CMS Documentation

Release 2.0.0-beta-303-g4743ffd

Broad Institute

September 26, 2018

Contents

CHAPTER 1

Contents

About CMS 2.0

Composite of Multiple Signals (CMS) refers to a family of tests applied to population genetic datasets in order to (i) identify genomic regions that may have been subject to strong recent positive selection (a 'sweep') and (ii) to narrow signals of selection within such regions, in order to identify tractable lists of candidate variants for experimental scrutiny. In both of these cases, CMS requires (a) phased variation data for several populations, along with (b) the identity of the ancestral allele for a majority of sites listed. It was developed with humans in mind (e.g., the 1000 Genomes Project) but could in principle be applied to any diploid species with data in VCF or TPED format.

In its current instantiation (**'CMS 2.0'**), it includes scripts to (i) calculate a variety of selection metrics for each population, (ii) model the demographic history of the dataset using an exploratory approach, (iii) generate probability distributions for each selection metric from data simulated from demographic models, (iv) generate composite scores and (v) visualize signals of selection in the UCSC Genome Browser.

Background

The method used in CMS is described in greater detail in the following papers:

A Composite of Multiple Signals distinguishes causal variants in regions of positive selection Sharon R. Grossman, Ilya Shylakhter, Elinor K. Karlsson, Elizabeth H. Byrne, Shannon Morales, Gabriel Frieden, Elizabeth Hostetter, Elaine Angelino, Manuel Garber, Or Zuk, Eric S. Lander, Stephen F. Schaffner, and Pardis C. Sabeti *Science* 12 February 2010: **327** (5967), 883-886.Published online 7 January 2010 [DOI:10.1126/science.1183863]

Identifying recent adaptations in large-scale genomic data Grossman SR, Andersen KG, Shlyakhter I, Tabrizi S, Winnicki S, Yen A, Park DJ, Griesemer D, Karlsson EK, Wong SH, Cabili M, Adegbola RA, Bamezai RN, Hill AV, Vannberg FO, Rinn JL; 1000 Genomes Project, Lander ES, Schaffner SF, Sabeti PC. *Cell* 14 February 2013: **152** (4), 883-886.Published online 7 January 2010 [DOI:10.1016/j.cell.2013.01.035]

Coalescent simulations

CMS uses simulated population genetic data for a variety of purposes. For the purpose of flexibility, this pipeline is optimized for use with cosi 2, but it would theoretically be straightforward to substitute other simulators supporting selection.

Cosi2: an efficient simulator of exact and approximate coalescent with selection. Shlyakhter I, Sabeti PC, Schaffner SF. *Bioinformatics* 1 December 2014: **30** (23), 3427-9.Published online 22 August 2014 [DOI:10.1093/bioinformatics/btu562]

Installation

Step 1: Install Conda

To use conda, you need to install the Conda package manager which is most easily obtained via the Miniconda Python distribution. Miniconda can be installed to your home directory without admin priviledges.

Step 2: Configure Conda

Software used by the cms project is distributed through the bioconda channel for the conda package manager. It is necessary to add this channel to the conda config:

```
conda config --add channels bioconda
conda config --add channels r
conda config --add channels conda-forge
```

Step 3: Make a conda environment and install cms

It is recommended to install cms into its own conda directory. This ensures its dependencies do not interfere with other conda packages installed on your system. First clone the source repository from Github:

```
git clone git@github.com:broadinstitute/cms.git
```

A new conda environment can be created with the following command, which will also install relevant cms dependencies. It is recommended to use the Python3 version of the environment file:

```
conda env create -f conda-environment_py3.yaml -n cms-env3
```

Step 4: Activate the cms environment

In order to use cms, you will need to activate its conda environment:

```
source activate cms-env3
```

Command line tools

scans.py

This script contains command-line utilities for calculating EHH-based scans for positive selection in genomes, including EHH, iHS, and XP-EHH.

```
usage: scans.py subcommand
```

Sub-commands:

selscan_file_conversion

Process a bgzipped-VCF (such as those included in the Phase 3 1000 Genomes release) into a gzip-compressed tped file of the sort expected by selscan.

```
usage: scans.py selscan_file_conversion [-h] [--startBp STARTBP]
                                        [--endBp ENDBP] [--ploidy PLOIDY]
                                        [--considerMultiAllelic]
                                        [--rescaleGeneticDistance]
                                        [--includeLowQualAncestral]
                                        [--codingFunctionClassFile CODINGFUNCTIONCL
                                        [--sampleMembershipFile SAMPLEMEMBERSHIPFIL
                                        [--filterPops FILTERPOPS [FILTERPOPS ...]]
```

```
[--filterSuperPops FILTERSUPERPOPS [FILTERS
[--loglevel {DEBUG,INFO,WARNING,ERROR,CRITI
[--version] [--tmpDir TMPDIR]
[--tmpDirKeep]
inputVCF genMap outPrefix outLocation
chromosomeNum
```

Positional arguments:

inputVCF	Input VCF file
genMap	Genetic recombination map tsv file with four columns: (Chromosome, Position(bp), Rate(cM/Mb), Map(cM))
outPrefix	Output file prefix
outLocation	Output location
chromosomeNum	Chromosome number.

