sangeranalyseR

Jul 19, 2021

Contents

1 Why sangeranalyseR **1**

2 Main features **3**

3 What sangeranalyseR doesn't do **5**

4 User Manual **7**

5 User support **9**

6 Key contributors **11**

7 Documentation **13**
7.1 Installation 13
7.1.1 System requirements 13
7.1.2 Install from Bioconductor 13
7.1.3 Install the development version 14
7.1.4 Where to go from here ? 14
7.2 Quick Start Guide 14
7.2.1 Super-Quick Start (3 lines of code) 15
7.2.2 Step 1: Prepare your input files 15
7.2.3 Step 2: Load and analyse your data 15
7.2.4 Step 3 (optional): Explore your data 15
7.2.5 Step 4: Output your aligned contigs 16
7.2.6 Step 5 (optional): Generate an interactive report 16
7.2.7 A Reproducible Example 16
7.3 Beginners Guide 18
7.3.1 Step 1: Preparing your input files 18
7.3.2 Step 2: Loading and analysing your data 20
7.3.3 Step 3: Exploring your data with the Shiny app 20
7.3.4 Step 4: Outputting your aligned contigs 21
7.3.5 Step 5: Generating an interactive report 21
7.3.6 What's next ? 22
7.4 Advanced User Guide - *SangerRead* (**AB1**) 22
7.4.1 Preparing *SangerRead* **AB1** input 22
7.4.2 Creating *SangerRead* instance from **AB1** 23
7.4.3 Visualizing *SangerRead* trimmed read 25

7.4.4 Updating *SangerRead* quality trimming parameters . . . 25
7.4.5 Writing *SangerRead* FASTA file (AB1) . . . 25
7.4.6 Generating *SangerRead* report (AB1) . . . 26
7.4.7 Code summary (*SangerRead*, **ab1**) . . . 26
7.5 Advanced User Guide - *SangerContig* (**AB1**) . . . 27
7.5.1 Preparing *SangerContig* **AB1** inputs . . . 28
7.5.2 Creating *SangerContig* instance from **AB1** . . . 31
7.5.3 Updating *SangerContig* quality trimming parameters . . . 35
7.5.4 Launching *SangerContig* Shiny app . . . 35
7.5.5 Writing *SangerContig* FASTA files (AB1) . . . 41
7.5.6 Generating *SangerContig* report (AB1) . . . 44
7.5.7 Code summary (*SangerContig*, **AB1**) . . . 44
7.6 Advanced User Guide - *SangerAlignment* (**AB1**) . . . 47
7.6.1 Preparing *SangerAlignment* **AB1** input . . . 47
7.6.2 Creating *SangerAlignment* instance from **AB1** . . . 52
7.6.3 Updating *SangerAlignment* quality trimming parameters . . . 56
7.6.4 Launching *SangerAlignment* Shiny app . . . 57
7.6.5 Writing *SangerAlignment* FASTA files (AB1) . . . 64
7.6.6 Generating *SangerAlignment* report (AB1) . . . 67
7.6.7 Code summary (*SangerAlignment*, **AB1**) . . . 68
7.7 Advanced User Guide - *SangerRead* (**FASTA**) . . . 70
7.7.1 Preparing *SangerRead* **FASTA** input . . . 70
7.7.2 Creating *SangerRead* instance from **FASTA** . . . 71
7.7.3 Writing *SangerRead* FASTA files (FASTA) . . . 72
7.7.4 Generating *SangerRead* report (FASTA) . . . 72
7.7.5 Code summary (*SangerRead*, **fasta**) . . . 72
7.8 Advanced User Guide - *SangerContig* (**FASTA**) . . . 74
7.8.1 Preparing *SangerContig* **FASTA** input . . . 74
7.8.2 Creating *SangerContig* instance from **FASTA** . . . 77
7.8.3 Writing *SangerContig* FASTA files (FASTA) . . . 80
7.8.4 Generating *SangerContig* report (FASTA) . . . 81
7.8.5 Code summary (*SangerContig*, **FASTA**) . . . 81
7.9 Advanced User Guide - *SangerAlignment* (**FASTA**) . . . 83
7.9.1 Preparing *SangerAlignment* **FASTA** input . . . 83
7.9.2 Creating *SangerAlignment* instance from **FASTA** . . . 87
7.9.3 Writing *SangerAlignment* FASTA files (FASTA) . . . 90
7.9.4 Generating *SangerAlignment* report (FASTA) . . . 90
7.9.5 Code summary (*SangerAlignment*, **FASTA**) . . . 91
7.10 Q & A 93
7.10.1 What is a regular expression? . . . 93
7.10.2 How to deal with secondary peaks . . . 93
7.10.3 How to work with FASTA files for input . . . 93
7.11 User Manual (functions) . . . 93
7.11.1 SangerRead Constructor Parameters . . . 93
7.11.2 SangerContig Constructor Parameters . . . 94
7.11.3 SangerAlignment Constructor Parameters . . . 96
7.12 Frequently Asked Questions . . . 98
7.12.1 Q: What is the difference between two different trimming methods? . . . 98
7.13 Conclusion . . . 99
7.14 License . . . 99
7.15 Contact . . . 99
7.16 Help . . . 99
7.16.1 Inside help test . . . 99

CHAPTER 1

Why sangeranalyseR

sangeranalseR is an R package that provides fast, flexible, and reproducible workflows for assembling your sanger seuqencing data into contigs.

