A Tutorial for RADseq_Tools

Angel Rivera-Colon & Kira Long

1 Introduction:

Restriction site-Associated DNA sequencing (RADseq) is a modern genomic sequencing technique that has greatly advanced the field of population genomics. As a reduced representation-sequencing method, RADseq has decreased per-individual sequencing costs, making genome-wide studies more widely accessible to the research community (Catchen, et al. 2017). Additionally, it allows for the cost-effective sequencing and genotyping of multiple individuals, which is particularly important when discerning genetic variation in natural populations (Narum, et al. 2013). Moreover, RADseq's compatibility with multiple genomic approaches, including phylogenomics, de novo population genomics, and genomic scans, have made it a key player in modern population genomics (Andrews, et al. 2016).

Although RADseq studies require relatively little previous information on the genome of interest, the selection of restriction enzymes and its subsequent effect on marker density remains one of RADseq's limiting factors (Lowry, et al. 2017). The selection of a poor enzyme can then lead to genome undersampling, resulting in low marker density, or oversampling, affecting the resulting downstream coverage. Based on this, we developed an `R` package pipeline for the exploratory analysis of RADseq data to guide researchers in planning their study and improving data quality. `RADseq_tools` estimates important information for the initial experimental design, including number of cutsites, and marker density and distribution for different user-defined restriction enzyme sites and reference genomes. Even when the reference of interest is not available, the user can explore with available genomes in related taxa and use this information to extrapolate the marker properties for their experiments. In addition, this package can assist with determining the cost of a RADseq experiment by providing estimates of expected coverage, number of samples per lane, and required read throughput.

2 Getting started:

2.1 Verifying R version

To run *RADseq_Tools*, make sure you have a recent `R` installation. The package was written using `R version 3.4.2`, but should work with versions above `3.1.0`. To verify the installed `R` version, use:

```
> R.version.string
```

```
[1] "R version 3.4.2 (2017-09-28)"
```

2.2 Installation

Install the *RADseq_Tools* package from CRAN using:

```
> install.packages("RADseq_Tools")
```

Load the package for use with:

```
> library(RADseq_Tools)
```

Alternatively, source the functions using:

```
> source("./../R/Functions.R")
```

Additional information on the package can be found in:
https://github.com/angelgr2/radseq_tools

3 Tutorial:

This brief guide will provide examples of all the package's basic functions and arguments. It uses a test dataset available with the package. It consist of a small FASTA file containing the first 100Kb of sequence of six Three Spine Stickleback (*Gasterosteus aculeatus*) chromosomes.

3.1 Load and process reference sequence

To begin, we need to load the reference sequence into a `RADseq_Tools` object. This new `sequence` object can then be used to search for restriction sites. Create a variable containing the path to the reference sequence.

```
> fasta_dir <- "./../inst/extdata/test_geno.fa.gz"
```

Because this test dataset is available with the package installation, it can also be obtained from the `extdata` by using:

```
> fasta_dir <- system.file("inst/extdata",
+                          "test_geno.fa.gz",
+                          package = "RADseq_Tools")
```

This path variable is then the input of the `process_fasta` function, which reads the file provided in the path, converting multi-line sequence objects into single line sequences. Sequences bellow 10kb are filtered and not included in the resulting sequences object. The output of the function is a vector containing the sequence strings for each of the chromosomes/scaffofds in the FASTA file. Here, we are creating the object `mySeqs` using the output of process_fasta, and displaying the first few nucleotides for each of the chromosomes.

```
> mySeqs <- process_fasta(fasta_dir, 10000)
> substr(mySeqs, 1, 60)
```

```
[1] "CTCTTTGTTTTCAGGTGTGGAATGTGCTTTCTACCACGGCTACAAATACTACAAAGGATG"
[2] "TGACTATTAAGGCGTTTGCAGGCTGAGAGAAGCCAGTCTTGAATGCTACCCCCTTTTGAG"
[3] "GCTAGCCTGTTTAAACCAAACCATCGGTGTGTATGATTACTTGCGCCCACACCCGGTCTC"
[4] "GTATTATATAGTAAATACTATACATTTTCTCTACAGATAGTACAGTGAGTTTACTCTACA"
[5] "GCTAATATTTTTATGGCTGAGTGGGAGGATTCAGTCCTGAAAATGTGTCCTAAAAAACCA"
[6] "AACCCATCGCCTTATAGGCTGTACTTTATCCTCCAATGATGGAGCAGTTCTCCGCTGCGC"
```