Options:

--startBp=1	Coordinate in bp of start position. (default: %(default)s).
--endBp	Coordinate in bp of end position.
--ploidy=2	Number of chromosomes expected for each genotype. (default: %(default)s).
--considerMultiAllelic=False	Include multi-allelic variants in the output as separate records
--rescaleGeneticDistance=False	Genetic distance is rescaled to be out of 100.0 cM
--includeLowQualAncestral=False	Include variants where the ancestral information is low-quality (as indicated by lower-case x for AA=x in the VCF info column) (default: %(default)s).
--codingFunctionClassFile	A python class file containing a function used to code each genotype as '1' and '0'. coding_function(current_value, reference_allele, alternate_allele, ancestral_allele)
--sampleMembershipFile	The call sample file containing four columns: sample, pop, super_pop, gender
--filterPops	Populations to include in the calculation (ex. "FIN")
--filterSuperPops	Super populations to include in the calculation (ex. "EUR")
--loglevel=DEBUG	Verboseness of output. [default: %(default)s]
	Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION
--version, -V	show program's version number and exit
--tmpDir=/tmp	Base directory for temp files. [default: %(default)s]
--tmpDirKeep=False	Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_ehh

Perform selscan's calculation of EHH.

```
usage: scans.py selscan_ehh [-h] [--gapScale GAPSCALE] [--maf MAF]
                            [--threads THREADS] [--window WINDOW]
                            [--cutoff CUTOFF] [--maxExtend MAXEXTEND]
                            [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXCEPTIO
                            [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                            inputTped outFile locusID
```

Positional arguments:

inputTped	Input tped file
outFile	Output filepath
locusID	The locus ID

Options:

--gapScale=20000	Gap scale parameter in bp. If a gap is encountered between two snps > GAP_SCALE and < MAX_GAP, then the genetic distance is scaled by GAP_SCALE/GA (default: %(default)s).
--maf=0.05	Minor allele frequency. If a site has a MAF below this value, the program will not use it as a core snp. (default: %(default)s).
--threads=1	The number of threads to spawn during the calculation. Partitions loci across threads. (default: %(default)s).
--window=100000	When calculating EHH, this is the length of the window in bp in each direction from the query locus (default: %(default)s).
--cutoff=0.05	The EHH decay cutoff (default: %(default)s).
--maxExtend=1000000	The maximum distance an EHH decay curve is allowed to extend from the core. Set <= 0 for no restriction. (default: %(default)s).
--loglevel=DEBUG	Verboseness of output. [default: %(default)s] Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION
--version, -V	show program's version number and exit
--tmpDir=/tmp	Base directory for temp files. [default: %(default)s]
--tmpDirKeep=False	Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_ihs

Perform selscan's calculation of iHS.

```
usage: scans.py selscan_ihs [-h] [--gapScale GAPSCALE] [--maf MAF]
                            [--threads THREADS] [--skipLowFreq]
                            [--dontWriteLeftRightiHH] [--truncOk]
                            [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXCEPTIO
                            [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                            inputTped outFile
```

Positional arguments:

inputTped	Input tped file
outFile	Output filepath

Options:

--gapScale=20000 Gap scale parameter in bp. If a gap is encountered between two snps > GAP_SCALE and < MAX_GAP, then the genetic distance is scaled by GAP_SCALE/GA (default: %(default)s).

--maf=0.05 Minor allele frequency. If a site has a MAF below this value, the program will not use it as a core snp. (default: %(default)s).

--threads=1 The number of threads to spawn during the calculation. Partitions loci across threads. (default: %(default)s).

--skipLowFreq=False Do not include low frequency variants in the construction of haplotypes (default: %(default)s).

--dontWriteLeftRightiHH=False When writing out iHS, do not write out the constituent left and right ancestral and derived iHH scores for each locus.(default: %(default)s).

--truncOk=False If an EHH decay reaches the end of a sequence before reaching the cutoff, integrate the curve anyway. Normal function is to disregard the score for that core. (default: %(default)s).

--loglevel=DEBUG Verboseness of output. [default: %(default)s]

Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION

--version, -V show program's version number and exit

--tmpDir=/tmp Base directory for temp files. [default: %(default)s]

--tmpDirKeep=False Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_nsl

Perform selscan's calculation of nSL.

```
usage: scans.py selscan_nsl [-h] [--gapScale GAPSCALE] [--maf MAF]
                            [--threads THREADS] [--truncOk]
                            [--maxExtendNsl MAXEXTENDNSL]
                            [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXCEPTIO
                            [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                            inputTped outFile
```

Positional arguments:

inputTped Input tped file

outFile Output filepath

Options:

--gapScale=20000 Gap scale parameter in bp. If a gap is encountered between two snps > GAP_SCALE and < MAX_GAP, then the genetic distance is scaled by GAP_SCALE/GA (default: %(default)s).

--maf=0.05 Minor allele frequency. If a site has a MAF below this value, the program will not use it as a core snp. (default: %(default)s).

--threads=1 The number of threads to spawn during the calculation. Partitions loci across threads. (default: %(default)s).

--truncOk=False If an EHH decay reaches the end of a sequence before reaching the cutoff, integrate the curve anyway. Normal function is to disregard the score for that core. (default: %(default)s).

--maxExtendNsl=100 The maximum distance an nSL haplotype is allowed to extend from the core. Set <= 0 for no restriction. (default: %(default)s).

--loglevel=DEBUG Verboseness of output. [default: %(default)s]

Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION

--version, -V show program's version number and exit

--tmpDir=/tmp Base directory for temp files. [default: %(default)s]

--tmpDirKeep=False Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_xpehh

Perform selscan's calculation of XPEHH.

```
usage: scans.py selscan_xpehh [-h] [--gapScale GAPSCALE] [--maf MAF]
                              [--threads THREADS] [--truncOk]
                              [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXCEPT
                              [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                              inputTped outFile inputRefTped
```

Positional arguments:

inputTped Input tped file

outFile Output filepath

inputRefTped Input tped for the reference population to which the first is compared

Options:

--gapScale=20000 Gap scale parameter in bp. If a gap is encountered between two snps > GAP_SCALE and < MAX_GAP, then the genetic distance is scaled by GAP_SCALE/GA (default: %(default)s).

--maf=0.05 Minor allele frequency. If a site has a MAF below this value, the program will not use it as a core snp. (default: %(default)s).

--threads=1 The number of threads to spawn during the calculation. Partitions loci across threads. (default: %(default)s).

--truncOk=False If an EHH decay reaches the end of a sequence before reaching the cutoff, integrate the curve anyway. Normal function is to disregard the score for that core. (default: %(default)s).

