Documentation of Tools for Noise Removal from Pyrosequenced Amplicons (AmpliconNoiseV1.29)

30th July 2013

Usage and reference:

AmpliconNoise is a collection of programs for the removal of noise from 454 sequenced PCR amplicons. Typical usage involves three steps:

1) the removal of noise from the sequencing itself,

2) the removal of PCR point errors, and

3) removal of chimeric sequences using the program Perseus.

While steps 2-3 can be used on amplicon reads generated using most sequencing technologies, step 1 is currently only supported for sequences generated using [454 pyrosequencing](#).

When presenting work using AmpliconNoise, please refer to the following citation:

Quince *et al* (2011), 'Removing noise from pyrosequenced amplicons.', *BMC Bioinformatics* **12**, 38.

Installation:

Requirements

The programs have been tested on MacOsX 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 and the Gnu Science Library are installed:

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

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

The proprietary program *sffinfo* from 454 Genomics (Roche) is optional, but the convenience scripts *RunTitanium.sh,* and *RunFLX.sh,* assume that this is installed. These scripts also use usearch and this too should be downloaded and installed (http://www.drive5.com/usearch/). *sffinfo* is part of the analysis software and drivers that are shipped with the sequencing machines from Roche and can possibly be obtained from your sequencing facility, or ask them to deliver plain text versions of all flowgram (SFF) files. Alternatively, you can use the free software Flower by Ketil Malde (http://blog.malde.org/index.php/2009/07/03/a-set-of-tools-for-working-with-454-sequences) or the script *process_sff.py* of QIIME (http://www.qiime.org/) to convert the SFF-files to plain text versions. The convenience scripts are still run referring to the name of the SFF-file but will skip the initial parsing step. If using Flower, the script *RunTitanium.sh* needs to be edited by changing "SplitKeys.pl" to "SplitKeysFlower.pl" (line 76).

Installation procedure

First extract and unzip the programs:

```
tar -xvzf AmpliconNoiseV1.29.tar.gz
```

To compile the programs, move into the top directory and type:

```
make clean
make
```

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

If the programs compile without errors, type:

```
make install
```

This will place the executables in the bin directory. This directory and the Scripts directory need to be added to your $PATH in order to run the programs from the command line. If you extract *AmpliconNoiseV1.29.tar.gz* in your home directory ($HOME) then this command should be added to your *.bashrc* or *.profile* or equivalent. Edit this file and add these two lines:

```
export PATH=$HOME/AmpliconNoiseV1.29/bin:$PATH
export PATH=$HOME/AmpliconNoiseV1.29/Scripts:$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_Titanium.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. Add the two following lines to your *.bashrc* or *.profile* located in your home directory:

```
export AMPLICON_NOISE_HOME=$HOME/AmpliconNoiseV1.29/
export
PYRO_LOOKUP_FILE=$HOME/AmpliconNoiseV1.29/Data/LookUp_Ti
tanium.dat
export
SEQ_LOOKUP_FILE=$HOME/AmpliconNoiseV1.29/Data/Tran.dat
```

Then either open a new terminal window or source the file to activate these environment variables:

```
source ~/.bashrc
```

*If working with sequencing data generated using an older Pyrosequencing protocol or machine such as (non-Titanium) GS FLX or GS 20, then the *file LookUp_E123.dat* located in the same directory should be used instead of *LookUp_Titanium.dat*

Testing the installation by running a single dat file

The directory Test contains the shell script *Run.sh* which will run through the entire de-noising process for a single dat file. This is the file format AmpliconNoise uses to store flowgrams. A smallish file consisting of 2,094 GS FLX reads, which will process on a reasonably new MacBook in ten or twenty minutes *C005.dat* is included. This should be run as follows:

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

If this works correctly the 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.*

Running AmpliconNoise on a complete barcoded 454 run

In the directory *Scripts* in the AmpliconNoise installation directory, there are two scripts for running the typical analysis workflow. These are *RunTitanium.sh* and

RunFLX.sh. These can be executed directly on the raw output from pyrosequencing data supplied as flowgram (SFF) files (file suffix ".sff").

In addition there are a number of other scripts for less standard pipelines.
- RunTitaniumA.sh: This will run 454 and IT data sets that comprise acyclic flows and non-standard key signatures. It is currently being tested but will replace RunTitanium.sh in the next release.
- RunTitaniumFast.sh: A script that splits even small samples with uclust in order to speed up denoising at a cost of sensitivity.
- RunBoth.sh: Another script under testing to run amplicons with dual barcodes. This will be fully documented in future updates.

RunTitanium.sh and RunFLX.sh

RunTitanium.sh is used for GS FLX Titanium or GS FLX Titanium+ datasets whereas *RunFLX.sh* is used for GS FLX or GS20 datasets.

In addition to the sff file, a comma-separated text-file named *keys.csv* needs to be present in the same directory. This file should contain one line for each barcode containing the sample name, comma and the barcode sequence, excluding the primer sequence used, e.g.:

```
Sample07,CTCGCGTGTC
```

If barcodes have not been used in your data set simply include a single line in this file with an empty barcode field e.g.

Sample,

In addition, a primer sequence has to be supplied, either by passing it as a second argument to the script or by making a FASTA-formatted file containing only the primer sequence named *primer.fasta,* e.g.:

```
>787F
ATTAGATACCCNGGTAG
```

Note that this file may contain degenerated base characters, such as 'N'.

As a default, the script is run using 4 parallel processes. To change this value, edit the script or make a local copy of it and edit that. Change line "nodes=4" to as many processes as you would like to run. The number of nodes should not exceed the cores on a server or the total number of cores on a cluster.

Both scripts are then run by passing the command and the name either of an sff file (if sffinfo is installed) or the already text translated file e.g.

```
RunTitanium.sh all MySamples.sff
```

Or

```
RunTitanium.sh all MySamples.sff.txt
```

Alternatively each step in the process can be run separately. There are six steps in all. Each is run by passing the corresponding command as the first argument to the script:

`split`: generates one file for each sample named (SampleName.raw) comprising all flowgrams from that sample found by insisting on a perfect match to the primer and barcode (this command requires the sff or sff.txt file as a second argument e.g. `RunTitanium.sh split MySamples.sff.txt`)

`filter`: generates filtered dat files

`pyronoise`: performs the PyroNoise flowgram clustering steps

`seqnoise`: performs the SeqNoise sequence clustering step to remove PCR errors

`perseus`: runs Perseus chimera classifier

`perseusd`: runs PerseusD chimera classifier

`otus`: builds OTUs from the denoised sequences

Output of AmpliconNoise

The different processes of the AmpliconNoise workflow generate several files, most important of which are (using "SampleX" as an example sample name):

SampleX_F_Good.fa
This file contains the unique sequences after removal of sequencing noise, PCR point errors and chimeras, in FASTA format. The last number of each sequence name given in the FASTA header, indicated after the underscore character, represents the number of reads that share this unique sequence after cleaning. For example the fasta header ">LA_RNA_0_8" indicates that this sequence represent eight reads, from the sample *LA_RNA*.

All_Good_C01.fa, All_Good_C03.fa, All_Good_C05.fa, All_Good_C10.fa
These files contain representative sequences for each OTU (Operational Taxanomic Unit) after a 1%, 3%, 5%, 10% maximum linkage clustering of the unique, de-noised sequences.

All_Good_C01.csv, All_Good_C03.csv, All_Good_C05.csv, All_Good_C10.csv
These files contain frequencies for each OTU (Operational Taxanomic Unit) after a 1%, 3%, 5%, 10% average linkage clustering of the unique, de-noised sequences in each of the samples.

If complete linkage clustering is desired then simply remove the –a flag from the FCluster command at line 328 of the script.

AN_stats.txt
This tab-separated text file contains statistics about the run, such as number of reads and unique sequences before and after de-noising and chimera filtering.

SampleX_[..]_cd.mapping
This file maps each unique sequence back to the name of its reads.

Step-by-step description of workflow parameters

This section explains the parameters of the RunTitanium.sh script. The other script *RunFLX.sh* is almost identical just reparameterised for FLX.

```
bc=keys.csv
nodes=4
snodes=1
min_size=50
max_size=50000
```

These values specify the name of the barcodes metadata file *keys.csv*, the number of nodes to be used, the number of nodes used for data sets with less than 100 sequences, the minimum number of reads for a sample to be processed, and the maximum sample size above which it will be split into multiple parts for separate processing. After this we have:

```
mpiextra="--mca btl tcp,self"

export AMPLICON_NOISE_HOME=$HOME/AmpliconNoiseV1.29/
export
PYRO_LOOKUP_FILE=$AMPLICON_NOISE_HOME/Data/LookUp_Titaniu
m.dat
export SEQ_LOOKUP_FILE=$AMPLICON_NOISE_HOME/Data/Tran.dat
```

The first of these variables are extra flags to pass to mpirun. The rest define environment variables for the AmpliconNoise home directory and the parameters to use for PyroNoise and SeqNoise respectively. We then have variables for the noise removal programs themselves:

