ortho_seqs

Release 1.0.1

Saba Nafees

Jan 01, 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION:

This documentation accompanies the ortho_seqs python command line tool that computes multivariate tensor-based orthogonal polynomials based on DNA, RNA or protein sequence data and maps corresponding phenotypic information onto the sequence space.

CHAPTER

TWO

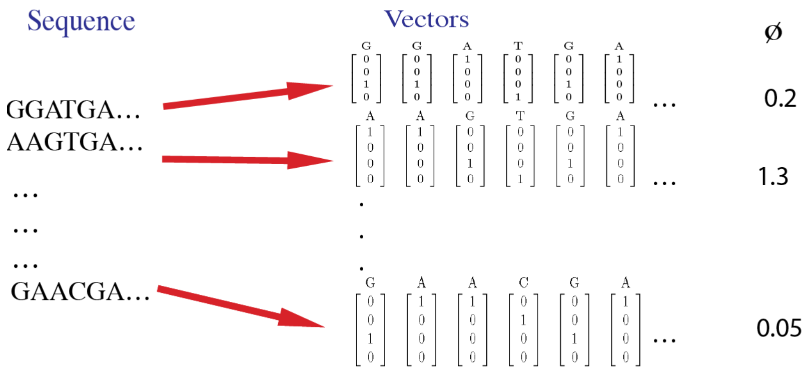
GUIDE

2.1 Background & Quickstart

2.1.1 ortho_seqs

Ortho_seqs is a command line to convert sequence data (DNA/protein) to tensor-valued orthogonal polynomials and project phenotypes onto the polynomial space. We do this by first converting the sequence information into 4-dimensional (for DNA) or 20-dimensional (for amino acids) vectors. The method can also be used for padded sequences to deal with unequal sequence lengths. Find out more about the approach in this paper Analyzing genomic data using tensor-based orthogonal polynomials with application to synthetic RNAs. The paper gives an example of this method as applied to a case of synthetic RNA from a previously published dataset. Another manuscript detailing the use of this method to understand binding affinities of transcription factors (TFs) is currently in progress.

For example, the sample data inputs for this tool are shown in this image. Here, each site in a sequence is first converted to a 4-dimensional vector. The input data includes phenotype values for each sequence.



2.1.2 Usage

First, install an environment with dependencies for this package:

```
conda create -n ortho_seqs pip
pip install -r requirements.txt
conda activate ortho_seq
```

or

```
conda env create -f conda_environment.yml
conda activate ortho_seq
```

Then, install the package:

```
python setup.py install
```

Gather the input file(s) needed.

There are three main ways to submit your sequence and phenotype files to *ortho_seqs*. The first method is to submit them separately, in their own .txt files. Recently, however, an update was added that allows you to submit them both in the same file. For this to apply: 1) The file must be either a .xlsx or a .csv file. 2) The sequences must be in the first column, and the phenotypes must be in the second column. 3) The columns must not have header names.

If you use a single file for the sequence and phenotype, you would submit the file path where you would submit the sequence file path, and leave the *–pheno_file* flag blank (*ortho_seqs* will set that flag to **None**).

The phenotypes must be real numbers.

Then, to run the commandline tool:

To start with a test example, you can run the sample command below:

```
ortho_seq orthogonal-polynomial ./ortho_seq_code/tests/data/nucleotide/first_order/test_
↪seqs_2sites_dna.txt --molecule DNA --pheno_file ./ortho_seq_code/tests/data/nucleotide/
↪first_order/trait_test_seqs_2sites_dna.txt --poly_order second --out_dir ../results_
↪ortho_seq_testing/DNA_2sites_test_run/
```

The above sample command line is building the tensor-valued orthogonal polynomial space based on the sequence data which consists of 12 sequences, each with two sites. Since these are DNA sequences, the vectors are 4-dimensional. These used to be flags for sites, dimensions, and population size, but new functionality will automatically calculate these. Corresponding to each sequence is a phenotype value (a real number) as given in the phenotype file. For DNA, the tool can run first and second order analyses currently. We'll implement third order in a future version. For amino acids, the current version supports first order analysis and we hope to expand this in the future.

