SEEKR functions

download_gencode

Description:

Enables download of fasta and gtf files from the GENCODE homepage. Note: if more than 50 transcript_ids in the gtf file cannot be matched to headers in the fasta file, there will be a warning. The transcript_id in the gtf file is located inside the 9th field. This warning will trigger often, because the typical GENCODE gtf contains many more annotations than are present in the set of lncRNA.fa sequences.

Python Example:

Download the GENCODE release 33 of human lncRNAs in fasta format and rename it as human_v33.fa.gz. Also download the GENCODE release 33 of human lncRNAs in gtf format and rename it human_v33.gtf.gz. Unzip and remove both .gz files.

```python
from seekr import fasta
downloader = fasta.Downloader()
downloader.get_gencode(biotype='lncRNA', species='human',
                       gtf=True, release='33',
                       fasta_path='human_v33.fa.gz',
                       gtf_path='human_v33.gtf.gz', unzip=True)
```

Console Example:

Download the GENCODE release M25 of all mouse transcripts in fasta format and rename it as vM25_all.fa.gz. Download the GENCODE release M25 of all mouse transcripts in gtf format (gencode.vM25.chr_patch_hapl_scaff.annotation.gtf.gz) and rename it vM25_all.gtf.gz. Unzip and remove both .gz files.

```
$ seekr_download_gencode all -s mouse -g -r M25 -fp
vM25_all.fa.gz -gp vM25_all.gtf.gz
```

If release (-r) is omitted, the latest release will be downloaded:

```
$ seekr_download_gencode all -s mouse -g
```

You may also keep both .gz files zipped by using -z:

```
$ seekr_download_gencode all -s mouse -g -z
```

Only download the fasta file (and not the gtf), by omitting -g:

```
$ seekr_download_gencode all
```

To save output file(s) in another location:

`-fp /Users/username/Desktop/vM25_all.fa.gz`

Input:

biotype (biotype, positional argument): Name of the GENCODE sequence set to download. Must be one of 'all', 'pc' or 'lncRNA'. 'all': Nucleotide sequences of all transcripts on the reference chromosomes. 'pc': Nucleotide sequences of coding transcripts on the reference chromosomes. 'lncRNA': Nucleotide sequences of long non-coding RNA transcripts on the reference chromosomes.

species (-s, --species): Name of species. Must be 'human' or 'mouse'.

gtf (-g, --gtf): Whether to download the accompanying gtf file (gencode.vXX.chr_patch_hapl_scaff.annotation.gtf.gz). Default is False.

release (-r, --release): Name of GENCODE release to download. For human use a number string (e.g. '33'). For mouse releases, start with the letter 'M' followed by a number (e.g. 'M25'). If set to None, or omit -r in command line, the latest release will be downloaded.

fasta_path (-fp, --fasta_path): Path and name for the saved fasta. Default is None, or omit -fp in command line, which will save file with a name consisting of GENCODE release and biotype.

gtf_path (-gp, --gtf_path): Path and name to save the gtf file if any. Default is None, or omit -fp in command line, which will save the file with a name consisting of GENCODE release, biotype and the suffix: .chr_patch_hapl_scaff.annotation.gtf.gz.

unzip (-z, --zip): Within the Python module, unzip is set to True by default; .gz files will automatically be unzipped and deleted after downloading. Within the command line module, unzip is also True by default. If -z is called, .gz files will not be unzipped.

filter_gencode

Description:

Function that enables filtering of fasta files from GENCODE by 1): sequence length; 2) Transcripts that harbor Ensembl_canonical tags; 3): transcript isoform number; 4): duplicated sequences. Please use as input only those fasta and gtf files downloaded from GENCODE, other formats are not tested. Ensure fasta and gtf files are from the same release and same species.

Python Example:

Filter GENCODE vM33 mouse lncRNA transcripts and only keep those that are greater than or equal to 500 nts long; the feature type 'transcript'; a tag 'Ensembl_canonical' and isoform number '201' in the transcript_name. And finally remove all duplicated sequences.

```
from seekr import filter_gencode
headers,seqs =
filter_gencode.filter_gencode(fasta_path='vM33.lncRNA_transcripts.fa',
                              gtf_path='vM33.long_noncoding_RNAs.gtf',
                              len_threshold=500,canonical=True,
                              isoform='201',rm_dup=True,outputname='test')
```

Console Example:

Filter GENCODE vM33 mouse lncRNA transcripts and only keep those that are greater than or equal to 500 nts long; the feature type 'transcript'; a tag 'Ensembl_canonical' and isoform number '201' in the transcript_name. And finally remove all duplicated sequences.

```
$ seekr_filter_gencode gencode.vM33.lncRNA_transcripts.fa -gtf
gencode.vM33.long_noncoding_RNAs.gtf -len 500 -can -iso 201 -rd -
o test
```

Filter to retain transcripts greater than or equal to 500nts in length:

```
$ seekr_filter_gencode gencode.vM33.lncRNA_transcripts.fa -len
500 -o test
```

Filter to retain transcripts labelled with the Ensembl_canonical tag and remove duplicated sequences:

```
$ seekr_filter_gencode gencode.vM33.lncRNA_transcripts.fa -gtf
gencode.vM33.long_noncoding_RNAs.gtf -can -rd -o test
```

Filter to retain only those isoforms whose annotation numbers end in ‘01’:

```
$ seekr_filter_gencode gencode.vM33.lncRNA_transcripts.fa -gtf
gencode.vM33.long_noncoding_RNAs.gtf -iso '[0-9]01' -o test
```

To save output file in another location:

`-o /Users/username/Desktop/test`

Input:

`fasta_path (fasta, positional argument)`: path to the fasta file to be filtered.

`gtf_path (-gtf, --gtf_path)`: path to the gtf file that contains annotations of the sequences in the fasta file. Only needed for Ensembl_canonical or isoform filtering. Otherwise, set gtf_path=None (omit the -gtf argument in console).

`len_threshold (-len, --len_threshold)`: the length threshold for filtering, if no length filtering is needed, set len_threshold=0 (default) (omit the -len argument in console). Transcripts of lengths equal to or greater than the threshold will be retained.

`canonical (-can, --canonical)`: Retain only those sequences that carry an 'Ensembl_canonical' tag. (default, False).

`isoform (-iso, --isoform)`: whether to filter by isoform number. In older releases of GENCODE, the tag 'Ensemble_canonical' does not exist, and users may wish to reduce complexity in transcript annotations by filtering on transcript isoform suffix, for example, 201, 202, 203, etc. The -iso option supports regular expressions. For example, '[0-9]01' will match 101, 201, 301 up to 901. Or, if -iso is set to '201', only transcripts whose name ends in the isoform number 201 will be retained (e.g. 'Gm37180-201'). Please refer to GECODE for further details: https://www.gencodegenes.org/pages/data_format.html

`rm_dup (-rd, --rmdup)`: remove duplicated sequences, rm_dup=False is the default. If rm_dup=True, sequences that are exactly the same will be removed and only the first occurrence will be kept.

`outputname (-o, --outputname)`: the name of the output fasta file, default is 'test', a trailing part '.fa' will be added to ensure fasta format.

norm_vectors

Description:

Produce the mean and standard deviation vectors used to standardize *k*-mer counts in other SEEKR functions. Ideally generated from a large set of transcripts, such as the complete collection of lncRNAs annotated in a particular species. Normalization vectors are specific to *k*-mer length; separate sets of vectors must be generated for each *k*-mer length being studied.

