you've finished selecting your options, click Run Analysis!

The output directory will contain *Output HDF5 files*, *Automatically generated plots*, and tab-separated files.

CHAPTER 4

SeqLib configuration details

Most parameters are specified within SeqLib objects. Experiment, Condition, and Selection objects have only a name (and output directory if at the root). *Analysis options*, such as scoring method, are chosen at run time.

Sequencing libraries have *General parameters*, *Sequence file parameters*, and other parameter groups depending on the type:

SeqLib type	Barcode	Variant	Identifier	Overlap
Barcoded Variant	X	X		
Barcoded Identifier	X		X	
Overlap		X		X
Basic		X		
Barcodes Only	X			
Identifiers Only			X	

See *SeqLibs* for descriptions of each type.

General parameters

- Name

  The object name should be short, descriptive, and not conflict with other object names in the analysis.

- Output Directory

  Path to the output directory. This field only appears for the root object.

- Time Point

  The time point must be an integer. All Selections require an input library as time point 0. Time point values may refer to the round of selection or hour of sampling.

- Counts File

  Required for Counts File Mode. Path to an HDF5 file or tab-separated value file that contains counts for this time point. Raw counts from that file will be used for this SeqLib. If an HDF5 file is provided, all tables in the “raw/” group are copied. Sequence file parameters will be ignored. The file must have the suffix ”.h5” for HDF5 or one of ”.txt” ”.tsv” or ”.csv” for tab-separated value files.

Note: Tab-separated value files must have exactly two columns separated by a tab. The first line of the file must have the column heading “counts” preceded by a single tab character. The first column contains the barcode, identifier, or HGVS variant string depending on the type of raw counts required by the SeqLib type. The second column contains the count for that element.

Sequence file parameters

Enrich2 accepts sequence files in FASTQ format. These files may be processed while compressed with gzip or bzip2. The file must have the suffix ”.fq” or ”.fastq” before compression.

- Reads

  Required for FASTQ File Mode. Path to a FASTQ file containing the sequencing reads. For overlap SeqLibs, there are fields for Forward Reads and Reverse Reads.

- Reverse

  Checking this box will reverse-complement reads before analysis. Not present for Overlap SeqLibs.

Read filtering parameters

Filters are applied after read trimming and any read merging.

- Minimum Quality

  Minimum single-base quality. If a single base in the read has a quality score below this value, the read will be discarded.

- Average Quality

  Average read quality. If the mean quality score of all bases in the read is below this value, the read will be discarded.

- Maximum N’s

  Maximum number of N nucleotides. If the read contains more than this number of bases called as N, the read will be discarded. This should be set to 0 in most cases.

- Remove Unresolvable Overlaps

  Present for Overlap SeqLibs only. Checking this box discards merged reads with unresolvable discrepant bases (see *Overlap parameters*).

- Maximum Mutations

  Present for SeqLibs with variants only. Maximum number of mutations. If the variant contains more than this number of differences from wild type, the variant is discarded (or aligned if that option is enabled under *Variant parameters*).

Barcode parameters

- Barcode-variant File

Not present for barcode-only SeqLibs. Path to a tab-separated file in which each line contains a barcode followed by its identifier or linked variant DNA sequence. This file may be processed while compressed with gzip or bzip2.

- Minimum Count

  Minimum barcode count. If the barcode has fewer counts than this value, it will not be scored and will not contribute to counts of its variant or identifier.

- Trim Start

  Position of the first base to keep when trimming barcodes. All subsequent bases are kept if Trim Length is not specified. Reverse-complementing occurs before trimming. Bases are numbered starting at 1.

- Trim Length

  Number of bases to keep when trimming barcodes. Starts at the first base if Trim Start is not specified. Reverse-complementing occurs before trimming.

Variant parameters

- Wild Type Sequence

  The wild type DNA sequence. This sequence will be compared to reads or the barcode-variant map when calling variants. All sequences must have the same length and starting position.

- Wild Type Offset

  Integer added to every variant nucleotide position. Used to place variants in the context of a larger sequence.

- Protein Coding

  Checking this box will interpret the wild type sequence as protein coding. The wild type sequence must be in frame.

- Use Aligner

  Checking this box will enable Needleman-Wunsch alignment. Insertion and deletion events will be called.

Warning: Using the aligner will dramatically increase run time, and is not recommended for most users.

- Minimum Count

  Minimum variant count. If the variant has fewer counts than this value, it will not be scored and will not contribute to counts of any synonymous elements.

Identifier parameters

- Minimum Count

  Minimum identifier count. If the identifier has fewer counts than this value, it will not be scored.

Overlap parameters

Overlapping read pairs reduce the likelihood of calling sequencing errors as variants. Paired-end Illumina reads are generated such that they overlap in the target region.

When Enrich2 combines forward and reverse reads into merged reads, base quality values in the overlapping region are defined as the higher quality value at each position. Mismatches are resolved by assuming the base with the higher quality value is correct. If mismatched bases have the same quality value, the position is considered unresolvable and replaced by an ‘X’ base.

- Forward Start

  Position of the first overlapping base in the forward read. Bases are numbered starting at 1.

- Reverse Start

  Position of the first overlapping base in the reverse read before reverse complementing. Bases are numbered starting at 1.

- Overlap Length

  Number of bases in the overlapping region.

- Maximum Mismatches

  Maximum number of mismatches in the overlapping region. If a merged read has more than this number of mismatches, the read pair will be discarded.

- Overlap Only

  Checking this box will trim the merged reads to the overlapping region.

CHAPTER 5

Output HDF5 files

Enrich2 stores data in an HDF5 file for each Experiment, Selection, and SeqLib analysis object. The name of the HDF5 file is the object's name plus the suffix "_<obj>.h5", where <obj> is the object type ("exp", "sel", or "lib"). Each file has multiple tables that can be queried and retrieved as pandas data frames (see *Example notebooks*).

Each Experiment, Selection, and SeqLib has its own directory inside "Results/tsv/" containing tab-separated value files for users who want to work with other tools, such as R or Excel.

Table organization

HDF5 files organize tables into groups like directories in a file system. Enrich2 has two top-level groups, "/main" (used for most tables) and "/raw" (used exclusively in SeqLibs to store raw counts). The first subgroup is typically the element type (variant, barcode, etc.), followed by the kind of data (counts, scores, etc.).

Note: When the "Force recalculation" analysis option is chosen, the "/main" tables are deleted from all HDF5 files in this analysis, and regenerated based on the "/raw" count data.

Enrich2 uses NaN (Not a Number) values to represent missing data, such as zero counts or scores that could not be calculated.

List of tables by object type

Experiment

Most experiment tables use a pandas MultiIndex for column names. The MultiIndex levels are: condition, selection (if applicable), and data value. See the pandas advanced indexing documentation for more information on how to work with these objects.

- "/main/<element>/counts"

Counts of elements that appear in at least one time point in the experiment.

- “/main/<element>/scores”

  Condition-level scores, standard errors, and epsilon (change in the standard error after the last iteration of the random-effects model) for all elements scored in all selections of at least one condition.

- “/main/<element>/scores_shared”

  Selection-level scores and standard errors for each element with at least one condition-level score.

- “/main/<element>/scores_shared_full”

  Selection-level scores and standard errors for each element scored in at least one selection.

- “/main/<element>/scores_pvalues_wt”

  z-scores and p-values for each variant or synonymous element with a condition-level score. The null hypothesis is that the element’s score is equal to wild type.

- “/main/barcodemap”

  Barcode-variant or barcode-identifier map for all barcodes that appear in the Experiment. Only present for Barcoded Variant or Barcoded Identifier SeqLibs.

Selection

- “/main/<element>/counts”

  Counts of elements that appear in all time points in the selection.

- “/main/<element>/counts_unfiltered”

  Counts of elements that appear in at least one time point in the selection.

- “/main/<element>/scores”

  Scores, standard errors, standard error percentiles, and method-specific values (e.g. regression slope and intercept) for all elements counted in all time points in the selection.

- “/main/<element>/weights”

  Regression weights for each element at each time point in weighted least squares regression.

- “/main/<element>/log_ratios”

  Y-values for each element at each time point in weighted and ordinary least squares regression.

- “/main/barcodemap”

  Barcode-variant or barcode-identifier map for all barcodes that appear in the Selection. Only present for Barcoded Variant or Barcoded Identifier SeqLibs.

SeqLib

- “/main/<element>/counts”

  Counts of elements after minimum count filtering and barcode mapping.

- “/raw/<element>/counts”

  Counts of elements taken directly from the FASTQ data.

- “/raw/filter”

Number of reads removed for each FASTQ filtering option.

- “/raw/barcodemap”

  Barcode-variant or barcode-identifier map for barcodes that appear in this SeqLib. Only present for Barcoded Variant or Barcoded Identifier SeqLibs.