```
#hardcoded parameters for AmpliconNoise

length=400
#truncation length

spyro=60
#PyroNoise cluster size

cpyro=0.01
```

```
#PyroNoise cluster init

sseq=25
#SeqNoise cluster size

cseq=0.08
#SeqNoise cluster init

alpha=-7.5
#Perseus logit intercept

beta=0.5
#Perseus logit gradient

minflows=360

maxflows=720
```

Of these the most important and the ones most likely to be adjust by the user would be the last two. To be less strict on filtering reduce minflows so to 240 for example. The other parameters are explained in the description of the individual programs below.

Brief description of RunTitanium.sh script

The complete script is complex but we will attempt to give an overview of the most salient steps. The workflow is started by reading the primer sequence from file primer.fasta. The script is split into separate subroutines corresponding to six steps mentioned above. The first subroutine split generates a plain-text version of the flowgram (SFF) file if such a file does not already exists (line 66):

```
if [ ! -f ${stub}.sff.txt ]; then
    echo "Generating .sff.txt file"
    sffinfo $1 >${stub}.sff.txt
fi
```

The plain-text flowgram file is then quality filtered and parsed into one or more *.dat* files containing only the identifiers and flow values of those reads that pass quality filtering (filter subroutine – line 93). The dataset is split into one *.dat* file for each barcode. Only exact matches to the given barcodes are retrieved.

The pre-filtering control also removes all reads with fewer than 360 flows before the first empty flow cycle or degenerate base (flow intensity between 0.5 and 0.7).

Following pre-filtering, then steps pyronoise, seqnoise and perseus are repeated for each sample using *RunTitanium / RunFLX.*

In the first step of pyronoise, distances between flowgrams are calculated using *PyroDist (line 173)*:

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

Then, hierarchical clustering with complete linkage is carried out using *FCluster* to provide input file for *PyroNoise*. (Some intermediate files are also removed):

```
echo "Clustering .fdist file"
FCluster -in ${stub}.fdist -out ${stub}_X > ${stub}.fout

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

Next, the flowgrams are iteratively clustered according to the EM algorithm implemented in *PyroNoise*, to remove pyrosequencing noise. An initial clustering cut-off of $cpyro and cluster size of $spyro are used (see Section Programs for details).

```
echo "Running PyroNoiseM"

echo "Running PyronoiseM for ${stub}" mpirun $mpiextra -
np $nodes PyroNoiseM -din ${stub}.dat -out ${pstub} -lin
${stub}_X.list -s $spyro -c $cpyro > ${pstub}.pout
```

The ends of reads are often noisy, so next, we truncate these to 400 bp (220 for FLX reads). This position can be moved to change the balance between remaining noise and sequence length and , barcodes and primer sequences are removed.

```
Parse.pl ${barcode}${primer} $length < ${pstub}_cd.fa >
${pstub}_T${length}.fa
```

The next step of the process is to remove PCR errors in the subroutine seqnoise. To begin we calculate the PCR-error-corrected distances between sequences using *SeqDist*:

```
echo "Running SeqDist for ${stub}"

mpirun $mpiextra -np $tnodes SeqDist -in
${pstub}_T${length}.fa > ${pstub}_T${length}.seqdist
```

Complete linkage clustering (again using *FCluster*) is carried out to provide input to *SeqNoise*:

```
echo "Clustering SeqDist output for ${stub}"
```

```
FCluster -in ${pstub}_T${length}.seqdist -out
${pstub}_T${length} > ${pstub}_T${length}.fcout
```

SeqNoise implements the sequence clustering algorithm that removes PCR errors:

```
echo "Running SeqNoise"

mpirun $mpiextra -np $tnodes SeqNoise -in
${pstub}_T${length}.fa -din ${pstub}_T${length}.seqdist
-lin ${pstub}_T${length}.list -out ${sstub} -s $sseq -c
$cseq -min ${pstub}.mapping > ${sstub}.snout
```

PerseusD is then run in the subroutine perseus to screen for and remove chimeric sequences.

Finally we build OTUs from the de-noised sequences in subroutine otus.

Running larger datasets

Any samples with more reads than the parameter max_size will be automatically split and run with the subroutine pyronoisesplit.