Python Example:

Generate mean and standard deviation vectors from all lncRNA transcripts in the mouse GENCODE release vM25 (gencode.vM25.lncRNA.fa) at *k*-mer length *k* = 4 and save as bkg_mean_4mers.npy and bkg_std_4mers.npy.

seekrBasicCounter initializes the object and bkg_norm.get_counts() gets length-normalized *k*-mer counts for all input sequences then calculates the mean and std.dev for each *k*-mer word across all sequences, then stores these as bkg_norm.mean and bkg_norm.std, respectively.

```
from seekr.kmer_counts import BasicCounter as seekrBasicCounter

bkg_norm = seekrBasicCounter('vM25_lncRNA.fa',
                             k=4, log2='Log2.post',
                             binary=True, label=False, leave=True,
                             silent=True, alphabet='AGTC')

bkg_norm.get_counts()

mean_path = 'bkg_mean_4mers.npy'
std_path = 'bkg_std_4mers.npy'
np.save(mean_path, bkg_norm.mean)
np.save(std_path, bkg_norm.std)
```

Console Example:

Use all lncRNA transcripts in the mouse release M25 (gencode.vM25.lncRNA.fa) as the background sequence pool to produce standardization vectors at k-mer length *k* = 5 and save them as mean_5mers.npy and std_5mers.npy.

```
$ seekr_norm_vectors vM25_lncRNA.fa -k 5 -l Log2.post -mv
bkg_mean_5mers.npy -sv bkg_std_5mers.npy
```

To save output file in another location:

`-mv /Users/username/Desktop/bkg_mean_5mers.npy`

Input:

fasta (fasta, positional argument): path and name of the fasta file that would be used to generate standardization vectors.

k (-k, --kmer): the length of *k*-mer, default is 6.

log2 (-l, --log2): whether and when to log2 transform *k*-mer counts, possible options are: 'Log2.pre', 'Log2.none', and 'Log2.post', default is 'Log2.post'. Within norm_vectors, the 'Log2.none' and 'Log2.post' options are effectively the same, and result in no alterations to the length-normalized *k*-mer count data. The 'Log2.pre' option causes a count of ‘1’ to be added to all k-mers (after normalizing for lncRNA length) followed by a log2 transformation of counts.

mean_path (-mv, --mean_vector): path and name to save the output mean vector.

std_path (-sv, --std_vector): path and name to save the output standard deviation vector.

binary: set the format of the output normalization vectors. Saves as numpy array if True (default), else saves as csv. This function is specific to python and does not exist for command line.

label: whether to save row (transcript headers) and column (*k*-mer words) labels in the output files. This function is specific to python and does not exist for the command line.

leave: determines whether to keep or clear the progress bar upon completion of the iteration. Default is True, which keeps the tqdm progress bar visible after generating normalization vectors. Set to False if this function is used within another tqdm loop, which allows the tqdm progress bar to be cleared after each round inside the loop. This function is specific to python and does not exist for the command line.

silent: whether to show tqdm progress bar. Default is False. Set to True to turn off the progress bar. This function is specific to python and does not exist for the command line.

alphabet: set the valid letters to include in *k*-mer. Default is 'ATGC'. This function is specific to python and does not exist for the command line.

kmer_counts

Description:

Generates a *k*-mer count matrix of m rows by n columns, where m is the number of input transcripts in a fasta file and n is 4^*k*-mer. The *k*-mer count matrix can be standardized by the provided vectors (mean and standard deviation) and log2 transformed (or not) as specified by the user.

Python Example:

Generate a matrix of z-scores for each 4-mer for all sequences within test_lncRNA.fa. Use the vectors bkg_mean_4mers.npy and bkg_std_4mers.npy generated from all lncRNA transcripts in the mouse release M25 (gencode.vM25.lncRNA.fa) at *k*-mer length k = 4 (from norm_vectors), to standardize the counts. Log2 transform z-scores after *k*-mer count standardization. Save the final matrix as a .csv file with row and column labels. The matrix is also saved inside python as the variable: test_count. seekrBasicCounter initializes the object and test_count.make_count_file() performs all the actions. The output test_count.counts can be used for further analysis, such as calculating Pearson correlations.

```
from seekr.kmer_counts import BasicCounter as seekrBasicCounter
from seekr.pearson import pearson as seekrPearson

test_count = seekrBasicCounter(infasta='test_lncRNA.fa',
                               outfile='test_counts.csv',
                               k=4, binary=False, label=True,
                               mean='bkg_mean_4mers.npy',
                               std='bkg_std_4mers.npy',
                               log2='Log2.post', leave=True,
                               silent=True, alphabet='ACGT')

test_count. make_count_file()

sim = seekrPearson(test_count.counts,test_count.counts)
```

Console Example:

Generate a matrix of z-scores for each 4-mer for all sequences within test_lncRNA.fa. Use the vectors bkg_mean_4mers.npy and bkg_std_4mers.npy generated from all lncRNA transcripts in the mouse release M25 (gencode.vM25.lncRNA.fa) at *k*-mer length *k* = 4 (from norm_vectors), to standardize the counts. Log2 transform z-scores. Save the final z-score matrix as a binary .npy file without rows or column labels. The output .npy file can be used for further analysis such as calculating Pearson correlations.

```
$ seekr_kmer_counts test_lncRNA.fa -o test_counts.npy -k 4 -b -rl
-l Log2.post -mv bkg_mean_4mers.npy -sv bkg_std_4mers.npy -a ATGC
$ seekr_pearson test_counts.npy test_counts.npy -o
test_vs_self.csv
```

To produce *k*-mer counts per kb without standardization, set the --log2 Log2.none, --uncentered and --unstandardized flags in command line, and do not provide the mean and std normalization vectors. In this example, the nucleotide alphabet ATGC is specified by the -a flag:

```
$ seekr_kmer_counts test_lncRNA.fa -o test_counts.npy -k 4 -b -rl
-l Log2.none -uc -us -a ATGC
```

To save output as .csv file, omit -b and -rl argument and change output file suffix as well:

```
$ seekr_kmer_counts test_lncRNA.fa -o test_counts.csv -k 4 -l
Log2.none -uc -us -a ATGC
```

Input:

infasta (fasta, positional argument): path and name of the fasta file for *k*-mer counts / z-scores.

outfile (-o --outfile): path and name of the file to save the *k*-mer counts / z-scores. Make sure to name the file with the format (.csv, .npy or .txt) that is consistent with other arguments, such as binary (--binary) and labels (--remove_labels).

k (-k, --kmer): the length of *k*-mer, default is 6

binary (-b, --binary): set the format of the output normalization vectors. In python, saves as numpy array if True (default), else saves as csv. Using the command line, call -b in the argument to save as numpy array; otherwise, the default will save as a .csv.

label (-rl, --remove_labels): whether to save row and column labels in the output files. In python, labels are set to False by default. Using the command line, labels are saved by default, and removed by calling -rl.

mean, std: This is only for python. Optional path to mean and std vector numpy arrays. Possible values can be: True, False, a string as a path, or a python variable that is np.array. If path is provided, the files must be in .npy format. If mean is set to False, mean vector will not be subtracted from the *k*-mer count matrix; if std is set to False, std vector will not be divided from the *k*-mer count matrix.

(-mv, --mean_vector) (-sv, --std_vector) (-uc, --uncentered) (-us, --unstandardized): Options for command line only, corresponding to the arguments mean, std in python.

-mv and -sv are the optional paths to mean and std vector numpy files. None by default.

-uc, do not mean center *k*-mer counts.

-us, do not divide mean-centered counts by the standard deviation.

To standardize the *k*-mer counts matrix with previously calculated standardization vectors:

```
$ -mv bkg_mean_4mers.npy -sv bkg_std_4mers.npy
```

To bypass both standardization steps:

```
$ -uc -us
```

log2 (-l, --log2): whether and when to perform a log2 transformation. Options are: 'Log2.pre', 'Log2.none', and 'Log2.post'. Default is 'Log2.post'. Under 'Log2.pre', (1+ the length-normalized k-mer counts) are log2 transformed prior to z-score calculation. Under 'Log2.post' z-scores are made non-negative by adding 1 plus the absolute value of the minimum_z_score to the entire matrix, and then log2 transformed. In practice, Log2pre and Log2post should yield similar results but we have a slight preference for using Log2post especially when evaluating short sequences, because Log2post does not alter raw *k*-mer counts whereas Log2pre does. We favor log2 tranformation over non-transformation because *k*-mer count distributions in mammalian transcriptomes are usually best fit by lognormal distributions.

leave: determines whether to keep or clear the progress bar upon completion of the iteration. This function is specific to python and does not exist for command line.

silent: whether to show tqdm progress bar. Default is False, which means the tqdm progress bar would be shown. Set to True to turn off the progress bar. This function is specific to python and does not exist for command line.

alphabet (-a, --alphabet): set the valid letters to include in kmer. Default is 'ATGC'.

pearson

Description:

Generate a matrix of Pearson similarities from two *k*-mer count files.

