

Documentation of Tools for Noise Removal from Pyrosequenced Amplicons (AmpliconNoiseV1.23)

Installation:

The programs have been tested on MacOs X and Linux – Windows is not supported. A cluster is not necessary but reasonable size data sets will only run on a cluster or good server. A version of Message Passing Interface (MPI) is necessary to install the programs. Open MPI is a good choice:

<http://www.open-mpi.org/>

In addition the chimera checker Perseus requires that both mafft:

<http://mafft.cbrc.jp/alignment/software/>

and the Gnu Science Library are installed:

<http://www.gnu.org/software/gsl/>

To install first unzip the programs:

unzip AmpliconNoiseV1.23.zip

move into the top directory and type:

make clean

make

this will compile the programs. Any errors here may require changing the default C (cc) and C-MPI compilers (mpicc) in the individual makefiles associated with the executables.

make install

will place the executables in the bin directories. This and the Scripts directory need to be added to your path. If you unzip AmpliconNoiseV1.23.zip in your home directory /home/whoeverIam/ then this command should be added to your .bashrc or .profile or equivalent:

**PATH=/home/whoeverIam/AmpliconNoiseV1.23/bin:
/home/whoeverIam/AmpliconNoiseV1.23/Scripts:\$PATH**

export PATH

You should also set environment variables to specify the location of look up tables used by the programs. These define the noise distributions. The

following commands ensure that the file LookUp_E123.dat is always used for PyroDist and PyroNoise and Tran.dat by SeqDist and SeqNoise. Having set these the programs can be run anywhere otherwise they can only be run from inside the bin directory:

```
PYRO_LOOKUP_FILE=/home/whoeverIam/AmpliconNoiseV1.23/Data/LookUp_E123.dat
SEQ_LOOKUP_FILE=/home/whoeverIam/AmpliconNoiseV1.23/Data/Tran.dat
export PYRO_LOOKUP_FILE
export SEQ_LOOKUP_FILE
```

Programs

FCluster:

-in string distance input file name
-out string output file stub

Options:

-r resolution
-a average linkage
-w use weights
-i read identifiers
-s scale dist.

This performs a simple hierarchical clustering. It reads a distance file in text format (**-in**).

The first line in the text file gives the number of entities to be clustered N. This is then optionally followed by N ids if the (**-i**) flag is set as separate lines. Otherwise the $N(N-1)/2$ pairwise distances follow as individual lines. The distances d_{ij} are specified in order $i = 1 \dots N, j = 1 \dots i$.

The program performs complete linkage clustering as default but average linkage can be specified by the (**-a**) flag. Average linkage accounting for weights is possible with (**-a -w**) the weights are then taken from the ids which must have format

Name1_Weight1

...

NameN_WeightN

The program produces three output files stub.list, stub.otu, stub.tree when stub is specified by (**-out**):

stub.list has format (similar to Dotur)

d NClusters Cluster1 .. ClusterN

where d is the distance at which clusters formed. N is the number of clusters at this cutoff and then each cluster is specified as a comma separated list of entries either indexed 0 to N -1 or by ids if the (-i) flag is specified.

stub.otu simply gives the cluster sizes in the same format. Clusters are outputted at separations of 0.01 by default but this can be change by (-r) flag.

stub.tree is the hierarchical in newick tree format

Finally the distances can be scaled by their maximum using the (-s) flag.

Examples:

To perform complete linkage hierarchical clustering:

```
FCluster -in test.fdist -out test_M
```

Or to use average linkage with weights and ids in output:

```
FCluster -i -a -w -in test.ndist -out test_A
```

(this requires distance file with ids)

FClusterM:

-in string **distance input file name**

-out string **output file stub**

Options:

-r **resolution**

-a **average linkage**

-w **use weights**

-i **read identifiers**

-s **scale dist.**

This performs a simple hierarchical clustering. It reads a distance file in text format (-in) that has a full distance matrix. The first line in the text file gives the number of entities to be clustered N. This is then optionally followed by N ids if the (-i) flag is set as separate lines. Otherwise the N*N pairwise distances follow as individual lines. The distances d_{ij} are specified in order $i = 1 \dots N, j = 1 \dots N$. For clustering this matrix is converted into its symmetric equivalent $d_{ij} = 0.5 * (d_{ij} + d_{ji})$. This is suitable for clustering the output of SeqDistM.

