ImPoStAR: Implement Population Structure Analyses in R
Quick Start Guide

***impostar* v 0.0.0.9**

R. M. Hamner
23 Apr 2018

Description

The *impostar* package implements appropriate resampling strategies to model the total sampling error associated with either Sanger sequencing (population sampling error) or next-generation sequencing (population + sequencer sampling error) in two commonly applied statistical tests: (1) a binomial logistic regression test for a genetic cline and (2) the analysis of molecular variance (AMOVA) test for genetic structure. The novel method that accounts for the additional sequencer sampling error associated with next-generation sequencing is essential for controlling the level of false positive ('impostar') loci. For additional information, see: Hamner, R.M., J.D. Selwyn, E. Krell, S.A. King, and C.E. Bird. In review. Modeling next-generation sequencer sampling error in pooled population samples dramatically reduces false positives in genetic structure tests.

Installation

Download and install R (https://www.r-project.org/), if you do not already have it.
The RStudio (https://www.rstudio.com/) interface is also recommended.
Please refer to the documentation supplied with these programs for assistance installing them.

In R, install the *devtools* package. If it is already installed, skip to the next step to load it.
```
install.packages("devtools")
```

Load *devtools.*
```
library(devtools)
```

Install the *impostar* from GitHub using the following command:
```
install_github("cbirdlab/impostar")
```

NOTE: this may take several minutes if you do not already have the dependencies. Only if the following direct dependencies do not automatically install with *impostar*, remove the "#" at the beginning of the line and run the following commands to install them and their dependencies.
```
# install.packages("dplyr")
# install.packages("parallel")
# install.packages("Rcpp")
# source("https://bioconductor.org/biocLite.R")
# biocLite(c("VariantAnnotation"))
```

If the installation fails because of a specified dependency package, install that package directly and then run the command to install *impostar* again. Although not a direct dependency, installation often fails if "GenomeInfoDb" is not installed. If you get this error, run the following and then run the command to install *impostar* again.
```
source("https://bioconductor.org/biocLite.R")
biocLite("GenomeInfoDb")
```

Load *impostar*. This will automatically load the dependencies.
```
library(impostar)
```

Running *impostar*

If you have a VCF file ready to go, continue on to "Convert VCF to ImPoStAR format", and then to the analysis you want to perform.

If creating the input data file from scratch (this will be painful), proceed to the "Input Files" section of the analysis you want to perform to see the files and format required. If you run into issues, check out the "Simulating Data" section to generate a sample input file with the correct format and headings.

If you just want to test drive *impostar,* carry on and use the sample datasets as described in the 'Examples' of each section.

Convert VCF to *impostar* format

Use the *vcf2impostar* function to reformat a VCF file into the format required to run a logistic regression test or AMOVA in *impostar*.

Input Files

To run *vcf2impostar*, you'll need to supply the VCF file that you want to convert to *impostar* format, and if you have pooled sequences you'll need a design file too.

The VCF file must follow standard VCF format and contain fields indicating the number of reference (RO) and alternate reads (AO), respectively, as this information is extracted for the analysis.

The design file is only required if your sequences represent pools of individuals**.** For file conversion, the only information that needs to be in this file is the number of individuals in each pool. This information can be supplied in a .csv or .txt (tab-delimited) file or can be created as a data.frame directly in R.
The design file for 3 pools each having 20 individuals should look like this:

```
n
20
20
20
```

To create this directly in R, run the following:

```
mydesignFile <- data.frame(n=rep(20,3))
```

It is ok to have additional information in this file, as long as the number of individuals in each pool is in the first column. Note that the design file required to run an AMOVA requires additional information about hierarchical population structure, so you can use the same file here. Here is an example where one level of population structure is included.

```
mydesignFile <- data.frame(n=rep(20,3), Sample=c(seq(1:3))
```

will produce this:

```
n  Sample
20      1
20      2
20      3
```

Setting Function Parameters

The basic format of the function is:

```
data.impostar <- vcf2impostar(vcfFile = myvcfFile, designFile = mydesignFile,
ploidy = 2, savecsv = FALSE)
```

The first part “data.impostar”, indicates the name of the data frame that will result.