--loglevel=DEBUG Verboseness of output. [default: %(default)s]

Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION

--version, -V show program's version number and exit

--tmpDir=/tmp Base directory for temp files. [default: %(default)s]

--tmpDirKeep=False Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_norm_nsl Undocumented

Normalize Selscan's nSL output

```
usage: scans.py selscan_norm_nsl [-h] [--bins BINS]
                                 [--critPercent CRITPERCENT]
                                 [--critValue CRITVALUE] [--minSNPs MINSNPS]
                                 [--qbins QBINS] [--winSize WINSIZE] [--bpWin]
                                 [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXC
                                 [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                                 inputFiles [inputFiles ...]
```

Positional arguments:

inputFiles	A list of files delimited by whitespace for joint normalization. Expected format for iHS/nSL files (no header): <locus name> <physical pos> <freq> <ihh1/sL1> <ihh2/sL0> <ihs/nsl> Expected format for XP-EHH files (one line header): <locus name> <physical pos> <genetic pos> <freq1> <ihh1> <freq2> <ihh2> <xpehh>

Options:

--bins=100	The number of frequency bins in [0,1] for score normalization (default: %(default)s)
--critPercent=-1.0	Set the critical value such that a SNP with iHS in the most extreme CRIT_PERCENT tails (two-tailed) is marked as an extreme SNP. Not used by default (default: %(default)s)
--critValue=2.0	Set the critical value such that a SNP with |iHS| > CRIT_VAL is marked as an extreme SNP. Default as in Voight et al. (default: %(default)s)
--minSNPs=10	Only consider a bp window if it has at least this many SNPs (default: %(default)s)
--qbins=20	Outlying windows are binned by number of sites within each window. This is the number of quantile bins to use. (default: %(default)s)
--winSize=100000	GThe non-overlapping window size for calculating the percentage of extreme SNPs (default: %(default)s)
--bpWin=False	If set, will use windows of a constant bp size with varying number of SNPs
--loglevel=DEBUG	Verboseness of output. [default: %(default)s] Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION
--version, -V	show program's version number and exit
--tmpDir=/tmp	Base directory for temp files. [default: %(default)s]
--tmpDirKeep=False	Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_norm_ihs Undocumented

Normalize Selscan's iHS output

```
usage: scans.py selscan_norm_ihs [-h] [--bins BINS]
                                 [--critPercent CRITPERCENT]
                                 [--critValue CRITVALUE] [--minSNPs MINSNPS]
                                 [--qbins QBINS] [--winSize WINSIZE] [--bpWin]
                                 [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,EXC
                                 [--version] [--tmpDir TMPDIR] [--tmpDirKeep]
                                 inputFiles [inputFiles ...]
```

Positional arguments:

inputFiles A list of files delimited by whitespace for joint normalization. Expected format for iHS/nSL files (no header):

<locus name> <physical pos> <freq> <ihh1/sL1> <ihh2/sL0> <ihs/nsl> Expected format for XP-EHH files (one line header): <locus name> <physical pos> <genetic pos> <freq1> <ihh1> <freq2> <ihh2> <xpehh>

Options:

--bins=100 The number of frequency bins in [0,1] for score normalization (default: %(default)s)

--critPercent=-1.0 Set the critical value such that a SNP with iHS in the most extreme CRIT_PERCENT tails (two-tailed) is marked as an extreme SNP. Not used by default (default: %(default)s)

--critValue=2.0 Set the critical value such that a SNP with |iHS| > CRIT_VAL is marked as an extreme SNP. Default as in Voight et al. (default: %(default)s)

--minSNPs=10 Only consider a bp window if it has at least this many SNPs (default: %(default)s)

--qbins=20 Outlying windows are binned by number of sites within each window. This is the number of quantile bins to use. (default: %(default)s)

--winSize=100000 GThe non-overlapping window size for calculating the percentage of extreme SNPs (default: %(default)s)

--bpWin=False If set, will use windows of a constant bp size with varying number of SNPs

--loglevel=DEBUG Verboseness of output. [default: %(default)s]

Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION

--version, -V show program's version number and exit

--tmpDir=/tmp Base directory for temp files. [default: %(default)s]

--tmpDirKeep=False Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

selscan_norm_xpehh Undocumented

Normalize Selscan's XPEHH output

```
usage: scans.py selscan_norm_xpehh [-h] [--bins BINS]
                                   [--critPercent CRITPERCENT]
                                   [--critValue CRITVALUE] [--minSNPs MINSNPS]
```

```
                                        [--qbins QBINS] [--winSize WINSIZE]
                                        [--bpWin]
                                        [--loglevel {DEBUG,INFO,WARNING,ERROR,CRITICAL,E
                                        [--version] [--tmpDir TMPDIR]
                                        [--tmpDirKeep]
                                        inputFiles [inputFiles ...]
```

Positional arguments:

inputFiles A list of files delimited by whitespace for joint normalization. Expected format for iHS/nSL files (no header):

<locus name> <physical pos> <freq> <ihh1/sL1> <ihh2/sL0> <ihs/nsl> Expected format for XP-EHH files (one line header): <locus name> <physical pos> <genetic pos> <freq1> <ihh1> <freq2> <ihh2> <xpehh>

Options:

--bins=100 The number of frequency bins in [0,1] for score normalization (default: %(default)s)

--critPercent=-1.0 Set the critical value such that a SNP with iHS in the most extreme CRIT_PERCENT tails (two-tailed) is marked as an extreme SNP. Not used by default (default: %(default)s)

--critValue=2.0 Set the critical value such that a SNP with |iHS| > CRIT_VAL is marked as an extreme SNP. Default as in Voight et al. (default: %(default)s)

--minSNPs=10 Only consider a bp window if it has at least this many SNPs (default: %(default)s)

--qbins=20 Outlying windows are binned by number of sites within each window. This is the number of quantile bins to use. (default: %(default)s)