Amino acids/nucleotides that do not appear in any sequence will be removed from the alphabet when the letters are being converted to first order vectors. For example, if the residue 'R' (Arginine) never occurs in the sequence dataset, the first order vectors will now have 19 dimensions (instead of 20) and 20 dimensions (instead of 21) if the sequences are padded with 'n'. This is done to greatly reduce runtime for larger sequence datasets and for longer sequences. When the program will run, it will return this sentence:

```
Will be computing p sequences with s sites, and each vector will be d-dimensional.
```

Where p represents the population size (number of rows in sequence file), s represents the number of sites, and d represents the number of amino acids/nucleotides detected in the sequence file (adds on 1 for lowercase n's). For the above example, the program will return

```
Will be computing 12 sequences with 2 sites, and each vector will be 4-dimensional.
```

Along with regressions on each site independent of one another and onto two sites at a time, the above command also computes *Fest* which is the phenotype estimated by the regressions. This shows that the mathematical calculations are done correctly as we now have an equation that accurately captures our initial data points. This only works here for sequences with 2 sites. If we had more sites, we'd need to do higher order calculations in order to capture all our combinations. Therefore, when running the tool with more sites, as will probably be the case for most users, even just going up to second order gives us useful information about our system. First order tells us the importance of each site (independent of any correlations it might have with another site) and second order tells the importance of pairs of nucleotides independent of other pairs. Please take a look at the paper linked above to learn more about this method.

Flags & Functionality

```
--pheno_file
```

Input a file with phenotype values corresponding to each sequence in the sequence file. If you have a .xlsx or .csv file, do NOT use this flag (more details above in the **Gather the Input Files Needed** section).

```
   --molecule

Currently, you can provide DNA or protein sequences. Here, you can also provide␣
↪sequences of unequal lengths, where sequences will be padded with lowercase 'n's until␣
↪it has reached the length of the longest sequence.
```

–poly_order

```
The order of the polynomials that will be constructed. Currently, one can do first and␣
↪second order for DNA and first order for protein.
```

–out_dir

```
Directory where results can be stored.
```

–precomputed

```
Let's say you have a case where you have the same set of sequences but two different␣
↪corresponding sets of phenotypes. You can build your sequence space and then project␣
↪the first set of phenotypes onto this space. Then, if you wish to see how the other␣
↪set of phenotypes maps onto the same sequence space, you can use this flag so that you
↪'re not wasting time and memory to recompute the space. When doing this, be sure to␣
↪add your results from the first run to the **out_dir** when rerunning the command with␣
↪the **precomputed** flag.
```

–alphbt_input

```
Used to group amino acids/nucleotides together, or specify certain amino acids/
↪nucleotides. If you don't want to group anything, don't include this flag when running␣
↪*ortho_seqs*. For example, putting *ASGR* will tell the program to have 6 dimensions:␣
↪one for each amino acid specified, and one for *z*, where every unspecified amino acid␣
↪will be converted to *z*, and one for *n* (whenever sequences have unequal lengths,␣
↪*ortho_seqs* will pad the shorter sequences with *n*). You can also comma-separate␣
↪amino acids/nucleotides to group them. For example, putting *AS,GR* will make the␣
↪vectors 4-dimensional, one for *AS*, one for *GR*, one for every other amino acid, and␣
↪one for *n*.

There are also built-in groups:

**protein_pnp** will group by polar and non-polar amino acids, every other amino acid,␣
↪and *n*.
```

(continues on next page)

