PoolParty Pipeline Example 2

by: Stuart C. Willis, updated 6/2022

Synopsis:
This exercise uses 150-bp paired-end sequences from steelhead trout (found in the 'Example2' folder) to illustrate some of the advanced features of *PoolParty*. We will assume the first example, which requires a shorter run time and covers many basic aspects of *PoolParty*, has already been run, and that all dependencies are therefore installed and operational. Depending on computation resources, this Example2 may require several hours to complete all 3 modules (mostly mapping and SNP scoring).

We will use a hypothetical example to detect regions of strong genomic differences between three stocks of steelhead, one of which has a history of anthropogenic selection, specifically a history of being strongly graded for fish that return after two years in the ocean (as opposed to one) and during the summer (as opposed to winter). In this example2, we will be aligning 300 FASTQ files from 150 samples in 3 groupings to a pruned reference genome consisting of 3 chromosomes (Omy_USDA1.1_Omy05.25.28.fasta).

As in first example, each paired set of read files is listed in a tab-delimited sample file, wherein the sample grouping (e.g. population or phenotype) for each individual is listed also. Note that in this Example2 the content of the fastq files have been pruned to include mappable reads from only these chromosomes, and file sizes and mapping rates will reflect that. The sample (population) groupings in this example are as follows: 02_LEW_ELEW and 04_WIL_EAGL are the control populations, and 03_WAS_SKAM has a history of selection.

1) Directory Structure and File Unpacking

We assume that you have downloaded the poolparty scripts and data as described in the Github README. Navigate to the example2 folder in a shell window. Before we begin, we need to download and unarchive the read files (about 7Gb of data). This archive also contains the sample list file, which should remain in the same directory.

```
> wget https://figshare.com/ndownloader/files/34951362 -O
        Omy_USDA1.1_Omy05.25.28.fasta
> cd samples
> wget https://figshare.com/ndownloader/files/34645955 -O example2_fastq.tar.gz
> tar -xvzf example2_fastq.tar.gz
> ls *.fastq.gz | wc -l
```

Should return 300, 2 paired files for each of 150 samples.

2) Genome Preparation

PoolParty requires a reference genome indexed for its dependencies. Assuming the *bwa* and *samtools* dependencies are in the $PATH, input the genome fasta name and the path to the picard jar on the command line. Change to the directory containing the genome fasta the active directory (in this Example2, the directory containing/above the samples directory).

```
> cd ..
> bash ~/bin/poolparty/PPprep_genome_cl.sh Omy_USDA1.1_Omy05.25.28.fasta
        ~/miniconda3/envs/poolparty_env/share/picard-2.20.2-0/picard.jar
```

Note: shell commands not preceded with ">" are part of the previous line; please ensure that if you copy/paste these lines, no extra carriage return is pasted with it, lest only part of the command be executed. After running these commands a number of files with additional extensions will be produced: *.sa, .fai., .dict, .pac, .bwt, .ann, and .amb*.

It is important that genome name serves as a prefix to these the files (e.g. *Omy_USDA1.1_Omy05.25.28.fasta.dict*).

3) PPalign

In this Example2, we will demonstrate some of the useful features of *PPalign*, including the ability to run this module piecemeal, which is helpful for checking that sufficient sequencing depth has been achieved, incorporating data over several stages, or managing resource (processor/memory) usage for different steps. This is possible both through optional switches and the fact that PPalign checks to see that the expected files (e.g. trimmed reads, filtered alignments) are already present, and if found, will not repeat these steps. Note that when combining new data with old, it is important to delete old files that list the sample groups or individuals to expect, i.e. $OUTPOP_poplist.txt and $OUTDIR/pops/$OUTPOP_files_for_pops.txt if including new sample groups, and/or the $OUTDIR/pops/pop_{POP}.txt files if including new individuals for previously used sample groups.

a) The sample list

In this example, the sample list has been provided, named *example2_samples_150.txt*. As this file is not named *samplelist.txt*, this file will be specified in the *config* file as SAMPLELIST, and must be in the directory specified as INDIR in the *config* file. The sample list contains one *fastq* file per line with its corresponding sample group (*tab-delimited, with unix line endings, including a carriage return on the last line*). In this example, we have 150 libraries (pairs of read files) that belong three different sample groups as described.

Take a look at *example2_samples_150.txt*:

```
> head -n6 samples/example2_samples_150.txt
```

```
OmyBsnp10-05CG01_i207_101_L1022-subset-R1.fastq.gz     02_LEW_ELEW
OmyBsnp10-05CG01_i207_101_L1022-subset-R2.fastq.gz     02_LEW_ELEW
OmyBsnp10-05CG02_i207_102_L1022-subset-R1.fastq.gz     02_LEW_ELEW
OmyBsnp10-05CG02_i207_102_L1022-subset-R2.fastq.gz     02_LEW_ELEW
OmyBsnp10-05CG03_i207_103_L1022-subset-R1.fastq.gz     02_LEW_ELEW
OmyBsnp10-05CG03_i207_103_L1022-subset-R2.fastq.gz     02_LEW_ELEW
```

See the first example for guidance on naming conventions. From this excerpt, we see two *fastqs* for each of three libraries (individuals): OmyBsnp10-05CG01, OmyBsnp10-05CG03, and OmyBsnp10-05CG03, all from sample group 02_LEW_ELEW. Notice that this sample group name is alphanumeric (combination of letters and numbers), with leading zeros on all numbers. Check to see how the sample groups will be sorted by *PPalign* (this will also be output in $OUTPOP_files_for_pops.txt and in the log after normalization):

```
> cut -f2 samples/example2_samples_150.txt | sort | uniq
```

b) Running PPalign *only to produce input for* PPstats.

Two options for *PPalign* are to run only to align data (produce BAM files), useful if the user expects to add more data in the future, and to create the mpileup file needed to run *PPstats*, useful if the user expects to later add more data but wishes to view coverage statistics to guide future sequencing runs. We will demonstrate the latter here. See the first example for additional parameter explanation. We will also demonstrate a feature that allows the user to certify that the sample list is correctly formatted (being interpreted correctly), as *PPalign* will confirm the number of single/paired samples and sample groups identified.

The configuration file contains directory locations, parameters, and dependency locations. **You may need to edit *pp_align _example2_stats.config* to tailor directories and dependencies to your system.** Note, that there cannot be any spaces between "=" and the definition of the variable; the script will exit with an error if so.

Open *pp_align _example2_stats.config*

#Input/Output#
INDIR=samples

Full or relative path to the directory where the *fastq* files and the sample list are.

SAMPLELIST=example2_samples_150.txt

This specifies the name of the sample list, should it be other than *samplelist.txt*.

OUTDIR=poolparty_example2

Full or relative path to directory for output files (will be created if it does not exist). Be sure that you have write permission in this or working directory.

OUTPOP=example2

This will be the prefix to most files that are written.

GENOME=Omy_USDA1.1_Omy05.25.28.fasta

Full or relative path to the genome assembly.

SCAHEAD=

This reference genome contains only three chromosomes. Leave this blank.

#Run Parameters#*
THREADZ=4

Four should be enough for this example.