It adds to a list of already widely-used tools, like Geneious, CodonCode Aligner and Phred-Phrap-Consed;. What makes it different from these tools is that it's free, it's open source, and it's in R.

CHAPTER 2

Main features

- **Pure R environment**: As far as we know, this is the first package that allows end-to-end analysis of Sanger sequencing data in a pure R environment.
- **Automated data analysis**: Given appropriately-named input files, a lot of the data analysis can be automated. Once you've set up an appropriate workflow for your data, you can run it again in seconds.
- **Interactive Shiny apps**: Local Shiny apps mean you visualize the data at many levels, view chromatograms, and adjust things like trimming parameters.
- **Exporting and importing FASTA files**: sangeranalyseR is primarily designed with loading raw `ab1` files in mind, but it can also load sequencesin **FASTA** format. Aligned results and trimmed reads can be written into **FASTA** file format.
- **Thorough report**: A single command creates a comprehensive interactive HTML report that provides a huge amount of detail on the analysis.

CHAPTER 3

What sangeranalyseR **doesn't** do

One really important feature that sangeranalyseR doesn't have is the ability to edit bases by hand. R is just not the right language for this. If you need to edit your reads by hand, we suggest doing that in another tool like Geneious, then exporting your reads as FASTA files and following the instructions for using sangeranalyseR with FASTA input.

CHAPTER 4

User Manual

If you are already familiar with sangeranalyseR and want to have a quick look at function signatures, please refer to sangeranalyseR user manual

CHAPTER 5

User support

Please go through the *Documentation* below first. If you have questions about using the package, a bug report, or a feature request, please use the GitHub issue tracker here:

https://github.com/roblanf/sangeranalyseR/issues

CHAPTER 6

Key contributors

The first (and not very good) version of the package was written by Rob Lanfear (at ANU in Australia), in collaboration with Kirston Barton and Sarah Palmer (then both at the University of Sydney). The second and far far better version of the package was written by Kuan-Hao (Howard) Chao at ANU. (This section was written by Rob Lanfear, lest you think Howard wrote it!)

CHAPTER 7

Documentation

7.1 Installation

7.1.1 System requirements

- R >= 4.0.0 (current)
- Rstudio (recommended)

7.1.2 Install from Bioconductor

sangeranalyseR is on Bioconductor 3.12 development now.

To install this package, start R (version "4.0") and enter:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

# The following initializes usage of Bioc devel
BiocManager::install(version='devel')

BiocManager::install("sangeranalyseR")
```

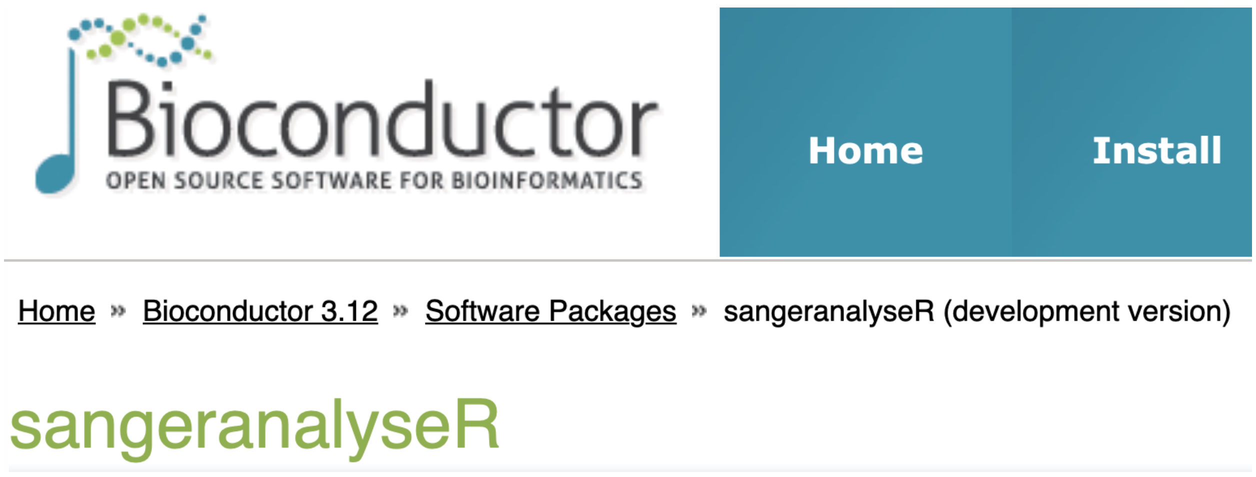

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS
Home
Install
Home » Bioconductor 3.12 » Software Packages » sangeranalyseR (development version)
sangeranalyseR

Fig. 1: Figure 1. sangeranalyseR on Bioconductor 3.12 development.

7.1.3 Install the development version

If you haven't installed the `devtools` package before, please install it first:

```
install.packages("devtools")
```

Then run the following code in your R console to install the newest version from Github.

