RAREsim Vignette

Megan Null

This vigette describes how to use the RAREsim R package to simulate rare variant genetic data.

The example below simulates a one cM block on chromosome 19. Here, RAREsim simulates haplotypes to match target data from the African ancestry group from gnomAD v2.1 (Karczewski, et al., 2020).

Install the package

```
library(RAREsim)
```

The source code for all functions within the RAREsim package can be found at https://github.com/meganmichelle/RAREsim. The package currently must be downloaded through github using devtools.

RAREsim has three main steps: (1) simulate genetic data with an abundance of rare variants using HAPGEN2 (Su, 2011), (2) estimate the expected number of variants in MAC bins, and (3) probabilistically prune the rare variants to match the estimated number of variants in each MAC bin.

An example simulation with HAPGEN2 can be found on the RAREsim Example Code github page. By simulating with default parameters and input haplotypes with information at all sequencing bases, including monomorphic sites, HAPGEN2 simulates an abundance of rare variants.

In order to emulate real sequencing data, RAREsim prunes the simulated variants by returning all or a subset of alternate alleles back to reference. In order to prune, RAREsim first estimates the expected number of variants within MAC bins. The number of variants in each MAC bin can either estimated using default parameters, modifying default parameters, or fitting target data. If the exact sample size of observed sequencing data is to be simulated, the observed data can be matched directly.

The Number of Variants function

For a given region, the *Number of Variants* function estimates the number of variants per Kb, $f_{nvariant}(n)$, for a sample size n. Estimating the number of variants can be achieved by 1) Fitting target data to estimate parameters, 2) Using default parameters, or 3) directly inputting parameters to the function. Additionally, a user may directly input the number of variants expected in the region (e.g. 1000 variants).

1) Fitting Target Data

Target data is used to estimate ϕ and ω to optimize the function $f_{nvariant}(n) = \phi n^{\omega}$ to fit the target data.

The Number of Variants target data consists of various sample sizes (n) and the observed number of variants per Kb in the region of interest. Ancestry specific data is advised. Data should be formatted with the first column as the number of individuals (n) and the second column as the observed number of variants per Kb in the region of interest (per_{kb}).

Here we will fit the example data for the African ancestry population. Example data is available in the R package for each of the four ancestries: African (AFR), East Asian (EAS), Non-Finnish European (NFE), and South Asian (SAS).

```r
# load the target data
data("nvariant_afr")
print(nvariant_afr, row.names  =  FALSE)
```

```
##     n     per_kb
##    10  0.2627568
##    20  0.6831678
##    50  1.5239897
##   100  2.7326712
##   200  4.3092123
##   500  7.6199485
##  1000 12.1919176
##  2000 19.3914551
##  3070 25.2246571
##  5000 33.4226707
##  5040 33.7905302
##  8128 45.1941773
```

The target data is used to estimate ϕ and ω within a least squares loss function, optimizing using sequential quadratic programming (SQP). This optimization is implemented via the *fit_nvariant* function.

```r
nvar <- fit_nvariant(nvariant_afr)
nvar
```

```
## $phi
## [1] 0.1638108
##
## $omega
## [1] 0.6248848
```

The output of the *fit_nvariant* function are the parameters phi (ϕ) and omega (ω), respectively. The estimated parameters can then be used to determine the expected number of variants per Kb within the region of interest, given the number of individuals to be simulated, N_{sim}.

For example, to simulate the sample size observed in the target data, ($N_{sim} = 8128$), we calculate $\hat{f_{nvariant}}(N_{sim}) = \hat{\phi} N_{sim}^{\hat{\omega}}$. This can be done with the *nvariant* function.