(continued from previous page)

```
**essential** groups by essential and non-essential amino acids, every other amino acid,␣
↪and *n*. \
Group 1: Essential - ILVFWHKTM \
Group 2: Non-Essential - Everything else \
Group 3: n \
(Source: https://www.ncbi.nlm.nih.gov/books/NBK557845/)

**alberts** groups by categories set by Alberts. \
Group 1: Basic - KRH \
Group 2: Acidic - DE \
Group 3: AVLIPFMWGC \
Group 4: Everything else \
Group 5: n \
(Source: https://www.ncbi.nlm.nih.gov/books/NBK21054/)

**sigma** groups by categories set by Sigma. \
Group 1: Aliphatic - AILMV \
Group 2: Aromatic - FYV \
Group 3: Polar Neutral - NQCST \
Group 4: Acidic - KRH \
Group 5: Basic - DE \
Group 6: Other - G \
Group 7: Other - P \
Group 8: n \
(Source: https://www.sigmaaldrich.com/US/en/technical-documents/technical-article/
↪protein-biology/protein-structural-analysis/amino-acid-reference-chart)

**hbond** groups by strength of hydrogen bond attractions. \
Group 1: Can Make Hydrogen Bonds - NQSTDERKYHW \
Group 2: Can Not Make Hydrogen Bonds - Everything else \
Group 3: n \
The first group is able to make hydrogen bonds, whereas the second group is not.

**hydrophobicity** groups by hydrophobicity. \
Group 1: Very Hydrophobic - LIFWVM \
Group 2: Hydrophobic - CYA \
Group 3: Neutral - TEGSQD \
Group 4: Hydrophilic - Everything else \
Group 4: n \
The first group is very hydrophobic, the second group is slightly hydrophobic, the third␣
↪group is neutral, and the last group is hydrophilic.
```

–min_pct

```
When ortho_seqs is run, a .csv file of covariances will be saved in the specified path.␣
↪This matrix of covariances is one of the main results of the program (as shown in
↪{sequence_file_name}.npz output below). The csv file will contain the covariance of␣
↪each nucleotide at each site with another nucleotide at another site (or amino acids␣
↪at each site).
Suppose there are 5 covariance values of 2, 1, 0, 0, -1. For the percentiles, all unique␣
↪*magnitudes* will be considered when assigning covariances, which will be 2, 1, and 0.␣
↪0 will be the 0th percentile (therefore, assigning 0 to the *--min_pct* flag will␣
↪return every covariance), 1 (and -1) will be 33.33..., and 2 will be 66.66...␣
↪Specifying 50 as *--min_pct* will only return the row with the covariance of 2, since␣
↪only 66.6...>50.
```

(continues on next page)

(continued from previous page)

```
The min_pct flag is short for minimum percentile, which will remove any covariances
from the .csv file that are below the given percentile. The default value is 75.
```

–pheno_name

```
Let's say you know that your phenotype values represent IC50 values. You could then add␣
↪*--pheno_name IC50* as a flag, and on the rFon1D plot that is automatically generated,␣
↪the y-axis label will include IC50. Default is **None**.

# Results & Outputs

The tool will provide updates as the run is progressing regarding which parts of the␣
↪calculations are done being computed. For example, when the mean is computed, it'll␣
↪say "computed mean". All the different elements that it is computing are different␣
↪parts of building the multivariate tensor-valued orthogonal polynomial space based on␣
↪the sequence information. To get a general idea of what the calculations mean, please␣
↪refer to the supplementary methods in the paper linked above.
The program will save outputs in [npz format](https://numpy.org/doc/stable/reference/
↪generated/numpy.savez.html). See below for what is stored.
```

{sequence_file_name}.npz

```
This will store the calculations that went into constructing the polynomial space. This␣
↪also includes information about the statics of our sequence space, such as mean,␣
↪variance and a matrix of covariances. See figures 4 and for ideas on how mean and the␣
↪matrix of covariances can be visualized. All of these calculations go into building␣
↪the orthogonal polynomial space based on sequence information and at this point of the␣
↪program, we have not connected the phenotype (the functional variable) with the␣
↪sequence information.
```

{sequence_file_name}_covs_with_F.npz

```
This will store the covariance of the phenotype (or trait) with the polynomials. This is␣
↪when we start connecting the phenotype with the sequence space.
```