--winSize=100000 GThe non-overlapping window size for calculating the percentage of extreme SNPs (default: %(default)s)

--bpWin=False If set, will use windows of a constant bp size with varying number of SNPs

--loglevel=DEBUG Verboseness of output. [default: %(default)s]

Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION

--version, -V show program's version number and exit

--tmpDir=/tmp Base directory for temp files. [default: %(default)s]

--tmpDirKeep=False Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

store_selscan_results_in_db

Aggregate results from selscan in to a SQLite database via helper JSON metadata file.

```
usage: scans.py store_selscan_results_in_db [-h]
                                            [--loglevel {DEBUG,INFO,WARNING,ERROR,C
                                            [--version] [--tmpDir TMPDIR]
                                            [--tmpDirKeep]
```

```
                                                    inputFile outFile
```

Positional arguments:

inputFile	Input *.metadata.json file
outFile	Output SQLite filepath

Options:

--loglevel=INFO	Verboseness of output. [default: %(default)s] Possible choices: DEBUG, INFO, WARNING, ERROR, CRITICAL, EXCEPTION
--version, -V	show program's version number and exit
--tmpDir=/tmp	Base directory for temp files. [default: %(default)s]
--tmpDirKeep=False	Keep the tmpDir if an exception occurs while running. Default is to delete all temp files at the end, even if there's a failure.

cms_modeller.py

This script contains command-line utilities for exploratory fitting of demographic models to population genetic data.

```
usage: cms_modeller.py [-h] {target_stats,bootstrap,point,grid,optimize} ...
```

Sub-commands:

target_stats Perform per-site(/per-site-pair) calculations of population summary statistics for model target values.

```
usage: cms_modeller.py target_stats [-h] [--freqs] [--ld] [--fst]
                                    [--modelpath MODELPATH] [--printOnly]
                                    inputTpeds recomFile regions out
```

Positional arguments:

inputTpeds	comma-delimited list of unzipped input tped files (only one file per pop being modelled; must run chroms separately or concatenate)
recomFile	file defining recombination map for input
regions	tab-separated file with putative neutral regions
out	outfile prefix

Options:

--freqs=False	calculate summary statistics from within-population allele frequencies
--ld=False	calculate summary statistics from within-population linkage disequilibrium
--fst=False	calculate summary statistics from population comparison using allele frequencies
--modelpath=cms/model/	path to model directory containing executables
--printOnly=False	print rather than execute pipeline commands

bootstrap Perform bootstrap estimates of population summary statistics from per-site(/per-site-pair) calculations in order to finalize model target values.

```
usage: cms_modeller.py bootstrap [-h] [--in_freqs IN_FREQS]
                                 [--nFreqHistBins NFREQHISTBINS]
                                 [--in_ld IN_LD]
                                 [--mafcutoffdprime MAFCUTOFFDPRIME]
                                 [--nphysdisthist NPHYSDISTHIST]
                                 [--in_fst IN_FST]
                                 [--ngendisthist NGENDISTHIST]
                                 nBootstrapReps out
```

Positional arguments:

nBootstrapReps	number of bootstraps to perform in order to estimate standard error of the dataset (should converge for reasonably small n)
out	outfile prefix

Options:

--in_freqs	comma-delimited list of infiles with per-site calculations for population. One file per population – for bootstrap estimates of genome-wide values, should first concatenate per-chrom files
--nFreqHistBins=6	number of bins for site frequency spectrum and p(der|freq)
--in_ld	comma-delimited list of infiles with per-site-pair calculations for population. One file per population – for bootstrap estimates of genome-wide values, should first concatenate per-chrom files
--mafcutoffdprime=0.2	for D' calculations, only use sites with MAF > mafcutoffdprime
--nphysdisthist=14	nbins for r2 LD calculations
--in_fst	comma-delimited list of infiles with per-site calculations for population pair. One file per population-pair – for bootstrap estimates of genome-wide values, should first concatenate per-chrom files
--ngendisthist=17	nbins for D' LD calculations

point Run simulates of a point in parameter-space.

```
usage: cms_modeller.py point [-h] [--cosiBuild COSIBUILD]
                             [--dropSings DROPSINGS] [--genmapRandomRegions]
                             [--stopAfterMinutes STOPAFTERMINUTES]
                             [--calcError CALCERROR]
                             [--targetvalsFile TARGETVALSFILE] [--plotStats]
                             [--printOnly]
                             inputParamFile nCoalescentReps outputDir
```

Positional arguments:

inputParamFile	file with model specifications for input
nCoalescentReps	number of coalescent replicates to run per point in parameter-space
outputDir	location in which to write cosi output

Options:

--cosiBuild=coalescent	which version of cosi to run?

--dropSings randomly thin global singletons from output dataset (i.e., to model ascertainment bias)

--genmapRandomRegions=False cosi option to sub-sample genetic map randomly from input

--stopAfterMinutes cosi option to terminate simulations

--calcError file specifying dimensions of error function to use. if unspecified, defaults to all. first line = stats, second line = pops

--targetvalsFile file containing target values for model

--plotStats=False visualize goodness-of-fit to model targets

--printOnly=False print rather than execute pipeline commands

grid Perform grid search: for specified parameters and intervals, define points in parameter-space to sample and compare.

```
usage: cms_modeller.py grid [-h] [--cosiBuild COSIBUILD]
                            [--dropSings DROPSINGS] [--genmapRandomRegions]
                            [--stopAfterMinutes STOPAFTERMINUTES]
                            [--calcError CALCERROR]
                            inputParamFile nCoalescentReps outputDir
                            grid_inputdimensionsfile
```

Positional arguments:

inputParamFile file with model specifications for input

nCoalescentReps number of coalescent replicates to run per point in parameter-space

outputDir location in which to write cosi output

grid_inputdimensionsfile file with specifications of grid search. each parameter to vary is indicated: KEY INDEX [VALUES]

Options:

--cosiBuild=coalescent which version of cosi to run?