FastaUnique – dereplicates fasta file

-in string **input file name**

Options:

This program simply dereplicates a fasta file of sequences. Sequences of different length are only compared up to the smaller length and if identical up to that smaller length are judged the same sequence. Dereplicated sequences with ids that are a combination of the founding sequence id and the number of identical sequences found i.e.

>founderID_weight

The mapping of sequences to the uniques is given by a .map file generated with the name fastaname.map where fastaname is obtained by parsing .fa of the original file name. This has a line for each unique sequence in format:

OriginalIdx, NewIdx, ParentID, I: Idx_1,...Idx_I:ID_1,...,ID_I

where I is the number of sequences mapping to the unique.

Example:

FastaUnique -in Test.fa > Test_U.fa

NDist - pairwise Needleman-Wunsch sequence distance matrix from a fasta file

-in string fata file name

Options:

-i output identifiers

This program generates a distance matrix from a fasta file of the format required by FCluster. It uses a simple implementation of the exact Needleman-Wunsch algorithm to perform pairwise alignments using a fixed gap penalty of 1.5. Distances are then calculated according to the 'QuickDist' algorithm basically counting mismatched nucleotides as a distance of one and with a cost of one for a gap regardless of length and then normalizing by number of comparisons (Huse et al. Genome Biology 2007). Output is to standard out.

The only option (**-i**) is to output identifiers suitable for running FCluster with **-i**.

This is an MPI program allowing the calculation of distances to spread across multiple cores and/or nodes.

Example:

mpirun -np 32 NDist -in Test.fa > Test.ndist

Perseus - slays monsters

-sin string seq file name

Options:

-s integer

-tin string reference sequence file

-a output alignments

-d use imbalance

-rin string lookup file name

The Perseus algorithm given an input fasta file (**-sin**) takes each sequence in turn and searches for the closest chimeric match using the other sequences as possible parents. It finds the optimum parents and breakpoints. It only searches for parents amongst species of equal or greater abundance where abundance is obtained from the fasta ids:

>ID_weight

Never run multiple copies of Perseus in the same directory! The (**-a**) flag outputs all the chimeric alignments and is useful for verifying if sequence truly is chimeric. The (**-d**) flag uses a slightly different algorithm including a penalty for imbalance on branches of the tree formed by the chimera and parents which may give better results in some instances. Perseus uses a nucleotide transition file and (**-rin**) allows this file to be set otherwise it defaults to the SEQ_LOOKUP_FILE variable and if this is not set the header variable LOOKUP_FILE which is set to “../Data/Tran.dat”.

We recommend removing degenerate primers before running Perseus.

It produces a lot of info but ... the critical portion are the x=12th, y=13th, and z=14th tokens. If $x < 0.15$ and $y \geq 0.0$ and z is larger than about 15 then this is a chimera.

The (**-s**) controls skew i.e. how much greater in frequency a sequence has to be to be a putative parent. This default to one – higher values can reduce the false positive rate.

The (**-tin**) option allows sequences other than the queries to be used as references. This can be used to split a file for running across threads or on a cluster (see example below).

Example usage:

```
sed 's/^ATTAGATACCC\w{1}GGTAG//' C005_s60_c01_T220_s30_c08_cd.fa > C005_s60_c01_T220_s30_c08_P.fa
```

```
Perseus -sin C005_s60_c01_T220_s30_c08_P.fa > C005_s60_c01_T220_s30_c08_P.per
```

To split a fasta file into four sections each in its own directory and then run Perseus in the background on each separately before recombining the output:

```
Split.pl Uneven1_s25_P.fa 4
```

```
cd Split0
```

```
Perseus -sin Split0.fa -tin ../Uneven1_s25_P.fa > Split0.per&
```

```
cd ../Split1
```

```
Perseus -sin Split1.fa -tin ../Uneven1_s25_P.fa > Split1.per&
```

```
cd ../Split2
```

```
Perseus -sin Split2.fa -tin ../Uneven1_s25_P.fa > Split2.per&
```

```
cd ../Split3
```

```
Perseus -sin Split3.fa -tin ../Uneven1_s25_P.fa > Split3.per&
```

```
../Scripts/Join.pl Split*/*per > Uneven1_s25_P.per
```