{trait_file_name}_Fm.npz

```
This contains the mean trait value. This is a scalar.
```

{trait_file_name}_regressions.npz

```
This set of files contains the main results which includes the following:
1. **rFon1D**: This is the regression of the trait onto the first order conditional␣
↪polynomial orthogonalized within. This tells us the regression of the phenotype onto␣
↪each site and onto each nucleotide (or amino acid) at that site independent of any␣
↪correlations that site might have with other sites. For the case of nucleotides, this␣
↪can be visualized as bar plots as shown in Figure 6 in the paper linked above.
2. **rFon2D**: This gives 4 matrices which give the regression of the pheonotype onto␣
↪(site1)x(site1), (site 1)x(site 2), (site 2)x(site 1) and (site 2)x(site 2), in that␣
↪order. The second matrix here is the important one and it is the same as rFon12. See␣
↪description of rFon12.
3. **rFon12**:  This is the regression of the trait onto *pairs* of sites for given␣
↪nucleotides at each site. These are regressions on (site 1)x(site 2) independent of␣
↪first order associations. Since we're looking at 2 sites at a time and there's a␣
↪possibility of having 4 nucleotides at each site (for the case of DNA), we can␣
↪visualize this via a 4x4 matrix as shown in Figure 8 in the paper linked above.
```

(continues on next page)

(continued from previous page)

```
# The rf1d class
The newest update to *ortho_seqs* involves adding a new class of objects, called *rf1d*␣
↪(short for rFon1D). To declare an *rf1d* object, you must supply a numpy ndarray of␣
↪the rFon1D values (supplied via the regressions.npz file that is output when *ortho_
↪seqs* is run), along with the alphabet input, and the molecule type.
Once you have done this, you can explore the rFon1D values, and see if there are any␣
↪trends. You can:
1. Print out a summary of the object using .summary()
2. Make a bar plot (just like the rFon1D bar plot automatically generated) using .plot_
↪bar()
3. Print out the extreme rFon1D values using .sort()
4. Remove any insignificant, but nonzero, rFon1D values using the .trim() function␣
↪(**NOTE:** this will change the ndarray in the rf1d object, and to recover old data,␣
↪you must reinstatiate an rf1d object)
5. Make a histogram of rFon1D values using .plot_hist()


# To run the GUI (currently in development)
A GUI version of the CLI is being actively developed in order to make it easier for␣
↪users to utilize the tool. The GUI allows the user to upload the sequence and␣
↪phenotype files via an upload button, specify the molecule, the polynomial order they␣
↪wish to run, and whether the sequence space was already computed or not (via the␣
↪precomputed button). The GUI is in its primitive form and will include further updates␣
↪resembling the cli in future versions.

![GUI - early version](https://github.com/snafees/ortho_seqs/blob/gui_draft/GUI_in_
↪development.png?raw=true)
```

cov*hist*{trait_file_name}.png

```
This is a histogram of all non-zero covariances. Its bin width is 0.5.
```

cov_data*frame*{trait_file_name}.csv

```
This file is a csv file of covariances between every item at every site. This includes␣
↪the item ID and site for both items in the pair used to calculate the covariance, the␣
↪covariance value, the covariance magnitude, and an ID for the pair (s1-g2,s3-g4␣
↪represents the pairing of an element from the first group in the alphabet at the␣
↪second site, and an element from the third group at the fourth site).
```

rFon1D*graph*{trait_file_name}.png ``` This is a bar plot of all nonzero rFon1D values of every item at every site.

2.1.3 Support

If you have specific or general questions, feel free to open an issue and we'll do our best to address them. If you have any comments, suggestions or would like to chat about this method or similar ideas, feel free to reach out via email at saba.nafees314@gmail.com.

2.1.4 Roadmap

We hope to implement third order analysis for DNA in the near future. For amino acids, we hope to implement second order analysis. We'll add visualization ideas soon but if you have any thoughts on this, please feel free to reach out.