--dropSings randomly thin global singletons from output dataset (i.e., to model ascertainment bias)

--genmapRandomRegions=False cosi option to sub-sample genetic map randomly from input

--stopAfterMinutes cosi option to terminate simulations

--calcError file specifying dimensions of error function to use. if unspecified, defaults to all. first line = stats, second line = pops

optimize Perform optimization algorithm (scipy.optimize) to fit model parameters robustly.

```
usage: cms_modeller.py optimize [-h] [--cosiBuild COSIBUILD]
                                [--dropSings DROPSINGS]
                                [--genmapRandomRegions]
                                [--stopAfterMinutes STOPAFTERMINUTES]
                                [--calcError CALCERROR] [--stepSize STEPSIZE]
                                [--method METHOD]
                                inputParamFile nCoalescentReps outputDir
                                optimize_inputdimensionsfile
```

Positional arguments:

inputParamFile file with model specifications for input

nCoalescentReps number of coalescent replicates to run per point in parameter-space

outputDir location in which to write cosi output

optimize_inputdimensionsfile file with specifications of optimization. each parameter to vary is indicated: KEY INDEX

Options:

--cosiBuild=coalescent which version of cosi to run?

--dropSings randomly thin global singletons from output dataset (i.e., to model ascertainment bias)

--genmapRandomRegions=False cosi option to sub-sample genetic map randomly from input

--stopAfterMinutes cosi option to terminate simulations

--calcError file specifying dimensions of error function to use. if unspecified, defaults to all. first line = stats, second line = pops

--stepSize scaled step size (i.e. whole range = 1)

--method=SLSQP algorithm to pass to scipy.optimize

likes_from_model.py

This script contains command-line utilities for generating probability distributions for component scores from pre-specified demographic model(s).

```
usage: likes_from_model.py [-h]
                           {generate_sel_bins,get_sel_traj,run_neut_sim,run_sel_sim,run_rep
                           ...
```

Sub-commands:

generate_sel_bins Pre-processing step: generate directories with parameter files divided by selDAF, according to specified bins

```
usage: likes_from_model.py generate_sel_bins [-h] [--freqRange FREQRANGE]
                                             [--nBins NBINS]
                                             outputDir
```

Positional arguments:

outputDir location to write cosi output

Options:

--freqRange=.05-.95 range of final selected allele frequencies to simulate, e.g. .05-.95

--nBins=9 number of frequency bins

get_sel_traj Run forward simulations of selection trajectories to populate selscenarios by final allele frequency before running coalescent simulations for entire sample.

```
usage: likes_from_model.py get_sel_traj [-h] [--maxAttempts MAXATTEMPTS]
                                        [--cosiBuild COSIBUILD]
                                        [--freqRange FREQRANGE]
                                        [--nBins NBINS]
                                        traj_outputname inputParamFile
```

Positional arguments:

traj_outputname write to file

inputParamFile file with model specifications for input

Options:

--maxAttempts=100 maximum number of attempts to generate a trajectory before re-sampling selection coefficient / start time

--cosiBuild=coalescent which version of cosi to run

--freqRange=.05-.95 range of final selected allele frequencies to simulate, e.g. .05-.95

--nBins=9 number of frequency bins

run_neut_sim run neutral simulations

```
usage: likes_from_model.py run_neut_sim [-h] [--checkOverwrite]
                                        [--dropSings DROPSINGS]
                                        [--cosiBuild COSIBUILD]
                                        writeBase inputParamFile
```

Positional arguments:

writeBase write prefix

inputParamFile file with model specifications for input

Options:

--checkOverwrite=True Undocumented

--dropSings=0.25 randomly thin global singletons from output dataset to model ascertainment bias

--cosiBuild=coalescent which version of cosi to run

run_sel_sim run sims with selection

```
usage: likes_from_model.py run_sel_sim [-h] [--checkOverwrite]
                                       [--dropSings DROPSINGS]
                                       [--cosiBuild COSIBUILD]
                                       traj_infilename writeBase
                                       inputParamFile
```

Positional arguments:

traj_infilename selection trajectory

writeBase write prefix

inputParamFile file with model specifications for input

Options:

--checkOverwrite=True Undocumented

--dropSings=0.25 randomly thin global singletons from output dataset to model ascertainment bias

--cosiBuild=coalescent which version of cosi to run

run_repscores run composite score calculations for simulated tped

```
usage: likes_from_model.py run_repscores [-h] [--simRecomFile SIMRECOMFILE]
                                         [--checkOverwrite] [--cmsdir CMSDIR]
                                         score_basedir inputTpedFile
```

Positional arguments:

score_basedir parent directory in which to generate/populate folders for each composite score

inputTpedFile TPED file with simulate data for putative selected population

Options:

--simRecomFile=/n/home08/jvitti/params/test_recom.recom location of input recom file

--checkOverwrite=True Undocumented

--cmsdir=/n/home08/jvitti/cms/cms/ location of CMS scripts (TEMPORARY; conda will obviate)

get_neut_norm_params jointly normalize scores from neutral replicates to get parameters with which to normalize all replicates

```
usage: likes_from_model.py get_neut_norm_params [-h] [--checkOverwrite]
                                                [--cmsdir CMSDIR]
                                                [--edge EDGE]
                                                [--chromlen CHROMLEN]
                                                [--score SCORE]
                                                [--simpop SIMPOP]
                                                [--altpop ALTPOP]
                                                [--nrep NREP]
                                                modeldir
```

Positional arguments:

modeldir location of component score folders for demographic scenario

Options:

--checkOverwrite=True Undocumented

--cmsdir=/n/home08/jvitti/cms/cms/ location of CMS scripts (TEMPORARY; conda will obviate)

--edge=250000 use interior of replicates; define per-end bp. (e.g. 1.5Mb -> 1Mb: 250000)

--chromlen=1500000 per bp (1.5mb = 1500000)