Python Example:

Calculate the Pearson correlation coefficients of all possible pairs of sequences within the test_count matrix (derived from sequences in test_lncRNA.fa).

```
from seekr.kmer_counts import BasicCounter as seekrBasicCounter
from seekr.pearson import pearson as seekrPearson

test_count = seekrBasicCounter(infasta='test_lncRNA.fa',
                               outfile='test_counts.csv',
                               k=4, binary=False, label=True,
                               mean='bkg_mean_4mers.npy',
                               std='bkg_std_4mers.npy',
                               log2='Log2.post', leave=True,
                               silent=True, alphabet='ACGT')

test_count. make_count_file()

sim = seekrPearson(test_count.counts,test_count.counts,outfile=None)
```

Console Example:

Calculate the Pearson correlation coefficients of all possible pairs of *k*-mer count rows within the test_counts.npy matrix (counts derived from sequences in test_lncRNA.fa). Save the output as a .npy binary file.

```
$ seekr_pearson test_counts.npy test_counts.npy -o
test_vs_self.npy -bi -bo
```

Calculate the Pearson correlation coefficients of all possible pairs of *k*-mer count rows within the test_counts.csv matrix (counts derived from sequences in test_lncRNA.fa). Save the output as a .csv file.

```
$ seekr_pearson test_counts.csv test_counts.csv -o
test_vs_self.csv
```

Input:

counts1 (counts1, positional argument): Within python, counts1 is the .counts attribute of the BasicCounter object (i.e. test_count.counts). This is the matrix of *k*-mer counts. Dimensions are equal to # of transcripts by # of *k*-mers. The counts1 attribute cannot be a file or a path to a file. For the command line, counts1 refers to the path and name of the count file produced by seekr_kmer_counts. The format can be either .npy or .csv.

counts2 (counts2, positional argument): Within python, counts2 is the .counts attribute of the BasicCounter object (i.e. test_count.counts). This is the matrix of *k*-mer counts. Dimensions are equal to # of transcripts by # of *k*-mers. The counts2 attribute cannot be a file or a path to a file. It can be the same as the counts1 input. For the command line, counts2 refers to the path and name of the count file produced by kmer_counts. The format can be either .npy or .csv. counts2 can be identical to the counts1 file.

outfile (-o --outfile): path and name of the file to save the pearson correlation matrix. The output file can be either a binary .npy file or a .csv file. The .csv file will contain the row and column names (sequence names inherited from counts1 and counts2, if they existed). The .npy files lack row and column names.

(-bi, --binary_input) (-bo, --binary_output): indicate the type of input and output files. If -bi is called in the argument, it indicates the input counts files are of .npy format. If -bi is omitted in the argument, it indicates that the input counts files are of .csv format. The same applies to -bo. Please make sure these arguments are consistent with the input/output counts file format: if you input a .npy file, please make sure to also call -bi in the argument; if you input a .csv file, please make sure to omit -bi from the argument.

find_dist

Description:

Fit distributions to the set of Pearson's r values between pairwise comparisons in a set of sequences. End-users can also choose to use a distribution that is comprised of the actual Pearson's r values. Distributions are used to calculate p-values that estimate significance of *k*-mer similarity between sequences.

Python Example:

Calculate all possible pairwise Pearson correlations between all unique lncRNAs in the GENCODE mouse vM25 set, using a *k*-mer size of k = 4 and the 'Log2.post' option, then subset 100000 of the correlation values to fit the 'common10' distributions in scipy stats. Use the KS test to quantify goodness of fit. Show the progress bar for the model fitting step, do not plot the model fits against the actual data, and save the fitted models to test_fitres.csv under current directory.

```
from seekr import find_dist
fitres = find_dist.find_dist(inputseq='default', k_mer=4,
                             log2='Log2.post',models='common10',
                             subsetting=True,
                             subset_size = 10000,
                             fit_model=False, statsmethod='ks',
                             progress_bar=True,
                             plotfit=None,outputname='test_fitres')
```

Console Example:

Calculate all possible pairwise pearson correlations between all unique lncRNAs in the GENCODE mouse vM25 set, using a *k*-mer size of k = 4 and the 'Log2.post' option, then subset 100000 of the datapoints to fit the 'common10' distributions in scipy stats. Use the KS test to quantify goodness of fit. Show the progress bar for the model fitting step, plot the model fits against the actual data (modelfit.pdf), and save the fitted models to test_fitres.csv under current directory.

```
$ seekr_find_dist default -k 4 -l Log2.pre -mdl common10 -sbt -
sbs 100000 -fm -statm ks -pb -pf modelfit -o test_fitres
```

To not save the model fit plot, leave out -pf and its argument

To not save the model fit result, leave out -o and its argument

To specify defined models: -mdl 'norm,expon,pareto'

Minimal code with all settings to default, fitting the models and save the fitting results:

```
$ seekr_find_dist default -fm -o test_fitres
```

Minimal code with all settings to default, not fitting the models, directly output the background numpy array:

```
$ seekr_find_dist default -o test_fitres
```

Input:

inputseq (fasta, positional argument): path to the fasta file containing the sequences to model. Default (inputseq='default') uses GENCODE's mouse vM25 unique lncRNA sequences.

k_mer (-k, --kmer): *k*-mer size, default *k* = 4

log2 (-l, --log2): whether and how to use log2 transformation in z-score calculations: 'Log2.pre', 'Log2.none', and 'Log2.post'. Default is 'Log2.post'.

models (-mdl, --models): groups of candidate models for fitting, options are: 'all','common10' (default), or a list of distributions input by the end-user, for example ['norm','expon','pareto'] in python and -mdl 'norm,expon,pareto' in the console.

'all' - fit all available distributions from scipy stats, both rv_continuous and rv_discrete: https://docs.scipy.org/doc/scipy/reference/stats.html#continuous-distributions

'common10' - fit the 10 common distributions in scipy stats ('cauchy', 'chi2', 'expon', 'exponpow', 'gamma', 'lognorm', 'norm', 'pareto', 'rayleigh', 't')

common10 distributions are determined from get_common_distributions() from the Fitter library with 'powerlaw' replaced by 'pareto'.

subsetting (-sbt, --subsetting): True (default) or False. whether or not to use a subset of the pearson correlation values for fitting or output. For large datasets, subsetting is recommended to save time.

subset_size (-sbs, --subset_size): the size of the subset to use for fitting or output. Default is 100000.

fit_model (-fm, --fit_model): True (default) or False. whether or not to fit the data to specific distributions. Users can set fit_model=False, which will return the actual data as a npy array, without fitting to any distributions. In this way, p values can be calculated based on the actual data instead of any fitted distribution.

statsmethod (-statm, --statsmethod): the method used to quantify the goodness of fit of the distributions used vs data. Options are: 'ks' (default, Kolmogorov-Smirnov test), 'mse' (Mean Squared Error), 'aic' (Akaike Information Criterion), 'bic' (Bayesian Information Criterion). Smaller values returned by all four methods generally indicate a better fit.

KS: measures the largest difference between the observed cumulative distribution function (CDF) of the data and the expected CDF of the model. It is sensitive to differences in both the center and tails of the distribution and it provides an absolute measure of goodness-of-fit, which is intuitive and easy to understand. However, KS can be sensitive to the sample size, and may not have sufficient power to detect subtle differences in distributions when the sample size is small. Moreover, it does not penalize for model complexity, so overfitting could be a potential problem.

MSE: is an error-based metric that calculates the mean squared error between the actual data and the fitted data. It is intuitive and easy to understand but it is sensitive to outliers and does not take into account the number of parameters in the model.

AIC and BIC are likelihood-based metrics which have advantages over error-based metrics, in that they balance the goodness-of-fit against the complexity of the model; they also have broader applicability especially for models where the mean and variance are not well-defined (such as cauchy). AIC tends to favor models that fit the data very closely and can sometimes lead to selection of overly complex models. BIC incorporates a penalty term for the number of parameters in the model and the sample size, and therefore can result in the selection of simpler models compared to AIC. The output of AIC and BIC are unitless. They are most useful for comparing different models.

progress_bar (-pb, --progress_bar): True or False (default). whether or not to show the progress bar when fitting the data to the distributions.

plotfit (-pf, --plotfit): None (default, meaning no plot will be saved) or full path of file to save the plot of the fitted distributions (red dash line) vs the actual data (blue solid line) for all distributions in models. A trailing ‘.pdf’ will be added to the output file name to ensure pdf format

outputname (-o, --outputname): path of the output csv file, default is None, which means not save the output to a csv file. A trailing ‘.csv’ will be added to outputname.

find_pval

Description:

Calculate p values for a matrix of seekr_pearson correlation values, based on the output of find_dist, which is either a list of distributions or a npy array. The output of this function is a dataframe of p values, with the row names (input 1) and column names (input 2) being assigned to the same values as the headers of the two input sequence files.