To specify the VCF file you want to convert, replace “myvcfFile” with the name of your VCF file inside quotes, like this: vcfFile = “myvcfFile.vcf”

Or, if you already have your VCF file read in as an R object, you can put the name of the R object (myvcfRobject) with no quotes, like this: vcfFile = myvcfRobject

If your sequences represent **individuals**, set “designFile=NULL” or just leave this parameter out.
If your sequences represent **pools of individuals**, you need to specify a design file. See “Input Files” above for the correct formatting.

Next, set the ploidy of your organism. This setting defaults to 2, so can be omitted if you are working with diploids.

The converted data will be available as an R object (data.impostar, in this example).
If you want to save the resulting converted data as a csv file for future use, set “savecsv = TRUE”.

Now that it’s all set up, let’s run it!

Example

If you don’t have your own data yet, try this built in example:

```
# load example vcf file
myvcfFile <- impostar:::exVCF

# generate an example designFile
mydesignFile <- data.frame(n=rep(20,3), Sample=c(seq(1:3)))

# run vcf2impostar format converter
data.impostar <- vcf2impostar(vcfFile = myvcfFile, designFile = mydesignFile)
```

Interpreting Output

Take a look at the resulting data frame.

```
data.impostar
```

It should look like this:

	snpID	REF	ALT	RD1	RD2	RD3	AD1	AD2	AD3	DP1	DP2	DP3	TotAlleles1	TotAlleles2	TotAlleles3
1	dDocent_Contig_3:146_G/A	G	A	34	35	11	1	0	8	35	35	19	40	40	40
2	dDocent_Contig_5:189_T/A	T	A	23	19	3	3	0	18	26	19	21	40	40	40
3	dDocent_Contig_6:140_T/C	T	C	13	46	16	0	5	26	13	51	42	40	40	40
4	dDocent_Contig_9:197_T/A	T	A	12	17	2	4	0	18	16	17	20	40	40	40
5	dDocent_Contig_10:11_A/T	A	T	12	10	1	9	0	10	21	10	11	40	40	40

Each row is one SNP locus. The columns are:

snpID	unique name for each SNP
REF	reference allelic state
ALT	alternate allelic state
RD1…*n*	number of reference reads per individual or pool
AD1…*n*	number of alternate reads individual or per pool
DP1…*n*	total read depth
TotAlleles1…*n*	total number of alleles (2n) per individual or pool

Each individual or pool is indicated by the trailing number in the column names; for example, the first individual or pool is represented by RD1, AD1, DP1, and TotAlleles1.

Now you're ready to analyze population structure with minimized false positive 'impostar' loci!

Logistic Regression: Assessing genetic clines

The runLogRegTest function performs a binomial logistic regression test for a genetic cline among three pools of individuals, implementing the appropriate resampling strategies to model the total sampling error associated with either Sanger sequencing (population sampling error) or next-generation sequencing (population + sequencer sampling error).

Input File

The required input file is a csv or data frame. The current implementation is designed for exactly three populations (pooled sequences or individual data collapsed into three pools) arranged in order along a geographic gradient and cannot have missing data.

Each row is a SNP, and the required columns are:

snpID	unique name for each SNP
RD1…*n*	number of reference reads per pool
AD1…*n*	number of alternate reads per pool
DP1…*n*	total read depth
TotAlleles1…*n*	total number of alleles (2n) per pool

Each pool is indicated by the trailing number in the column names; for example, the first pool is represented by RD1, AD1, DP1, and TotAlleles1. Additional columns can be included. These could include information of interest or required for other functions, such as:

REF	reference allelic state
ALT	alternate allelic state

```
                    snpID REF ALT RD1 RD2 RD3 AD1 AD2 AD3 DP1 DP2 DP3 TotAlleles1 TotAlleles2 TotAlleles3
1 dDocent_Contig_3:146_G/A   G   A  34  35  11   1   0   8  35  35  19          40          40          40
2 dDocent_Contig_5:189_T/A   T   A  23  19   3   3   0  18  26  19  21          40          40          40
3 dDocent_Contig_6:140_T/C   T   C  13  46  16   0   5  26  13  51  42          40          40          40
4 dDocent_Contig_9:197_T/A   T   A  12  17   2   4   0  18  16  17  20          40          40          40
5 dDocent_Contig_10:11_A/T   A   T  12  10   1   9   0  10  21  10  11          40          40          40
```

Setting the Parameters

The basic format of the function is:

```
LogRegResults <- runLogRegTest(dataFile, OutputBase=NULL, mainDirectory=getwd(),
NumBS=1000, NGSdata=T, nCores=NULL)
```

The first part, "LogRegResults", indicates the name of the data frame that will result.

The dataFile is the only required input. If reading in from a csv file wrap the file name in quotes (dataFile="mydataFile.csv"); if already an R data.frame, do not wrap it in quotes (dataFile=mydataFile). See "Input Files" above for the correct formatting of the files.

For the sake of speed with our example, we'll only run 50 iterations. In a real analysis, this number should be increased considerably (default is 10,000).

The next step is to set the appropriate model for the sampling variance. If you're analyzing next-generation sequencing data make sure NGSdata is set to TRUE (default), so that the total sampling variance is modeled in the appropriate way. If you're analyzing Sanger sequence data, or want to see how many false positive "impostar" loci you would have gotten by ignoring the additional sampling error associated with next-generation sequencing, NGSdata should be set to FALSE to implement the appropriate method.

You're now ready to run the analysis using the default settings for the remaining parameters. To find out more information about additional parameters, run:

```
?runLogRegTest
```

Examples

If you are following on from the *vcf2impostar* example, use the dataset from that as your input:

```
LogRegResults <- runLogRegTest(data.impostar, NumBS=50, NGSdata = T)
```

Alternatively, you can simulate an example data file to try and run it:

```
simdata <- simulate_data(rep(50, 3), rep(100, 3), c(0.4, 0.5, 0.6), 5,
file_name=T)

LogRegResults <- runLogRegTest(simdata, NumBS=50)
```

Interpreting the Output

The runLogRegTest function returns a data.frame. Have a look at it by running:

```
LogRegResults
```

The results from the data.impostar example above should look similar to this:

```
                    snpID REF ALT RD1 RD2 RD3 AD1 AD2 AD3 DP1 DP2 DP3 TotAlleles1 TotAlleles2 TotAlleles3
1 dDocent_Contig_3:146_G/A   G   A  34  35  11   1   0   8  35  35  19          40          40          40
2 dDocent_Contig_5:189_T/A   T   A  23  19   3   3   0  18  26  19  21          40          40          40
3 dDocent_Contig_6:140_T/C   T   C  13  46  16   0   5  26  13  51  42          40          40          40
4 dDocent_Contig_9:197_T/A   T   A  12  17   2   4   0  18  16  17  20          40          40          40
5 dDocent_Contig_10:11_A/T   A   T  12  10   1   9   0  10  21  10  11          40          40          40
  ObsRefAlleles1 ObsRefAlleles2 ObsRefAlleles3 ObsAltAlleles1 ObsAltAlleles2 ObsAltAlleles3 ObsRefFreq1
1             39             40             23              1              0             17       0.975
2             35             40              6              5              0             34       0.875
3             40             36             15              0              4             25       1.000
4             30             40              4             10              0             36       0.750
5             23             40              4             17              0             36       0.575
  ObsRefFreq2 ObsRefFreq3  ObsSlope ObsAbsSlope nBSwithObs SumPcountSlopes     SlopeP
1         1.0       0.575 -2.690203    2.690203         51               1 0.01960784
2         1.0       0.150 -2.435373    2.435373         51               1 0.01960784
3         0.9       0.375 -2.860862    2.860862         51               1 0.01960784
4         1.0       0.100 -1.704590    1.704590         51               1 0.01960784
5         1.0       0.100 -1.051104    1.051104         51               3 0.05882353
```