```
library(devtools)
install_github("roblanf/sangeranalyseR", ref = "develop")
library(sangeranalyseR)
```

After installing `sangeranalyseR`, load it in R console.

```
library(sangeranalyseR)
```

Now, you are ready to go !

7.1.4 Where to go from here ?

Please continue to the *Quick Start Guide* or the more detailed *Beginners Guide*.

7.2 Quick Start Guide

This page provides simple quick-start information for using sangeranalyseR with `AB1` files. Please read the *Beginners Guide* page for more details on each step.

If you haven't already, please follow the steps in the *Installation* page to install and load sangeranalyseR.

7.2.1 Super-Quick Start (3 lines of code)

The most minimal example gets the job done in three lines of code. More details below.

```
my_aligned_contigs <- SangerAlignment(ABIF_Directory      = "./my_data/",
                                      REGEX_SuffixForward = "_[0-9]*_F.ab1$",
                                      REGEX_SuffixReverse = "_[0-9]*_R.ab1$")

writeFasta(my_aligned_contigs)

generateReport(my_aligned_contigs)
```

7.2.2 Step 1: Prepare your input files

Put all your `AB1` files in a directory `./my_data/`. The directory can be called anything.

Name your files according to the convention `contig_index_direction.ab1`. E.g. `Drosophila_COI_1_F.ab1` and `Drosophila_COI_2_R.ab1` describes a forward and reverse read to assemble into one contig. You can have as many files and contigs as you like in one directory.

7.2.3 Step 2: Load and analyse your data

```
my_aligned_contigs <- SangerAlignment(ABIF_Directory      = "./my_data/",
                                      REGEX_SuffixForward = "_[0-9]*_F.ab1$",
                                      REGEX_SuffixReverse = "_[0-9]*_F.ab1$")
```

This command loads, trims, builds contigs, and aligns contigs. All of these are done with sensible default values, which can be changed. I

7.2.4 Step 3 (optional): Explore your data

```
launchApp(my_aligned_contigs)
```

This launches an interactive Shiny app where you can view your analysis, change the default settings, etc.

7.2.5 Step 4: Output your aligned contigs

```
writeFasta(my_aligned_contigs)
```

This will save your aligned contigs as a FASTA file.

7.2.6 Step 5 (optional): Generate an interactive report

```
generateReport(my_aligned_contigs)
```

This will save a detailed interactive HTML report that you can explore.

7.2.7 A Reproducible Example

If you are still confused about how to run sangeranalyseR and want to check whether it produces the results that you want, then check this section for more details. Here we demonstrate a simple and reproducible example for using sangeranalyseR to generate a consensus read from 8 sanger ab1 files (4 contigs and each includes a forward and a reverse read).

1. Prepare your input files & loading

The data of this example is in the sangeranalyseR package; thus, you can simply get its path from the library.

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
parentDir <- file.path(rawDataDir, 'Allolobophora_chlorotica', 'ACHLO')
```

2. Load and analyse your data

Run the following on-liner to create the sanger alignment object.

```
ACHLO_contigs <- SangerAlignment(ABIF_Directory     = parentDir,
                                 REGEX_SuffixForward = "_[0-9]*_F.ab1$",
                                 REGEX_SuffixReverse = "_[0-9]*_R.ab1$")
```

Following is the R shell output that you will get.