Python Example:

Calculate p values for the seekr_pearson correlation values between the sequences contained within test1.fa and test2.fa, using a *k*-mer size of *k* = 4, the 'Log2.post' option, and the pre-calculated standardization vectors bkg_mean_4mers.npy and bkg_std_4mers.npy. Calculate p values using the background distribution that is listed first in fitres list (i.e. the output of find_dist). Show progress bar for the p value calculation step and do not save the output as a csv file.

```python
from seekr import find_pval
pvals=find_pval.find_pval(seq1file='test1.fa', seq2file='test2.fa',
                          mean_path='bkg_mean_4mers.npy',
                          std_path='bkg_std_4mers.npy',
                          k_mer=4, fitres=fitres,
                          log2='Log2.post', bestfit=1,
                          outputname=None, progress_bar=True)
```

Console Example:

Calculate p values for the seekr_pearson correlation between the sequences contained within test1.fa and test2.fa, using a *k*-mer size of *k* = 4, the 'Log2.post' option, and the pre-calculated standardization vectors bkg_mean_4mers.npy and bkg_std_4mers.npy. Calculate p values using the background distribution that is listed first in the fitres.file (i.e. the output of find_dist). Show the progress bar for the p value calculation and save the output to test_pval.csv under current directory.

```
$ seekr_find_pval seqs1.fa seqs2.fa mean_4mer.npy std_4mer.npy 4
fitres.csv -ft npy -bf 1 -o test_pval -pb
```

Minimal code with all settings to default

```
$ seekr_find_pval seqs1.fa seqs2.fa mean_4mer.npy std_4mer.npy 4
fitres.csv
```

Input:

seq1file (seq1file, positional argument): path to the fasta file of first input sequence(s).

seq2file (seq2file, positional argument): path to the fasta file of second input sequence(s).

seq1file and seq2file can be the same file. In this case, the p values are calculated using the seekr correlation values between each pair of sequences.

mean_path (mean_path, positional argument): the path to the mean vector file for standardization of *k*-mer counts.

std_path (std_path, positional argument): the path to the standard deviation vector file for standardization of *k*-mer counts.

k_mer (kmer, positional argument): the *k*-mer length to be used in SEEKR, which should be the same *k*-mer length used in the mean and std files.

fitres (fitres_file, positional argument): the output of find_dist, which is either a list of distributions or a npy array.

(-ft, --fitres_type): the type of fitres file, which is either one of ['distribution', 'npy']. This argument is specific to the command line (does not exist in python) to assist file import.

log2 (-l, --log2): whether and how to use log2 transformation in z-score calculations: 'Log2.pre', 'Log2.none', and 'Log2.post'. Default is 'Log2.post'.

bestfit (-bf, --bestfit): the index of the distribution selected for use from the list of distributions returned by find_dist. Index starts from 1, not 0. Default is 1.

outputname (-o, --outputname): full path of the output file, default is None, which means the output is not saved to a csv file.

progress_bar (-pb, --progress_bar): whether to show the progress bar during p value calculation. Default is True.

adj_pval

Description:

This function performs multiple comparison correction for p values calculated by find_pval, using the statsmodels.stats.multitest.multipletests function in python. The input is a dataframe of p values, where the row names (input 1) and column names (input 2) correspond to the headers of the two original input sequence files. The output is a dataframe of adjusted p values that has the same dimensions as the input dataframe. If the input dataframe is a symmetric matrix (i.e. the sequences in the rows and columns are the same) the number of multiple comparisons is automatically corrected to be half the matrix size, excluding the diagonal, and in the output dataframe, the upper triangle of the matrix (excluding the diagonal) is filled with the adjusted p values, while the lower triangle and the diagonal is filled with NaN. If the input dataframe is not a symmetric matrix, the total matrix is used for multiple comparison correction.

Python Example:

Use bonferroni correction with family-wise error rate at 0.05 to adjust the p values in pvals which is a dataframe and the direct output from the find_pval function. Save the output as test_adj_pval.csv.

```
from seekr import adj_pval
adjpvals=adj_pval.adj_pval(pvals, method='bonferroni',
                           alpha=0.05, outputname='test_adj_pval')
```

Console Example:

Use bonferroni correction with family-wise error rate at 0.05 to adjust the p values in test_pval.csv (the output of the find_pval function) and save the output as test_adj_pval.csv.

```
$ seekr_adj_pval test_pval.csv bonferroni -a 0.05 -o
test_adj_pval
```

Minimal code with all settings to default

```
$ seekr_adj_pval test_pval.csv bonferroni
```

Input:

pvals (pval_path, positional argument): a dataframe of p values (python) or the path to the p value .csv file, with the row names (input 1) and column names (input 2) representing the headers of the input sequences from find_pval.

method (method, positional argument): the method used for multiple comparison correction; can be 'bonferroni', 'sidak', 'holm-sidak', 'holm', 'simes-hochberg', 'hommel', 'fdr_bh', 'fdr_by', 'fdr_tsbh', 'fdr_tsbky'. Please refer to statsmodels.stats.multitest.multipletests for details.

alpha (-a, --alpha): the desired alpha level (family-wise error rate, FWER); default is 0.05. The FWER is the probability of making one or more false discoveries, or Type I errors, when performing multiple hypotheses tests. alpha represents the desired significance level for the entire set of tests, not just for an individual hypothesis test. For example, if you set alpha to 0.05, you are aiming to have no more than a 5% chance of making at least one Type I error across all the tests you are conducting.

outputname (-o, --outputname): full path of the output file, default is None, which means the output is not saved to a csv file. The .csv appendix will be automatically added to any file name to ensure csv format.

kmer_heatmap

Description:

A customizable heatmap function to enable visualization of the results of kmer_count, seekr_pearson, or find_pval. Can perform hierarchial clustering on both rows and columns.

Python Example:

Plot a heatmap for the df dataframe (which contains Pearson’s r values). Color the r value of -1 as #1b7837 (green) and the r value of 1 #c51b7d (purple). Encode a change of color at the r value of 0 (#ffffff, white). Draw dendrograms for columns and rows. Use correlation as a distance metric and complete as the linkage method. Save the plot as as kmer_heatmap.pdf in the current directory.

```
from seekr import kmer_heatmap
kmer_heatmap.kmer_heatmap(df, datamin=-1, datamax=1,
                          thresh_value=0.00,
                          color_range=['#1b7837', '#ffffff', '#c51b7d'],
                          cluster=True,
                          distmetric='correlation', linkmethod='complete',
                          hmapw_ratio=0.3, hmaph_ratio=0.3,
                          x_tick_size=6, y_tick_size=6,
                          cbar_font_size=16, outputname='kmer_heatmap',
                          hformat='pdf', hdpi=300)
```

Console Example:

Plot heatmap for the input pval.csv file. Color the p value of 0 as #1b7837 (green) and the p value 1 as #c51b7d (purple). Encode a change of color at 0.05 (#ffffff, white). Draw dendrograms for columns and rows. Use correlation as a distance metric and complete linkage. Save the plot as kmer_heatmap.pdf in /Users/username/Desktop/.

```
$ seekr_kmer_heatmap pval.csv 0 1 -th 0.05 -cr
'#1b7837,#ffffff,#c51b7d' -cl -distm correlation -linkm complete
-wratio 0.3 -hratio 0.3 -xts 16 -yts 16 -cfs 16 -o
/Users/username/Desktop/kmer_heatmap -hf pdf -hd 300
```

Minimal code with all settings to default

```
$ seekr_kmer_heatmap pval.csv 0 1
```

Input:

df (df_file, positional argument): a pandas dataframe with row and column names. df is usually the output of kmer_count (z-scores), seekr_pearson (r values) or find_pval (p values).

datamin (datamin, positional argument): the minimum possible value of the data. For r values, -1. For p values, 0.

datamax (datamax, positional argument): the maximum possible value of the data. For r values, 1. For p values, 1.

thresh_value (-th, --thresh_value): the threshold value. Default is 0.05. If using a 3-color palette, this corresponds to the middle color. thresh_value can be used to separate the data into two colors based on a threshold. For example, if using r values, thresh_value=0 will separate the data into positive and negative correlations. If using p values, thresh_value=0.05 will separate the data into significant and non-significant p values.

color_range (-cr, --color_range_str): color palette to use for the heatmap, has to be a list of 2 or 3 hex color strings. Default is ['#1b7837', '#ffffff', '#c51b7d'] in python, which is [green, white, magenta]. From the console, it is -cr '#1b7837,#ffffff,#c51b7d'. If using three colors, the middle color corresponds to the thresh_value. If using two colors, the thresh_value will be ignored. The first and last colors correspond to the datamin and datamax respectively.

cluster (-cl, --cluster): whether to perform hierarchical clustering on both rows and columns. Default is set to True in python. If cluster is set to False, only the heatmap will be plotted, and the row and column orders will be the same as those in the input dataframe. From the console, if you call -cl or --cluster in the argument, hierarchical clustering will be performed for both columns and rows. If -cl is omitted from the argument, only the heatmap will be plotted, and the row and column orders will be the same as those in the input dataframe.