To classify sequences use Class.pl with suggested parameters for V5:

```
Class.pl C005_s60_c01_T220_s30_c08_P.per -6.6925 0.5652 >  
C005_s60_c01_T220_s30_c08_P.class
```

generates a file:

```
seqname x y z probabilityofbeingchimeric
```

We can split up the original fasta file at 50% probability of being chimeric:

```
FilterGoodClass.pl C005_s60_c01_T220_s30_c08_P.fa  
C005_s60_c01_T220_s30_c08_P.class 0.5 2>  
C005_s60_c01_T220_s30_c08_Good.fa > C005_s60_c01_T220_s30_c08_Chi.fa
```

PerseusD - slays monsters

```
-sin string seq file name
```

Options:

```
-c float,float set alpha,beta default = -5.54,0.33
```

```
-s integer set skew default = 2
```

```
-tin string reference sequence file
```

```
-a output alignments
```

```
-b do not use imbalance
```

```
-rin string lookup file name
```

PerseusD differs in algorithm and output from Perseus. It only tests against parents that have been classified as non-chimeric. It also only tests for possible parents amongst sequences that are at least twice as abundant as the query. These changes reduce false positives but at the cost that

sensitivity is also slightly reduced. They were inspired by the strategy adopted in uchime (Edgar et al 2011 *to appear* Bioinformatics). This program should be preferred when a few chimeras can be tolerated and false positives cannot. Unlike Perseus it needs to perform classification itself. Usage is just like Perseus except that it generates .class files rather than .per equivalent to running Perseus and then Class.pl:

Example usage:

```
Perseus -sin C005_s60_c01_T220_s30_c08_P.fa >  
C005_s60_c01_T220_s30_c08_P.class
```

The out format is therefore of this form:

SeqName x y z p

where p is the probability of the sequence being chimeric. Never run multiple copies of PerseusD in the same directory! PerseusD uses the imbalance penalty as default. The (-b) flag turns this off. The flag (-c **alpha,beta**) allows different alpha and beta parameters to be passed to the program these default to values for the V5 region trained through logistic regression. These work well generally though. Other parameters are as for Perseus.

PyroDist - pairwise distance matrix from flowgrams

-in	string	flow file name
-out	stub	out file stub
Options:		
-ni		no index in dat file
-rin	string	lookup file name

This program calculates a distance matrix between flowgrams. Input (-in) is to a .dat file containing flowgrams in a simple format. The first line has the number of flowgrams followed by the number of flows: N M. Each of the N flowgram entries has the format: id length1 flow1 flow2 ... flowM where id is just an identifier, length is the number of 'clean' flows, followed by all M flows (although only length will ever be used).

The distances are calculated according to the algorithm in Quince et al. 2009 except that alignment of flowgrams no longer occurs. This requires a look up table for the intensity distributions about the homopolymer length. By default this is read in from a file set in the header file by the constant LOOKUP_FILE which is set to “../Data/LookUp_E123.dat” a well configured distribution for 454 GSFLX implementation. Consequently the program can only be run from the bin directory to maintain this relative path. However, to allow the program to run anywhere the environment variable PYRO_LOOKUP_FILE can be set as described in the installation instructions or the path to a lookup file can be passed with the (-rin) flag.

The optional flag (**-ni**) is necessary if the flowgram file contains no ids.

Output is to a distance matrix in flat format of name stub.fdist where stub is set by the (**-out**) flag. Status information is sent to stdout this can be safely ignored if the program runs correctly.

This is an MPI program allowing the calculation of distances to spread across multiple cores and/or nodes.

Example:

```
mpirun -np 32 PyroDist -in Test.fa -out Test > Test.fdist
```

generates distance matrix Test.fdist

PyroNoise - clusters flowgrams without alignments

-din	string	flow file name
-out	string	cluster input file name
-lin	string	list file
Options:		
-v	verbose	
-c	double	initial cut-off
-ni		no index in dat files
-s	double	precision
-rin	file	lookup file name

This program uses an EM algorithm to construct de-noised sequences by clustering flowgrams as described in Quince et al. 2009 but without alignments. It takes as input (**-din**) a flowgram file of the format described above and an initial hierarchical clustering (**-lin**) generated by running FCluster on the output of PyroDist. Output files are generated with the stub specified by flag (**-out**).