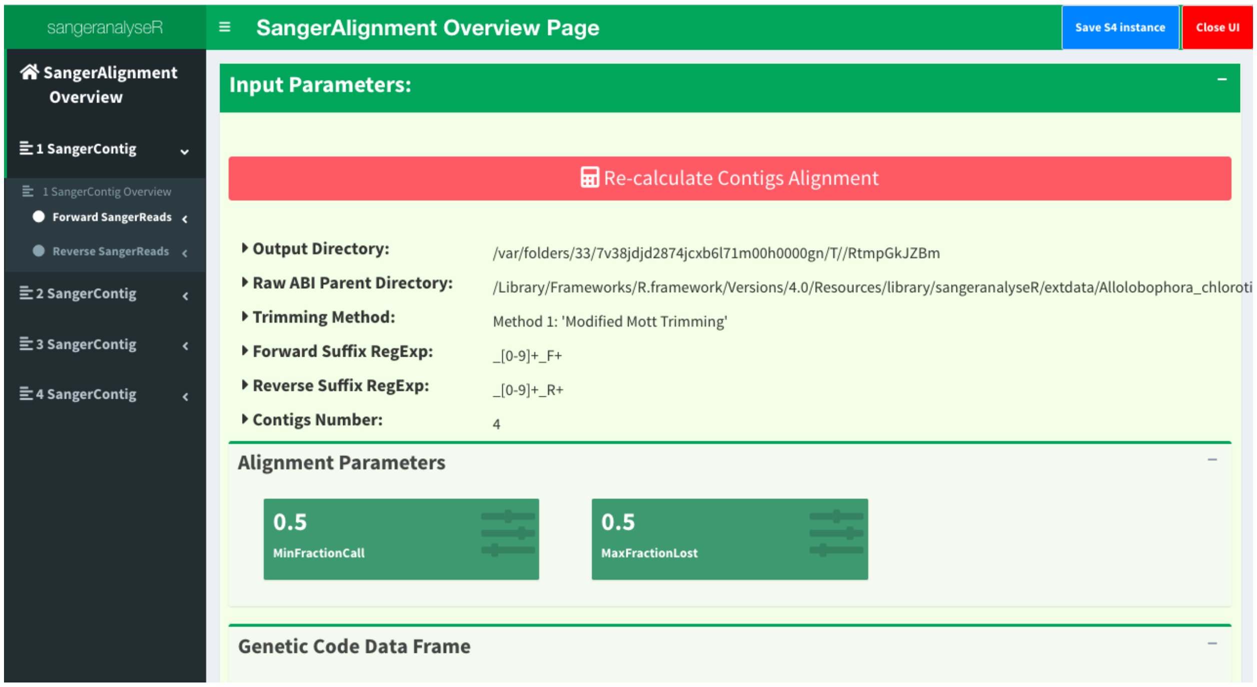
3. Explore your data

Launch the Shiny app to check the visualized results.

```
launchApp(ACHLO_contigs)
```

Following is the R shell output that you will get.

And a Shiny would popup as showed in Figure 1



Input Parameters:

- Output Directory: /var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpGkJZBm
- Raw ABI Parent Directory: /Library/Frameworks/R.framework/Versions/4.0/Resources/library/sangeranalyseR/extdata/Allolobophora_chloroti
- Trimming Method: Method 1: 'Modified Mott Trimming'
- Forward Suffix RegExp: _[0-9]+_F+
- Reverse Suffix RegExp: _[0-9]+_R+
- Contigs Number: 4

Alignment Parameters

0.5 MinFractionCall

0.5 MaxFractionLost

Fig. 2: Figure 1. *SangerAlignment* Shiny dashboard.

4. Output your aligned contigs

Write each contig and the aligned consensus read into FASTA files.

```
writeFasta(ACHLO_contigs)
```

Following is the R shell output that you will get.

And you will get three FASTA files:

(1) `Sanger_all_trimmed_reads.fa`

(2) `Sanger_contigs_alignment.fa`

(3) `Sanger_contigs_unalignment.fa`

5. Generate an interactive report

Last but not least, generate an Rmarkdown report to store all the sequence information.

```
generateReport(ACHLO_contigs)
```

For more detailed analysis steps, please choose one the following topics :

- *Beginners Guide*
- *Advanced User Guide - SangerRead (AB1)*
- *Advanced User Guide - SangerContig (AB1)*
- *Advanced User Guide - SangerAlignment (AB1)*
- *Advanced User Guide - SangerRead (FASTA)*
- *Advanced User Guide - SangerContig (FASTA)*
- *Advanced User Guide - SangerAlignment (FASTA)*

7.3 Beginners Guide

If you haven't already, please follow the steps in the *Installation* page to install and load sangeranalyseR.

This guide is for users who are starting with **AB1** (`.ab1`) files. If you are starting with **FASTA** (`.fasta` or `.fa`) files, please read through this guide then follow the slightly different path for those starting with FASTA data here: *Advanced User Guide - SangerAlignment (FASTA)*.

7.3.1 Step 1: Preparing your input files

sangeranalyseR takes as input a group of **AB1** files, which it then groups together into contigs. Once the individual contigs are built, all the contigs are aligned and a simple phylogenetic tree is made. This section explains how you should organize your files before running sangeranalyseR.

First, prepare a directory and put all your **AB1** files inside it (there can be other files in there too, sangeranalyseR will ignore anything without a *AB1* file extension). Files can be organised in as many sub-folders as you like. sangeranalyseR will recursively search all the directories inside `ABIF_Directory` and find all files that end with **AB1**.

Second, give sangeranalyseR the information it needs to group reads into contigs. To do this, sangeranalyseR needs two pieces of information about each read: the direction of the read (forward or reverse), and the contig that it should be grouped into. There are two ways you can give sangeranalyseR this information:

- using the file name itself
- using a three-column csv file

We'll cover both approaches using the following example. Imagine you have sequenced four contigs with a forward and reverse read, all from the same species, but from different locations. In this case you might have arranged your data something like *Figure_1*, below.