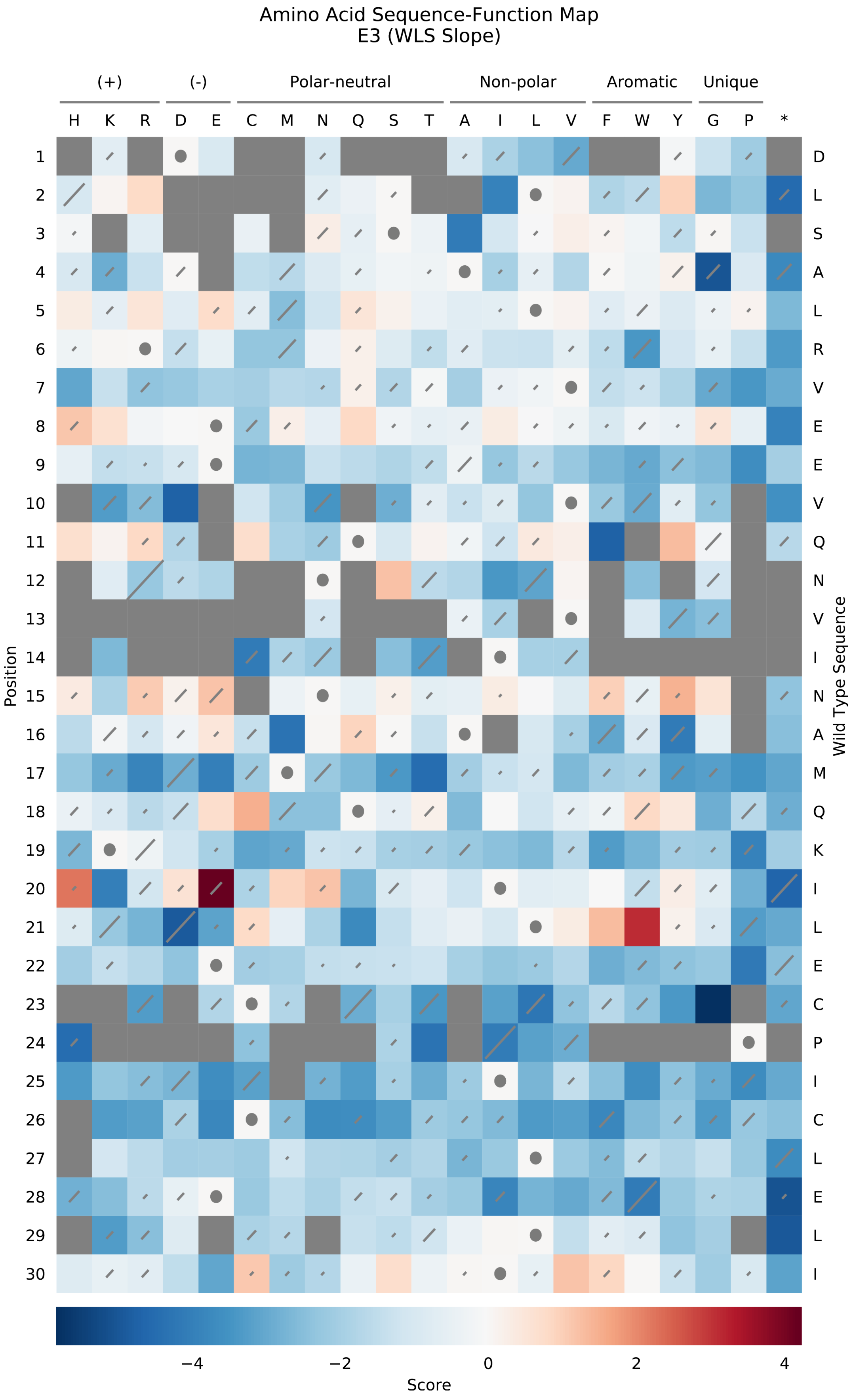
CHAPTER 6

Automatically generated plots

In addition to providing structured output to allow users to create their own plots, Enrich2 produces default visualizations for each analysis. Experiment, Selection, and SeqLib objects each have their own directory inside "Results/plots/". Plots are saved in PDF format, and many of the files contain multiple pages.

Experiment plots

- Sequence-function map


Amino Acid Sequence-Function Map
E3 (WLS Slope)
(+)
(-)
Polar-neutral
Non-polar
Aromatic
Unique
H K R D E C M N Q S T A I L V F W Y G P *
Position
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Wild Type Sequence
D L S A L R V E E V Q N V I N A M Q K I L E C P I C L E L I
−4
−2
0
2
4
Score

Visualization of scores and standard errors for single changes from wild type. Separate protein- and nucleotide-level sequence-function maps are generated.

Cell color indicates the score for the single change (row) at the given position (column). Positive scores (in red) indicate better performance in the assay, and negative scores (in blue) indicate worse performance. Grey squares denote changes that were not measured. Diagonal lines in each cell represent the standard error for the score, and are scaled such that the highest standard error on the plot covers the entire diagonal. Standard errors that are less than 2% of this maximum value are not plotted. Cells containing circles have the wild type residue at that position. Custom amino acid ordering and groups can be specified by running Enrich2 in command line mode and using the `--sfmap-aa-file` option. Each line of the file begins with an optional label followed by a single tab character and then a comma-separated list of single-letter amino acid codes. All amino acid codes must be present exactly once.

The following amino acid grouping files are provided:

Default (`click to download`)

This grouping is used when no file is specified. Reference

```
(+)        H,K,R
(-)        D,E
Polar-neutral        C,M,N,Q,S,T
Non-polar        A,I,L,V
Aromatic        F,W,Y
Unique        G,P
        *
```

Helical Propensity (`click to download`)

Reference

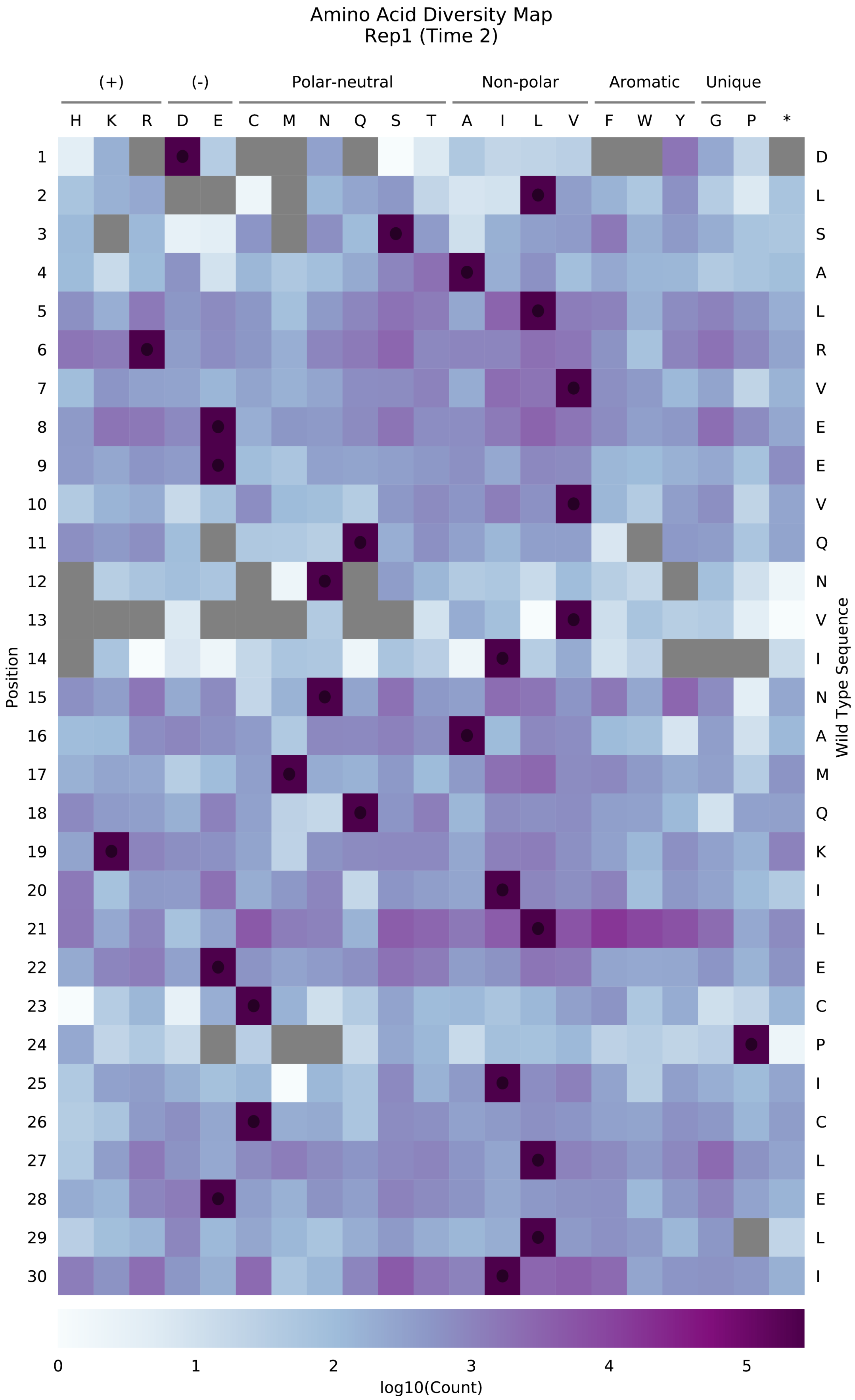
```
High helical propensity        A,L,R,M,K,Q,E,I,W
Low helical propensity        S,Y,F,V,H,N,T,C,E
Disruptive        G,P
        *
```

Selection plots

- Sequence-function map

  As above.