2.1.5 Contribution

We hope to make the tool run faster as with higher dimenions and higher order analysis of longer sequence data, we can run into memory and time issues. Any thoughts on this or visualization are welcome.

2.1.6 Authors and acknowledgements

The derivation of the method and the construction of an initial version of the program was done by Dr. Sean Rice who served as Saba's Ph.D. advisor. Thank you to Isaac Griswold-Steiner for helping write the function to compute generalized inner and outer products. Thank you to Pranathi Vemuri for helping with the very initial draft of the CLI, adding CI integration testing, and to Phoenix Logan for helping write unit-tests. Thank you to AhmetCan for helping initiatie the first GUI version. Thank you, Aaron, for always being ready to review PRs and for your insights/help in the development process. Thank you to Vijayanta Jain and Saugato Rahman Dhruba for being the guinea pigs and running lots of sample commands, discussing the mathematics with me, and for their ideas on visualizations. Their efforts are deeply appreciated!

2.1.7 License

MIT

2.2 Tutorial: Running a sample dataset (protein sequences)

This document will walk you through the steps of how to run a dataset on ortho_seqs, and what the various outputs are. This tutorial uses protein sequence data from *The Intrinsic Contributions of Tyrosine, Serine, Glycine, and Arginine to the Affinity and Specificity of Antibodies* by Birtalan & Sidhu et al., 2008.

2.2.1 1. Setting Up Your Computer to Run ortho_seqs

The first thing you have to do (aside from gathering data!) is set up your computer to run ortho_seqs.

> You first need to have Miniconda installed on your computer, in order to do the shell commands. To do so, follow the link here, and choose the appropriate version, with regards to your computer.

After you have installed Minoconda, open up Terminal, or an equivalent Command-Line Interface (CLI). Run either this:

```
conda create -n ortho_seqs pip
conda activate ortho_seqs
pip install -r requirements.txt
```

Or, alternatively:

```
conda env create -f conda_environment.yml
conda activate ortho_seqs
```

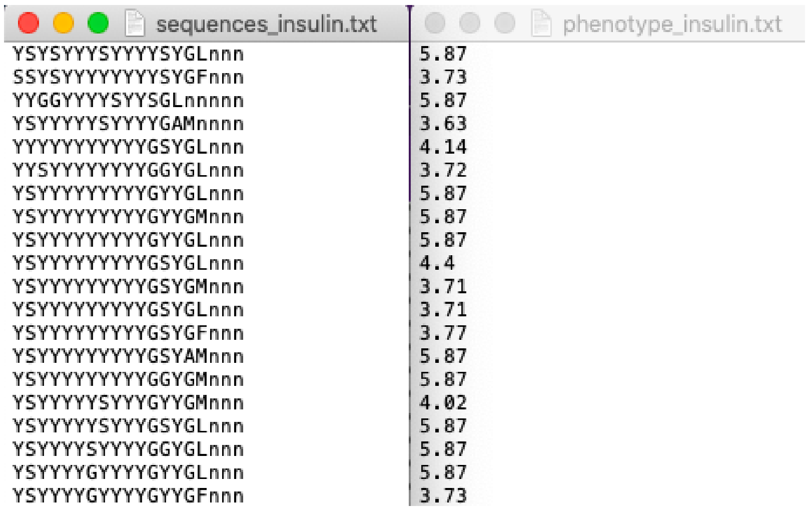
To activate ortho_seqs on your device. You will also need to run:

```
conda install openpyxl
python setup.py install
```

This line must be run every time ortho_seqs is updated, so you are using the most recent version. If the above steps have worked, congrats! You now have ortho_seqs on your computer. It's time to input some data.