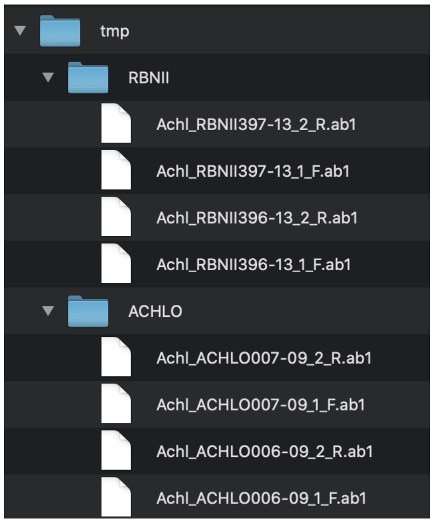


Fig. 3: Figure 1. Input ab1 files inside the parent directory, `./tmp/`.

When using the filenames to group the reads, you'll need to specify three parameters: `ABIF_Directory`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`:

- `ABIF_Directory`: this is the directory that contains all the **AB1** files. In this example, the reads are in the `/tmp/` directory, so for convenience we'll just say that `ABIF_Directory` should be `/path/to/tmp/`. In your case, it should be the absolute path to the folder that contains your reads.
- `REGEX_SuffixForward`: This is a regular expression (if you don't know what this is, don't panic - it's just a way of recognising text that you will get the hang of fast), which tells sangeranalyseR how to use the end of a filename to determine a forward read. All the reads that are in forward direction have to contain this in their filename suffix. There are lots of ways to do this, but for this example, one uesful way to do it is `_[0-9]*_F.ab1$`. This regular expression just says that the forward suffix is an underscore, followed at least one digit from 0-9, followed by another underscore then 'F', and ends with `.ab1`. The regex does not have to match to the end of the file name, but it's important to realise is that whatever comes before the part of the filename captured by this regex is by default the contig name. So in this case the regex also determines that the contig name for the first read is 'Achl_RBNII397-13'.
- `REGEX_SuffixReverse`: This is just the same as for the forward read, except that it determines the suffix for reverse reads. All the reads that are in reverse direction have to contain this in their filename suffix. In this example, its value is `_[0-9]*_R.ab1$`. I.e. all we've done is switch the 'F' in the forward read for an 'R' in the reverse read.

If you don't want to use the regex method, you can use the csv method instead. To use this method, just set `processMethod` parameter to `csv` and prepare an input **.csv** file with three columns:

- reads: the full file name (just the name, not the path) of the read to be grouped
- direction: "F" or "R" for forward and reverse reads, respectively
- contig: the name of the contig that reads should be grouped into

Following is an example of how you should organize your csv file in this example:

7.3.2 Step 2: Loading and analysing your data

After preparing the input files, you can create and align your contigs with just a single line of R code. In technical jargon, we are creating a *SangerAlignment* S4 instance.

It's important to note that this function is designed to be both *simple* and *flexible*. It's simple in that it has sensible defaults for all the usual things like trimming reads. But it's flexible in that you can change any and all of these defaults to suit your particular data and analyses. Here we just cover the simplest usage. The more flexible things are covered in the Advanced sections of the user guide.

So, let's create our contigs from our reads, and align them.

Here's how to do it using the regex method:

```
my_aligned_contigs <- SangerAlignment(ABIF_Directory      = "/path/to/tmp/",
                                      processMethod        = "REGEX",
                                      REGEX_SuffixForward = "_[0-9]*_F.ab1$",
                                      REGEX_SuffixReverse = "_[0-9]*_R.ab1$")
```

Here's how to do it using the csv file method

```
my_aligned_contigs <- SangerAlignment(ABIF_Directory      = "/path/to/tmp/",
                                      processMethod        = "CSV",
                                      CSV_NamesConversion = "/path/to/csvfile")
```

`my_aligned_contigs` is now a *SangerAlignment* S4 object which contains all of your reads, all the information on how they were trimmed, processed, and aligned, their chromatograms, and an alignment and phylogeny of all of your assembled contigs. The next section explains how to start digging into the details of that object.

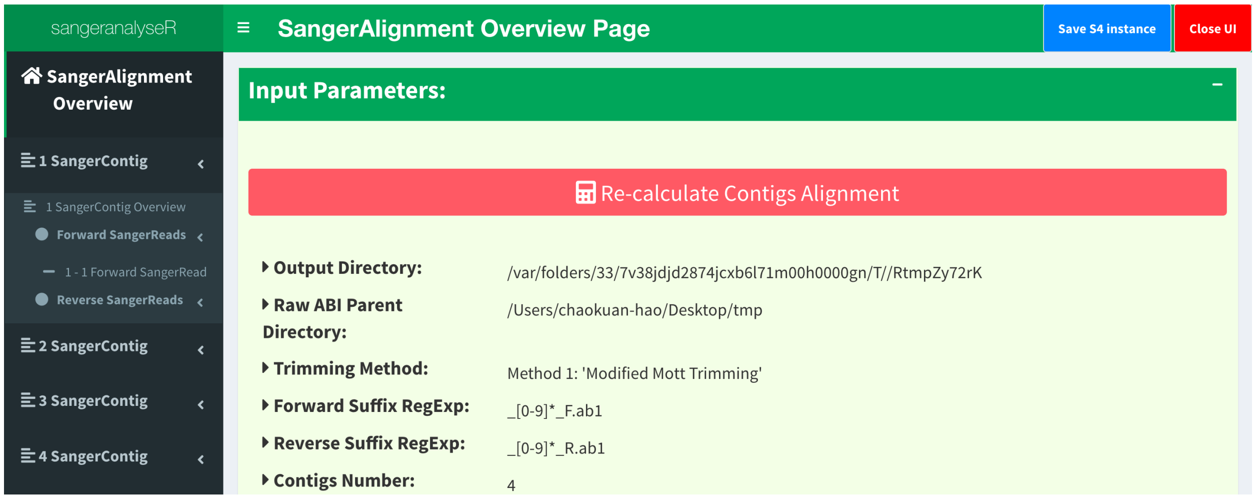
7.3.3 Step 3: Exploring your data with the Shiny app