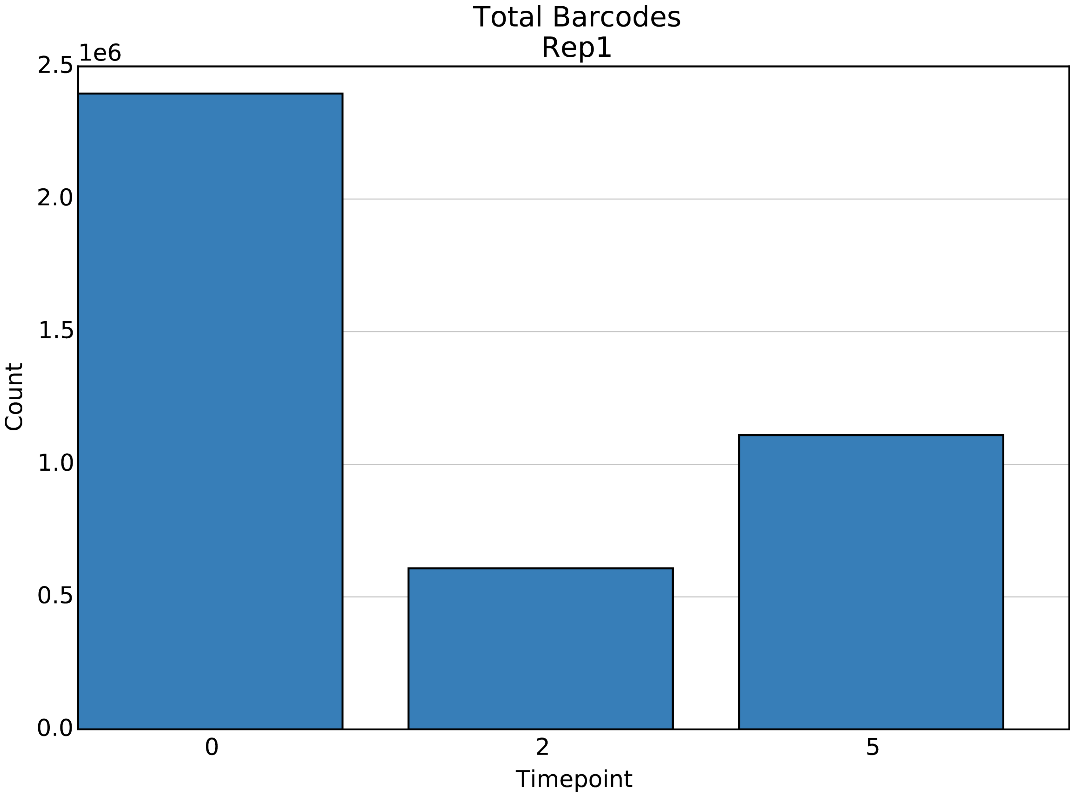
- Diversity map


Amino Acid Diversity Map
Rep1 (Time 2)
(+)
(-)
Polar-neutral
Non-polar
Aromatic
Unique
H
K
R
D
E
C
M
N
Q
S
T
A
I
L
V
F
W
Y
G
P
*
Position
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
Wild Type Sequence
D
L
S
A
L
R
V
E
E
V
Q
N
V
I
N
A
M
Q
K
I
L
E
C
P
I
C
L
E
L
I
0
1
2
3
4
5
log10(Count)

Variant frequencies are visualized in the style of a sequence-function map. Separate protein- and nucleotide-level diversity maps for each time point are generated.

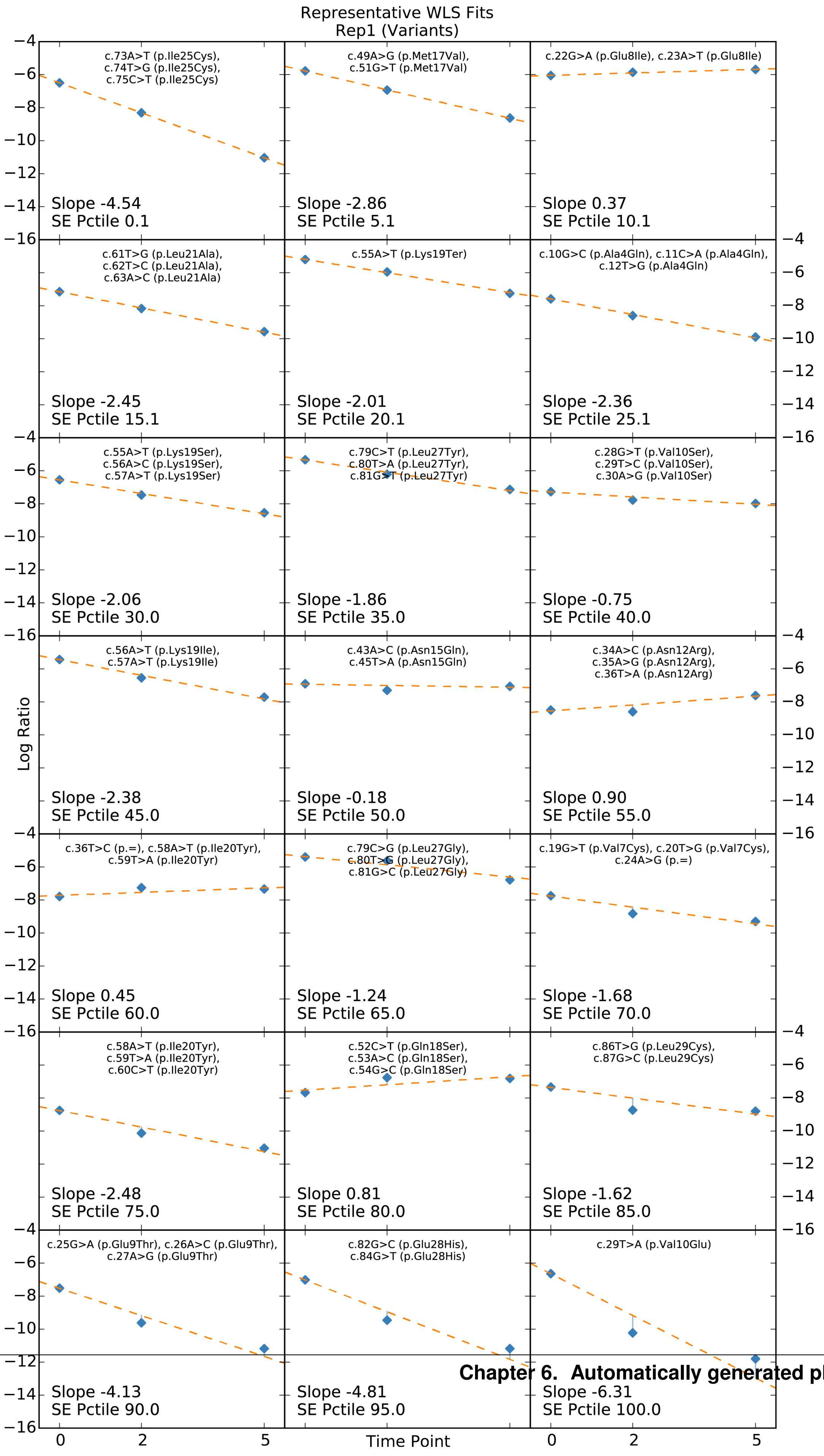
Custom amino acid ordering and groups can be specified by running Enrich2 in command line mode and using the `--sfmap-aa-file` option. *See above* for more details.