BQUAL=20
MAPQ=5
SNPQ=20
MINLENGTH=50
INWIN=3
MAF=0.005

Global minor allele frequency. Removes SNP calls where the minor allele is below MAF.

KMEM=Xmx4g
MINDP=5

The minimum depth of coverage summed across all libraries required to retain a SNP.

#Run-types#
SPLITDISC=off
INDCONT=off

no effect at this stage

QUALREPORT=on

#Optional Parameters#
ALIGNONLY=off

quit after alignments are created. Optional states: on/off

STATSONLY=on

quit after making stats.mpileup (used in PPstats). Optional states: on/off

CONFIRM=on

confirm with the user that read pairing and population counts were interpreted correctly. Interactive; cannot be run in background!

USEVCF=

no effect at this stage

PARALLEL=off

no effect at this stage

DIVIDE=

no effect at this stage

#Dependency Locations# **MODIFY AS NECESSARY**

```
BCFTOOLS=bcftools  #modify if not in the $PATH
FASTQC=fastqc  #modify if not in the $PATH
BWA=bwa  #modify if not in the $PATH
SAMBLASTER=samblaster  #modify if not in the $PATH
SAMTOOLS=samtools #modify if not in the $PATH
PICARDTOOLS=~/miniconda3/envs/poolparty_env/share/picard-2.20.2-0/picard.jar
BBMAPDIR=~/miniconda3/envs/poolparty_env/bin
POOL2=~/bin/popoolation2_1201
```

PPalign will ensure that all dependencies are installed and throw an error if not. Ensure that the correct command and/or directory is set for each dependency. Note that R, perl, and java languages are required as well.

Without making a link to the *PPalign* script, we can simply run the script with bash by specifying the location of the script, and the config file as an argument. Make sure you are in the directory where the config file is located and run *PPalign* as follows*:*

```
> bash ~/bin/poolparty/PPalign.sh pp_align_example2_stats.config|& tee
log.align_stats_only_example2.txt
```

This will write all alerts to *log.align_stats_only_example2.txt* in the current directory (while also displaying to screen), which contains a lot of useful information. The script will ask the user to confirm that the correct sample list file has been found (respond “yes” if so) and whether the correct number of read files, samples (individuals), and number of sample groups have been properly inferred (respond “yes” if so).

The script will take some time, possible several hours, to finish (feel free to increase the number of processors if more are available). The script can be interrupted with CTRL+C. When finished, check the log file for errors, and if desired, look at the trim or mapping statistics. Proceed to the next *PPalign* stage or skip to *PPstats* (and if desired return to finish *PPalign*; it won’t hurt either way). The file necessary for *PPstats* is called *example2_stats.mpileup*.

c) Running PPalign *only to score SNPs.*

If the above executed properly, the next stage, identifying (“calling”) SNPs, can proceed exactly where the first stage left off. This stage will identify SNPs, filter them according to thresholds in the config file, and then estimate (non-normalized!) allele counts and frequencies for each sample grouping. The config file for this next stage, *pp_align_example2_scoreSNPs.config*, will look very similar to the last file, with the following modifications:

```
#Optional Parameters#
STATSONLY=off
```

Turn this off so the script proceeds to SNP calling.

```
PARALLEL=on
```

divide SNP calling into chunks of chromosomes/scaffolds for GNU parallel; now all the chromosomes and scaffolds will be divided amongst the (available) processors specified with THREADZ. In this example there are only 3 chromosomes, but each is >1 million bp, so even in this example SNP calling will be faster with parallel (turn it off and try if you don’t believe!). In most cases, most base pairs are contained within the “chromosomes”, and specifying more processors than the number of chromosomes will not provide any meaningful increase in speed. For scaffold-only genomes, scaffolds will be randomly sorted (so not all the large scaffolds end up on one processor) and divided in approx. even numbers across processors. Users should note that *PPalign* does not use job scheduling, and assumes that the number of processors specified by THREADZ will actually be available when called. If not, the system will throw an error, and *PPalign* will try **1** additional time to score SNPs on those subsets in parallel, try one more in serial (on a single thread) if any fail a second time, and if any subsets are still missing, will exit with an indication why. It is therefore useful to be judicious with the number of processors specified (THREADZ). Optional states: on/off

With this config file, run *PPalign* as follows*:*

```
> bash ~/bin/poolparty/PPalign.sh pp_align_example2_scoreSNPs.config |& tee
        log.align_scoreSNPs_example2.txt
```

This will write all alerts to *log.align_scoreSNPs_example2.txt* in the current directory (while also displaying to screen), the end of which should specify how many SNPs were found. The script will ask the user to confirm the sample list file etc. We could turn this off since we confirmed it before, but it's good practice to always leave it on.

As indicated, the output from this stage will be a VCF file (and $OUTPOP_variants.txt) listing the variants (SNPs and indels) found, and files containing allele counts ($OUTPOP.sync) and frequencies ($OUTPOP.fz) for those SNPs for each sample group. These counts/frequencies are non-normalized, meaning any imbalance in read numbers across samples (individuals) could bias these numbers. We could therefore proceed directly into the normalization by changing the INDCONT switch in the config file and running *PPalign* as before. However, we used fairly low values for minimum depth and minor allele frequency in this stage. Let's presume we might wish to increase those thresholds before normalization. Instead of re-running *PPalign*, we can just filter the VCF file using any number of utilities and then direct *PPalign* to create new *sync*/*fz* files from those SNPs. (One could do this AND normalize, but for the sake of clarity, we'll divide those steps).

First, for reasons you'll understand below, we should rename the VCF output by *PPalign*. The VCF file *PPalign* produces will be called $OUTPOP.VCF.

```
> mv example2.VCF example2_original.VCF
```

Now let's filter this VCF file with *BCFtools*. With no other flags, VCFtools should report all the variants identified:

```
> bcftools view  -i  'MIN(DP)>'5' & MIN(QUAL)>'20' & MAF[0]> '0.005' '
        example2_original.VCF | grep -v "#" | wc -l
```

As these are the values we specified in the config file, this filtering should return ~2.64 million variants, the number indicated in the log. *BCFtools* is filtering the VCF by depth, minor allele frequency, and quality, 'grep' is getting rid of comment/INFO lines, and 'wc' is counting the number of lines passing those filters. If we increase the depth threshold to 10 (MINDP) and the minor allele frequency threshold to 0.05 (MAF), we get a new VCF file with ~1.57 million variants (which we count again with 'grep' and 'wc').

```
> bcftools view -i  'MIN(DP)>'10' & MIN(QUAL)>'20' & MAF[0]> '0.05' '
        example2_original.VCF > example2_filtered.VCF
> grep -v "#" example2_filtered.VCF | wc -l
```

Feel free to try other depth, qual, or MAF values and see how many SNPs are retained.

d) running PPalign *with a custom variants list*

PPalign will take a list of variants (for that genome assembly!) specified as a VCF file. *PPalign* does not do any filtering of the input VCF, so this must be done by the user ahead of time (see above; note that additional filters can be applied later with *PPanalyze*). *PPalign* will rename this VCF file as $OUTPOP.VCF, so be aware or point the script to a copy that it can rename.

```
> cp example2_filtered.VCF example2_filtered_copy.VCF
```