sangeranalseR includes a Shiny app that allows you to see, interact with, and adjust the parameters of your aligned contigs. For example, you can adjust things like the trimming parameters, and see how that changes your reads and your contigs.

To launch the interactive Shiny app use the `launchApp` function as follows

```
launchApp(my_aligned_contigs)
```

Figure_2 shows what the Shiny app looks like. On the left-hand side of *Figure_2*, there is a navigation menu that you can click to get more detail on every contig and every read. You can explore this app to get a lot more detail and make adjustments to your data. (Note that sangeranalyseR doesn't allow for editing individual bases of reads though - that's just not something that R is good for).



Input Parameters:

Output Directory:	/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpZy72rK
Raw ABI Parent Directory:	/Users/chaokuan-hao/Desktop/tmp
Trimming Method:	Method 1: 'Modified Mott Trimming'
Forward Suffix RegExp:	_[0-9]*_F.ab1
Reverse Suffix RegExp:	_[0-9]*_R.ab1
Contigs Number:	4

Fig. 4: Figure 2. *SangerAlignment* Shiny app user interface.

7.3.4 Step 4: Outputting your aligned contigs

Once you're happy with your aligned contigs, you'll want to save them somewhere.

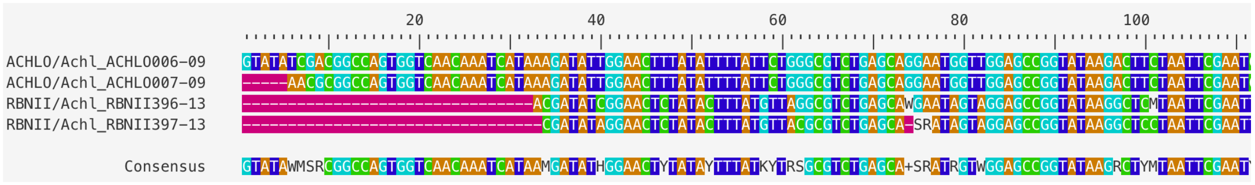
The following function can write the *SangerAlignment* object into FASTA files. You just need to tell it where with the `outputDir` argument. Here we just wrote the alignment to the same folder that contains our reads.

```
writeFasta(my_aligned_contigs, outputDir = "/path/to/tmp/")
```

7.3.5 Step 5: Generating an interactive report

Last but not least, it is useful to store all the results in a report for future reference. You can generate a detailed report by running the following one-line function. *Figure_3* and *Figure_4*.

```
generateReport(my_aligned_contigs)
```



```
                                       20                  40                  60                  80                 100
ACHLO/Achl_ACHLO006-09   GTATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAAT
ACHLO/Achl_ACHLO007-09   -----AACGCGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAAT
RBNII/Achl_RBNII396-13   --------------------------------ACGATATCGGAACTCTATACTTTATGTTAGGCGTCTGAGCAWGAATAGTAGGAGCCGGTATAAGGCTCMTAATTCGAAT
RBNII/Achl_RBNII397-13   ---------------------------------CGATATAGGAACTCTATACTTTATGTTACGCGTCTGAGCA-SRATAGTAGGAGCCGGTATAAGGCTCCTAATTCGAAT

             Consensus   GTATAWMSRCGGCCAGTGGTCAACAAATCATAAMGATATHGGAACTYTATAYTTTATKYTRSGCGTCTGAGCA+SRATRGTWGGAGCCGGTATAAGRCTYMTAATTCGAAT
```

Fig. 5: Figure 3. An alignment of all contigs in the *SangerAlignment* object.

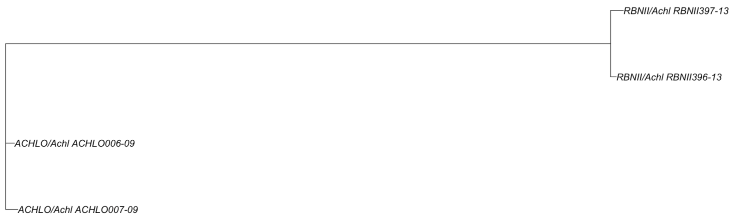


Fig. 6: Figure 4. A phylogenetic tree with contigs as the leaf nodes. This can help diagnose any issues with your contigs.