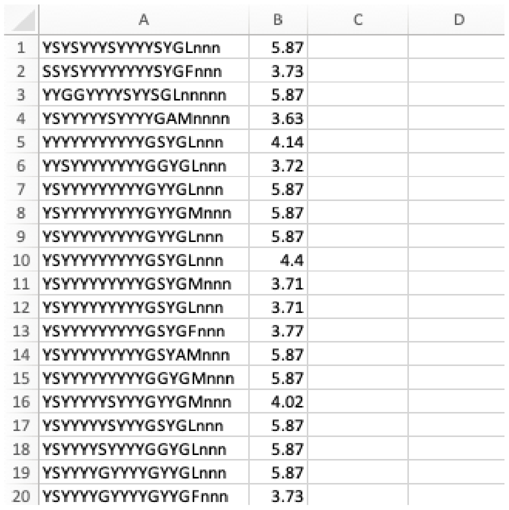
2.2.2 2. Your dataset

The data that is input to ortho_seqs must include a column of sequences, and a column of their corresponding phenotype values. These two columns can either be separate .txt files, or a single .xlsx or .csv file. Take, for instance, our toy example, which is a dataset originating from a paper titled The Intrinsic Contributions of Tyrosine, Serine, Glycine, and Arginine to the Affinity and Specificity of Antibodies by Sidhu et al. In this work, the authors constructed synthetic antibody Fab libraries to measure the impact of four different amino acids, Tyr, Ser, Gly and Arg on antigen recognition. Affinity and specificity data for these antigen-binding Fabs is provided for 3 different antigens (insulin, VEGF, and HER2). For this tutorial, we look at CDRH3 sequences of Fabs binding to insulin, along with corresponding phenotypes which is given by Specificity ELISA Signal Optical Density (see Figure 4a in the paper). This will be referred to as the "Sidhu dataset" for this tutorial. The dataset, when input into ortho_seqs, should look like



```
YSYSYYYSYYYYSYGLnnn     5.87
SSYSYYYYYYYYSYGFnnn     3.73
YYGGYYYYSYYSGLnnnnn     5.87
YSYYYYYSYYYYGAMnnnn     3.63
YYYYYYYYYYYYGSYGLnnn    4.14
YYSYYYYYYYYYGGYGLnnn    3.72
YSYYYYYYYYYYGYYGLnnn    5.87
YSYYYYYYYYYYGYYGMnnn    5.87
YSYYYYYYYYYYGYYGLnnn    5.87
YSYYYYYYYYYYGSYGLnnn    4.4
YSYYYYYYYYYYGSYGMnnn    3.71
YSYYYYYYYYYYGSYGLnnn    3.71
YSYYYYYYYYYYGSYGFnnn    3.77
YSYYYYYYYYYYGSYAMnnn    5.87
YSYYYYYYYYYYGGYGMnnn    5.87
YSYYYYYSYYYGYYGMnnn     4.02
YSYYYYYSYYYGSYGLnnn     5.87
YSYYYYSYYYYGGYGLnnn     5.87
YSYYYYGYYYYGYYGLnnn     5.87
YSYYYYGYYYYGYYGFnnn     3.73
```

or



	A	B	C	D
1	YSYSYYYSYYYYSYGLnnn	5.87		
2	SSYSYYYYYYYYSYGFnnn	3.73		
3	YYGGYYYYSYYSGLnnnnn	5.87		
4	YSYYYYYSYYYYGAMnnnn	3.63		
5	YYYYYYYYYYYGSYGLnnn	4.14		
6	YYSYYYYYYYYGGYGLnnn	3.72		
7	YSYYYYYYYYYGYYGLnnn	5.87		
8	YSYYYYYYYYYGYYGMnnn	5.87		
9	YSYYYYYYYYYGYYGLnnn	5.87		
10	YSYYYYYYYYYGSYGLnnn	4.4		
11	YSYYYYYYYYYGSYGMnnn	3.71		
12	YSYYYYYYYYYGSYGLnnn	3.71		
13	YSYYYYYYYYYGSYGFnnn	3.77		
14	YSYYYYYYYYYGSYAMnnn	5.87		
15	YSYYYYYYYYYGGYGMnnn	5.87		
16	YSYYYYYSYYYGYYGMnnn	4.02		
17	YSYYYYYSYYYGSYGLnnn	5.87		
18	YSYYYYSYYYYGGYGLnnn	5.87		
19	YSYYYYGYYYYGYYGLnnn	5.87		
20	YSYYYYGYYYYGYYGFnnn	3.73		

Note that for .xlsx (and .csv) files, the first column must be the sequences, and the second column must be the phenotypes. In addition, there must not be any header names for any files.