Also, we need to delete, move, or rename any prior $OUTPOP_variants.txt, $OUTPOP.mpileup, $OUTPOP.sync, and $OUTPOP.fz files that are present. *PPalign* avoids re-creating them unnecessarily, and will skip these steps if it sees them, but in this case, we want the script to recreate them.

```
> mkdir OLD
> mv example2.mpileup example2.sync example2.fz example2_variants.txt OLD
```

To use a VCF as a guide for which sites should be assayed ("scored"), the following parameter must be modified in the config file to point to the VCF file the user wished *PPalign* to use. You will notice the next config file, *pp_align_example2_useVCF.config*, only differs in this line:

```
#Optional Parameters#
USEVCF=example2_filtered_copy.VCF
```

We use the copy to preserve the provenance of the original. Run PPalign similarly to before

```
> bash ~/bin/poolparty/PPalign.sh pp_align_example2_useVCF.config |& tee
        log.align_example2_useVCF.txt
```

This analysis should complete faster than the last one, since the only tasks are re-creation of the variants-only mpileup, sync, and fz files (check that the time stamp on these files has changed). When it's finished, we can query how many SNPs we ended up with (a careful reading of the log file will reveal this too). The following command counts lines in the (non-normalized) *sync* file:

```
> wc -l example2.sync
```

and returns 1,206,451 SNPs. So, we directed *PPalign* to ~1.57 million SNPs and we ended up with 1.2 million. In between SNP calling (or reading the VCF) and allele counting and frequency estimation is the implementation of the indel filtering window, controlled by the INWIN parameter, which omits sites within the specified basepairs to any indel position (set to 3bp in our Example2).

e) running PPalign *to normalize individual contributions*

At this point, if each "sample" corresponded to data from a pooled set of (non-barcoded) individuals, we would move on to *PPstats* and/or *PPanalyze*. But, as described, *PPalign* utilizes a function which down-samples the contributions of barcoded samples to reduce bias ("normalization"). This function proceeds by creating a *sync* file for each population (containing counts for each library or individual), a pseudo-genotyping procedure, and summing these pseudo-genotype counts (see the first example for an explanation of this normalization). Because this procedure requires holding the counts for individuals in memory in R, it can be very demanding on RAM. The RAM requirements of these scripts can be managed through user-tunable subsetting of the data (the DIVIDE parameter), which limits the number of individuals from each sample group (population) held in memory at each step, writing temporary files, and summing together across subsets. While more robust to crashes based on memory availability, the user should be aware this takes approximately linearly longer as the number of subsets increases (specification for DIVIDE gets smaller).

The numbers of individuals in each sample group for Example2 are 59, 48, and 42 for the sample groups 02_LEW_ELEW, 03_WAS_SKAM, and 04_WIL_EAGL respectively. The default for DIVIDE (if left blank in the *config* file) is 100, which will tally counts in a single group for this and most projects. Here, let's use a DIVIDE of 25, which will direct the script to tally each sample group in 3, 2, and 2 subsets, respectively. For normalization, we specify 'on' for the INDCONT parameter in order for DIVIDE to take effect. You'll notice those two parameters have been changed in the next *config* file, *pp_align_example2_norm.config*, and now USEVCF is *empty*.

```
#Run Parameters#
MAF=0.05
MINDP=10

#Run-types#
INDCONT=on

#Optional Parameters#
USEVCF=
DIVIDE=25
```

We can go ahead and change MINDP and MAF to match our VCF filtering, but they would have no and little effect at this stage (the R script that converts allele counts in the *sync* file to allele frequencies in the *fz* file identifies sites below MAF threshold after normalization, but additional MAF filters can also be applied in *PPanalyze*). Run *PPalign* as below:

```
> bash ~/bin/poolparty/PPalign.sh pp_align_example2_norm.config |& tee
        log.align_example2_norm.txt
```

It is worth noting that normalization depends heavily on R scripts, as does the last stage of non-normalized SNP scoring, which converts the allele counts in the *sync* file to allele frequencies in the *fz* file. The disk I/O requirements of very large datasets (dozens of samples per sample group and tens of millions of SNPs) are occasionally more than the system can sustain, and the *sync* to *fz* conversion will fail. To overcome this, we have provided a script, *PP_sync_to_fz.sh*, which converts the sync and writes output line by line. While slower, it is not subject to disk I/O-based crashes. If *PPalign* appears to run smoothly and the *sync* but no *fz* file are produced, try using this script instead. It takes a minor allele frequency as an optional filtering parameter. E.g.

```
> bash ~/bin/poolparty/PPsync_to_fz.sh example2.sync 0.05
```

When *PPalign* and/or *PPsync_to_fz* is finished, you can confirm how many SNPs we ended up with using a similar command as above (e.g. 'wc -l example2_norm.sync'), or perusing the log file.

Contained within the log file is another tidbit we should note for later:

```
ALERT: Normalized sync file order is:
1 = pop_02_LEW_ELEW
2 = pop_03_WAS_SKAM
3 = pop_04_WIL_EAGL
```

This reflects the sorting that *PPalign* did and the column order in the *sync*/*fz* files, i.e. the values the user should specify to *PPanalyze* to direct it which sample groups (populations) to operate upon. This same information is available in the pops/$OUTFILE_files_for_pops.txt file also.

4) PPstats

PPstats is an optional module that produces stats and plots to assess sequencing and alignment performance. It only requires *$OUTFILE_stats.mpileup* created by *PPalign* and the genome index* (*Omy_USDA1.1_Omy05.25.28.fasta.fai*) to generate stats.

a) The config file.

```
#Specify Files#
FAI=Omy_USDA1.1_Omy05.25.28.fasta.fai
MPILEUP=poolparty_example2/example2_stats.mpileup
        Full or relative path to the $OUTPOP_stats.mpileup file
OUTDIR=poolparty_example2/stats
        Full or relative path to the directory to which output files should be written
OUTFILE=example2_stats.txt
        Name of the file containing basic stats.
POPFILE=poolparty_example2/pops/example2_files_for_pops.txt
        Full or relative path to the $OUTPOP_files_for_pops.txt file, used for plotting.

#Parameters#
SCAFP= #leave blank if no scaffolds; otherwise PPstats will throw an error
THREADZ=4
MINCOV=10
        min coverage PER sample group
MAXCOV=1000
        max coverage PER sample group
```

b) Running PPstats

In the directory containing the config file, run:

```
> bash ~/bin/poolparty/PPstats.sh pp_stats_example2.config |& tee log.stats_example2.txt
```