7.3.6 What's next ?

Now you've finished the *Beginners Guide*, you should have a good overview of how to use the package. To dig a lot deeper into what you can do and why you might bother, there are also a set of advanced guides that focus on the three levels at which you can analyse Sanger data in the sangeranalyseR package. You can analyse individual reads with the *SangerRead* object, individual contigs with the *SangerContig* object, and alignments of two or more contigs (as we focussed on in this intro) with teh *SangerAlignment* object.

If you want to start the analysis from **AB1** files, please choose the analysis level and read the following three links.

- *Advanced User Guide - SangerRead (AB1)*
- *Advanced User Guide - SangerContig (AB1)*
- *Advanced User Guide - SangerAlignment (AB1)*

If you want to start the analysis from **FASTA** files, please choose the analysis level and read the following three links.

- *Advanced User Guide - SangerRead (FASTA)*
- *Advanced User Guide - SangerContig (FASTA)*
- *Advanced User Guide - SangerAlignment (FASTA)*

7.4 Advanced User Guide - *SangerRead* (AB1)

SangerRead is in the bottommost level of sangeranalyseR (*Figure_1*), and each *SangerRead* object corresponds to a single read (one **AB1** file) in a Sanger sequencing experiment. *SangerRead* class extends *sangerseq* class from sangerseqR package and contains input parameters and results of quality trimming and chromatogram. In this section, we are going to go through detailed sangeranalyseR data analysis steps in *SangerRead level* with **AB1** file input.

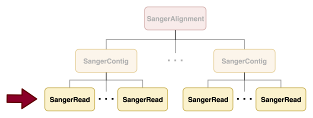


Fig. 7: Figure 1. Hierarchy of classes in sangeranalyseR, *SangerRead* level.

7.4.1 Preparing *SangerRead* AB1 input

The main input file format to create *SangerRead* instance is **AB1**. Before starting the analysis, users need to prepare one target **AB1** file, and in this example, it is in the sangeranalyseR package; thus, you can simply get its path by running the following codes:

```
inputFilesPath <- system.file("extdata/", package = "sangeranalyseR")
A_chloroticaFFN <- file.path(inputFilesPath,
                             "Allolobophora_chlorotica",
                             "ACHLO",
                             "Achl_ACHLO006-09_1_F.ab1")
```

The only hard regulation of the filename, `Achl_ACHLO006-09_1_F.ab1` in this example, is that the input file must have **.ab1** as its file extension. There are some suggestions about the filename in the note below:

Note:

- **AB1** file should be indexed for better consistency with file-naming regulation for *SangerContig* and *SangerAlignment*.
- Forward or reverse direction should be specified in the filename.

Figure_2 shows the suggested file-naming strategy. The filename should contain four main parts: **"Contig name"**, **"Index number"**, **"Direction"** and **"ab1 file extension"**.

- **"Contig name"** : `Achl_RBNII397-13`
- **"Index number"** : `1`
- **"Direction"** : `F`
- **"ab1 file extension"** : `.ab1`

Achl_RBNII397-13_1_F.ab1

Fig. 8: Figure 2. *SangerRead* filename regulation.

In *SangerRead* section, it is not compulsory to follow the file-naming regulation because users can directly specify the filename in input (see *Creating SangerRead instance from AB1*); however, in the *SangerContig* and *SangerAlignment*, sangeranalyseR will automatically group files, so it is compulsory to have systematic file-naming strategy. For more details, please read *Advanced User Guide - SangerContig (AB1)* and *Advanced User Guide - SangerAlignment (AB1)*. *Figure_3* shows the suggested **AB1** file-naming regulation.

```
[Consensus Read Name] + _ + [index] + _ + [F,R] + .ab1
```

Fig. 9: Figure 3. Suggested **AB1** file-naming regulation - *SangerRead*.

7.4.2 Creating *SangerRead* instance from AB1

After preparing the *SangerRead* input **AB1** file, `A_chloroticaFFN` , the next step is to create a *SangerRead* instance by running `SangerRead` constructor function or `new` method. The constructor function is a wrapper for the `new` method which makes instance creation more intuitive. The inputs include **Basic Parameters**, **Trimming Parameters**, and **Chromatogram Parameters**, and all of them have default values. In the example below, we show both *SangerRead* creation methods with important parameters.

```
# using `constructor` function to create SangerRead instance
sangerReadF <- SangerRead(readFeature           = "Forward Read",
                          readFileName          = A_chloroticaFFN,
                          geneticCode           = GENETIC_CODE,
                          TrimmingMethod        = "M1",
                          M1TrimmingCutoff      = 0.0001,
                          M2CutoffQualityScore  = NULL,
                          M2SlidingWindowSize   = NULL,
                          baseNumPerRow         = 100,
                          heightPerRow          = 200,
                          signalRatioCutoff     = 0.33,
                          showTrimmed           = TRUE)

# using `new` method to create SangerRead instance
sangerReadF <- new("SangerRead",
                   readFeature           = "Forward Read",
                   readFileName          = A_chloroticaFFN,
                   geneticCode           = GENETIC_CODE,
                   TrimmingMethod        = "M1",
                   M1TrimmingCutoff      = 0.0001,
                   M2CutoffQualityScore  = NULL,
                   M2SlidingWindowSize   = NULL,
                   baseNumPerRow         = 100,
                   heightPerRow          = 200,
                   signalRatioCutoff     = 0.33,
                   showTrimmed           = TRUE)
```