Parameter values for phi (ϕ), omega (ω), and the sample size (n) found previously are used here.

```r
nvariant(phi = nvar$phi, omega = nvar$omega, N = 8128)
```

```
## [1] 45.46027
```

2) Using Default Parameters

RAREsim also provides ancestry specific default parameters for phi (ϕ), omega (ω). To use the default parameters, the ancestry must be specified: African (AFR), East Asian (EAS), Non-Finnish European (NFE), or South Asian (SAS).

```r
nvariant(N=8128, pop = 'AFR')
```

```
## [1] 43.66395
```

3) Directly Inputting Parameters

Finally, parameters can be directly input into the *Number of Variants* function.

```r
nvariant(phi = 0.1638108, omega = 0.6248848, N = 8128)
```

```
## [1] 45.46026
```

Total Number of Variants in the Region

The example data here is a cM block with 19,029 bp. Thus, to calculate the total expected number of variants in the region, we multiple the expected number of variants per Kb (*nvariant*) by 19.029.

```
19.029*nvariant(phi = 0.1638108, omega = 0.6248848, N = 8128)
```

```
## [1] 865.0633
```

The Allele Frequency Spectrum (AFS) Function

The *AFS* function inputs a MAC (z) and outputs the proportion of variants at MAC = z, ($f_{afs}(z)$). This is done by estimating α and β to optimize the function $f_{afs}(z) = \frac{b}{(\beta+z)^{\alpha}}$. Here b ensures that the sum of the individual rare allele count proportions equals the total proportion of rare variants, p_{rv}.

The *AFS* function inputs a data frame with the upper and lower boundaries for each bin and proportion of variants within each respective bin. The default bins used here and within the evaluation of RAREsim are:

MAC = 1
MAC = 2
MAC = 3 - 5
MAC = 6 - 10
MAC = 11 - 20
MAC = 21 - MAF = 0.5%
MAC = 0.5% - MAF = 1%

Estimating the AFS can be achieved by 1) Fitting target data to estimate parameters, 2) Using default parameters, or 3) directly inputting parameters to the function. Additionally, a user may directly input the proportion of variants in each MAC bin.

1) Fitting Target Data

Below is an example of the AFS target data for the African ancestry group. The first two columns identify the lower and upper boundaries of each MAC bin. The third column specifies the observed proportion of variants within each MAC bin in the target data.

```
# load the data
data("afs_afr")
colnames(afs_afr)[3] <- 'Prop'
print(afs_afr)
```

```
##   Lower Upper       Prop
## 1     1     1 0.50257998
## 2     2     2 0.16305470
## 3     3     5 0.08255934
## 4     6    10 0.05882353
## 5    11    20 0.03715170
## 6    21    81 0.05675955
## 7    82   162 0.01754386
```

To fit the *AFS* function (*fit_afs*), RAREsim requires the data frame with MAC bins and proportion of variants (shown above), the number of subjects to simulate N, and the total proportion of rare variants, p_{rv}. Here, we will simulate the sample size observed in gnomAD, $N = 8128$. The function estimates the parameters alpha (α), beta (β), and b, and includes the estimated proportion of variants based on calculations from the fitted parameters, as shown below.

```r
af <- fit_afs(Observed_bin_props = afs_afr)
print(af)
```

```
## $alpha
## [1] 1.594622
##
## $beta
## [1] -0.2846474
##
## $b
## [1] 0.297495
##
## $Fitted_results
##   Lower Upper       Prop
## 1     1     1 0.50753380
## 2     2     2 0.12582725
## 3     3     5 0.12226962
## 4     6    10 0.06152310
## 5    11    20 0.04187244
## 6    21    81 0.04709594
## 7    82   162 0.01235050
```

2) Using Default Parameters

As with the *Variants per Kb* function, default parameters can be used to estimate the parameters for the *AFS* function with the *afs* function. As the default parameters are ancestry specific, the ancestry needs to be specified as pop = AFR, EAS, NFE, or SAS when default parameters are used. The parameters alpha (α), beta (β), and b can be specified, or default parameters can be used. Both implementations of the function require a MAC bin dataframe, with the bins specified.