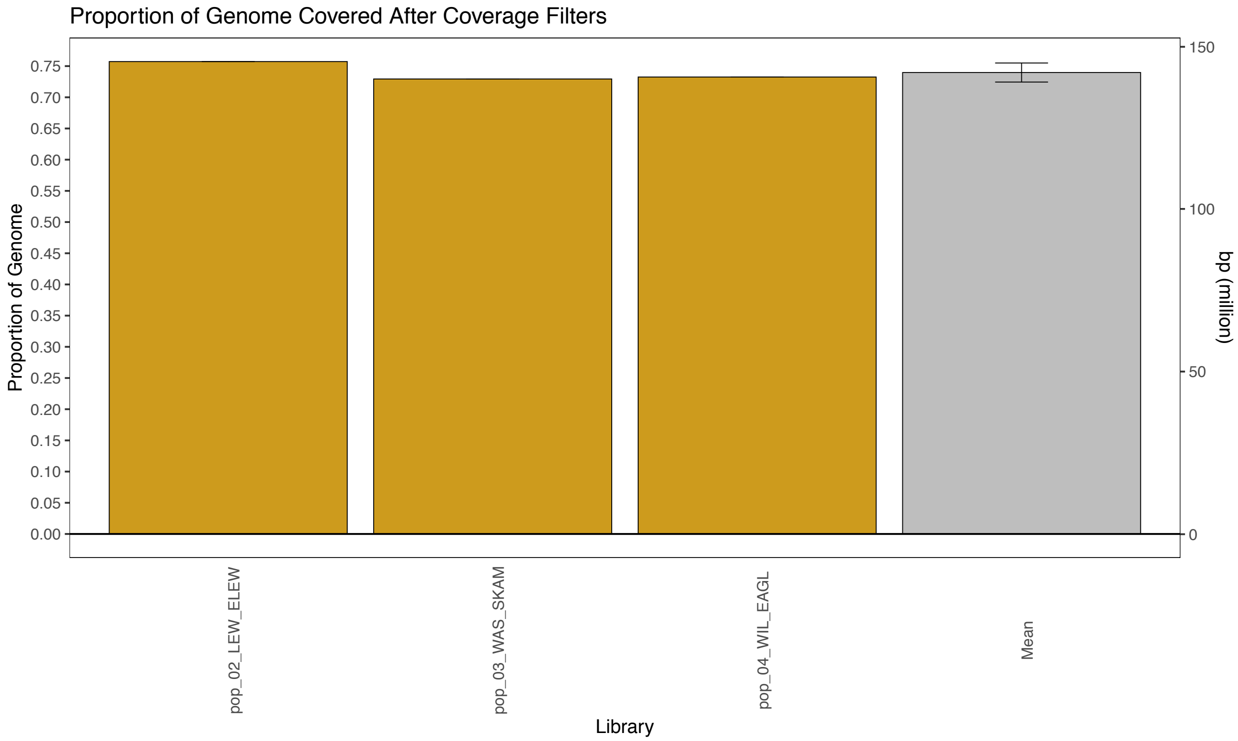
PPstats will provide rough status updates for the 8 blocks it runs through.

c) Output files

In the specified output folder, there will be a number of *txt* files and *pdfs*

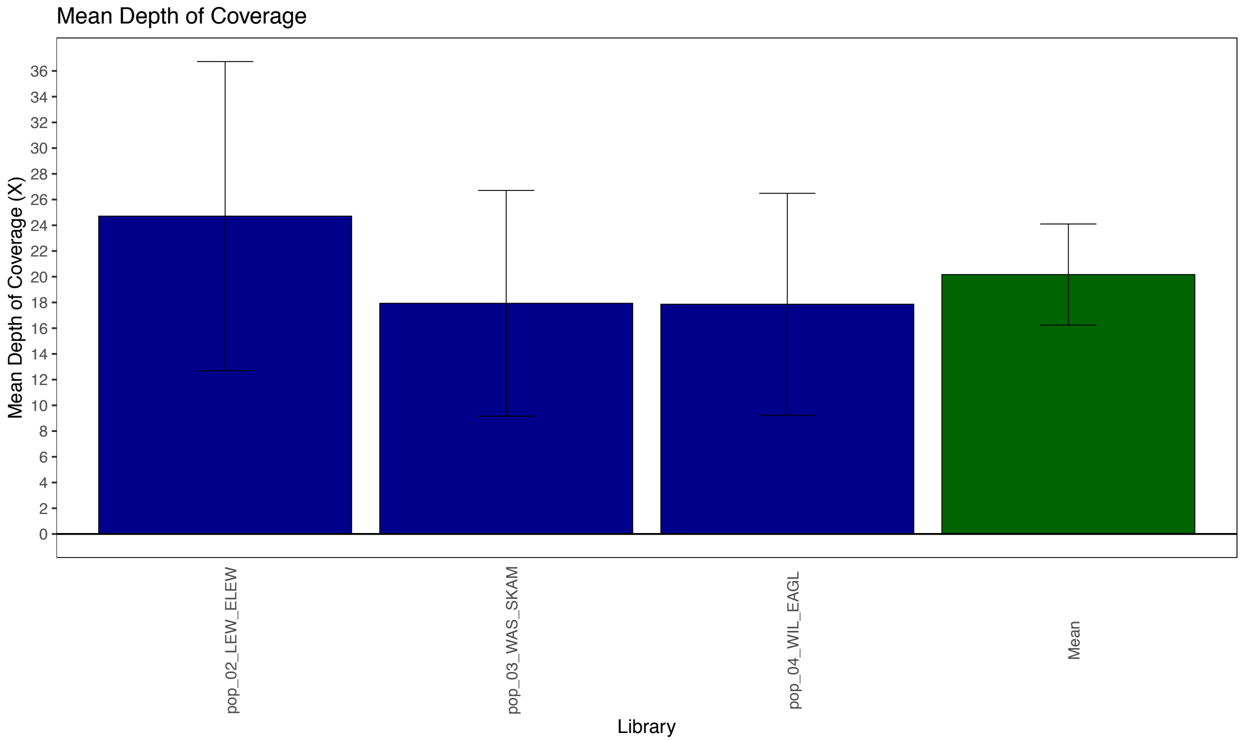
The first example discusses a number of the outputs that *PPstats* produces. Here we will focus on a couple of them.