Similar to the input, each row is a SNP. The columns from snpID to TotAlleles3 are copied from the input. New columns created during the analysis are

ObsRefAlleles1…*n*	number of reference alleles observed in a given pool (scaled from reads)
ObsAltAlleles1…*n*	number of alternate alleles observed in a given pool (scaled from reads)
ObsRefFreq1…*n*	reference allele frequency observed in a given pool
ObsSlope	observed rate of change in the log odds ratio of the logistic regression line
ObsAbsSlope	absolute value of ObsSlope
nBSwithObs	number of iterations + 1 for the observed data
SumPcountSlopes	total number of iterations where the absolute value of the simulated change in the log odds ratio is ≥ the absolute value of the observed change in the log odds ratio
SlopeP	p-value used to evaluate significance (SumPcountSlopes/nBSwithObs)

The two columns to focus on are “ObsSlope” and “SlopeP”, as these provide information on the strength of the genetic cline and statistical significance, respectively. The sign of the ObsSlope indicates whether the arbitrarily selected reference allele is increasing or decreasing, but remember as these are biallelic SNPs, the alternate allele will show the exact opposite pattern.

AMOVA: Assessing population structure

The runAMOVA function performs an analysis of molecular variance (AMOVA) test for genetic structure, implementing the appropriate resampling strategies to model the total sampling error associated with either Sanger sequencing (population sampling error) or next-generation sequencing (population + sequencer sampling error). The current implementation works with pooled data that does not have any missing data.

Input Files

The required **design file** contains the number of individuals in each pool and information about the hierarchical structure of the samples. Each row is a different pool. The first column (required) is the number of individuals in each pool. The second column (required) represents the lowest (or only) level of hierarchical structure. Optional additional columns represent increasing levels of nested structure.
Below is an example of the design file for 3 pools (one on each row) each having 20 individuals (“n”), where each pool is considered a different population “Sample”.

```
n  Sample
20      1
20      2
20      3
```

This information can be supplied in a .csv or .txt (tab-delimited) file, or can be created as a data.frame directly in R. To create the example above directly in R, run the following:

```
mydesignFile <- data.frame(n=rep(20,3), Sample=c(seq(1:3)))
```

The required **data file** is a csv file or data frame. The current implementation cannot handle missing data. Each row is one SNP and the required columns for runAMOVA are:

snpID	unique name for each SNP
RD1…*n*	number of reference reads per pool
AD1…*n*	number of alternate reads per pool

```
                     snpID RD1 RD2 RD3 AD1 AD2 AD3 DP1 DP2 DP3
1 dDocent_Contig_3:146_G/A  34  35  11   1   0   8  35  35  19
2 dDocent_Contig_5:189_T/A  23  19   3   3   0  18  26  19  21
3 dDocent_Contig_6:140_T/C  13  46  16   0   5  26  13  51  42
4 dDocent_Contig_9:197_T/A  12  17   2   4   0  18  16  17  20
5 dDocent_Contig_10:11_A/T  12  10   1   9   0  10  21  10  11
```

Additional columns can be included. These could include information of interest or required for other functions, such as:

REF	reference allelic state
ALT	alternate allelic state
DP1…*n*	total read depth
TotAlleles1…*n*	total number of alleles (2n) per pool

Each pool is indicated by the trailing number in the column names; for example, the first pool is represented by RD1, AD1, DP1, and TotAlleles1.