The cut-off for the initial clustering is specified by (**-c**) generally this should be quite small 0.01 is a good value for most data sets. The parameter (**-s**) controls the cluster size. The larger this is the tighter the clusters – 60.0 is a reasonable value here but smaller may remove more pyrosequencing noise. If these parameters are not set they default to these values.

The parameter (**-rin**) allows a look up file to be specified otherwise the program uses the environment variable PYRO_LOOKUP_FILE if that is not set it defaults to the global variable LOOKUP_FILE found in PyroNoise.h currently “../Data/LookUp_E123.dat”. This will work provided the executable is run from the bin directory to maintain this relative path to the files in ../Data.

The option (**-v**) outputs extra debug information to standard out.

Information on cluster convergence is output to standard out and after running the program produces a number of files:

- 1) stub_cd.fasta: a fasta file of de-noised sequences. The ids are formed as ">stub_index_weight" where weight are the number of reads mapping to that sequence, and index is just an arbitrary cluster number.
- 2) stub_cd.qual: qualities for the denoised sequences see Quince et al. (unpublished).
- 3) stub.mapping: contains a line for each de-noised sequence giving the read that characterizes that sequence followed by a tab separated list of flowgram reads (specified by their ids read from dat file) that map to it.
- 4) directory stub: contains a fasta file for each de-noised sequence, i_index.fasta, of reads that map to it.

This is an MPI program allowing the calculation of distances to spread across multiple cores and/or nodes.

Example:

```
mpirun -np 32 PyroNoise -din Test.dat -out Test_s60_c01 -lin Test_X.list -s 60.0 -c 0.01 > Test_s60_c01.pout
```

PyroNoiseM

This version of PyroNoise has the exact same usage as above but stores flowgram distances in memory. It is useful for Titanium data where the calculation of these distances may be the limiting step.

SeqDist - pairwise distance matrix from a fasta file

-in string fasta file name
Options:
-i output identifiers
-rin string lookup file name

This program generates a distance matrix of the format required by FCluster from a fasta file. It uses a an implementation of the exact Needleman-Wunsch algorithm to perform pairwise alignments. Distances account for nucleotide transition probabilities as a result of PCR errors. There is a different cost for homopolymer (4.0) and normal gaps (15.0). The probabilities, actually -log of, are read from a look up table. By default this is from a file set in the header file by the constant LOOKUP_FILE which is set to "../Data/Tran.dat" configured for a standard polymerase. Consequently the program can only be run from the bin directory to maintain this relative path. However, to allow the program to run anywhere the environment variable SEQ_LOOKUP_FILE can be set as described in the installation instructions or the path to a lookup file can be passed with the (-rin) flag.

The option (**-i**) is to output identifiers suitable for running FCluster with **-i**.

This is an MPI program allowing the calculation of distances to spread across multiple cores and/or nodes.

Example:

```
mpirun -np 32 SeqDist -in Test.fa > Test.seqdist
```

SeqDistM

This version of SeqNoise has the exact same usage as above but generates an asymmetric distance matrix NXN distance matrix that is appropriate for SeqNoiseM.

SeqNoise - clusters sequences

-in	string	fasta sequence file name
-din	string	sequence distances file name
-out	string	cluster input file name
-lin	string	list file

Options:

-min	mapping file	
-v	verbose	
-c	double	initial cut-off
-s	double	precision
-rin	string	lookup file name

This program uses an EM algorithm to remove PCR noise by clustering sequences as described in Quince et al. (unpublished). The same distance metric as described in SeqDist is used. It takes as input (**-in**) a fasta file (with frequencies defined in ids as >id_weight), (**-din**) a flat matrix of sequence distances generated by SeqDist and an initial hierarchical clustering (**-lin**) generated by running FCluster on the output of SeqDist. Output files are generated with the stub specified by flag (**-out**).

The cut-off for the initial clustering is specified by (**-c**) generally this should be quite large 0.08 is a good value for most data sets. The parameter (**-s**) controls the cluster size. The larger this is the tighter the clusters – 30.0 is a reasonable value here but smaller may remove more noise and larger allow high resolutions OTUs. If these parameters are not set they default to these values.