Take a look at *example2_stats_prop_cov.pdf*



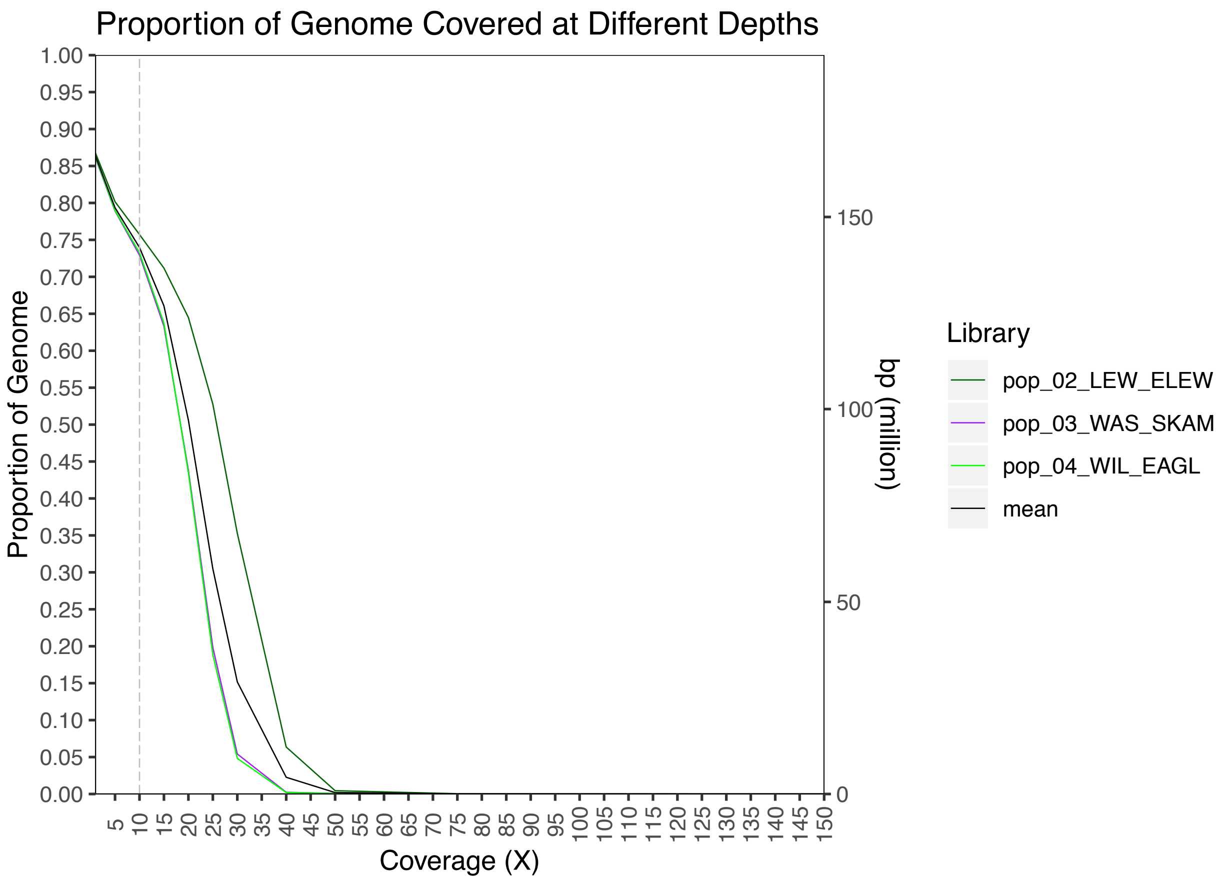
This plot depicts the proportion of the genome (consisting here of 3 chromosomes) that are covered by each of the sample groups, and the mean proportion covered above our specified minimum of 10 (MINCOV). A good rule of thumb to have sufficient overlap among sample groups to enable effective analyses is 50%; we are well above that with these samples. Note that the portion of the genome covered in each sample may not be randomly distributed, but the majority of bases per chromosome are covered at the specified depth.

Take a look at *example2_stats_mean_coverage.pdf*



This plot depicts the mean coverage over all base pairs, with a mean around 20 across sample groups. Another plot, *example2_stats_mean_filt_cov.pdf*, depicts coverage after eliminating sites that are covered at more than MINCOV and less than MAXCOV, should you wish to consider that. However, we may wish to know how the proportion of the genome covered would change if we specified different MINCOV values.

Take a look at *example2_stats_prop_at_covs.pdf*



Not surprisingly, as our minimum standard for coverage increases, the proportion of the genome decreases rapidly, but with a trade-off of increased accuracy in estimates of allele frequencies in the remaining sites. For example, if we imposed a standard of 30X coverage per sample group, we would only be observing about 5% of the genome in two of our sample groups. One of the downsides of including sites with lower coverage is that we are less sure than multiple individuals have been sampled, and thus that allele frequencies are representative of the sample groups we will process with *PPanalyze*. Fortunately, some output from the normalization procedure make it possible to filter out sites with poor individual representation.

5) PPanalyze

PPanalyze is the module of *PoolParty* that and performs comparative analyses with the primary intention of discovering differentiated genomic regions between specified comparisons, or creates input files for additional analyses. *PPanalyze* produces output formats that can quickly be sorted or plotted with various tools.

a) Blacklist for PPanalyze

PPanalyze includes the option to blacklist (omit from consideration or subset) sites present in the input *sync* and *fz* files. This is useful, for example, to include sites that did not pass MAF specifications after normalization or do not sample enough individuals. Some of these files are already prepared by *PPalign*, and others must be created from *PPalign* output. In the latter case, we include a script that will collect the statistics for how many individuals contributed reads to a given site for each sample group (contained in the pop_{POP} _snp_stats.txt files), and create a list of sites falling below a user-specified minimum value (*PPblacklist_minimum_individuals.sh*). Let's run that script as follows:

```
> cd poolparty_example2/inds
> bash ~/bin/poolparty/PPblacklist_minimum_individuals.sh 3
> wc -l min_3_ind_blacklist.txt
```

This produces a file, min_3_ind_blacklist.txt, which lists 56,701 sites that contained reads from fewer than 3 individuals in any sample group (population). I assume you can predict whether more or fewer sites would be identified if we increased this threshold (try it if you wish). The name of the blacklist file will change with the specified minimum.

PPanalyze only accepts a single blacklist, which means the user should combine (uniquely) sites from multiple black lists. We can combine our minimum-individual list with the lists that *PPalign* automatically creates identifying those with more than 2 alleles (indicating likely paralogous sites) in at least half of the sample groups and those not passing the MAF threshold after normalization (though do note that *PPanalyze* applies another MAF filter, but which applies only to the subset of sample groups included in that *PPanalyze* run). First let's see how many sites are included in each of those.

```
> cd ../filters
> wc -l example2_norm_MAF_fail.txt
> wc -l example2_poly_half.txt
```

This indicates that 1,170 and 20,529 sites met these criteria. Each of these files contains only chromosome-*TAB*-position; to combine them together, use some simple Linux utilities to sort and de-duplicate the positions.

```
> cat ../inds/min_3_ind_blacklist.txt example2_norm_MAF_fail.txt example2_poly_half.txt
        | sort | uniq > example2_normMAF_polyHALF_min3_blacklist.txt
> wc -l example2_normMAF_polyHALF_min3_blacklist.txt
```

This combined file has 77,258 sites, not quite the sum of each file, meaning that a few were in fact in two or more sets.

a) The PPanalyze *config file.*

The config file uses some of the previously generated files from *PPalign* to properly filter and analyze user-specified comparisons.

```
#Populations for analysis#
POPS=1:3,2
```

Specify the sample groups (actually columns from the *sync*/*fz* files) that will be compared, separated by a comma (“,“), where colons (“:”) indicate samples whose differences should be subordinated. In this Example2, we wish to highlight differences between a steelhead strain with a history of anthropogenic selection, and ignore differences between the other two (control) populations. Recall the line in the PPalign output (or in pops/$OUTFILE_files_for_pops.txt) that indicates the order of populations in the sync/fz files: our two control lineages are columns 1 and 3, while the experimental lineage is column 2. So the above arrangement focuses on sample groups 1+3 vs. 2, ignoring differences between 1 and 3, as desired. However, be aware that the *sync*/*fz* files created by *PPanalyze* will have columns now re-sorted (again) by this order, i.e. all sample groups on either side of the colon (“:”) will be output first, and then within groups. While no CMH test is run in this example, noting this reformatting would become important when running CMH.

```
#Input files and names#
PREFIX=example2_analyze
COVFILE=poolparty_example2/filters/example2_coverage.txt
SYNC=poolparty_example2/example2_norm.sync
FZFILE=poolparty_example2/example2_norm.fz
BLACKLIST=poolparty_example2/filters/example2_normMAF_polyHALF_min3_blacklist.txt
```