If following on from the "Convert VCF to *impostar* format" section, the data.impostar dataset is a good example:

	snpID	REF	ALT	RD1	RD2	RD3	AD1	AD2	AD3	DP1	DP2	DP3	TotAlleles1	TotAlleles2	TotAlleles3
1	dDocent_Contig_3:146_G/A	G	A	34	35	11	1	0	8	35	35	19	40	40	40
2	dDocent_Contig_5:189_T/A	T	A	23	19	3	3	0	18	26	19	21	40	40	40
3	dDocent_Contig_6:140_T/C	T	C	13	46	16	0	5	26	13	51	42	40	40	40
4	dDocent_Contig_9:197_T/A	T	A	12	17	2	4	0	18	16	17	20	40	40	40
5	dDocent_Contig_10:11_A/T	A	T	12	10	1	9	0	10	21	10	11	40	40	40

Setting Function Parameters

The basic format of the function is:

```
AMOVAresults <- runAMOVA(designFile=design, dataFile=simdata,
outputFile="AMOVAoutput", NresamplesToStop=1000, maxPermutations=10000, multi.core
= T, do.bootstrap = T)
```

The first part, "AMOVAresults", indicates the name of the data frame that will result.

Your designFile and dataFile must both be specified. If reading in from a file, wrap the name in quotes (e.g., designFile = "design.csv"); if already an R data.frame, do not wrap it in quotes (e.g., designFile = design). See "Input Files" above for the correct formatting of the files.

For the sake of speed with our example, we'll restrict the number of iterations to 20 (maxPermutations=20), with the option to stop after 10 (NresamplesToStop=10) if the if the resampled F is consistently ≥ the observed F. (i.e., it is not going to be significant, and you don't want to wait the full set of iterations to confirm this). In a real analysis, these numbers should be increased considerably (defaults are NresamplesToStop=1000, maxPermutations=10,000), and can both be set to the same number if you want to run the full number of iterations without stopping early.

The next step is to set the appropriate model for the sampling variance. If you're analyzing next-generation sequencing data, make sure "NGSdata" is set to TRUE (default), so that the total sampling variance is modeled in the appropriate way. If you're analyzing Sanger sequence data, or want to see how many false positive "impostar" loci you would have gotten by ignoring the additional sampling error associated with next-generation sequencing, "NGSdata" must be set to FALSE to implement the appropriate method.

You are now ready to run the analysis using the default settings for the remaining parameters. To find out more information about additional parameters, run

```
?runAMOVA
```

Examples

If you're following on from the "Convert VCF to *impostar* format" section, you can run the AMOVA using the data and design files created there:

```
AMOVAresults <- runAMOVA(designFile= mydesignFile, dataFile=data.impostar,
NresamplesToStop=10, maxPermutations=20, do.bootstrap = T)
```

Alternatively, you can simulate an example data file and run it:

```
# create design file
design <- data.frame(n=rep(20,3), Sample=c(1,2,3))
# simulate data file
simdata <- simulate_data(rep(50, 3), rep(100, 3), rep(0.5, 3), 5, file_name=T)
# run AMOVA
AMOVAresults <- runAMOVA(designFile=design, dataFile=simdata, NresamplesToStop=10,
maxPermutations=20, do.bootstrap = T)
```

Interpreting the Output

The runAMOVA function returns a list of AMOVA tables. Have a look at the output by running:

```
AMOVAresults
```

The output from the first two SNPs of the example above using data.impostar and mydesignFile looks like this:

```
[[1]]
        df    SS         MS sigma squared        f  p #permutations
Sample   2  4.55 2.27500000    0.05457799 0.372652  0            20
Within 117 10.75 0.09188034    0.09188034 0.372652 NA            NA
Total  119 15.30         NA    0.14645833       NA NA            NA

[[2]]
        df     SS         MS sigma squared         f  p #permutations
Sample   2 16.850 8.42500000    0.20860043 0.7203468  0            20
Within 117  9.475 0.08098291    0.08098291 0.7203468 NA            NA
Total  119 26.325         NA    0.28958333        NA NA            NA
```

Each element [[#]] of the list, contains the AMOVA table for one SNP, numbered in the order they appeared in the input file.