This is the first two columns of the AFS target data.

```r
mac <- afs_afr[,c(1:2)]
```

Using the MAC bins as input and specifying an African ancestry, the default parameters are used below to estimate the proportion of variants within each bin.

```r
afs(mac_bins = mac, pop = 'AFR')
```

```
##   Lower Upper       Prop
## 1     1     1 0.51573550
## 2     2     2 0.12461241
## 3     3     5 0.12033906
## 4     6    10 0.06047216
## 5    11    20 0.04122759
## 6    21    81 0.04658283
## 7    82   162 0.01229418
```

3) Directly Inputting Parameters

The afs function can inputs the parameters alpha, beta, and b, along with the MAC bin endpoints.

```r
afs(alpha = 1.594622, beta =  -0.2846474, b  = 0.297495, mac_bins = mac)
```

```
##   Lower Upper       Prop
## 1     1     1 0.50753386
## 2     2     2 0.12582721
```

```
## 3     3     5 0.12226954
## 4     6    10 0.06152304
## 5    11    20 0.04187238
## 6    21    81 0.04709586
## 7    82   162 0.01235047
```

Expected Number of Variants per MAC bin

Using the parameter estimates from the *Number of Variants* and *AFS* functions, the expected number of variants in each MAC bin can be estimated. An example using the total number of varants and estimated proportion of variants per MAC bin is shown below.

```
bin_estimates <- expected_variants(Total_num_var = 865.0633, mac_bin_prop = af$Fitted_results)
print(bin_estimates)
```

```
##   Lower Upper Expected_var
## 1     1     1    439.04886
## 2     2     2    108.84854
## 3     3     5    105.77096
## 4     6    10     53.22138
## 5    11    20     36.22231
## 6    21    81     40.74097
## 7    82   162     10.68396
```

The output of the *expected_variants* function is the exected number of variants in each MAC bin within the simulation region. This output (shown above) is input for the pruning function.

The *Number of Variants* and *AFS* function can also be calculated within the *expected_variants* function.

```
bin_estimates <- expected_variants(Total_num_var = 19.029*nvariant(phi = 0.1638108, omega = 0.6248848, l
print(bin_estimates)
```

```
##   Lower Upper Expected_var
## 1     1     1    446.14384
## 2     2     2    107.79762
## 3     3     5    104.10090
## 4     6    10     52.31224
## 5    11    20     35.66447
## 6    21    81     40.29710
## 7    82   162     10.63524
```

Pruning Variants

In order to use RAREsim to prune simulated data, genetic data must be simulated with HAPGEN2 with all sequencing bases, including monomorphic variants, added to the input haplotypes. HAPGEN2 will simulate an abudance of rare variants to allow for variant pruning. Additionally, a MAC file (count of the number of alternate alleles at each bp) enables an efficient and fast pruning process. It is recommended to create the MAC file within the process of simulating data with HAPGEN2, as shown in the example code that is available on the RAREsim Example code github page.

Below is an example MAC file created from the haplotypes simulated for the African ancestry group and the region of interest. Each row represents one bp in the haplotype file.

```
data("MAC_afr")
```

Pruning happens in two stages: 1) RAREsim theoretically decides which variants should be pruned and 2) Prunes the haplotype and legend files

1) Theoretically Prune

Pruning variants requires a MAC file from the simulated data and the expected number of variants within each MAC bin (product of the *expected_variants* function).

```
ToPrune <- prune_variants(MAC = MAC_afr, expected = bin_estimates)
head(ToPrune$ToRemove, row.names = FALSE)
```

```
##      line Current_mac New_mac
## 382   382         137       0
## 616   616          99       0
## 1736 1736         105       0
## 2060 2060         117       0
## 2227 2227         105       0
## 5314 5314         147       0
```

```
head(ToPrune$ToChange)
```

```
## NULL
```

The output from *prune_variants* includes the variants to prune, notated by what line they are in the haplotype file, the current allele count, and the new minor allele count. Variants that will either have all or a subset of minor alleles removed.

2) Pruning Implementation

There is (or soon will be) an example on the RAREsim github page that shows how to implement the pruning.

Pruning requires the haplotype and legend files that are being pruned and the output from the *prune_variants* function.