3.2 Find restriction sites in the reference sequences

The first part of finding the restriction cutsite position is defining the query to be searched, the sequence of the restriction enzyme. The package data contains sequences for four commonly used restriction enzymes stored in a vector called `renz`. They can be accesed in the following way:

```
> data(renz)
> renz

      pstI       sbfI      ecoRI    hindIII
  "CTGCAG" "CCTGCAGG"   "GAATTC"   "AAGCTT"
```

In this example, we are using the enzyme `SbfI`, which cuts at the sequence `CCTGCAGG`.

```
> SbfI <- renz["sbfI"]
> SbfI

      sbfI
"CCTGCAGG"
```

Once the restriction site sequence is defined, it will be searched in the stored sequences using the `find_cuts` function. Using `gregexpr`, it will find all matches of the restriction site query in the reference sequences. The resulting output is a list of vectors, containing all the per-chromosome/scaffold match positions.

Here, we see the cutsite position for the first three test sequences:

```
> myCuts <- find_cuts(mySeqs, SbfI)
> myCuts[c(1,2,3)]

[[1]]
[1] 11812 33253 49707 65489 65747 65834 67213 89247 92064

[[2]]
[1]   330 37897 75355 89527

[[3]]
[1]  3941 15515 28777 40042 47416 63187 70615 98324
```

3.3 Find cutsite distribution

After determining the position of the cutsites, it is also important to determine the distribution of cutsites in the genome, as it gives us an idea of the number of cutsites and the average inter-cutsite distance. The distribution can be calculated using the `cutsite_distance` function. Using the cutsite position list resulting from the `find_cuts` function, it will output a single vector containing the inter-cutsite distance for all cutsites in the genome.

```
> cutDist <- cutsite_distance(myCuts)
> cutDist[1:10]
```

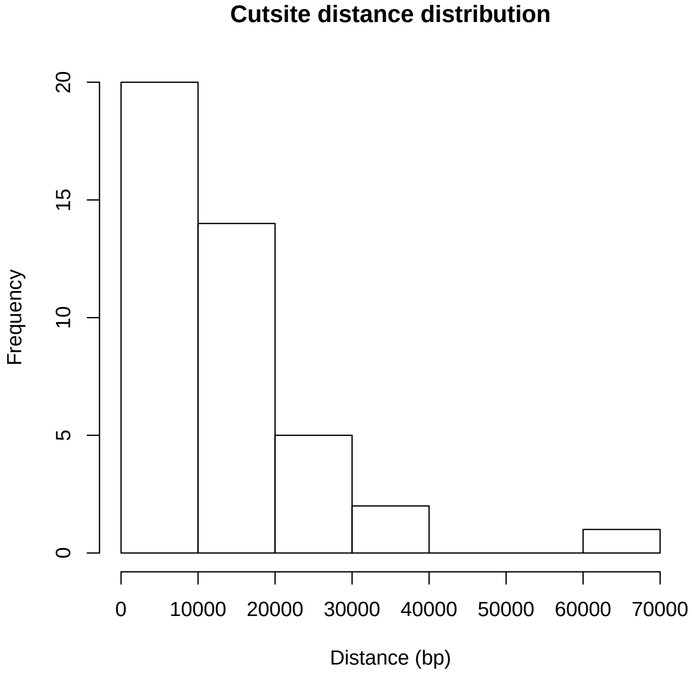
```
 [1] 11812 21441 16454 15782   258    87  1379 22034  2817   330
```

The resulting vector is compatible with all `R` vector functions, thus other statistical measures can be obtained. For example:

```
> summary(cutDist)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
     87    3098   10863   12603   15733   69883 
```

```
> hist(cutDist, xlab="Distance (bp)",
+      ylab="Frequency", main="Cutsite distance distribution" )
```



3.4 Find number of cutsites

In addition to finding the inter-cutsite distance distribution, we can also determine the total number of cutsites using the number_cutsites functions. It takes as input the original cutsite positions object product of the find_cuts function, and outputs a single numeric value.

```
> nCuts <- number_cutsites(myCuts)
> nCuts
```

```
[1] 42
```