- Counts per time point



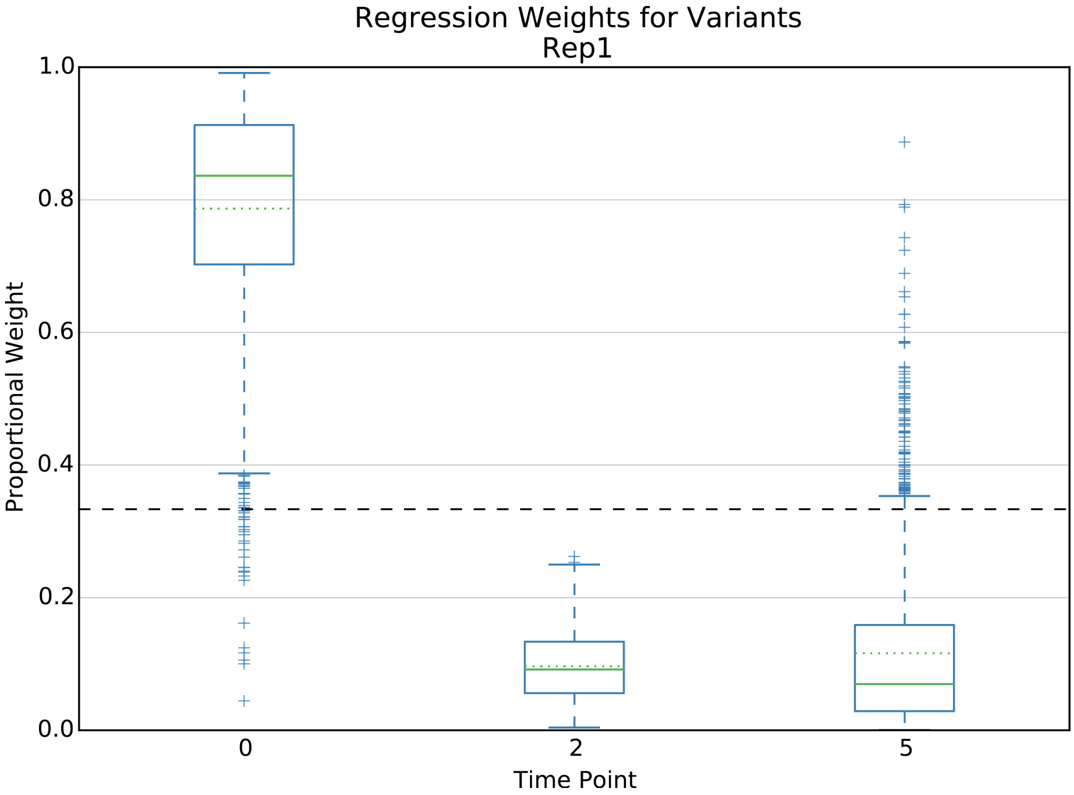
Bar plots showing the total element count in each time point. One plot for each element type.

- Representative regression fits


Representative WLS Fits
Rep1 (Variants)
Log Ratio
Time Point
0
2
5
−4
−6
−8
−10
−12
−14
−16
c.73A>T (p.Ile25Cys), c.74T>G (p.Ile25Cys), c.75C>T (p.Ile25Cys)
Slope -4.54
SE Pctile 0.1
c.49A>G (p.Met17Val), c.51G>T (p.Met17Val)
Slope -2.86
SE Pctile 5.1
c.22G>A (p.Glu8Ile), c.23A>T (p.Glu8Ile)
Slope 0.37
SE Pctile 10.1
c.61T>G (p.Leu21Ala), c.62T>C (p.Leu21Ala), c.63A>C (p.Leu21Ala)
Slope -2.45
SE Pctile 15.1
c.55A>T (p.Lys19Ter)
Slope -2.01
SE Pctile 20.1
c.10G>C (p.Ala4Gln), c.11C>A (p.Ala4Gln), c.12T>G (p.Ala4Gln)
Slope -2.36
SE Pctile 25.1
c.55A>T (p.Lys19Ser), c.56A>C (p.Lys19Ser), c.57A>T (p.Lys19Ser)
Slope -2.06
SE Pctile 30.0
c.79C>T (p.Leu27Tyr), c.80T>A (p.Leu27Tyr), c.81G>T (p.Leu27Tyr)
Slope -1.86
SE Pctile 35.0
c.28G>T (p.Val10Ser), c.29T>C (p.Val10Ser), c.30A>G (p.Val10Ser)
Slope -0.75
SE Pctile 40.0
c.56A>T (p.Lys19Ile), c.57A>T (p.Lys19Ile)
Slope -2.38
SE Pctile 45.0
c.43A>C (p.Asn15Gln), c.45T>A (p.Asn15Gln)
Slope -0.18
SE Pctile 50.0
c.34A>C (p.Asn12Arg), c.35A>G (p.Asn12Arg), c.36T>A (p.Asn12Arg)
Slope 0.90
SE Pctile 55.0
c.36T>C (p.=), c.58A>T (p.Ile20Tyr), c.59T>A (p.Ile20Tyr)
Slope 0.45
SE Pctile 60.0
c.79C>G (p.Leu27Gly), c.80T>G (p.Leu27Gly), c.81G>C (p.Leu27Gly)
Slope -1.24
SE Pctile 65.0
c.19G>T (p.Val7Cys), c.20T>G (p.Val7Cys), c.24A>G (p.=)
Slope -1.68
SE Pctile 70.0
c.58A>T (p.Ile20Tyr), c.59T>A (p.Ile20Tyr), c.60C>T (p.Ile20Tyr)
Slope -2.48
SE Pctile 75.0
c.52C>T (p.Gln18Ser), c.53A>C (p.Gln18Ser), c.54G>C (p.Gln18Ser)
Slope 0.81
SE Pctile 80.0
c.86T>G (p.Leu29Cys), c.87G>C (p.Leu29Cys)
Slope -1.62
SE Pctile 85.0
c.25G>A (p.Glu9Thr), c.26A>C (p.Glu9Thr), c.27A>G (p.Glu9Thr)
Slope -4.13
SE Pctile 90.0
c.82G>C (p.Glu28His), c.84G>T (p.Glu28His)
Slope -4.81
SE Pctile 95.0
c.29T>A (p.Val10Glu)
Slope -6.31
SE Pctile 100.0

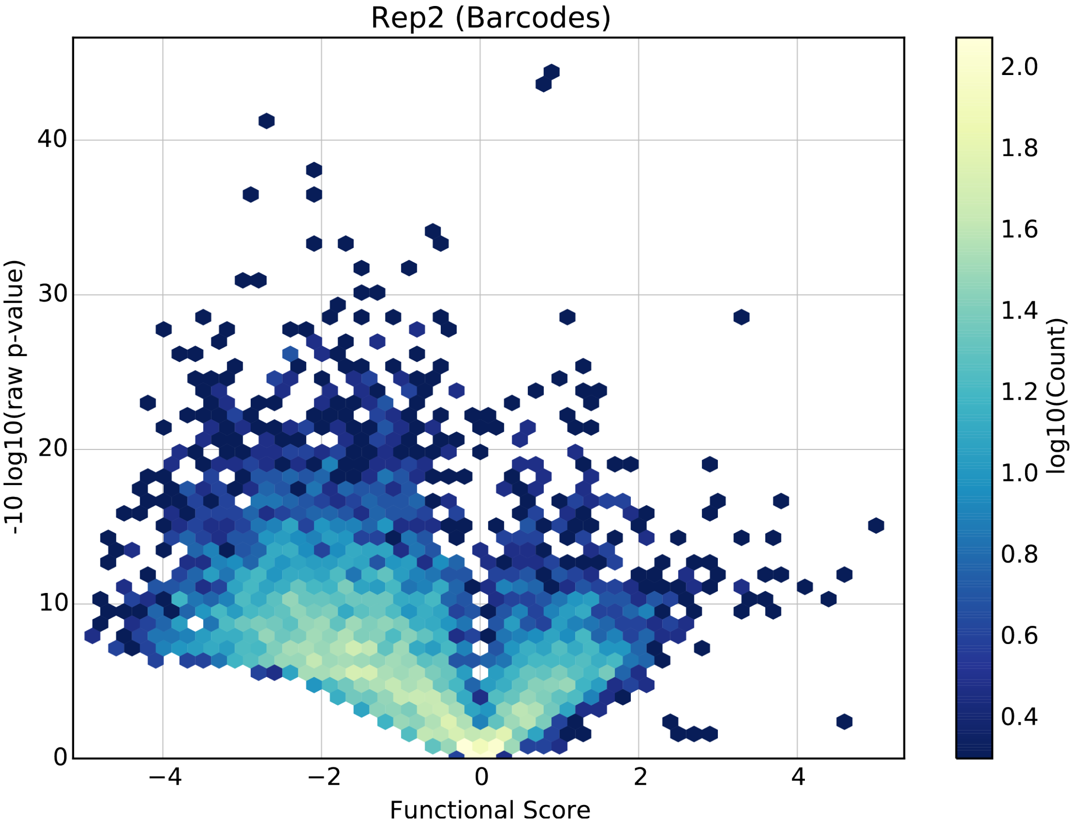
Present for linear regression scoring methods only. Linear fits for the element closest to each 5th percentile (0, 5, 10, ..., 95, 100). Used for diagnostic purposes and setting standard error filtering cutoffs. One plot for each element type.