Distmetric (-distm, --distmetric): the distance metric to use for the dendrogram. Default is ‘correlation’. Common distmetric options are: ‘euclidean’, ‘cityblock’, ‘correlation’, ‘cosine’, ‘jaccard’, ‘hamming’. See the documentation for scipy.spatial.distance.pdist for the full list of valid distmetrics.

Linkmethod (-linkm, --linkmethod): the linkage method to use for the dendrogram. Default is ‘complete’. Common linkmethod options are: ‘single’, ‘complete’, ‘average’, ‘weighted’, ‘centroid’, ‘median’, ‘ward’. Check the documentation for scipy.cluster.hierarchy.linkage for the full list of valid linkmethods.

Hmapw_ratio (-wratio, --hmapw_ratio): a ratio factor to control the heatmap width based on number of columns. Default is 0.3. hmapw_ratio has to be a positive number (>0). If wider heatmap is desired, increase the ratio. If narrower heatmap is desired, decrease the ratio.

Hmaph_ratio (-hratio, --hmaph_ratio): a ratio factor to control the heatmap height based on number of rows. Default is 0.3. hmaph_ratio has to be a positive number (>0). If taller heatmap is desired, increase the ratio. If shorter heatmap is desired, decrease the ratio.

X_tick_size (-xts, --x_tick_size): the font size for the x tick labels or the column labels. Default is 16.

Y_tick_size (-yts, --y_tick_size): the font size for the y tick labels or the row labels. Default is 16.

Cbar_font_size (-cfs, --cbar_font_size): the font size for the colorbar ticks. Default is 16.

Outputname (-o, --outputname): the path and name to save the output heatmap.

Hformat (-hf, --hformat): the format of the output heatmap. Default is ‘pdf’. Other options are: ‘eps’, ‘jpeg’, ‘jpg’, ‘pgf’, ‘png’, ‘ps’, ‘raw’, ‘rgba’, ‘svg’, ‘svgz’, ‘tif’, ‘tiff’, ‘webp’

hdpi (-hd, --hdpi): the dpi of the output heatmap. Default is 300.

kmer_dendrogram

Description:

Function to draw dendrograms to visualize hierarchical clustering results of seekr.pearson or find_pval.

Python Example:

Plot a dendrogram for the rows of the input dataframe df, using the distance metric of correlation and the complete linkage method, and save the plot as kmer_dendrogram.pdf in the current directory.

```
from seekr import kmer_dendrogram
kmer_dendrogram.kmer_dendrogram(df, dendro_direct='row',
                                distmetric='correlation',
                                linkmethod='complete',
                                plot_ht=8, wd_ratio=0.5,
                                leaf_font_size = 16,
                                outputname='kmer_dendrogram',
                                pformat='pdf', pdpi=300)
```

Console Example:

Plot a dendrogram for the columns of the input pval.csv file, using the distance metric of correlation and the complete linkage method, and save the plot as kmer_dendrogram.pdf in the current directory.

```
$ seekr_kmer_dendrogram pval.csv -dd column -distm correlation -
linkm complete -ph 8 -wratio 0.5 -lfs 16 -o kmer_dendrogram -pf
pdf -d 300
```

Minimal code with all settings to default

```
$ seekr_kmer_dendrogram pval.csv
```

Input:

df (df_file, positional argument): a pandas dataframe with row and column names. df can be either the output of seekr.pearson (r values) or find_pval (p values).

dendro_direct (-dd, --dendro_direct): the direction to perform hierarchical clustering. Can only be either 'row' (default) or 'column'.

distmetric (-distm, --distmetric): the distance metric to use for the dendrogram. Default is 'correlation'. Common distmetric options are: 'euclidean', 'cityblock', 'correlation', 'cosine', 'jaccard', 'hamming'. Check the documentation for scipy.spatial.distance.pdist for the full list of valid distmetrics.

linkmethod (-linkm, --linkmethod): the linkage method to use for the dendrogram. Default is 'complete'. Common linkmethod options are: 'single', 'complete', 'average', 'weighted', 'centroid', 'median', 'ward'. Check the documentation for scipy.cluster.hierarchy.linkage for the full list of valid linkmethods.

plot_ht (-ph, --plot_ht): height of the dendrogram plot. must be a positive number. Default is 8.

wd_ratio (-wratio, --wd_ratio): a ratio factor to control the dendrgram width based on number of leaves. Default is 0.5. wd_ratio has to be a positive number (>0). if wider dendrogram is desired, increase the ratio. If narrower dendrogram is desired, decrease the ratio.

leaf_font_size (-lfs, --leaf_font_size): the font size of the leaves, which corresponds to either the row or column names of the input dataframe. Default is 16.

outputname (-o, --outputname): the name and path to save the output dendrogram.

pformat (-pf, --pformat): the format of the output dendrogram. Default is 'pdf'. Other options are: 'eps', 'jpeg', 'jpg', 'pgf', 'png', 'ps', 'raw', 'rgba', 'svg', 'svgz', 'tif', 'tiff', 'webp'.

pdpi (-d, --pdpi): the dpi of the output dendrogram. Default is 300.

kmer_leiden

Description:

Create a Leiden community network that categorizes relationships between the *k*-mer similarities in a set of input fasta sequences. This function can also plot the resulting Leiden community network, but for better visualization, end-users will wish to suppress the plotting option and import the community.csv file into Gephi (not plot the network using Python).

Python Example:

Perform seekr pearson correlation on testld.fa using a *k*-mer size of *k* = 4 and the standardization vectors bkg_mean_4mer.npy and bkg_std_4mer.npy. Correlation r values less than 0.1 will be set to 0. Perform Leiden community detection on the correlation matrix using the algorithm RBERVertexPartition and resolution of 1.0. Set the seed to have reproducible community assignment results. The plot edge will be gradient colored based on the pearson correlation values. The plot will be saved as kmer_leiden.pdf under current directory. The corresponding nodes and edges files will not be saved

```python
from seekr import kmer_leiden
kmer_leiden.kmer_leiden(inputfile='testld.fa',
                        mean='bkg_unique_mean_4mers.npy',
                        std='bkg_unique_std_4mers.npy',
                        k=4, algo='RBERVertexPartition',
                        rs=1.0,pearsoncutoff=0, setseed=True,
                        edgecolormethod='gradient',
                        edgethreshold=0.1, labelfontsize=15,
                        plotname='kmer_leiden', csvfile=None)
```

Console Example:

Perform seekr pearson correlation on ldseq.fa using a *k*-mer size of *k* = 4 and the standardization vectors mean_4mer.npy and std_4mer.npy. Correlation r values less than 0 will be set to 0. Perform Leiden community detection on the correlation matrix using the algorithm RBERVertexPartition and the resolution of 1.0. Set the seed to have reproducible community assignment results. Set the edge threshold to 0.1; edges with correlation values greater than 0.1 will be colored black, otherwise they will be colored grey. Save the plot as kmer_leiden.pdf in the current directory and save the corresponding nodes and edge files as testdata_nodes_leiden.csv and testdata_edges_leiden.csv also in the current directory.

```
$ seekr_kmer_leiden ldseq.fa mean_4mer.npy std_4mer.npy 4 -a
RBERVertexPartition -r 1.0 -pco 0 -sd -ec threshold -et 0.1 -lfs
12 -pn kmer_leiden -cf testdata
```