TestTitaniumFast.sh

An additional script is provided which uses usearch to perform preclustering of even small samples and hence runs faster than the standard script. This comes at a small cost in terms of sensitivity to remove noise.

Testing the scripts

In the directories TestFLX, TestTitanium, TestTitaniumFast configuration files are available for testing the RunFLX.sh, RunTitanium.sh and RunTitaniumFast.sh scripts respectively. The data sets themselves have to be downloaded, however. To test the RunFLX.sh script:

```
cd TestFLX
wget
http://userweb.eng.gla.ac.uk/christopher.quince/Data/Arti
ficialGSFLX.sff.txt.gz
gunzip ArtificialGSFLX.sff.txt.gz
RunFLX.sh all ArtificialGSFLX.sff.txt
```

If this script runs correctly it will generate the file AN_stats.txt as follows:
Sample Total Pre-filtered Unique Chimeric CleanSeq CleanReads
Artificial   40258 31045 60   17   43   31025

And to test the RunTitanium.sh script:

```
wget
http://userweb.eng.gla.ac.uk/christopher.quince/Data/Tita
nium.sff.txt.gz
gunzip Titanium.sff.txt.gz
RunTitanium.sh all Titanium.sff.txt
```

If this script runs correctly it will generate the file AN_stats.txt as follows:
Sample Total Pre-filtered Unique Chimeric CleanSeq CleanReads
Titanium   35765 25311 307   118   189   25149

Testing RunTitaniumFast.sh proceeds as above and should give:

Sample Total Pre-filtered Unique Chimeric CleanSeq CleanReads
Titanium   35765 25311 252   107   145   25168

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). This code has been optimized by David Hunt of Tessella.

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...N, j = 1..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 take 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)

FastaUnique:
-in   string   input file name

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:

```
OriginallIdx, 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 >= 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
```

..this generates a file with format:

```
seqname x y z probability_of_being_chimeric
```

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 'UCHIME improves sensitivity and speed of chimera detection', *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 distrubution 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 standard out and 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.fdout
```

This generates the 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 paramter (-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.fa*: 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.fa*, 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
```

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. (2011). 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 paramter (-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.fa*: 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.fa*, 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.fa -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
```

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.

SplitClusterClust

-din   string dat filename
-min   string map filename
-uin   string uclust filename
-m   integer min size

This program splits up *dat* files (-din) using output of uclust parsed with Sub.pl. The mapping of unique sequences to reads in the dat file is specified by a .map file (-min). 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 a plain-text flowgram (*.sff.txt*) file. It takes the primer as a first argument and an output stub as the second. It reads from standard input. It should be used for GS FLX 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 standard input. It should be used for GS FLX 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 file. It takes the primer as a first argument and an output stub as the second. It reads from standard input. 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. For example:

```
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 standard input according to barcode sequences. Requires a file *keys.csv* with format:

```
SampleName1, Barcode1
…
SampleNameN, BarcodeN
```

The primer is the first argument of the script. The second is the *keys.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 standard error. Any linkers must be included in the barcodes.

```
./SplitKeys.pl TGCTGCCTCCCGTAGGAGT keys.csv <
FV9NWLF01.sff.txt 2> Err.fa
```

SplitKeysFlower.pl

Separates out a flower file generated generated by Ketil Malde's program Flower (http://blog.malde.org/index.php/2009/07/03/a-set-of-tools-for-working-with-454-sequences). It reads from standard input according to barcode sequences. Requires a file *keys.csv* with format:

```
SampleName1, Barcode1
…
SampleNameN, BarcodeN
```

The primer is the first argument of the script. The second is the *keys.csv* file. This script generates .raw files that then have to be filtered and reformatted using *Clean360.p*l. 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 standard error. Any linkers must be included in the barcodes:

```
./SplitKeysFlower.pl TGCTGCCTCCCGTAGGAGT Keys.csv <
FV9NWLF01.flower.txt 2> Err.fa
```

SubsampleDat.pl

Randomly subsamples a .dat-file according to the specified number of reads. To produce the .dat-file Sub1000.dat from All.dat with 100 clean reads, use e.g.:

```
SubsampleDat.pl All.dat 1000 > Sub1000.dat
```

Qiime_Typical.pl

Generates OTU consensus sequences with format suitable for *Qiime*. Takes fractional sequence difference for OTU construction as the first argument, a fasta file of denoised sequences for the second and list file from *NDist* for the third. See the tutorial for more 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 using:

```
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.