--score=ihs Undocumented

--simpop=1 Undocumented

--altpop=2 Undocumented

--nrep=100 Undocumented

norm_from_neut_params normalize component scores according to neutral distribution

```
usage: likes_from_model.py norm_from_neut_params [-h] [--selbin SELBIN]
                                                 [--scenario_dir SCENARIO_DIR]
                                                 [--checkOverwrite]
                                                 [--cmsdir CMSDIR]
                                                 [--score SCORE]
                                                 [--simpop SIMPOP]
                                                 [--altpop ALTPOP]
                                                 [--nrep NREP]
                                                 modeldir
```

Positional arguments:

modeldir	location of component score folders for demographic scenario

Options:

--selbin	e.g. 0.10 – if excluded, normalize neutral simulates
--scenario_dir=neut	neut/selsim dir
--checkOverwrite=True	Undocumented
--cmsdir=/n/home08/jvitti/cms/cms/	location of CMS scripts (TEMPORARY; conda will obviate)
--score=ihs	Undocumented
--simpop=1	Undocumented
--altpop=2	Undocumented
--nrep=100	Undocumented

likes_from_scores Collate scores from simulated data in order to generate component test probability distributions.

```
usage: likes_from_model.py likes_from_scores [-h] [--nrep_neut NREP_NEUT]
                                             [--nrep_sel NREP_SEL]
                                             [--binMerge] [--foldDist]
                                             [--save_suffix SAVE_SUFFIX]
                                             [--save_dir SAVE_DIR]
                                             [--save_likelihoods] [--ihs]
                                             [--delihh] [--nsl] [--xpehh]
                                             [--deldaf] [--fst] [--edge EDGE]
                                             [--chromlen CHROMLEN]
                                             [--freqRange FREQRANGE]
                                             [--nBins NBINS]
                                             modeldir
```

Positional arguments:

modeldir	location of component score folders for demographic scenario

Options:

--nrep_neut=1000	number of neutral replicates
--nrep_sel=500	number of replicates per selection scenario bin
--binMerge=False	if included, generate higher-order groupings of selscenario frequency bins
--foldDist=False	if included, take absolute value of score values (fold distribution over y-axis)

--save_suffix optional string to add to filenames (e.g. to avoid overwrite)

--save_dir if included, specify alternate outpot location

--save_likelihoods=False in addition to plotting, save data

--ihs=False visualize likelihoods for iHS

--delihh=False visualize likelihoods for delIHH

--nsl=False visualize likelihoods for nSL

--xpehh=False visualize likelihoods for XP-EHH

--deldaf=False visualize likelihoods for delDAF

--fst=False visualize likelihoods for Fst

--edge=250000 use interior of replicates; define per-end bp. (e.g. 1.5Mb -> 1Mb: 250000)

--chromlen=1500000 per bp (1.5mb = 1500000)

--freqRange=.05-.95 range of final selected allele frequencies to simulate, e.g. .05-.95

--nBins=9 number of frequency bins

plot_likes_vs_scores useful QC step; visualize the function that converts score values into likelihood contributions towards composite scores

```
usage: likes_from_model.py plot_likes_vs_scores [-h]
                                                [--default_nregion_snps DEFAULT_NRE
                                                inputLikesPrefix
```

Positional arguments:

inputLikesPrefix filename minus causal/linked/neutral.txt

Options:

--default_nregion_snps=5000 presumptive number of SNPs in region (determines prior distributions for within-region CMS calculations)

write_master_likes specify and bundle together input distributions to pass to CMS. This allows the user to specify models, frequency of SNPs used to generated p(score|sel), etc.

```
usage: likes_from_model.py write_master_likes [-h]
                                              [--likes_masterDir LIKES_MASTERDIR]
                                              [--model MODEL]
                                              [--likes_inDir LIKES_INDIR]
                                              [--score SCORE]
                                              [--simpop SIMPOP]
                                              [--altpop ALTPOP] [--nrep NREP]
```

Options:

--likes_masterDir=/n/regal/sabeti_lab/jvitti/ location of likelihood tables, defined

--model=nulldefault ...

--likes_inDir=/n/regal/sabeti_lab/jvitti/ ...

--score=ihs Undocumented

--simpop=1 Undocumented

--altpop=2 Undocumented

--nrep=100 Undocumented

composite.py

This script contains command-line utilities for manipulating and combining component statistics

```
usage: composite.py [-h]
                    {freqscores,delihh_from_ihs,ihh_from_xp,xp_from_ihh,composite_sims,comp
                    ...
```

Sub-commands:

freqscores Calculate allele frequency-based scores (Fst and delDAF) for a pair of populations.

```
usage: composite.py freqscores [-h] [--modelpath MODELPATH]
                               inTped1 inTped2 recomFile outfile
```

Positional arguments:

inTped1 input tped 1

inTped2 input tped 2

recomFile input recombination file

outfile file to write

Options:

--modelpath=cms/cms/model/ path to model directory containing executables

delihh_from_ihs Calculate delIHH values from iHS output files.

```
usage: composite.py delihh_from_ihs [-h] readfile writefile
```

Positional arguments:

readfile input ihs file

writefile delihh file to write

ihh_from_xp extract per-pop iHH values from XP-EHH and write to individual files to facilitate arbitrary population comparisons

```
usage: composite.py ihh_from_xp [-h] inXpehh outIhh takePop
```

Positional arguments:

inXpehh input xpehh file

outIhh write to file

takePop write for first (1) or second (2) pop in XP-EHH file?

xp_from_ihh Calculate XP-EHH based on two per-pop iHH files.

```
usage: composite.py xp_from_ihh [-h] [--cmsdir CMSDIR] [--printOnly]
                                inIhh1 inIhh2 outfilename
```

Positional arguments:

inIhh1 input ihh file 1

inIhh2 input ihh file 2

outfilename write to file

Options:

--cmsdir=/idi/sabeti-scratch/jvitti/cms/cms/ TEMPORARY, will become redundant with conda packaging