To suppress plotting, leave out -pn and its argument

To suppress saving nodes and edges files, leave out -cf and its argument

To turn set seed off to test robustness of community assignments, remove -sd from the command

Minimal code with all settings to default

```
$ seekr_kmer_leiden ldseq.fa mean_4mer.npy std_4mer.npy 4 -cf
testdata
```

Input:

inputfile (fasta, positional argument): path to the input fasta file; ensure all input sequences have unique headers, which will be used for node labels.

mean (mean_path, positional argument): path to the mean vector file for k-mer count standardization. Typically calculated using the seekr_norm_vector function.

std (std_path, positional argument): path to the standard deviation vector file for k-mer count standardization. Typically calculated using the seekr_norm_vector function.

k (kmer, positional argument): the *k*-mer length to use in SEEKR; should be consistent with the *k*-mer length used to generate the mean and std dev. files.

algo (-a, --algo): the algorithm used for Leiden community detection. Default is 'RBERVertexPartition'. Other options are: 'ModularityVertexPartition', 'RBConfigurationVertexPartition','RBERVertexPartition', 'CPMVertexPartition', 'SurpriseVertexPartition','SignificanceVertexPartition'.

rs (-r, --rs): the resolution parameter used in Leiden algorithm, default is 1.0. Larger rs values lead to more clusters and smaller values lead to less clusters. Testing ranges from .1 to 5 is often reasonable. In our experience, values from 1-3 have been most useful.

pearsoncutoff (-poc, --pearsoncutoff): set seekr pearson r value (from seekr.pearson) to 0 if it is less than the pearsoncutoff. pearsoncutoff enables removal of weak correlations when constructing Leiden communities from a large number of nodes. pearsoncutoff default is 0, which will remove all negative correlations.

setseed (-sd, --setseed): if set to True, set the initial value of the random-number generator. Enables community detection results to be reproduced from run to run.

edgecolormethod (-ec, --edgecolormethod): the method used to set the color of edges that connect communities. Default is 'gradient', which is a gradient scale of grey for edges based on their weights (pearson correlation); darker edges represent higher weights. The other option available option is 'threshold', which will color grey those edges whose weights are smaller than the threshold, and color black those edges whose weights are larger than threshold.

edgethreshold (-et, --edgethreshold): the threshold used in edgecolormethod 'threshold', default is 0.1.

labelfontsize (-lfs, --labelfontsize): the font size of the node label, default is 12.

plotname (-pn, --plotname): the name and path of your output plot, default is None, which suppress the plotting.

plotname example: if given a string, such as 'test_kmer_leiden', the suffix .pdf will be added automatically.

csvfile (-cf, --csvfile): the name and path of the nodes and edges .csv files. Default is None, which suppress saving these files. The edges and nodes csv files are readily importable to Gephi for better network visualization options.

kmer_count_barplot

Description:

Function that enables users to create barplots of z-scores showing the relative abundance for select *k*-mers in a select set of sequences.

Python Example:

Perform BasicCounter to count *k*-mers in the sequences from test.fa file, using a *k*-mer size of 4 and the standardization vectors mean_4mer.npy and std_4mer.npy. Log2 transform z scores and arrange *k*-mers by the summed difference from the mean among all sequences in ascending order; *k*-mers with the smallest summed difference from the mean among all sequences will be plotted first. Plot the 10 *k*-mers with the smallest summed differences in barplot.pdf in the current directory.

```
from seekr import kmer_count_barplot
kmer_count_barplot.kmer_count_barplot(inputfile='test.fa',
                                      mean='mean_4mers.npy',
                                      std='std_4mers.npy',
                                      log2='Log2.post',
                                      k=4,sortmethod='ascending',
                                      topkmernumber=10,
                                      xlabelsize=20,ylabelsize=20,
                                      xticksize=20,yticksize=20,
                                      legendsize=12,outputname='barplot',
                                      pformat='pdf',pdpi=300)
```

Console Example:

Perform BasicCounter to count *k*-mers in the sequences from test.fa file, using a *k*-mer size of 4 and the standardization vectors mean_4mer.npy and std_4mer.npy. Log2 transform z scores and arrange *k*-mers by the summed difference from the mean among all sequences in ascending order; *k*-mers with the smallest summed difference from the mean among all sequences will be plotted first. Plot the 10 *k*-mers with the smallest summed differences in barplot.pdf in the current directory.

```
$ seekr_kmer_count_barplot test.fa mean_4mer.npy std_4mer.npy 4 -
l Log2.post -sm ascending -tn 10 -xls 20 -yls 20 -xts 20 -yts 20
-ls 12 -o kmer_count_barplot -pf pdf -d 300
```

Minimal code with all settings to default

```
$ seekr_kmer_count_barplot test.fa mean_4mer.npy std_4mer.npy 4
```

Input:

inputfile (fasta, positional argument): path to a fasta file with unique headers for each sequence. Limit is 10 sequences.

mean (mean_path, positional argument): path to the mean vector file for *k*-mer count standardization. Typically calculated using the seekr_norm_vector function.

std (std_path, positional argument): path to the standard deviation vector file for *k*-mer count standardization. Typically calculated using the seekr_norm_vector function.

k (kmer, positional argument): the *k*-mer length to use in SEEKR; should be consistent with the *k*-mer length used to generate the mean and std dev. files.

log2 (-l, --log2): whether and how to use log2 transformation in z-score calculations: 'Log2.pre', 'Log2.none', and 'Log2.post'. Default is 'Log2.post'.

sortmethod (-sm, --sortmethod): how to sort *k*-mers based on summed difference from the mean among all sequences. Options: 'ascending' (default), 'descending'.

topkmernumber (-tn, --topkmernumber): the number of *k*-mers for which to plot summed differences, default is 10.

xlabelsize (-xls, --xlabelsize): x axis label font size, default is 20.

ylabelsize (-yls, --ylabelsize): y axis label font size, default is 20.

xticksize (-xts, --xticksize): x tick label font size, default is 20.

yticksize (-yts, --yticksize): y tick label font size, default is 20.

legendsize (-ls, --legendsize): legend font size, default is 12.

outputname (-o, --outputname): the path and name to save the output file, default is current working directory.

pformat (-pf, --pformat): the format of the output file, default is 'pdf'. Other options are: 'eps', 'jpeg', 'jpg', 'pgf', 'png', 'ps', 'raw', 'rgba', 'svg', 'svgz', 'tif', 'tiff', 'webp'.

pdpi (-d, --pdpi): the dpi of the output file, default is 300.

kmer_msd_barplot

Description:

Function to make barplots of mean and standard deviations of z-scores for select *k*-mers words across a set of input sequences.

Python Example:

Perform BasicCounter to count *k*-mers in the sequences from test.fa file, using a *k*-mer size of 4 and the standardization vectors mean_4mer.npy and std_4mer.npy. Log2 transform z scores and arrange *k*-mers by the standard deviation of their transformed z scores in ascending order; the k-mers with the smallest standard deviations will be plotted first. Plot the 10 least deviant *k*-mers as_msd_barplot.pdf in the current directory.

```
from seekr import kmer_msd_barplot
kmer_msd_barplot.kmer_msd_barplot(inputfile='test.fa',
                                  mean='mean_4mers.npy',
                                  std='std_4mers.npy',
                                  log2 = 'Log2.post',
                                  k=4, sortstat='sd',
                                  sortmethod='ascending',
                                  topkmernumber=10,
                                  xlabelsize=20, ylabelsize=20,
                                  xticksize=20, yticksize=20,
                                  outputname='kmer_msd_barplot',
                                  pformat='pdf', pdpi=300)
```

Console Example:

Perform BasicCounter to count *k*-mers in the sequences from test.fa file, using a *k*-mer size of 4 and the standardization vectors mean_4mer.npy and std_4mer.npy. Log2 transform z scores and arrange *k*-mers by the standard deviation of their transformed z scores in ascending order; *k*-mers with the smallest standard deviations will be plotted first. Plot the 10 least deviant *k*-mers as_msd_barplot.pdf in the current directory.

```
$ seekr_kmer_msd_barplot test.fa mean_4mer.npy std_4mer.npy 4 -l
Log2.post -ss mean -sm descending -tn 10 -xls 20 -yls 20 -xts 20
-yts 20 -o kmer_msd_barplot -pf pdf -d 300
```