Alternatively, the number of cutsites can be manually calculated from the length of the cutsite distribution object.

```
> length(cutDist)
```

```
[1] 42
```

3.5 Estimate per-sample coverage

Using the estimations of cutsite distribution, the user can estimate the expected per-sample coverage when running a sequencing experiment using the Per_Sample_Coverage function. The function requires the number of cutsites (likely calculated with the number_cutsites function), number of samples, and the type of sequencing machine used, either Illumina HiSeq2500 ("hiseq2500") or HiSeq4000 ("hiseq4000") with their corresponding read throughput. These can be accessed throught the illumina dataset available with the package.

```
> data(illumina)
> hiseq2500
```

```
    Low     Med      Hi
2.2e+08 3.1e+08 4.0e+08
```

```
> hiseq4000
```

```
    Low     Med      Hi
5.0e+08 7.5e+08 1.0e+09
```

The output is a vector contaning the estimated per-sample coverage for the machine's low, medium and high read throughput. Here's an example:

```
> nCuts <- 35000      # Number of cutsites.
> nSams <- 96         # Number of multiplexed samples
> illumina <- "hiseq2500"
> Per_Sample_Coverage(nCuts, nSams, illumina)
```

```
  Low   Med    Hi
32.74 46.13 59.52
```

```
> illumina <- "hiseq4000"
> Per_Sample_Coverage(nCuts, nSams, illumina)
```

```
  Low   Med    Hi
 74.4 111.6 148.8
```

3.6 Estimate samples per lane

The package can also be used to calculate an estimation of the number of samples that can be sequenced in a single Illumina sequencing lane for a given coverage. The function works similarly to `Per_Sample_Coverage`, as it takes the number of cutsites, a desired target coverage, and a type of Illumina sequencing machine. It outputs a vector containing calculated number of samples for the machine's low, medium and high read throughputs.

```
> nCuts <- 35000     # Number of cutsites.
> cover <- 45        # Target per-sample average coverage
> illumina <- "hiseq2500"
> Samples_Per_Lane(nCuts, cover, illumina)
```

```
Low Med  Hi
 70  98 127
```

```
> illumina <- "hiseq4000"
> Samples_Per_Lane(nCuts, cover, illumina)
```

```
Low Med  Hi
159 238 317
```

3.7 Estimate required sequencing reads

Given a fixed number of cutsites, number of samples, and target per-sample coverage, the `DNA_Reads_Per_Lane` function calculates an estimate of the required number of reads. It can be used to determine the type of sequencing machine, and number of sequencing lanes, necessary for the experiment.

```
> nCuts <- 35000     # Number of cutsites.
> nSams <- 96        # Number of multiplexed samples
> cover <- 45        # Target per-sample average coverage
> DNA_Reads_Per_Lane(nCuts, nSams, cover)
```

```
[1] 3.02e+08
```

4 Other applications:

An important application of the `RADseq_Tools` package is to compare the use of different datasets made with different restriction enzymes. The purpose of this application is to assist in the experimental design of RADseq projects, as choosing the right enzyme is fundamental for the success of the experiment.

To begin, we will process the test reference genome as before, storing the sequences in the `mySeqs` object.

```
> fasta_dir <- "./../inst/extdata/test_geno.fa.gz"
> mySeqs <- process_fasta(fasta_dir, 10000)
```

After storing the sequences, we will find cutsites using three different restriction enzymes: `SbfI`, `EcoRI` and `HindIII`. The cutsite sequences are available in the `renz` vector.

```
> # For SbfI
> SbfI <- renz["sbfI"]
> SbfI_Cuts <- find_cuts(mySeqs, SbfI)
> SbfI_Dist <- cutsite_distance(SbfI_Cuts)
> SbfI_num <- number_cutsites(SbfI_Cuts)

> # For EcoRI
> EcoRI <- renz["ecoRI"]
> EcoRI_Cuts <- find_cuts(mySeqs, EcoRI)
> EcoRI_Dist <- cutsite_distance(EcoRI_Cuts)
> EcoRI_num <- number_cutsites(EcoRI_Cuts)

> # For HindIII
> HindIII <- renz["hindIII"]
> HindIII_Cuts <- find_cuts(mySeqs, HindIII)
> HindIII_Dist <- cutsite_distance(HindIII_Cuts)
> HindIII_num <- number_cutsites(HindIII_Cuts)
```