The file we just created by combining several potential blacklists.

```
OUTDIR=poolparty_example2/example2_analysis
```

```
#Types of Analyses#
FST=on
SLIDINGFST=on
FET=on
NJTREE=on
```

Let’s run all of these analyses. In the last case, an NJ tree and PCA of three samples won’t be particularly informative, but the script invoked for this analysis also creates the SNP density files needed for plotting.

```
#Population Parameters#
MINCOV=10
MAXCOV=1000
MAF=0.05
```

These values reflect the filtering of the VCF file that we did, with two notable differences. First, PPanalyze applies the MINCOV standard individuals to each sample group (population), rather than globally. As such many fewer loci tend to pass this filter despite the same value. Second, PPanalyze applies the MAF filter in this comparison specific context, i.e. if a subset of sample groups are included, different (more or fewer) loci will be retained by this standard. MAXCOV is somewhat arbitrary here, but should be set to filter potentially paralogous loci, e.g. as some number of standard deviations above the median coverage (e.g. counted in the $OUTPOP_stats.mpileup with *awk*).

```
#FST Parameters#
FSTTYPE=karlsson
WINDOW=5000
STEP=500
NIND=150
```

```
#NJ Tree Parameters
STRWINDOW=10000
BSTRAP=2000
AFFILT=0.9
METHOD="mean"
```

See description in the first example. The AFFILT standard, only applied in the construction of PCA and NJ plots, is a useful approach to get rid of the majority of outlier loci so that these plots reflect mainly neutral population structure (in theory). For example, a value of AFFILT=0.9 would remove sites with strong differences in allele frequencies leaving only sites with allele frequencies <0.9. It may also be useful to note that, were the loadings from the PCA (showing contribution of individual alleles/loci to the observed patterns) of interest, all the METHODs available refer to real loci *except* “mean”, i.e. in those cases the loadings are artificial and only represent compound signal.

```
#Dependencies#
POOL2=~ /bin/popoolation2_1201
```

b) Running PPanalyze

Run *PPanalyze* similar to other modules, from the directory with the config file:

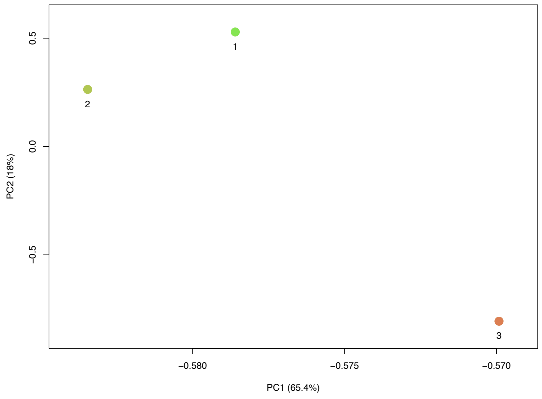
```
> bash ~/bin/poolparty/PPanalyze.sh pp_analyze_example2.config |& tee
log.analyze_example2.txt
```

Like *PPalign*, *PPanalyze* will try to avoid repeating things it has already done (with the same file prefix), and skip re-writing files already written. If repeating analysis with different parameters, use different OUTDIR and/or PREFIX names.

Alerts are printed that indicate important items, like the number of SNPs removed by various filters. Another important ALERT here is “ALERT: Pops 1 3 2 are now in the order of 1 2 3 in the subset sync file.” This line indicates the new order of samples after analyzing and/or sub-setting, i.e. in the $PREFIX.sync and fz files. As these are useful for the CMH and other tests, it would be important to note (and input for CMH) the NEW order of samples (columns).

d) Output files

PPanalyze creates subset *sync* and *fz* files for additional analyses, along with the outputs of FST and Fisher’s Exact Test that are useful in additional analyses. Just out of curiosity, let’s take a look at the PCA.



Intriguingly, in the neutral structure defined by PC1, which explains most of the genetic variance, the test lineage (Pop 2) is more similar to one of the control lineages (Pop 1) than the latter is to the other control lineage (Pop 3). This bodes well for identifying loci associated with the history of selection in this stock rather than artifacts of the comparison.

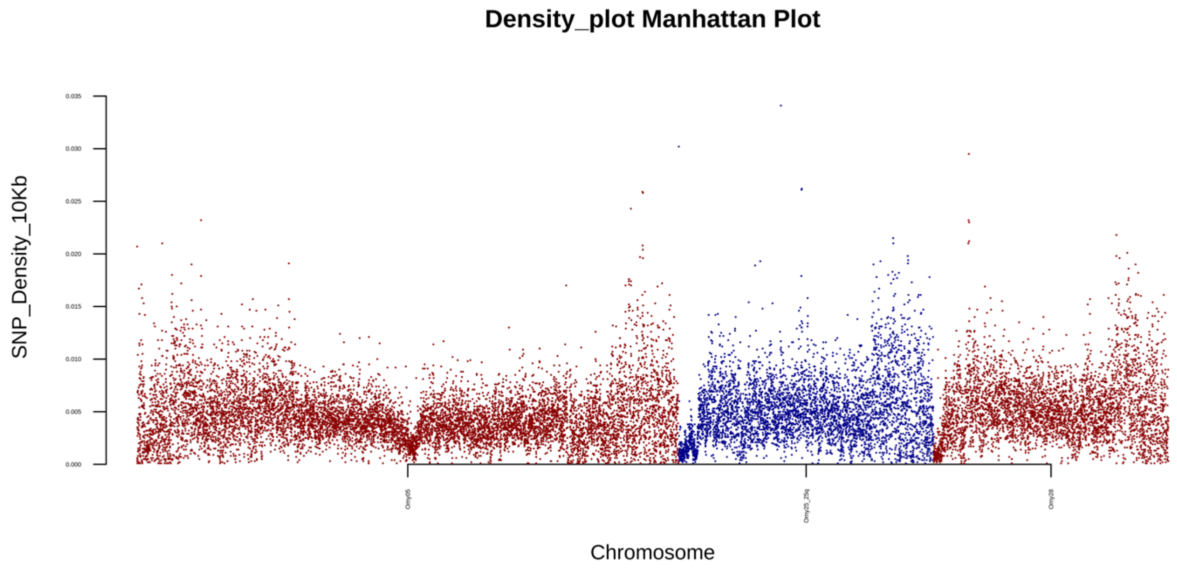
e) *Plotting results*

Provided with *PoolParty* is the PPmanhat.sh utility module, which uses *qqman* in R (Turner 2014). PPmanhat can directly plot a number of *PoolParty* outputs such as F_{ST}, FET, SF_{ST}, CMH, FLK, LocalScore, and SNP-density outputs. We also provide a script to run standard analysis plots for all chromosomes, *PPrun_analysis_plots.sh*, from within the analysis directory, to streamline plotting (see the first example for customization options). This script requires several command line inputs: the analysis prefix given to *PPanalyze*, whether or not to perform a CMH analysis (yes/no), whether or not to perform a FLK but no CMH analysis (yes/no), the path to the *PPmanhat* script, the path to the Popoolations2 folder, and optionally the prefix for assembly scaffolds.

```
> cd poolparty_example2/example2_analysis
> bash ~/bin/poolparty/PPrun_analysis_plots.sh
> bash ~/bin/poolparty/PPrun_analysis_plots.sh example2_analyze no no ~/bin/poolparty/
        ~/bin/popoolation2_1201
```