- Regression weights



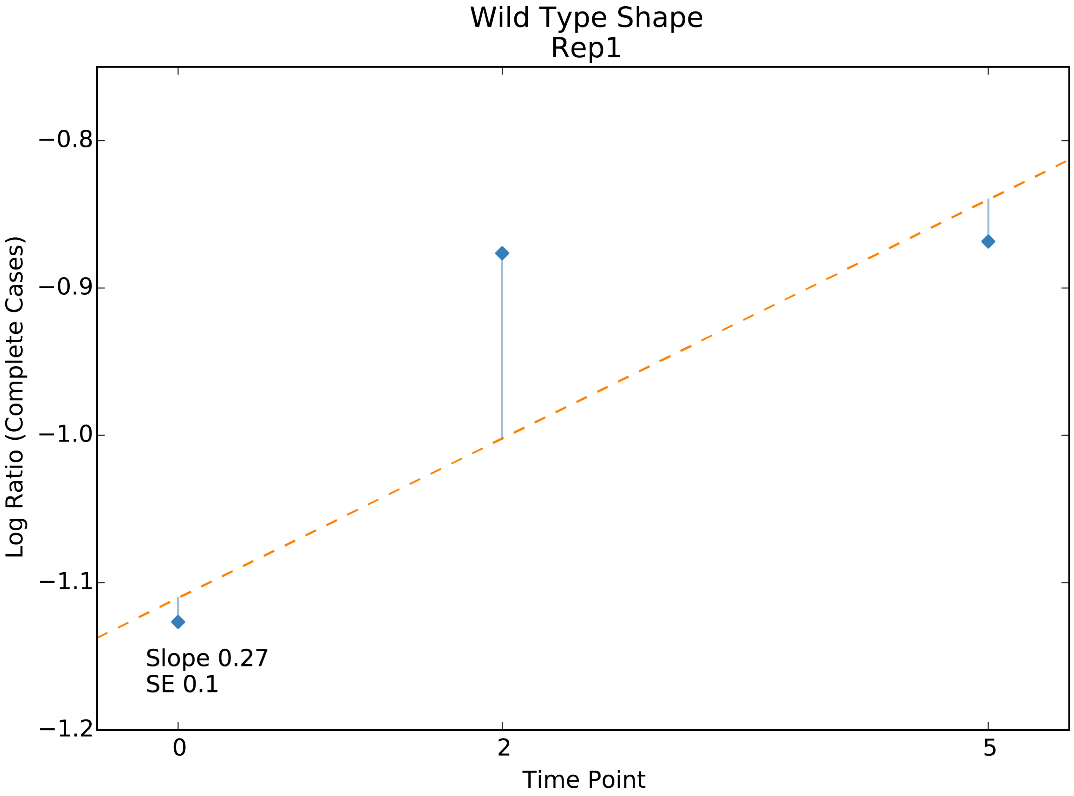
Present for weighted linear regression scoring method only. Boxplot of regression weights for each time point. Dashed line indicates uniform weight. One plot for each element type.

- Volcano plot



Present for linear regression scoring methods with variants only. Volcano plot of the raw p-value from a z-test under the null hypothesis that the element behaves the same as wild type vs. the element's score. One plot for each element type.

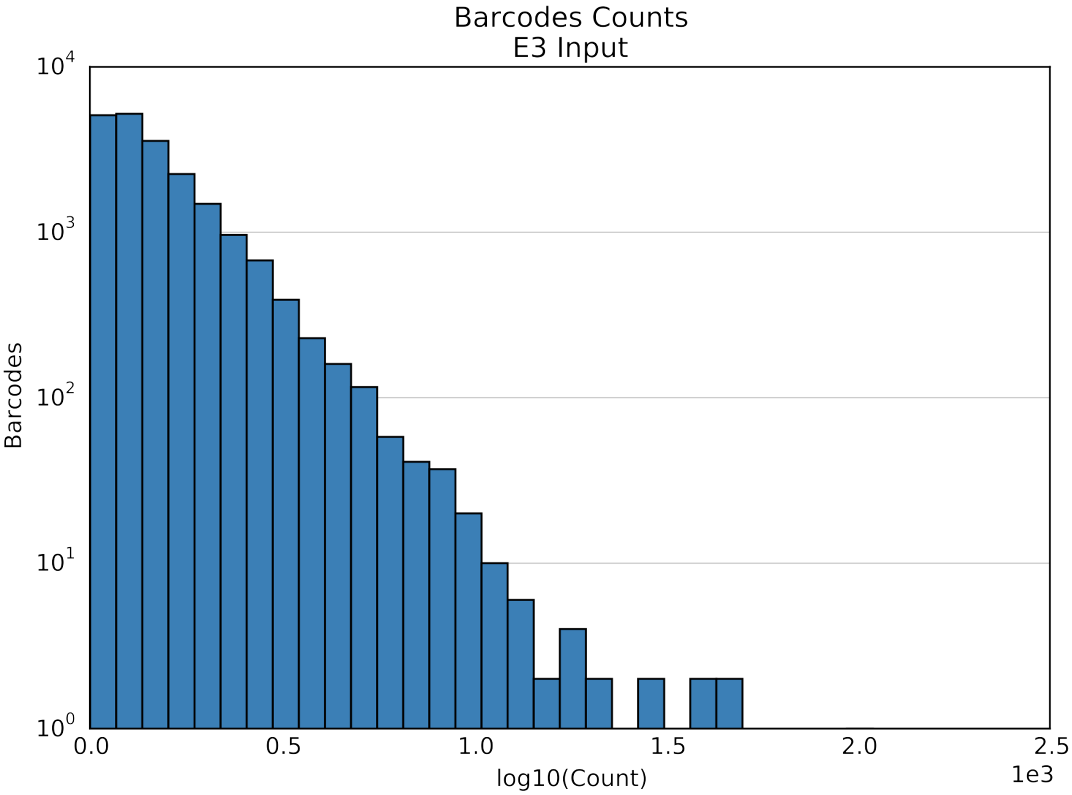
- Wild type shape



Present for linear regression scoring methods with variants only. Plot of the non-normalized linear fit of the wild type. Used to assess the effect of wild type correction.