By displaying the cutsites for just the first chromosome/scaffold, we can see that the cutsite positions in the genome for each enzyme is different for each enzyme:

```
> SbfI_Cuts[[1]]

[1] 11812 33253 49707 65489 65747 65834 67213 89247 92064

> EcoRI_Cuts[[1]]

[1]  8919 32571 58427 62641 90174

> HindIII_Cuts[[1]]

 [1]   211   795  1248  2639 11265 11611 11943 18494 21615 29090 32550 36482
[13] 41716 42491 44116 45708 53990 54429 56980 60812 64051 64074 65401 82684
[25] 86097 98196 98628
```

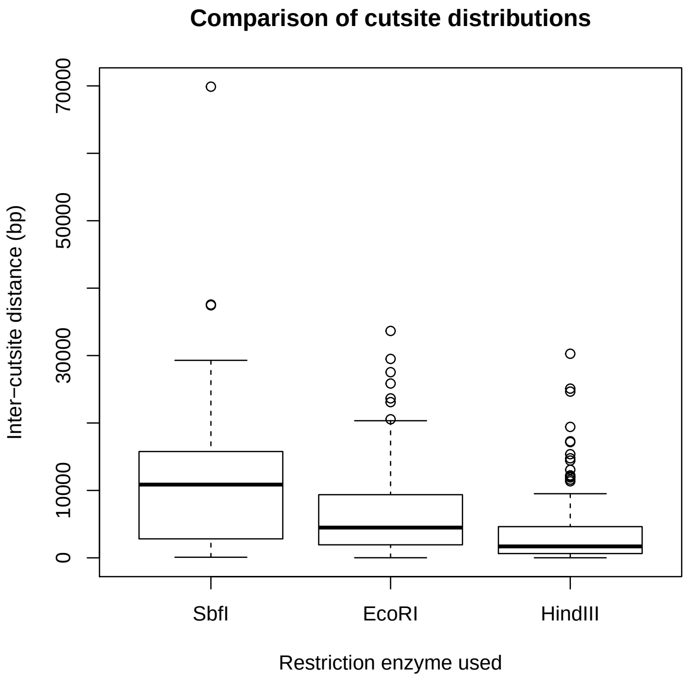
Also, the total number of cutsites is different:

```
> SbfI_num
```

```
[1] 42

> EcoRI_num

[1] 80

> HindIII_num

[1] 151
```

And have completely different inter-cutsite distance distributions:

```
> boxplot(SbfI_Dist, EcoRI_Dist, HindIII_Dist,
+         names = c("SbfI", "EcoRI", "HindIII"),
+         xlab="Restriction enzyme used",
+         ylab="Inter-cutsite distance (bp)",
+         main="Comparison of cutsite distributions")
```



With this information, the user can have an idea of how the different enzymes behave in the reference genome of interest, or related genomes. Based on this information, the user can then decide which enzyme to use based on marker desity and raw number of sites, among other variables.

5 References:

Andrews, K. R., Good, J. M., Miller, M. R., Luikart, G., & Hohenlohe, P. A. (2016). Harnessing the power of RADseq for ecological and evolutionary genomics. *Nature Reviews. Genetics*, 17(2), 81–92. doi.org/10.1038/nrg.2015.28

Catchen, J. M., Hohenlohe, P. A., Bernatchez, L., Funk, W. C., Andrews, K. R. and Allendorf, F. W. (2017), Unbroken: RADseq remains a powerful tool for understanding the genetics of adaptation in natural populations. *Mol Ecol Resour*, 17: 362–365. doi:10.1111/1755-0998.12669

Lowry DB, Hoban S, Kelley JL, et al. (2017). Responsible RAD: Striving for best practices in population genomic studies of adaptation. *Mol Ecol Resour.* 17:366–369. doi.org/10.1111/1755-0998.12677

Narum, S. R., Buerkle, C. A., Davey, J. W., Miller, M. R. and Hohenlohe, P. A. (2013), Genotyping-by-sequencing in ecological and conservation genomics. *Molecular Ecology*, 22: 2841–2847. doi: 10.1111/mec.12350