2.2.3 3. Executing Ortho_Seqs

We now turn towards our CLI to execute ortho_seqs. Using the Sidhu dataset, our input would look like:

```
ortho_seq orthogonal-polynomial ortho_seq_code/tests/data/nucleotide/onefile_tests/sidhu.
↪xlsx --molecule protein --poly_order first --out_dir ../onefile_tests/sidhu --alphbt_
↪input SYG,R --min_pct 40
```

Let's explore what these flags are, and how you can use them.

The file input (ortho_seq_code/. . . /sidhu.xlsx) is our sequence AND phenotype data.

```
--molecule
```

This flag is where you indicate what kind of molecule this is. This can be DNA, RNA, or protein. For the Sidhu dataset, the molecules are protein molecules.

```
--poly_order
```

This flag is to indicate the highest degree of polynomial order you want. Currently, DNA and RNA can go up to 2, and protein can only be 1. For the Sidhu dataset, we will look at first-order interactions.

```
--pheno_file
```

This flag is not in the example, because we don't need it. If you were to present your data as two separate .txt files, then this would be where you put the file path for the phenotype data, and the first file path is for your sequence data.

```
--out_dir
```

This flag indicates where you want the output files to go (more on what exactly is saved there later). If the folder path already exists, ortho_seqs will create a new directory with a very similar name, and it will tell you what the new path's name is.

```
--alphbt_input
```

(Note: "Characters" in the following section refer to the nucleotides for DNA, the bases for RNA, and all 21 amino acids for proteins, plus one additional character, "n", which indicates nothing is at that spot to deal with protein sequences of unequal lengths.) This flag indicates the groupings of characters you want. The default will be no groupings, or every character gets counted on its own. If you include (uppercase) letters here, then only those characters will be used (every other character, except "n", gets converted to a "z" and treated as one group). If you comma-separate somewhere in that group, then characters will be grouped based on what comma(s) they are in between. For the Sidhu dataset, to show proof of concept, the groupings will be:

1. SYG
2. R
3. Everything else (z)
4. n

If we were to leave out the commas, the groups would be:

1. S
2. Y
3. R
4. G
5. Everything else (z)
6. n

```
--min_pct
```

One output will be an .xlsx file containing all of the first-order covariances between each amino acid at each side with another amino acid at another site. However, this file can get pretty big pretty quick. Therefore, this flag will only print out covariance values whose magnitudes are at or above the PERCENTILE value specified. The default is 75, meaning it will only save the covariances which range from the 75th to the 100th percentiles in magnitude. To keep it at the default, leave out this flag when inputting what you want. For the Sidhu dataset, we want all magnitudes at or above the 40th percentile (as proof of concept).

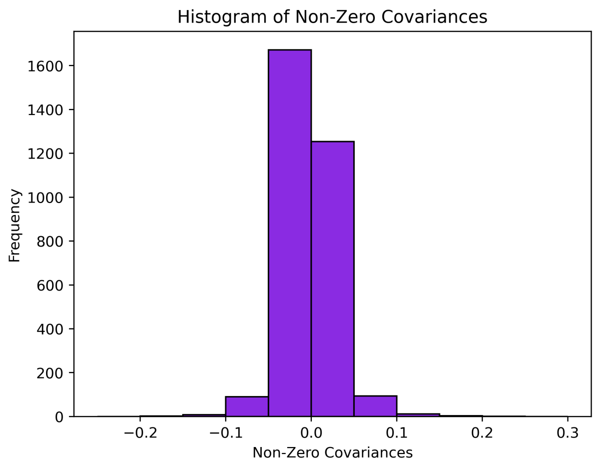
2.2.4 4. Obtained Outputs

CLI Outputs

The CLI will print out the g

Histogram and Spreadsheet of Covariances

The covariances between every character at every site with every other character at another site is recorded in a .csv file, and includes everything at or above the minimum percentile you specified in the input (or defaults to 75th percentile). In addition, the program outputs a histogram of the non-zero covariances, with the bin widths always being 0.5. For the Sidhu dataset, it looks like



And will have the file name cov_hist_{name}.png The .csv file has 8 columns.