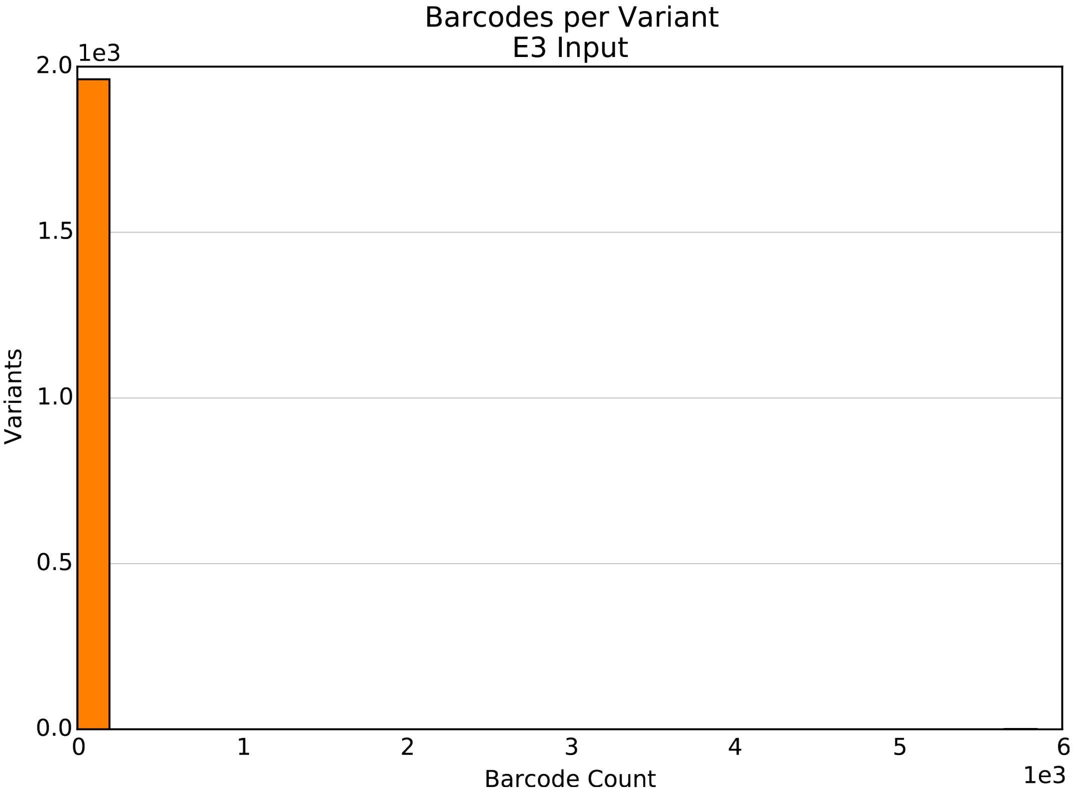
SeqLib plots

- Counts per element



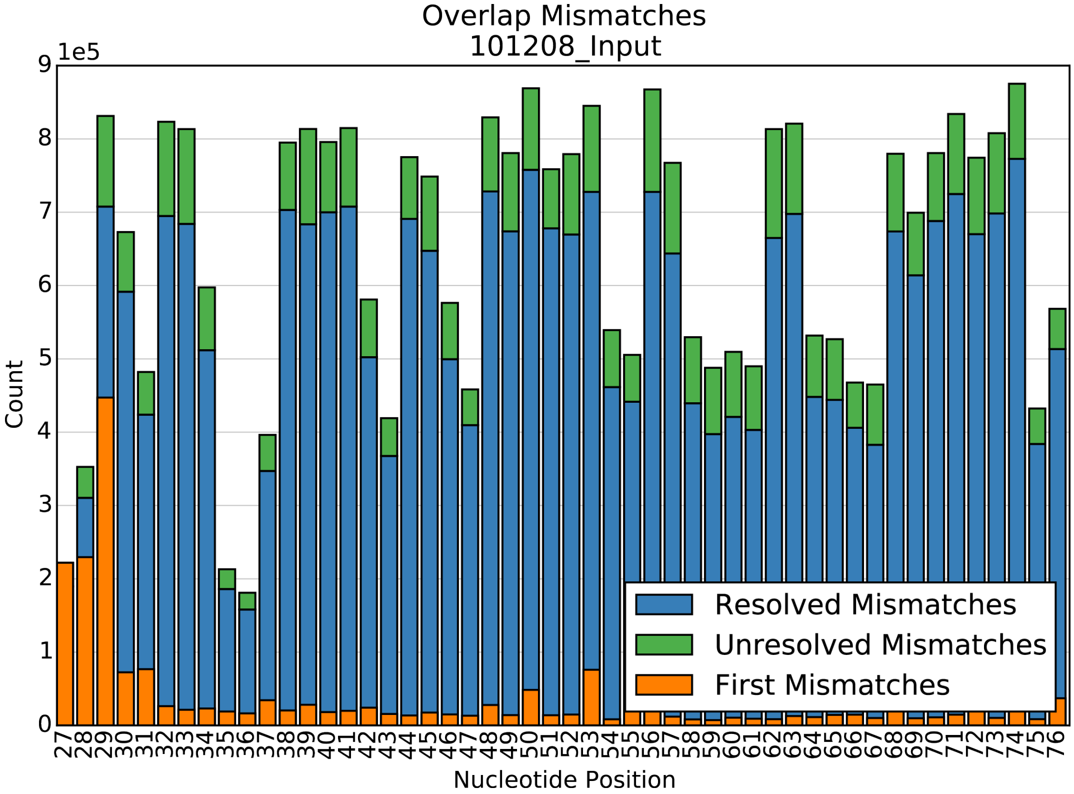
Histogram of element counts. Two plots for each element type, one with log-transformed x-axis and one without.

- Unique barcodes per element



Present for Barcoded Variant and Barcoded Identifier SeqLibs only. Histogram of unique barcodes per variant or identifier.

- Mismatches in overlapping reads



Present for Overlap SeqLibs only. Barplot of the number of resolved and unresolved mismatches at each position in the overlap region, and the number of times the first mismatch in a read pair occured at each position. Used for diagnosing misalignment of overlapping reads.

CHAPTER 7

Example notebooks

Begin exploring Enrich2 datasets with the following notebooks. They rely on the Enrich2 example dataset, so please perform that analysis before running any of these notebooks locally.

The notebooks can be run interactively by using the command line to navigate to the "Enrich2/docs/notebooks" directory and enter `jupyter notebook <notebook.ipynb>` where `<notebook.ipynb>` is the notebook file name.

The first two notebooks demonstrate using pandas to open an HDF5 file, extract its contents into a data frame, and perform queries on tables in the HDF5 file. For more information, see the pandas HDF5 documentation.

Selecting variants by input library count

This notebook gets scores and standard errors for the variants in a Selection that exceed a minimum count cutoff in the input time point, and plots the relationship between each variant's score and input count.

```
% matplotlib inline
```

```
from __future__ import print_function
import os.path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from enrich2.variant import WILD_TYPE_VARIANT
import enrich2.plots as enrich_plot
pd.set_option("display.max_rows", 10) # rows shown when pretty-printing
```

Modify the `results_path` variable in the next cell to match the output directory of your Enrich2-Example dataset.

```
results_path = "/path/to/Enrich2-Example/Results/"
```

Open the Selection HDF5 file with the variants we are interested in.

```
my_store = pd.HDFStore(os.path.join(results_path, "Rep1_sel.h5"))
```

The `pd.HDFStore.keys()` method returns a list of all the tables in this HDF5 file.

```
my_store.keys()
```

```
['/main/barcodemap',
 '/main/barcodes/counts',
 '/main/barcodes/counts_unfiltered',
 '/main/barcodes/log_ratios',
 '/main/barcodes/scores',
 '/main/barcodes/weights',
 '/main/synonymous/counts',
 '/main/synonymous/counts_unfiltered',
 '/main/synonymous/log_ratios',
 '/main/synonymous/scores',
 '/main/synonymous/weights',
 '/main/variants/counts',
 '/main/variants/counts_unfiltered',
 '/main/variants/log_ratios',
 '/main/variants/scores',
 '/main/variants/weights']
```

We will work with the “/main/variants/counts” table first. Enrich2 names the columns for counts `c_n` where `n` is the time point, beginning with `0` for the input library.

We can use a query to extract the subset of variants in the table that exceed the specified cutoff. Since we’re only interested in variants, we’ll explicitly exclude the wild type. We will store the data we extract in the `variant_count` data frame.

```
read_cutoff = 10
```

```
variant_counts = my_store.select('/main/variants/counts', where='c_0 > read_cutoff␣
↪and index != WILD_TYPE_VARIANT')
variant_counts
```

The index of the data frame is the list of variants that exceeded the cutoff.

```
variant_counts.index
```

```
Index([u'c.10G>A (p.Ala4Arg), c.11C>G (p.Ala4Arg), c.12T>A (p.Ala4Arg)',
       u'c.10G>A (p.Ala4Asn), c.11C>A (p.Ala4Asn)',
       u'c.10G>A (p.Ala4Asn), c.11C>A (p.Ala4Asn), c.12T>C (p.Ala4Asn)',
       u'c.10G>A (p.Ala4Ile), c.11C>T (p.Ala4Ile)',
       u'c.10G>A (p.Ala4Ile), c.11C>T (p.Ala4Ile), c.12T>A (p.Ala4Ile)',
       u'c.10G>A (p.Ala4Ile), c.11C>T (p.Ala4Ile), c.12T>C (p.Ala4Ile)',
       u'c.10G>A (p.Ala4Lys), c.11C>A (p.Ala4Lys), c.12T>A (p.Ala4Lys)',
       u'c.10G>A (p.Ala4Met), c.11C>T (p.Ala4Met), c.12T>G (p.Ala4Met)',
       u'c.10G>A (p.Ala4Ser), c.11C>G (p.Ala4Ser)',
       u'c.10G>A (p.Ala4Ser), c.11C>G (p.Ala4Ser), c.12T>C (p.Ala4Ser)',
       ...
       u'c.8C>T (p.Ser3Phe), c.60C>T (p.=)',
       u'c.8C>T (p.Ser3Phe), c.9T>C (p.Ser3Phe)', u'c.90C>A (p.=)',
       u'c.90C>G (p.Ile30Met)', u'c.90C>T (p.=)', u'c.9T>A (p.=)',
       u'c.9T>C (p.=)',
       u'c.9T>C (p.=), c.49A>T (p.Met17Ser), c.50T>C (p.Met17Ser), c.51G>A (p.
↪Met17Ser)',
       u'c.9T>C (p.=), c.62T>C (p.Leu21Ser), c.63A>T (p.Leu21Ser)',
       u'c.9T>G (p.=)'],
      dtype='object', length=1440)
```

We can use this index to get the scores for these variants by querying the "/main/variants/scores" table. We'll store the result of the query in a new data frame named `variant_scores`, and keep only the score and standard error (SE) columns.

```
variant_scores = my_store.select('/main/variants/scores', where='index in variant_
↪counts.index')
variant_scores = variant_scores[['score', 'SE']]
variant_scores
```

Now that we're finished getting data out of the HDF5 file, we'll close it.

```
my_store.close()
```