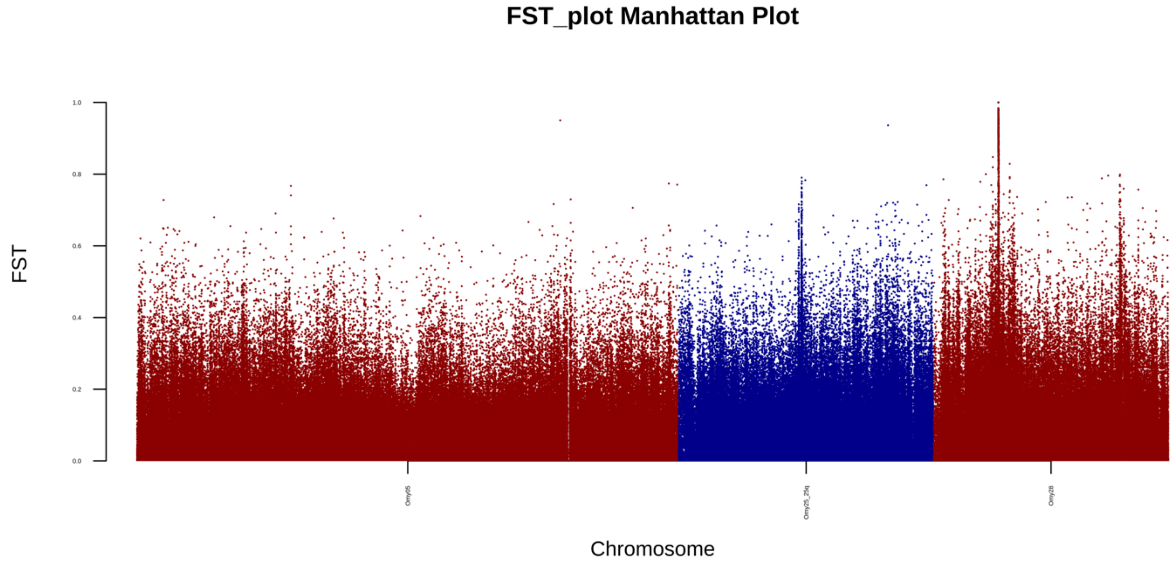
For this Example2 data set, we decline to run CMH (we would need at least two pairs of "replicates"), decline to run FLK (requires 4+ populations) and leave the scaffold prefix blank (since we have none; same if we were using an 'all-scaffold' genome). Once this runs, let's look at some of the results.

Take a look at Density_plot.png



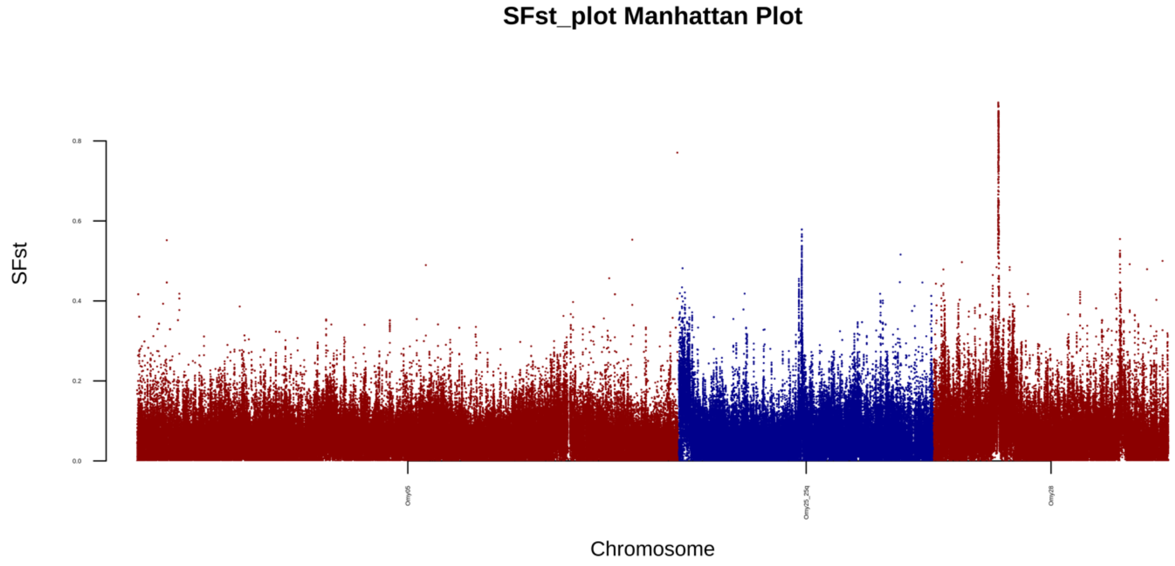
This plot shows the density and evenness of SNPs across our three chromosomes. Though some spots seem depauperate (center of Omy05, which is known from previous studies to contain one or more large inversions; beginning of Omy25 and Omy28), and a few are higher, the entire chromosomes are fairly well covered. If we should observe outlier loci in any of the especially poor or dense regions, this *might* indicate artifacts from assembly, mapping to repeat regions, or structural variants.

Take a look at Fst_plot.png



This plot shows raw F_{ST} of each SNP across chromosomes. There is a wide variation in F_{ST} across sites, and a few more and less dense areas of high FST. One challenge with raw F_{ST} is that individual loci may have high (or low) F_{ST} for a multitude of technical reasons and implying nothing about our axis of interest (test vs. control), so separating the wheat from the chafe is important. There are several further analyses intended to improve signal to noise ratio from allele frequency data. The first way to do this is to look at the mean F_{ST} of SNPs within sliding windows rather than individual SNPs.