Minimal code with all settings to default

```
$ seekr_kmer_msd_barplot test.fa mean_4mer.npy std_4mer.npy 4
```

Input:

inputfile (fasta, positional argument): path to fasta file with unique headers for each sequence.

mean (mean_path, positional argument): path to the mean vector file for k-mer count standardization. Typically calculated using the seekr_norm_vector function.

std (std_path, positional argument): path to the standard deviation vector file for k-mer count standardization. Typically calculated using the seekr_norm_vector function.

k (kmer, positional argument): the *k*-mer length to use in SEEKR; should be consistent with the *k*-mer length used to generate the mean and std dev. files.

log2 (-l, --log2): whether and how to use log2 transformation in z-score calculations: 'Log2.pre', 'Log2.none', and 'Log2.post'. Default is 'Log2.post'.

sortstat (-ss, --sortstat): whether to sort *k*-mers based on mean or standard deviation, options: 'mean' (default) and 'sd'.

sortmethod (-sm, --sortmethod): how to sort *k*-mers based , possible options: 'ascending' and 'descending' (default)

topkmernumber (-tn, --topkmernumber): the number of *k*-mers for which to plot summed differences, default is 10.

xlabelsize (-xls, --xlabelsize): x axis label font size, default is 20

ylabelsize (-yls, --ylabelsize): y axis label font size, default is 20

xticksize (-xts, --xticksize): x tick label font size, default is 20

yticksize (-yts, --yticksize): y tick label font size, default is 20

outputname (-o, --outputname): the name and path to save the output file, default is the current working directory and the file named 'test_kmer_msd_barplot.pdf'.

pformat (-pf, --pformat): the format of the output file, default is 'pdf'. Other options are: 'eps', 'jpeg', 'jpg', 'pgf', 'png', 'ps', 'raw', 'rgba', 'svg', 'svgz', 'tif', 'tiff', 'webp'

pdpi (-d, --pdpi): the dpi of the output file, default is 300.

kmer_comp_textplot

Description:

This function prints to a single file two sequences aligned by nucleotide position and highlights the location of a list of up to 10 user-specified *k*-mers within them. Overlapping *k*-mers will be highlighted with a priority given to the earlier *k*-mer in the list.

Python Example:

Plot and align the two input sequences. Highlight the *k*-mers of interest at all matched locations along the two input sequences using the default ‘tab10’ colors.

The *k*-mers of interest are 'ATCG' in red, 'GTAG' in pink, 'AAAA' in orange, and ‘GCGC’ in olive. Use a wrap length of 60 characters and save the textplot as comp_textplot.pdf in the current directory.

```
from seekr import kmer_comp_textplot
kmer_comp_textplot.kmer_comp_textplot(seq1file='test.fa',seq2file='test2.fa',
                                      words=['ATCG','GTAG','AAAA','GCGC'],
                                      color_vec='default', wraplen=60,
                                      char_spacing=1.0,line_spacing=0.5,
                                      seqfontsize=28,numfontsize=18,
                                      colorblockh=0.5,
                                      outputname='comp_textplot',
                                      plotformat='pdf', plotdpi=300)
```

Console Example:

Plot and align the two input sequences. Highlight the *k*-mers of interest in corresponding colors at all matched locations along the two sequences. The words of interest are 'ATTA' in '#d62728', 'AAAA' in '#e377c2', and 'ACTC' in '#ff7f0e'. Use a wrap length of 60 characters and save the textplot as comp_textplot.pdf in the current directory.

```
$ seekr_kmer_comp_textplot seq1.fa seq2.fa 'ATTA,AAAA,ACTC' -cv
'#d62728,#e377c2,#ff7f0e' -wl 60 -cs 1.0 -ls 0.5 -sfs 28 -nfs 18
-cbh 0.5 -o comp_textplot -pf pdf -d 300
```

Minimal code with all settings to default

```
$ seekr_kmer_comp_textplot seq1.fa seq2.fa 'ATTA,AAAA,ACTC'
```

Input:

seq1file, seq2file (seq1file, seq2file, positional arguments): input sequence 1 and 2 in fasta format. If seq1 and seq2 files include more than one sequence, only the first sequence will be plotted. The sequence from seq1file will be plotted in black while the sequence from seq2file will be plotted in grey.

words (words_str, positional argument): *k*-mers of interest, max 10 in the format of a list, python example: ['ATTA','AAAA','ACTC','CCTT','GGCC']; console example: 'ATTA,AAAA,ACTC'. If the list contains more than 10 *k*-mers, only the first 10 will be plotted.

color_vec (-cv, --color_vec_str): ‘default’ or a list of hex color for the words of interest (for example: python: ['#d62728','#e377c2','#ff7f0e'], console: '#d62728,#e377c2,#ff7f0e'). Default uses the 'tab10' pallette with rearrangements into a quasi-rainbow order. If using a custom, color vector, please ensure the length of the color vector is the same as the length of the *k*-mer list.

wraplen (-wl, --wraplen): wrapping length, how many nucleotides to wrap in each line, default is 60.

char_spacing (-cs, --char_spacing): space between characters in the plot, default is 1.0.

line_spacing (-ls, --line_spacing): line space between seq1, seq2 and number, default is 0.5.

seqfontsize (-sfs, --seqfontsize): sequence character font size, default is 28.

numfontsize (-nfs, --numberfontsize): sequence position number font size, default is 18.

colorblockh (-cbh, --colorblockh): the height of the highlight color block, default is 0.5, change this when changing seqfontsize.

outpoutname (-o, --outputname): the name and path of the output file, default is to plot in the current directory using the file name 'comp _textplot'.

plotformat (-pf, --plotformat): output format, default is 'pdf'. Other common options are 'png', 'jpg', 'svg', 'eps', 'tif', 'tiff', 'ps', 'webp'. Other options can be found here: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html

plotdpi (-d, --plotdpi): resolution of the output file, default is 300.

kmer_indi_textplot

Description:

This function highlights the location of a list of up to 10 user-specified *k*-mers within each sequence in an input file, and prints each sequence to a separate location. Overlapping *k*-mers will be highlighted with a priority given to the earlier *k*-mers in the list.

Python Example:

Highlight the *k*-mers of interest in default rearranged ‘tab10’ colors at all matched locations across all the input sequences. The *k*-mers of interest are 'ATCG' in red, 'GTAG' in pink, 'AAAA' in orange, and ‘GCGC’ in olive. Wrap length is 60 characters. For each input sequence, a textplot is saved with part of its fasta header as outputname in .pdf format under current directory.

```
from seekr import kmer_indi_textplot
kmer_indi_textplot.kmer_indi_textplot(seqfile='test.fa',
                                       words=['ATCG','GTAG','AAAA','GCGC'],
                                       color_vec='default', wraplen=60,
                                       char_spacing=1.0,line_spacing=0.5,
                                       seqfontsize=28,numfontsize=18,
                                       colorblockh=0.5,outputpath='',
                                       plotformat='pdf', plotdpi=300)
```

Console Example:

Highlight the *k*-mers of interest in default rearranged ‘tab10’ colors at all matched locations across all the input sequences. The *k*-mers of interest are 'ATCG' in red, 'GTAG' in pink, 'AAAA' in orange, and ‘GCGC’ in olive. Wrap length is 60 characters. For each input sequence, a textplot is saved with part of its fasta header as outputname in .pdf format under /Users/username/Desktop/textplot/.

```
$ seekr_kmer_indi_textplot seq.fa 'ATTA,AAAA,ACTC' -cv
'#d62728,#e377c2,#ff7f0e' -wl 60 -cs 1.0 -ls 0.5 -sfs 28 -nfs 18
-cbh 0.5 -op /Users/username/Desktop/textplot/ -pf pdf -d 300
```

To save under current directory, omit the -op argument.