The parameter (**-rin**) allows a look up file to be specified otherwise the program uses the environment variable SEQ_LOOKUP_FILE if that is not set it defaults to the global variable LOOKUP_FILE found in SeqNoise.h currently “../Data/Tran.dat”. This will work provided the executable is run from the bin directory to maintain this relative path to the files in ../Data.

The option (**-v**) outputs extra debug information to standard out.

The option (**-min**) allows a mapping file from a previous PyroDist step to be input. If used the program will use this information to map denoised sequences back to the original flowgram ids.

Information on cluster convergence is output to standard out and after running the program produces a number of files:

- 1) stub_cd.fasta: a fasta file of de-noised sequences. The ids are formed as ">stub_index_weight" where weight are the number of sequences mapping to that sequence, and index is just an arbitrary cluster number.
- 2) stub.mapping: contains a line for each de-noised sequence giving the input sequence defining the denoised cluster followed by a tab separated list of input sequences that map to that sequence.
- 3) directory stub: contains a fasta file for each de-noised sequence, i_index.fasta, of sequences that map to it.
- 4) Optional on (**-min**) if a mapping file is input then a file stub_cd.mapping containing a line for each de-noised sequence giving the id followed by a tab separated list of original reads that map to it.

This is an MPI program allowing the calculation of distances to spread across multiple cores and/or nodes.

Example:

```
mpirun -np 32 SeqNoise -in Test_s60_c01_T220.fasta -din
Test_s60_c01_T220.seqdist -lin Test_s60_c01_T220_S.list -out
Test_s60_c01_T220_s30_c08 -s 30.0 -c 0.08 -min Test_s60_c01.mapping >
Test_s60_c01_T220.snout
```

SeqNoiseM

This version of SeqNoise has the exact same usage as above but uses a slightly different algorithm for the centroid construction which will prefer longer sequences for centroid clusters. It may be preferred for Titanium data if read lengths are very uneven (std dev > 100) it requires input from SeqDistM.

SplitClusterEven

```
-din string dat filename
-min string map filename
-tin string tree filename
-s split size
-m min size
```

This program splits up dat files (**-din**) using a tree generated on unique sequences (**-tin**) input as a .tree file. The mapping of unique sequences to reads in the dat file is specified by a .map file (**-min**). The tree is the split in such a way as to maintain a maximum (**-s**) and minimum (**-m**) cluster size (measured on unique reads). The parameters **-s 2500** and **-m 250** will likely produce dat files of a good size although you should play around with these. The dat files are placed in directories labeled C000, ..., C00N+ where N is the number of clusters and the + simply indicates that this will be an aggregation of all small clusters.

Scripts:

Some useful Perl scripts are also provided in the Scripts directory:

FlowsFA.pl

This extracts flowgrams from the text translation of a .sff.txt. It takes the primer as a first argument and an output stub as the second. It reads from std in. It should be used for GSFLX reads. For example

```
FlowsFA.pl ATTAGATACCC[ACTG]GGTAG ArtificialGSFLX <
ArtificialGSFLX.sff.txt
```

Will generate the filtered .dat flowgram file ArtificialGSFLX.dat and a fasta file of the corresponding sequences ArtificialGSFLX.fa. Filtering requires that a minimum sequence length of 204 (changed by altering variable \$minLength) including key and primer is achieved before the first noisy signal (0.5-0.7 or no signal across all four bases). Flowgrams are then truncated at this point. If keys are used simply pass the entire key - linker - primer sequence to this script or use SplitKeys.pl described below.

FlowsFA360.pl

This extracts flowgrams from the text translation of a .sff.txt. It takes the primer as a first argument and an output stub as the second. It reads from std in. It should be used for GSFLX reads. For example

```
FlowsFA360.pl ATTAGATACCC[ACTG]GGTAG ArtificialGSFLX <
ArtificialGSFLX.sff.txt
```

Will generate the filtered .dat flowgram file ArtificialGSFLX.dat and a fasta file of the corresponding sequences ArtificialGSFLX.fa. Filtering requires that a minimum flowgram length of 360 including key and primer is achieved before the first noisy signal (0.5-0.7 or no signal across all four bases). All flowgrams are then truncated at 360. If keys are used simply pass the entire key - linker - primer sequence to this script or use SplitKeys.pl described below.