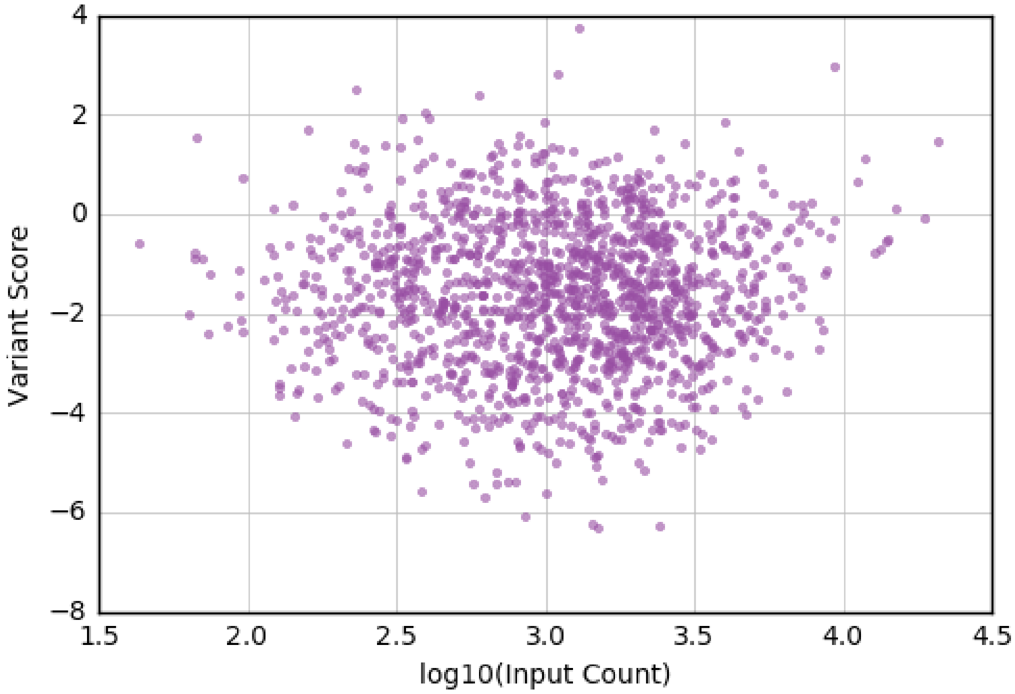
To more easily explore the relationship between input count and score, we'll add a column to the `variant_scores` data frame that contains input counts from the `variant_counts` data frame.

```
variant_scores['input_count'] = variant_counts['c_0']
variant_scores
```

Now that all the information is in a single data frame, we can make a plot of score vs. input count. This example uses functions and colors from the Enrich2 plotting library. Taking the log10 of the counts makes the data easier to visualize.

```
fig, ax = plt.subplots()
enrich_plot.configure_axes(ax, xgrid=True)
ax.plot(np.log10(variant_scores['input_count']),
        variant_scores['score'],
        linestyle='none', marker='.', alpha=0.6,
        color=enrich_plot.plot_colors['bright4'])
ax.set_xlabel("log10(Input Count)")
ax.set_ylabel("Variant Score")
```

```
<matplotlib.text.Text at 0x9e796a0>
```



Selecting variants by number of unique barcodes

This notebook gets scores for the variants in an Experiment that are linked to multiple barcodes, and plots the relationship between each variant's score and number of unique barcodes.

```
% matplotlib inline
```

```
from __future__ import print_function
import os.path
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from enrich2.variant import WILD_TYPE_VARIANT
import enrich2.plots as enrich_plot
pd.set_option("display.max_rows", 10) # rows shown when pretty-printing
```

Modify the `results_path` variable in the next cell to match the output directory of your Enrich2-Example dataset.

```
results_path = "/path/to/Enrich2-Example/Results/"
```

Open the Experiment HDF5 file.

```
my_store = pd.HDFStore(os.path.join(results_path, "BRCA1_Example_exp.h5"))
```

The `pd.HDFStore.keys()` method returns a list of all the tables in this HDF5 file.

```
my_store.keys()
```

```
['/main/barcodemap',
 '/main/barcodes/counts',
 '/main/barcodes/scores',
 '/main/barcodes/scores_shared',
 '/main/barcodes/scores_shared_full',
 '/main/synonymous/counts',
 '/main/synonymous/scores',
 '/main/synonymous/scores_pvalues_wt',
 '/main/synonymous/scores_shared',
 '/main/synonymous/scores_shared_full',
 '/main/variants/counts',
 '/main/variants/scores',
 '/main/variants/scores_pvalues_wt',
 '/main/variants/scores_shared',
 '/main/variants/scores_shared_full']
```

First we will work with the barcode-variant map for this analysis, stored in the "/main/barcodemap" table. The index is the barcode and it has a single column for the variant HGVS string.

```
bcm = my_store['/main/barcodemap']
bcm
```

To find out how many unique barcodes are linked to each variant, we'll count the number of times each variant appears in the barcode-variant map using a Counter data structure. We'll then output the top ten variants by number of unique barcodes.

```
variant_bcs = Counter(bcm['value'])
variant_bcs.most_common(10)
```

```
[('_wt', 5844),
 ('c.63A>T (p.Leu21Phe)', 109),
 ('c.39C>A (p.=)', 91),
 ('c.61T>A (p.Leu21Ile), c.63A>T (p.Leu21Ile)', 77),
 ('c.62T>A (p.Leu21Tyr), c.63A>T (p.Leu21Tyr)', 77),
 ('c.63A>G (p.=)', 73),
 ('c.72C>A (p.=)', 72),
 ('c.62T>G (p.Leu21Cys), c.63A>T (p.Leu21Cys)', 71),
 ('c.13C>A (p.Leu5Ile)', 70),
 ('c.62T>A (p.Leu21Ter)', 63)]
```

Next we'll turn the Counter into a data frame.

```
bc_counts = pd.DataFrame(variant_bcs.most_common(), columns=['variant', 'barcodes'])
bc_counts
```

The data frame has the information we want, but it will be easier to use later if it's indexed by variant rather than row number.

```
bc_counts.index = bc_counts['variant']
bc_counts.index.name = None
del bc_counts['variant']
bc_counts
```

We'll use a cutoff to choose variants with a minimum number of unique barcodes, and store this subset in a new index. We'll also exclude the wild type by dropping the first entry of the index.

```
bc_cutoff = 10
```

```
multi_bc_variants = bc_counts.loc[bc_counts['barcodes']  >= bc_cutoff].index[1:]
multi_bc_variants
```

```
Index([u'c.63A>T (p.Leu21Phe)', u'c.39C>A (p.=)',
       u'c.61T>A (p.Leu21Ile), c.63A>T (p.Leu21Ile)',
       u'c.62T>A (p.Leu21Tyr), c.63A>T (p.Leu21Tyr)', u'c.63A>G (p.=)',
       u'c.72C>A (p.=)', u'c.62T>G (p.Leu21Cys), c.63A>T (p.Leu21Cys)',
       u'c.13C>A (p.Leu5Ile)', u'c.62T>A (p.Leu21Ter)',
       u'c.63A>C (p.Leu21Phe)',
       ...
       u'c.88A>C (p.Ile30Arg), c.89T>G (p.Ile30Arg), c.90C>T (p.Ile30Arg)',
       u'c.76T>A (p.Cys26Lys), c.77G>A (p.Cys26Lys), c.78C>G (p.Cys26Lys)',
       u'c.22G>A (p.Glu8Ile), c.23A>T (p.Glu8Ile), c.24A>T (p.Glu8Ile)',
       u'c.49A>T (p.Met17Ser), c.50T>C (p.Met17Ser), c.51G>A (p.Met17Ser)',
       u'c.64G>A (p.Glu22Arg), c.65A>G (p.Glu22Arg)',
       u'c.77G>C (p.Cys26Ser), c.78C>G (p.Cys26Ser)',
       u'c.29T>A (p.Val10Glu), c.30A>G (p.Val10Glu)',
       u'c.50T>A (p.Met17Asn), c.51G>T (p.Met17Asn)',
       u'c.61T>A (p.Leu21Thr), c.62T>C (p.Leu21Thr), c.63A>G (p.Leu21Thr)',
       u'c.49A>G (p.Met17Ala), c.50T>C (p.Met17Ala)'],
      dtype='object', length=504)
```

We can use this index to get condition-level scores for these variants by querying the “/main/variants/scores” table. Since we are working with an Experiment HDF5 file, the data frame column names are a MultiIndex with two levels, one for experimental conditions and one for data values (see the pandas documentation for more information).

```
multi_bc_scores = my_store.select('/main/variants/scores', where='index in multi_bc_
↪variants')
multi_bc_scores
```

There are fewer rows in `multi_bc_scores` than in `multi_bc_variants` because some of the variants were not scored in all replicate selections, and therefore do not have a condition-level score.