The AMOVA table for a specific SNP can be viewed using its position number (in most cases, this will correspond to the row name in the input file). For example to see the results for the 5th SNP, run:

```
AMOVAresults[[5]]
```

Simulating Data

This function is not required for a typical analysis, but is included to provide an example of the correct input format, and for those who might want to explore type I and type II error.

The simulate_data function is used to create a biallelic SNP dataset similar to one produced by a next-generation sequencing project. The process simulates sampling alleles from a population, then sampling sequence reads from that sample of alleles.

Input Parameters

The user must specify:

- number of individuals in each population (nIndiv; a vector with an integer for each population)
- sequence reads per SNP per population (nReads; a vector with an integer for each population)
- reference allele frequencies per population (RefProb; a vector with a number (0-1) for each population)
- the number of populations is indicated by the length of the vectors for nIndiv, nReads, & RefProb)
- number of SNPs to be simulated (nSNPs; an integer)

Additionally, the user can choose to save the resulting dataset to a csv file by setting the file_name option to TRUE, which will assign a file name based on the number of individuals, reads, and allele frequencies, or by entering a character string for the file_name that specifies the name to be given to the file. The default is FALSE, meaning no file is saved.

The basic structure of the function is:

```
simdata <- simulate_data(nIndiv, nReads, RefProb, nSNPs, file_name=F)
```

Examples

To simulate 5 SNPs for 3 populations with 50 individuals sampled per population, 100 reads per SNP per population, a reference allele frequency of 0.5 in all populations, and an automatically named csv file saved to the working directory, run:

```
simdata <- simulate_data(rep(50, 3), rep(100, 3), rep(0.5, 3), 5, file_name=T)
```

To simulate 5 SNPs for 4 populations with unequal numbers of individuals, reads and allele frequencies per population, and a custom file name:

```
simdata <- simulate_data(nIndiv=c(20,45,50,55), nReads=c(100,115,200,150), RefProb
=c(0.1,0.15,0.5,0.9), nSNPs=5, file_name="mySimulatedData")
```

Output

The simulate_data function returns a data frame, and optionally saves it to a csv file. Have a look at the output by running:

```
simdata
```

The output will look similar to this:

```
                    snpID TotAlleles1 TotAlleles2 TotAlleles3 DP1 DP2 DP3 RefAl1 RefAl2 RefAl3 AltAl1 AltAl2 AltAl3
1 simSNPprob_0.5_0.5_0.5_1         100         100         100 100 100 100     48     52     60     52     48     40
2 simSNPprob_0.5_0.5_0.5_2         100         100         100 100 100 100     45     52     48     55     48     52
3 simSNPprob_0.5_0.5_0.5_3         100         100         100 100 100 100     51     44     49     49     56     51
4 simSNPprob_0.5_0.5_0.5_4         100         100         100 100 100 100     49     47     45     51     53     55
5 simSNPprob_0.5_0.5_0.5_5         100         100         100 100 100 100     49     57     64     51     43     36
  RD1 RD2 RD3 AD1 AD2 AD3
1  41  47  59  59  53  41
2  47  55  55  53  45  45
3  51  40  47  49  60  53
4  45  41  38  55  59  62
5  50  51  62  50  49  38
```

Each row of the data frame is one SNP locus. The columns contain:

TotAlleles1…*n*	total number of alleles (2n) per population
DP1…n	total read depth per population
RefAl1…n	number of reference alleles sampled per population
AltAl1…n	number of alternate alleles sampled per population
RD1…n	number of reference reads observed per population
AD1…n	number of alternate reads observed per population

Each population is indicated by the trailing number in the column names; for example, the first population is represented by TotAlleles1, DP1, RefAl1, AtlAl1, RD1 and AD1.

The output of this function will show the correct input file format for the runLogRegTest or runAMOVA functions included in this package.

Congratulations!

You can now minimize false positive 'impostar' loci in your next-generation sequencing projects!