FlowsMinMax.pl

This extracts flowgrams from the text translation of a .sff.txt. It takes the primer as a first argument and an output stub as the second. It reads from stdin. It should be used for Titanium reads. For example

```
FlowsMinMax.pl ACACACGTCGACTCCTACGGGAGGCAGCAG TitaniumV3 <
TitaniumV3.sff.txt
```

Will generate the filtered .dat flowgram file TitaniumV3.dat and a fasta file of the corresponding sequences TitaniumV3.fa for a key **ACACACGTCG** and primer **ACTCCTACGGGAGGCAGCAG**. Filtering requires that a minimum flowgram length of 360 including key and primer is achieved before the first noisy signal (0.5-0.7 or no signal across all four bases). All flowgrams are then truncated at 720. If keys are used simply pass the entire key - linker -primer sequence to this script in upper case or use SplitKeys.pl described below.

CountFasta.pl

Gives total read number mapping to a fasta file with weighted ids.

```
CountFasta.pl < Test_s60_c01_cd.fa
```

Truncate.pl

Truncates sequences in a fasta file e.g.

```
Truncate.pl 220 < Test_s60_c01_cd.fa > Test_s60_c01_T220.fa
```

SplitKeys.pl

Separates out an sff file read from stdin according to barcode sequences. Requires a file Tags.csv with format:

```
SampleName1, Barcode1
...
SampleNameN, BarcodeN
```

The primer is the first argument of the script. The second is the Tags.csv file. This script generates .raw files that then have to be filtered and reformatted using Clean360.pl. A shell script Clean.sh shows how to do this for multiple raw data files. Reads that do not match to any tag are output to stderr. Any linkers must be included in the barcodes.

```
./SplitKeys.pl TGCTGCCTCCCGTAGGAGT Tags.csv < FV9NWL01.sff.txt 2>
Err.fa
```

SplitKeysFlower.pl

Separates out a flower file generated generated by Ketil Malde's program (<http://blog.malde.org/index.php/2009/07/03/a-set-of-tools-for-working-with-454-sequences/>) read from stdin according to barcode sequences. Requires a file Tags.csv with format:

```
SampleName1, Barcode1
...
SampleNameN, BarcodeN
```

The primer is the first argument of the script. The second is the Tags.csv file. This script generates .raw files that then have to be filtered and reformatted using Clean360.pl. A shell script Clean.sh shows how to do this for multiple raw data files. Reads that do not match to any tag are output to stderr. Any linkers must be included in the barcodes.

```
./SplitKeysFlower.pl TGCTGCCTCCCGTAGGAGT Tags.csv <
FV9NWLf01.flower.txt 2> Err.fa
```

Qiime_Typical.pl

Generates OTU consensus sequences with format suitable for Qiime. Takes fractional sequence difference for OTU construction as the first argument. Fasta file of denoised sequences for the second and list file from ndist for the third. See tutorial for information. Example:

```
./Qiime_Typical.pl 0.03 All_Good.fa All_Good.list > All_Good_C03_Q.fa
```

Qiime_OTU.pl

Generates Qiime OTU tables. Takes fractional sequence difference for OTU construction as the first argument. RDP taxonomic classifications as second and sample suffix for third. Generate classifications from using Qiime (`assign_taxonomy.py -i All_Good_C03_Q.fa`) Example:

```
./Qiime_OTU.pl 0.03
rdp_assigned_taxonomy/All_Good_C03_Q_tax_assignments.txt TS <
All_Good.list > All_Good_C03.qiime
```

The file `All_Good_C03.qiime` can now be used directly in Qiime as an OTU table.

Example Analyses Test

The directory test contains a shell script Run.sh which will run through the entire de-noising process for a single dat file. A smallish file, 2094 reads,

which will process on a good MacBook in ten or twenty minutes C005.dat is included. This should be run as follows:

```
./Run.sh C005.dat
```

If this works correctly a de-noised file C005_s60_c01_T220_s30_c08_cd.fa with just 18 sequences will be generated. The file C005_s60_c01_T220_s30_c08_cd.mapping will map these back to the original reads. Other files reflecting the intermediate steps are also generated but in general they can be ignored. The list file giving complete linkage OTUs for these sequences is also produced C005_s60_c01_T220_s30_c08.list.