Minimal code with all settings to default

```
$ seekr_kmer_textplot seq.fa 'ATTA,AAAA,ACTC'
```

Input:

seqfile (seqfile, positional arguments): input sequences in fasta format. Can include more than one sequence, and each sequence will be plotted in a separate file. The section of the fasta header after the ‘>’ and before the first ‘|’ character (relevant for GENCODE sequences), or the whole header after > if there is no ‘|’ character, will be used as the name of the plot for each sequence. Ensure that the fasta header of the seqfile contains a unique ID for each sequence before the first ‘|’ character.

words (words_str, positional argument): *k*-mers of interest, max 10 in the format of a list, python example: ['ATTA','AAAA','ACTC','CCTT','GGCC']; console example: 'ATTA,AAAA,ACTC'. If the list contains more than 10 *k*-mers, only the first 10 will be plotted.

color_vec (-cv, --color_vec_str): ‘default’ or a list of hex color for the *k*-mers of interest (for example: python: ['#d62728','#e377c2','#ff7f0e'], console: '#d62728,#e377c2,#ff7f0e'). Default uses the 'tab10' pallette with rearrangements into a quasi-rainbow order.

wraplen (-wl, --wraplen): wrapping length, how many nucleotides to wrap in each line, default is 60.

char_spacing (-cs, --char_spacing): space between characters in the plot, default is 1.0.

line_spacing (-ls, --line_spacing): line space between sequence and number, default is 0.5.

seqfontsize (-sfs, --seqfontsize): sequence character font size, default is 28.

numfontsize (-nfs, --numberfontsize): sequence position number font size, default is 18.

colorblockh (-cbh, --colorblockh): the height of the highlight color block, default is 0.5.

outputpath (-op, --outputpath): the path for the output files. The name of each plot will be automatically generated based on the fasta header of each sequence.

plotformat (-pf, --plotformat): output format, default is 'pdf'. Other common options are 'png', 'jpg', 'svg', 'eps', 'tif', 'tiff', 'ps', 'webp'. Other options can be found here: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.savefig.html

or by running this code: matplotlib.pyplot.gcf().canvas.get_supported_filetypes()

plotdpi (-d, --plotdpi): resolution of the output file, default is 300.

Seekr Docker Image

Docker Installation

Install Docker on your computer. Please refer to the official website for installation instructions.

https://www.docker.com/get-started/

Then start the application and ensure it is running properly – you should see the docker icon on your task bar with the status indicated.

Pull Docker Image and Test Run

1. Start your command line tool, Terminal for MacOS and CMD for Windows. You can also use Powershell or Cygwin for Windows, but Cygwin might have interaction issues.

2. From the command line, pull the Docker Image:

```
docker pull calabreselab/seekr:latest
```

You can replace `latest` with a specific tag if needed.

3. Test Run the Docker Image:

```
docker run -it --rm calabreselab/seekr:latest
```

The -it tag sets it to interactive mode. If you don't need to run the Docker container in the interactive mode (i.e., you don't need a shell inside the container), you can simply omit the -it flag.

This will print the user manual out to the command line, which is basically the same that happens when the command 'seekr' is typed directly into the command line.

Run Docker Image from command line

You can run the seekr function from this Docker Image directly from command line with the following specified syntax.

```
docker run -v /path/to/your/files:/data calabreselab/seekr:latest
seekr_kmer_comp_textplot /data/seq1.fa /data/seq2.fa
'ATTA,AAAA,ACTC' -o /data/comp_textplot
```

In this command:

-v /path/to/your/files:/data: This mounts the directory /path/to/your/files from your host machine (where seq1.fa and seq2.fa are located) to the /data directory inside the Docker container. Replace /path/to/your/files with the actual path to your files.

seekr_kmer_comp_textplot /data/seq1.fa /data/seq2.fa 'ATTA,AAAA,ACTC' -o /data/test: This is the command that gets executed inside the Docker container.

Since we mounted our files to /data in the container, we reference them with /data/seq1.fa and /data/seq2.fa.

The /data folder is basically a mirror of the folder you specified in /path/to/your/files. So by specifying -o /data/comp_textplot (output with the path /data/ and plotname as in comp_textplot.pdf) we can have the output files directly written in to the folder in /path/to/your/files.

Please remember to specify your output path to /data/ otherwise it will not be saved to your folder on local machine and it would be hard to locate it even inside the Docker Container Filesystem (in this case, when the Docker Container is removed, your data will be deleted as well).

Examples of code mounts e:/test on Windows as the folder that contains the input and holds the output files:

```
docker run -v e:/test:/data calabreselab/seekr:latest
seekr_kmer_comp_textplot /data/test1.fa /data/test2.fa
'ATTA,AAAA,ACTC' -o /data/comp_textplot

docker run -v e:/test:/data calabreselab/seekr:latest
seekr_kmer_leiden /data/testld.fa /data/bkg_mean_4mers.npy
/data/bkg_std_4mers.npy 4 -s -o /data/kmer_leiden
```

Here, you need to add: docker run -v /path/to/your/files:/data calabreselab/seekr:latest before the command line code for seekr (see above for examples of all functions).

Run Docker Image with Jupyter Notebook

If you want to work with python code instead of directly calling from the command line, you can choose to run seekr using a Jupyter Notebook inside the Docker Container.

First you need to run Docker Container with Interactive Terminal:

```
docker run -it -p 8888:8888 -v /path/on/host:/path/in/container
calabreselab/seekr:latest /bin/bash
```

This command will start the container and give you a bash terminal inside it. The `-p 8888:8888` flag maps the port `8888` from the container to your host machine so that you can access the Jupyter Notebook server.

/path/on/host is the path to a directory on your local machine where you want the notebooks and other data to be saved.

/path/in/container is the directory inside the container where you want to access and save the files from Jupyter Notebook.

When you use Jupyter Notebook to create or save notebooks, they will be stored at /path/in/container inside the Docker container. Due to the volume mount, these files will also be accessible and stored at /path/on/host on your host machine so that you can later access the code and files even when the container is removed. Example of code:

```
docker run -it -p 8888:8888 -v e:/test:/data
calabreselab/seekr:latest /bin/bash
```

From here you will need to manually start Jupyter Notebook. From the bash terminal inside the Docker container:

```
jupyter notebook --ip=0.0.0.0 --port=8888 --NotebookApp.token=''
--NotebookApp.password='' --allow-root
```

The flags are explained as follows:

--ip=0.0.0.0: Allow connections from any IP address. This is important for accessing Jupyter from outside the container.

--port=8888: Run Jupyter on this port. You can change this if needed, but remember to adjust the `-p` flag when starting the Docker container.

--allow-root: Allow Jupyter to be run as the root user inside the Docker container.

--NotebookApp.token='': This disables the token-based security measure.

--NotebookApp.password='': This ensures that there's no password required to access the Jupyter server.

Disabling the token and password for Jupyter Notebook reduces security. It's generally okay for local development, but you should avoid doing this in production or any publicly accessible server.

Finally, you can access the Jupyter Notebook from your host machine's browser by entering this address:

http://localhost:8888/

You can run the python functions as demonstrated above. It would be convenient if all input files can be copied over to the folder you have mounted: /path/on/host. Pay attention that when you specify your input and output file route, you use the /path/in/container route, which is a mirror to your local folder. Example of code:

```
from seekr import kmer_comp_textplot
kmer_comp_textplot.kmer_comp_textplot(seq1file='/data/test1.fa',
                                      seq2file='/data/test2.fa',
                                      words=['ATCG','GTAG','AAAA','GCGC'],
                                      color_vec='default', wraplen=60,
                                      char_spacing=1.0, line_spacing=0.5,
                                      seqfontsize=28,numfontsize=18,
                                      colorblockh=0.5,
                                      outputname='/data/comp_textplot',
                                      plotformat='pdf', plotdpi=300)
```

Once you are done, you can click the shut-down button inside Jupyter Notebook to shut down the instance or you can just click Ctrl+C twice from the command line to kill the process. Then you will need to exit the Docker Container from the interactive session by typing 'exit' and enter.

Cleanup (Optional)

To Clean Up Docker Container:

- List all containers, including the stopped ones: `docker ps -a`
- To remove a specific container: `docker rm CONTAINER_ID_OR_NAME`
- To remove all stopped containers: `docker container prune`

Clean Up Docker Image (Optional):

If you want to remove the Docker image you used:

List all images: `docker images`

Remove a specific image: `docker rmi IMAGE_ID_OR_NAME`

Note: You'd typically only remove an image if you're sure you won't be using it again soon, or if you want to fetch a fresh version of it from a repository like Docker Hub.

Additional Cleanup (Optional):

Docker also maintains a cache of intermediate images and volumes. Over time, these can accumulate. To free up space:

Remove unused data: `docker system prune`

To also remove unused volumes (be careful, as this might remove data you want to keep): `docker system prune --volumes`

Remember to be cautious when cleaning up, especially with commands that remove data. Make sure you have backups of any essential data, and always double-check what you specify for deletion.