--printOnly=False print rather than execute pipeline commands

composite_sims calculate composite scores for simulations

```
usage: composite.py composite_sims [-h] [--regional_cms]
                                   [--scenariopop SCENARIOPOP]
                                   [--cmsdir CMSDIR] [--printOnly]
                                   [--nrep_sel NREP_SEL]
                                   [--nrep_neut NREP_NEUT]
                                   [--likes_masterDir LIKES_MASTERDIR]
                                   [--likes_nonSel LIKES_NONSEL]
                                   [--likes_freqSuffix LIKES_FREQSUFFIX]
                                   [--cutoffline CUTOFFLINE]
                                   [--includeline INCLUDELINE]
                                   [--writedir WRITEDIR]
                                   [--runSuffix RUNSUFFIX] [--simpop SIMPOP]
                                   [--model MODEL] [--checkOverwrite]
```

Options:

--regional_cms=False calculate within-region CMS rather than genome-wide CMS

--scenariopop sel directory

--cmsdir=/idi/sabeti-scratch/jvitti/cms/cms/ TEMPORARY, will become redundant with conda packaging

--printOnly=False print rather than execute pipeline commands

--nrep_sel=500 Undocumented

--nrep_neut=1000 Undocumented

--likes_masterDir=/n/regal/sabeti_lab/jvitti/clear-synth/sims_reeval/likes_masters/ location of likelihood tables, defined

--likes_nonSel=vsNeut do we use completely neutral, or linked neutral SNPs for our non-causal distributions? by default, uses strict neutral (CMSgw)

--likes_freqSuffix=allFreqs for causal SNPs, include suffix to specify which selbins to include

--cutoffline=250000 1250000 0 1 specify bounds to include/exclude in calculations, along with MAF filter and likes decomposition

--includeline=0 0 0 0 0 0 specify (0:yes; 1:no) which scores to include: iHS, ...

--writedir= specify relative path

--runSuffix add a suffix to .cms file (corresponds to runparamfile)

--simpop=1 simulated population

--model=nulldefault Undocumented

--checkOverwrite=False Undocumented

composite_emp calculate composite scores for empirical data

```
usage: composite.py composite_emp [-h] [--score_basedir SCORE_BASEDIR]
                                  [--regional_cms_chrom REGIONAL_CMS_CHROM]
```

```
                                        [--composite_writedir COMPOSITE_WRITEDIR]
                                        [--cmsdir CMSDIR] [--printOnly]
                                        [--likes_masterDir LIKES_MASTERDIR]
                                        [--likes_nonSel LIKES_NONSEL]
                                        [--likes_freqSuffix LIKES_FREQSUFFIX]
                                        [--cutoffline CUTOFFLINE]
                                        [--includeline INCLUDELINE]
                                        [--writedir WRITEDIR]
                                        [--runSuffix RUNSUFFIX] [--simpop SIMPOP]
                                        [--emppop EMPPOP] [--model MODEL]
                                        [--checkOverwrite]
```

Options:

--score_basedir=/n/regal/sabeti_lab/jvitti/clear-synth/1kg_scores/ Undocumented

--regional_cms_chrom if included, calculate within-region CMS (rather than CMS_gw) for specified bounds at this chromosome

--composite_writedir= write output to

--cmsdir=/idi/sabeti-scratch/jvitti/cms/cms/ TEMPORARY, will become redundant with conda packaging

--printOnly=False print rather than execute pipeline commands

--likes_masterDir=/n/regal/sabeti_lab/jvitti/clear-synth/sims_reeval/likes_masters/ location of likelihood tables, defined

--likes_nonSel=vsNeut do we use completely neutral, or linked neutral SNPs for our non-causal distributions? by default, uses strict neutral (CMSgw)

--likes_freqSuffix=allFreqs for causal SNPs, include suffix to specify which selbins to include

--cutoffline=250000 1250000 0 1 specify bounds to include/exclude in calculations, along with MAF filter and likes decomposition

--includeline=0 0 0 0 0 0 specify (0:yes; 1:no) which scores to include: iHS, ...

--writedir= specify relative path

--runSuffix add a suffix to .cms file (corresponds to runparamfile)

--simpop=1 simulated population

--emppop=YRI empirical population

--model=nulldefault Undocumented

--checkOverwrite=False Undocumented

normsims_genomewide normalize simulated composite scores to neutral

```
usage: composite.py normsims_genomewide [-h] [--nrep_sel NREP_SEL]
                                        [--nrep_neut NREP_NEUT]
                                        [--writedir WRITEDIR]
                                        [--runSuffix RUNSUFFIX]
                                        [--simpop SIMPOP] [--model MODEL]
                                        [--checkOverwrite]
```

Options:

--nrep_sel=500 Undocumented

--nrep_neut=1000 Undocumented

--writedir= specify relative path

--runSuffix add a suffix to .cms file (corresponds to runparamfile)

--simpop=1 simulated population

--model=nulldefault Undocumented

--checkOverwrite=False Undocumented

normemp_genomewide normalize CMS scores to genome-wide

```
usage: composite.py normemp_genomewide [-h] [--score_basedir SCORE_BASEDIR]
                                       [--cmsdir CMSDIR] [--printOnly]
                                       [--writedir WRITEDIR]
                                       [--runSuffix RUNSUFFIX]
                                       [--emppop EMPPOP]
```

Options:

--score_basedir=/n/regal/sabeti_lab/jvitti/clear-synth/1kg_scores/ Undocumented

--cmsdir=/idi/sabeti-scratch/jvitti/cms/cms/ TEMPORARY, will become redundant with conda packaging

--printOnly=False print rather than execute pipeline commands

--writedir= specify relative path

--runSuffix add a suffix to .cms file (corresponds to runparamfile)

--emppop=YRI empirical population

hapviz Visualize haplotypes for region

```
usage: composite.py hapviz [-h] [--startpos STARTPOS] [--endpos ENDPOS]
                           [--corepos COREPOS] [--title TITLE]
                           [--annotate ANNOTATE] [--maf MAF] [--dpi DPI]
                           inputfile out
```