Now that we’re finished getting data out of the HDF5 file, we’ll close it.

```
my_store.close()
```

We’ll add a column to the `bc_counts` data frame that contains scores from the `multi_bc_scores` data frame. To reference a column in a data frame with a MultiIndex, we need to specify all column levels.

```
bc_counts['score'] = multi_bc_scores['E3', 'score']
bc_counts
```

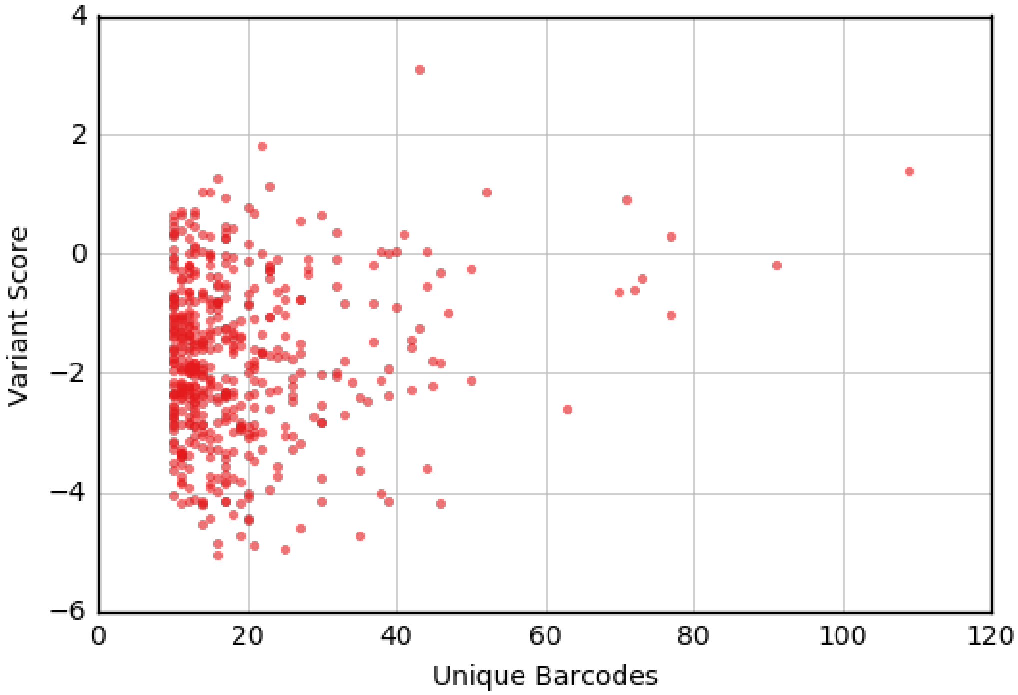
Many rows in `bc_counts` are missing scores (displayed as NaN) because those variants were not in `multi_bc_scores`. We’ll drop them before continuing.

```
bc_counts.dropna(inplace=True)
bc_counts
```

Now that we have a data frame containing the subset of variants we’re interested in, we can make a plot of score vs. number of unique barcodes. This example uses functions and colors from the Enrich2 plotting library.

```
fig, ax = plt.subplots()
enrich_plot.configure_axes(ax, xgrid=True)
ax.plot(bc_counts['barcodes'],
        bc_counts['score'],
        linestyle='none', marker='.', alpha=0.6,
        color=enrich_plot.plot_colors['bright5'])
ax.set_xlabel("Unique Barcodes")
ax.set_ylabel("Variant Score")
```

```
<matplotlib.text.Text at 0xd91fe80>
```



For more information on Enrich2 data tables, see *Output HDF5 files*.

CHAPTER 8

Appendix: API documentation

This page contains automatically generated documentation from the Enrich2 codebase. It is intended for developers and advanced users.

`storemanager` — Abstract class for Enrich2 data

This module contains the class definition for the `StoreManager` abstract class, the shared base class for most classes in the Enrich2 project. This class provides general behavior for the GUI and for handling HDF5 data files.

`StoreManager` class

`seqlib` — Sequencing library file handling and element counting

This module provides class definitions for the various types of sequencing library designs usable by Enrich2. Data for each FASTQ file (or pair of overlapping FASTQ files for overlapping paired-end data) is read into its own `SeqLib` object. If necessary, FASTQ files should be split by index read before being read by a `SeqLib` object. `SeqLib` objects are coordinated by `Selection` objects.

`SeqLib` and `VariantSeqLib` are abstract classes.

`SeqLib` class

`VariantSeqLib` class

`BarcodeSeqLib` class

`BcvSeqLib` class

`BcidSeqLib` class

`BasicSeqLib` class

`OverlapSeqLib` class

`IdOnlySeqLib` class

`SeqLib` helper classes

`Aligner` class

`WildTypeSequence` class

class `enrich2.wildtype.`**`WildTypeSequence`**(*parent_name*)

Container class for wild type sequence information. Used by `VariantSeqLib` objects and `Selection` or `Experiment` objects that contain variant information.

Requires a *parent_name* that associates this object with a StoreManager object for the purposes of error reporting and logging.

`duplicate`(*new_parent_name*)

Create a copy of this object with the *new_parent_name*.

Uses the configure and serialize methods to perform the copy.

`position_tuples`(*protein=False*)

Return a list of tuples containing the position number (after offset adjustment) and single-letter symbol (nucleotide or amino acid) for each position the wild type sequence.

`serialize`()

Format this object as a config object suitable for dumping to a config file.

`BarcodeMap` class

class `enrich2.barcodemap.`**`BarcodeMap`**(*mapfile*, *is_variant=False*)

Dictionary-derived class for storing the relationship between barcodes (keys) and variants (values). Requires the path to a *mapfile*, containing lines in the format `'barcode<whitespace>variant'` for each barcode expected in the library. This file can be plain text or compressed (`.bz2` or `.gz`).

Barcodes must only contain the characters `ACGT` and variants must only contain the characters `ACGTN` (lowercase characters are converted to uppercase).

Blank lines and lines that begin with `#` (comments) are ignored.

> *is_variant* is a boolean that is `True` if the barcodes are assigned to variant DNA sequences, or `False` if the barcodes are assigned to arbitrary identifiers. If this is `True`, additional error checking is performed on the variant DNA sequences.

`selection` — Functional score calculation using SeqLib count data

This module provides class definitions for the `Selection` class. This is where functional scores are calculated from the `SeqLib` count data. For time series data, each time point in the selection can have multiple `SeqLib` assigned to it, in which case the counts for each element will be added together. Each time series selection must have a time point 0 (the "input library").

`Selection` class

`Selection` helpers

`condition` — Dummy class for GUI

This module provides class definitions for the `Condition` classes. This class is required for proper GUI operation. All condition-related behaviors are in the `Experiment` class.

`Condition` class

`experiment` — Aggregation of replicate selections

This module provides class definitions for the `Experiment`. Functional scores for selections within the same condition are combined to generate a single functional score (and associated error) for each element in each experimental condition.

`Experiment` class

Enrich2 plotting

Text goes here.

Sequence-function map plotting

Text goes here.

Utility functions

Configuration object type detection

Functions for identifying the type of `StoreManager` derived object associated with a given configuration object (decoded from a JSON file as described here).

`enrich2.config_check.`**`element_type`**`(`*cfg*`)`

Get the type of `StoreManager` derived object specified by the configuration object.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns The class name of the `StoreManager` derived object specified by *cfg*.

Return type str

Raises `ValueError` – If the class name cannot be determined.

`enrich2.config_check.`**`is_condition`**`(`*cfg*`)`

Check if the given configuration object specifies a `Condition`.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns True if *cfg* if specifies a `Condition`, else False.

Return type bool

`enrich2.config_check.`**`is_experiment`**`(`*cfg*`)`

Check if the given configuration object specifies an `Experiment`.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns True if *cfg* if specifies an `Experiment`, else False.

Return type bool

`enrich2.config_check.`**`is_selection`**`(`*cfg*`)`

Check if the given configuration object specifies a `Selection`.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns True if *cfg* if specifies a `Selection`, else False.

Return type bool

`enrich2.config_check.`**`is_seqlib`**`(`*cfg*`)`

Check if the given configuration object specifies a `SeqLib` derived object.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns True if *cfg* if specifies a `SeqLib` derived object, else False.

Return type bool

`enrich2.config_check.`**`seqlib_type`**`(`*cfg*`)`

Get the type of `SeqLib` derived object specified by the configuration object.

Parameters **`cfg`** (*`dict`*) – decoded JSON object

Returns The class name of the `SeqLib` derived object specified by *cfg*.

Return type str

Raises `ValueError` – If the class name cannot be determined.

Dataframe and index helper functions

Variant helper functions

HGVS variant regular expressions

Enrich2 entry points

Python Module Index

c

`condition`, 45

e

`enrich2.config_check`, 45
`experiment`, 45

p

`plots`, 45

s

`selection`, 45
`seqlib`, 43
`sfmap`, 45
`storemanager`, 43

Index

B

BarcodeMap (class in enrich2.barcodemap), 44

C

condition (module), 45

D

duplicate() (enrich2.wildtype.WildTypeSequence method), 44

E

element_type() (in module enrich2.config_check), 45
enrich2.config_check (module), 45
experiment (module), 45

I

is_condition() (in module enrich2.config_check), 46
is_experiment() (in module enrich2.config_check), 46
is_selection() (in module enrich2.config_check), 46
is_seqlib() (in module enrich2.config_check), 46

P

plots (module), 45
position_tuples() (enrich2.wildtype.WildTypeSequence method), 44

S

selection (module), 45
seqlib (module), 43
seqlib_type() (in module enrich2.config_check), 46
serialize() (enrich2.wildtype.WildTypeSequence method), 44
sfmap (module), 45
storemanager (module), 43

W

WildTypeSequence (class in enrich2.wildtype), 44