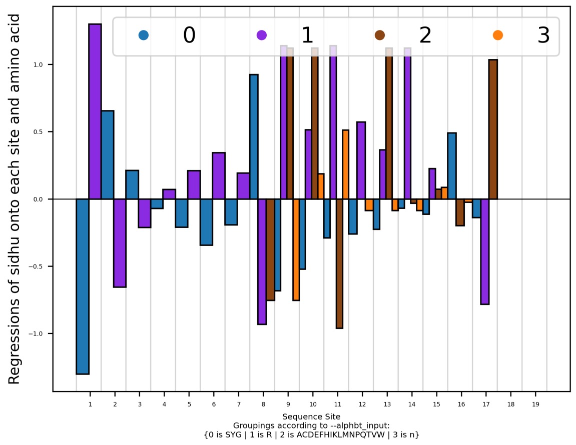
They are:

1. ID: Useful for searching for a specific pairing. Ordering will be s{Site 1)-g{Group 1}, s{Site 2)-g{Group 2}. For example, s1-g2,s10-g8 refers to the pairing between Group 2 at Site 1, and Group 8 at Site 10.
2. Magnitude: Absolute value of the covariance value, used to assign percentile values and plot histogram.
3. Covariance: The obtained covariance value.
4. First Site: The site of one of the groups of the covariance pairing. Site 1 for ID column.
5. First Group: The group the character belongs to, identifiable through the –alphbt_input dictionary. Group 1 for ID Column.
6. Second Site: The site of the other group of the covariance pairing. Site 2 for ID column.
7. First Group: The group the character belongs to, identifiable through the –alphbt_input dictionary. Group 2 for ID Column.
8. Percentile: The percentile the respective magnitude is, relative to the entire dataset (including magnitudes that were omitted from the .csv file).

E.g., in our case, say we have a value of -.051 for the covariance corresponding to s1-g1,s5-g2. This means that the group SYG at site 1 covaries negatively with an Arg at site 5. This covariance analysis tells us things about the statics of the sequence space. At this point, we have not projected our phenotype onto the sequence space. Here, we can discover patterns of covariation between amino acids at a given site with amino acids at other sites. When looking at the distribution of the magnitude of covariances, we can identify the ones at the tail ends of this distribution. This information is denoted in the output csv and the allows for the identification of highly covarying (negative or positive) sites. +++++++++ rFon1D Graph +++++++++

The main result for the first order analysis is the regression of the phenotype (in this case, ELISA values) onto the first order conditional polynomial (denoted as rFon1D). This tells us the effect of having a given amino acid at one site independent of its correlations with other amino acids at other sites. Here, we can use this result to understand the independent effects of a given amino acid at a given site on the phenotype.

One output is a graph of the nonzero rFon1D values. For the Sidhu dataset, it looks like



At the bottom, it lists the dictionary of the groups and their corresponding number, which then can be used to determine which color bar belongs to which group. The rFon1D graph will always have the name rFon1D_graph_{name}.png. The rFon1D values can also be found in the _regressions.npz file which can be opened up by the user in a jupyter notebook for further visualization.

2.3 Lincense

2.4 Get Further Help

If you have more questions, please refer to the github repo at https://github.com/snafees/ortho_seqs for existing issues or start an issue there. You can also refer to the PyPI page at https://pypi.org/project/ortho-seq-code/

2.4.1 Contact

If you have more questions or comments, please feel free to reach out to me at saba.nafees314@gmail.com. We look forward to hearing from you!

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search