Larger dat files could be processed with this script on a cluster simply by changing the value nodes to a larger number say 32 or whatever is appropriate. To explain the script in detail:

```
#!/bin/bash
```

```
nodes=2                   #no. of cluster nodes to use
```

(change this to a larger value if running on a cluster simply gives the number of nodes for the MPI programs)

```
file=$1; #first argument name of dat file
```

(any dat file with ids can be used)

```
echo $file  
stub=${file%.dat}
```

```
echo "Calculating .fdist file"
```

```
mpirun -np $nodes PyroDist -in $file -out ${stub} > ${stub}.fout
```

(first we calculate flowgram distances)

```
echo "Clustering .fdist file"
```

```
FCluster -in ${stub}.fdist -out ${stub}_X > ${stub}.fout
```

(then hierarchical cluster with complete linkage to provide input file for PyroNoise)

```
rm ${stub}.fdist  
rm ${stub}_X.otu ${stub}_X.tree
```

(remove some intermediate files)

echo "Running PyroNoise"

```
mpirun -np $nodes PyroNoiseM -din ${file} -out ${stub}_s60_c01 -lin  
${stub}_X.list -s 60.0 -c 0.01 > ${stub}_s60_c01.pout
```

(this performs the flowgram clustering according to the EM algorithm to remove pyrosequencing noise)

```
Truncate.pl 220 < ${stub}_s60_c01_cd.fa > ${stub}_s60_c01_T220.fa
```

(The end of reads are often noisy so we truncate to 220 base pairs. If degenerate primers are used they should perhaps be removed at this point although that is optional).

echo "Running SeqDist"

```
mpirun -np $nodes SeqDist -in ${stub}_s60_c01_T220.fa >  
${stub}_s60_c01_T220.seqdist
```

(Now we calculate the PCR error corrected distances between sequences)

```
FCluster -in ${stub}_s60_c01_T220.seqdist -out ${stub}_s60_c01_T220_S >  
${stub}_s60_c01_T220.fcout
```

(and complete linkage cluster to provide input to SeqNoise)

echo "Running SeqNoise"

```
mpirun -np $nodes SeqNoise -in ${stub}_s60_c01_T220.fa -din  
${stub}_s60_c01_T220.seqdist -lin ${stub}_s60_c01_T220_S.list -out  
${stub}_s60_c01_T220_s30_c08 -s 30.0 -c 0.08 -min  
${stub}_s60_c01.mapping > ${stub}_s60_c01_T220.snout
```

```
rm ${stub}_s60_c01_T220_S.otu ${stub}_s60_c01_T220_S.tree  
${stub}_s60_c01_T220.seqdist
```

(the sequence clustering algorithm that removes PCR errors)

echo "Clustering OTUs"

```
mpirun -np $nodes NDist -i -in ${stub}_s60_c01_T220_s30_c08_cd.fa >  
${stub}_s60_c01_T220_s30_c08.ndist
```

```
FCluster -i -in ${stub}_s60_c01_T220_s30_c08.ndist -out  
${stub}_s60_c01_T220_s30_c08 > ${stub}_s60_c01_T220_s30_c08.fcout
```

(Finally we build OTUs from the de-noised sequences)

```
rm ${stub}_s60_c01_T220_s30_c08.ndist  
echo "Removing intermediate files"  
rm *out  
exit 0
```

TestFull

The directory TestFull contains a shell script that illustrates the de-noising process for a larger sample that needs to be split to allow de-noising. This should only be run on a cluster or good server. It is assumed that a single sample without barcodes is used. The script takes an .sff file as an argument but in case sffinfo (a 454 program) is absent we have provided ArtificialGSFLX.sff.txt an sff that has already been converted into text.