Positional arguments:

inputfile input tped

out save image as file

Options:

--startpos define physical bounds of region

--endpos define physical bounds of region

--corepos=-1 partition haplotypes based on allele status at this position

--title title to give to plot

--annotate tab-delimited file where each line gives <chr.pos> <annotation>

--maf filter on minor allele frequency (e.g. .01, .05)

--dpi=300 image resolution

ml_region machine learning algorithm (within-region)

```
usage: composite.py ml_region [-h]
```

ucsc_viz Generate trackfiles of CMS scores for visualization in the UCSC genome browser.

```
usage: composite.py ucsc_viz [-h] [--posIndex POSINDEX]
                             [--scoreIndex SCOREINDEX] [--strip_header]
                             infile_prefix outfile
```

Positional arguments:

infile_prefix	prefix of file containing scores to be reformatted (e.g. 'score_chr' for files named scores_chr#.txt)
outfile	file to write

Options:

--posIndex=1	index for column of datafile containing physical position (zero-indexed)
--scoreIndex=-2	index for column of datafile containing score (zero-indexed)
--strip_header=False	if input files include header line

Sample workflow

CMS provides a computational framework to explore signals of natural selection in diploid organisms. This section describes how to do so at an abstract level.

Preliminary considerations

CMS is a computational and statistical framework for exploring the evolution of populations within a species at a genomic level. To that end, the user must first provide a dataset containing genotype calls for individuals in at least one putative selected population and at least one putative 'outgroup.' CMS 2.0 is designed to be flexible with respect to the number and configuration of input populations – that is, given input of however many populations, the user can easily calculate CMS scores for any configuration of these populations. Nonetheless, CMS still relies on the user to define these populations appropriately.

- To determine or confirm appropriate population groupings, identify outliers, etc., we recommend that users first characterize their dataset using such methods as similarity clustering (e.g. STRUCTURE), principal components analysis, or phylogenetic methods (see e.g. SNPRelate)
- Each population should be randomly thinned to the same number of individuals, none of whom should be related within the past few generations.
- Larger samples are generally preferable (e.g. 50-100+ diploid individuals per population). However, depending on such factors as the landscape of recombination in the species, the quality/density of genotype data, and the extent of neutral genetic divergence between represented populations, it may be possible to leverage smaller datasets. As CMS necessitates the generation of a demographic model for the given dataset, the user is advised to use their model to generate simulated data with which to perform power estimations.

Data formatting

CMS requires the user to provide population genetic (i.e., within-species) diversity data, including genotype phase and allele polarity.

- If your dataset is **unphased**, you can preprocess it using a program like Beagle or PLINK.

- The identity of the **ancestral allele** at each site is typically determined by comparison to outgroups at orthologous sites. Inferred ancestral sequence is available for a number of species through e.g. Ensembl via their ftp. You can use VCFtools to populate the "AA=" section of your VCF's INFO field.
- In most cases, the user will want to provide a **genetic recombination map**. If this is unavailable, CMS will assume uniform recombination rates when calculating haplotype scores. Human recombination maps are available from the HapMap Project.
- CMS works with TPED datafiles, and includes support to convert from VCF using the command line tool scans.py.

Demographic modeling

CMS combines several semi-independent component tests for selection in a Bayesian or Machine Learning framework. In the former case, a demographic model for the species in question is critical in order to furnish posterior distributions of scores for said component tests under alternate hypotheses of neutrality or selection. Put otherwise: a demographic model is a (conjectural) descriptive historical account of our dataset, including population sizes and migration rates across time, that can be used to generate simulated data that 'looks like' our original dataset. We then simulate many scenarios of selection in order and calculate the distributions of component scores for adaptive, linked, and neutral variants. These distributions form the basis of our Bayesian classifier. We can also circumvent the need to define posterior score distributions by using simulated data as training data for Machine Learning implementations of CMS.

Our modeling framework is designed to accomodate an arbitrary number of populations in a tree of arbitrary complexity; as such, it is designed to be exploratory, allowing users to iteratively perform optimizations while visualizing the effect on model goodness-of-fit. For rigorous demographic inference (i.e., in the case of a model with known topology and tractably few parameters), users may consider programs such as dadi or diCal.

Following Schaffner et al 2005 , our framework calculates a range of population summary statistics as target values, and defines error as the Root Mean Square discrepancy between target and simulated values. These summary statistics are calculated by bootstrap estimate from user-specified putative neutral regions. For human populations, the Neutral Regions Explorer is a useful resource.

The user must specify tree topology and ranges for parameter values. These can be added and removed as desired through the script params.py. After target values have been estimated and model topology defined, the user can iteratively search through subsets of parameter-space using cms_modeller.py with a masterfile specifying search input.

Calculating selection statistics

CMS packages a number of previously described population genetic tests for recent positive selection. Haplotype scores are calculated using selscan.

Combining scores

CMS 2.0 allows users to define CMS scores flexibly with respect to (i) number and identity of putative selected/neutral populations, (ii) assumed demographic model, (iii) input component scores, (iv) method of score combination. In each case the user should motivate their choices and consider how robust a putative signal of selection is to variation or arbitrariness in these factors.

Identifying regions

CMS is motivated by the need to resolve signals of selection – that is, to identify genetic variants that confer adaptive phenotypes. Because selective events can alter patterns of population genetic diversity across large genomic regions, we take a two-step approach to this goal: we first identify putative selected regions (using CMS, another framework,

prior knowledge, etc.), and then examine each region with CMS to identify a tractable list of candidate variants for further scrutiny.

Localizing signals

Once regions are defined, we can reapply our composite framework in order to thin our list of candidate variants for further scrutiny and prioritize those sites that have the strongest evidence of selection (or other compelling evidence, e.g. overlap with known or predicted functional elements).