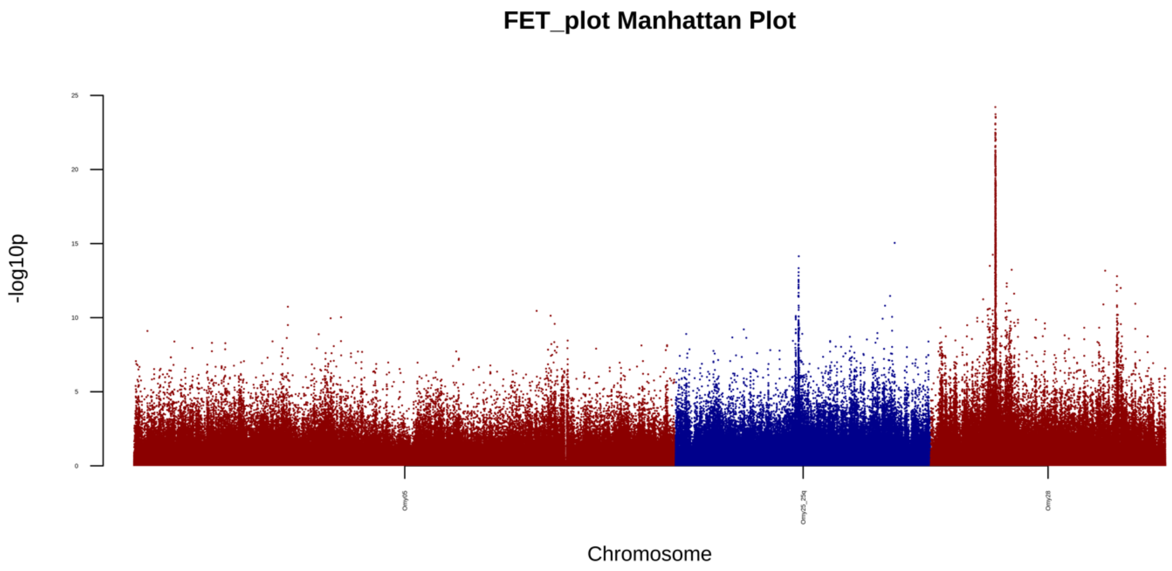
Take a look at SFst_plot.png



This plot shows that FST has been considerably smoothed (and generally lowered) across most of the genome. A few large peaks remain and much of the noise has been decreased from individual SNPs. One of those large peaks is at the beginning of Omy25 (the second chromosome in blue). We should be wary of this peak, since we know there is a lower SNP density here and the peak is not well defined. In contrast, we see additional large peaks on Omy25 and Omy28, and many much smaller peaks across all three chromosomes.

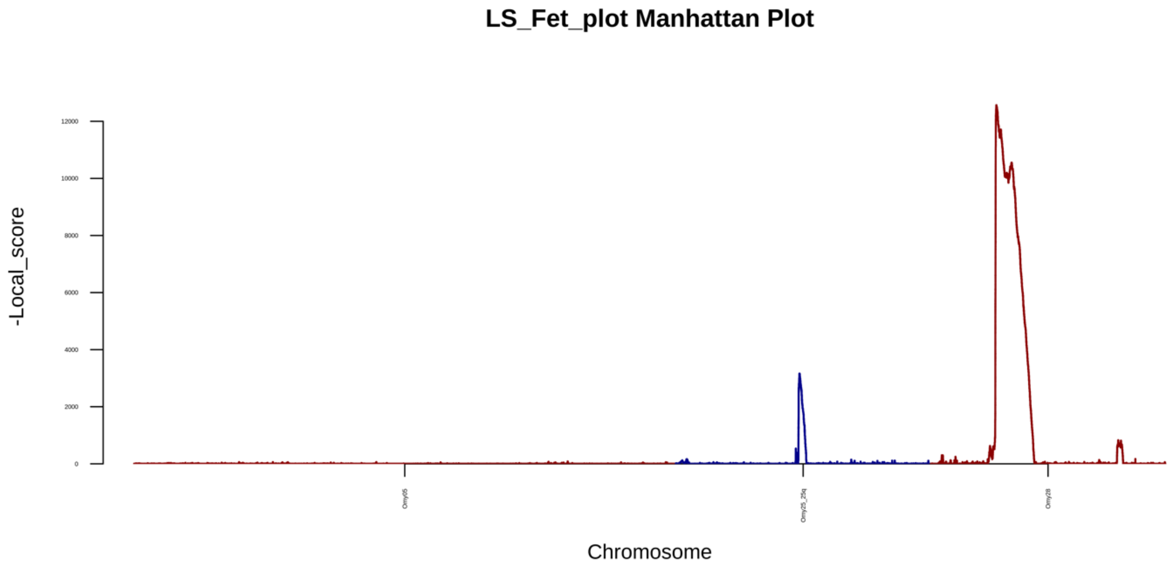
While Fst and SFst are useful indicators of differences in allele frequencies, Fisher's exact test (FET) provides estimates of p-values that may improve signal to noise since some differences in allele frequency and Fst may not be statistically significant.

Here is FET_plot.png:



There are some clear peaks in this plot, as above, in fact quite a few peaks, which makes it hard to know which few to focus on, and where each begins and ends. Another way of looking at these data would be testing for continuous regions of outliers (i.e. SNPs linked due to selection), such as by using the local score test, which attempts to factor out background rates of significance in *p*-values such as from Fisher's exact test or the Cochran–Mantel–Haenszel (CMH) test. If you choose to run local score, we strongly recommend reading Fariello *et al.* 2017 for guidance in choosing appropriate smoothing parameters (*X*), but our script does a fair job at estimating appropriate starting values.

Take a look at LS_Fet_plot.png



This plot identifies regions that stand out from the background rate of significance, in this case for significance values from Fisher's exact test (FET), though another variation is to run local score on p-values from the CMH test. While there are no peaks on Omy05, there is a strong peak on Omy25, a very strong peak on Omy28, and smaller peaks on both of these chromosomes. The local score script lists in a text file, at several significance values, the positions of the regions identified (e.g. LS_Fet_run_001sig.txt). One can use this information to identify any genomic positions with known or putative function, such as by querying a genomic feature file that contains predicted coding regions. Note that analyses with local score are susceptible to false positive (Type I error) even with the incorporated Bonferroni correction (Fariello *et al.* 2017), so *a priori* critical values should be considered carefully depending on the study design.

Our analysis has indicated that there are regions that strongly diverge between the two control lineages and the one with a known history of selection on chromosomes Omy25 and Omy28. These regions contain loci known to be associated with migration phenology, specifically age-at-maturity (age at first return migration) and run-timing (early or late in the year). As mentioned, fish in this stock have a history of being strongly graded for fish that return after two years in the ocean (as opposed to one) and during the summer (as opposed to winter). This appears to have resulted in strong selection on these genomic regions relative to other stocks.

6) References

Bonhomme, M., Chevalet, C., Servin, B., Boitard, S., Abdallah, J., Blott, S., & SanCristobal, M. (2010). Detecting selection in population trees: the Lewontin and Krakauer test extended. *Genetics*, 186(1), 241-262.

Fariello, M. I., Boitard, S., Mercier, S., Robelin, D., Faraut, T., Arnould, C., ... & Gourichon, D. (2017). Accounting for Linkage Disequilibrium in genome scans for selection without individual genotypes: the local score approach. *Molecular Ecology*.

Karlsson, E. K., Baranowska, I., Wade, C. M., Hillbertz, N. H. S., Zody, M. C., Anderson, N., ... & Comstock, K. E. (2007). Efficient mapping of mendelian traits in dogs through genome-wide association. Nature genetics, 39(11), 1321.

Krzywinski, M., Schein, J., Birol, I., Connors, J., Gascoyne, R., Horsman, D., ... & Marra, M. A. (2009). Circos: an information aesthetic for comparative genomics. *Genome research*, *19*(9), 1639-1645.

Turner, S.D. qqman: an R package for visualizing GWAS results using Q-Q and manhattan plots. biorXiv DOI: 10.1101/005165 (2014).

Wiberg, R. A. W., Gaggiotti, O. E., Morrissey, M. B., & Ritchie, M. G. (2017). Identifying consistent allele frequency differences in studies of stratified populations. *Methods in Ecology and Evolutio*