The script should be run in general as:

```
./Run.sh My.sff primer
```

and for test purposes:

```
./Run.sh ArtificialGSFLX.sff
```

```
#!/bin/bash
```

```
defaultPrimer="ATTAGATACCC\w{1}GGTAG" #default primer  
nodes=8 #no. of cluster nodes to use
```

```
sfffile=$1; #first argument name of sff file (necessary)  
primer=${2:-$defaultPrimer} #second argument primer as a Perl regular  
expression
```

(the second argument should be your primer else it defaults to our 787f)

```
stub=${sfffile%.sff};
```

```
echo $stub $sfffile $primer
```

```
# first generate sff text file if necessary  
if [ ! -f ${sfffile}.txt ]; then  
    echo "Generating .sff.txt file"  
    sffinfo $sfffile > ${sfffile}.txt  
fi
```

(generates text translation of sff file if necessary)

```
#generate flowgram and fasta files  
if [ ! -f ${stub}.dat ]; then  
    echo "Generating .dat file"  
    FlowsFA360.pl $primer $stub < ${sfffile}.txt  
fi
```

(extracts filtered dat and sequence files)

```
#get unique sequences  
if [ ! -f ${stub}_U.fa ]; then  
    echo "Getting unique sequences"
```

```
FastaUnique -in ${stub}.fa > ${stub}_U.fa  
fi
```

(generates a file of unique sequences)

```
#use NDist to get sequence distances  
if [ ! -f ${stub}_U_I.list ]; then  
    echo "Calculating sequence distances"  
    mpirun -np $nodes NDist -i -in ${stub}_U.fa > ${stub}_U_I.ndist  
fi
```

```
#use NDist to get sequence distances  
if [ ! -f ${stub}_U_I.list ]; then  
    echo "Cluster sequences..";  
#cluster sequences using average linkage and sequence weights  
    FCluster -a -w -in ${stub}_U_I.ndist -out ${stub}_U_I > ${stub}_U_I.fcout  
fi
```

```
rm ${stub}_U_I.ndist
```

(These two steps use the unique sequences to generate an average linkage tree based on sequence distance...)

```
SplitClusterEven -din ${stub}.dat -min ${stub}.map -tin ${stub}_U_I.tree -s  
5000 -m 1000 > ${stub}_split.stats
```

(that is then used to split up the .dat file)

```
echo "Calculating .fdist files"  
for c in C*  
do  
    if [ -d $c ] ; then  
        mpirun -np $nodes PyroDist -in ${c}/${c}.dat -out ${c}/${c} >  
${c}/${c}.fout  
    fi  
done
```

(Now we can start denoising each dat file separately beginning by calculating the flowgram distances)

```
echo "Clustering .fdist files"  
  
for c in C*  
do  
    if [ -d $c ] ; then  
        FCluster -in ${c}/${c}.fdist -out ${c}/${c}_X > ${c}/${c}.fout  
        rm ${c}/${c}.fdist  
    fi
```

done

(cluster them...)

```
echo "Running PyroNoise"
for dir in C*
do
    if [ -d $dir ] ; then
        mpirun -np $nodes PyroNoise -din ${dir}/${dir}.dat -out
        ${dir}/${dir}_s60_c01 -lin ${dir}/${dir}_X.list -s 60.0 -c 0.01 >
        ${dir}/${dir}_s60_c01.pout
    fi
done
```

(denoise them to get sequences)

```
cat C*/C*_s60_c01_cd.fa > All_s60_c01_cd.fa
cat C*/C*_s60_c01.mapping > All_s60_c01.mapping
```

```
Truncate.pl 220 < All_s60_c01_cd.fa > All_s60_c01_T220.fa
```

(in this case we can cat them together and do a single sequence noise removal step but in general we may first run SeqNoise in the separate directories before bringing them together for a final noise removal step)

```
echo "Running SeqDist"
mpirun -np $nodes SeqDist -in All_s60_c01_T220.fa >
All_s60_c01_T220.seqdist
```

```
FCluster -in All_s60_c01_T220.seqdist -out All_s60_c01_T220_S >
All_s60_c01_T220.fcout
```

```
rm All_s60_c01_T220.seqdist
```

```
echo "Running SeqNoise"
mpirun -np $nodes SeqNoise -din All_s60_c01_T220.fa -lin
All_s60_c01_T220_S.list -out All_s60_c01_T220_s30_c08 -s 30.0 -c 0.08 -min
All_s60_c01.mapping > All_s60_c01_T220_s30_c08.snout
```

(Finally we remove PCR error from our sequences)