The inputs of `SangerRead` constructor function and `new` method are the same. For more details about *SangerRead* inputs and slots definition, please refer to the sangeranalyseR reference manual. The created *SangerRead* instance, `sangerReadF`, is used as the input for the following functions.

Inside the R shell, you can run `sangerReadF` to get basic information of the instance or run `sangerReadF@objectResults@readResultTable` to check the creation result of every Sanger read after `sangerReadF` is successfully created.

Here is the output of `sangerReadF`:

```
SangerRead S4 instance
         Input Source :  ABIF
         Read Feature :  Forward Read
         Read FileName :  Achl_ACHLO006-09_1_F.ab1
      Trimming Method :  M1
      Primary Sequence : ␣
↪CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCTGGGCAGAGACCAACTATAC
   Secondary Sequence : ␣
↪CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCTGGGCAGAGACCAACTATAC
SUCCESS [2021-12-07 23:31:16] 'Achl_ACHLO006-09_1_F.ab1' is successfully created!
```

Here is the output of `sangerReadF@objectResults@readResultTable`:

```
                  readName creationResult errorType errorMessage inputSource   ␣
↪direction
1 Achl_ACHLO006-09_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
```

7.4.3 Visualizing *SangerRead* trimmed read

Before going to *Writing SangerRead FASTA file (AB1)* and *Generating SangerRead report (AB1)* pages, it is suggested to visualize the trimmed *SangerRead*. Run the `qualityBasePlot` function to get the result in *Figure_4*. It shows the quality score for each base pairs and the trimming start/end points of the sequence.

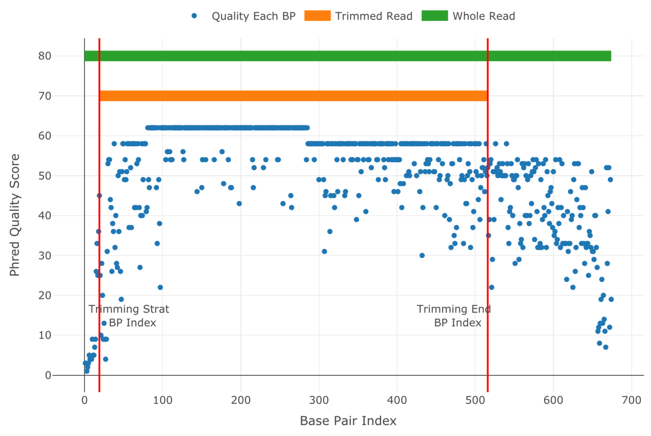


Fig. 10: Figure 4. *SangerRead* trimmed read visualization.

```
qualityBasePlot(sangerReadF)
```

7.4.4 Updating *SangerRead* quality trimming parameters

In the previous *Creating SangerRead instance from AB1* part, the constructor function applies the quality trimming parameters to the read. These parameters are not fixed. After instance creation, users can run `updateQualityParam` function which will change the *QualityReport* instance inside the *SangerRead* and update frameshift amino acid sequences.

```
newSangerRead <- updateQualityParam(sangerReadF,
                                    TrimmingMethod        = "M2",
                                    M1TrimmingCutoff      = NULL,
                                    M2CutoffQualityScore = 29,
                                    M2SlidingWindowSize   = 15)
```

7.4.5 Writing *SangerRead* FASTA file $_{\text{(AB1)}}$

After quality trimming, users can write `sangerReadF` into a **FASTA** file. Below is the one-liner that needs to be run. This function, `writeFasta`, mainly depends on `writeXStringSet` function in Biostrings R package. Users can further set the compression level through it.

```
writeFasta(sangerReadF,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA)
```

Users can download the `output FASTA file` of this example.

7.4.6 Generating *SangerRead* report (AB1)

Last but not least, users can save `sangerReadF` into a static **HTML** report by knitting **Rmd** files. In this example, `tempdir` function will generate a random path.

```
generateReport(sangerReadF,
               outputDir = tempdir())
```

SangerRead_Report_ab1.html is the generated *SangerRead* report html of this example. Users can access to '*Basic Information*', '*DNA Sequence*', '*Amino Acids Sequence*', '*Quality Trimming*' and '*Chromatogram*' sections inside this report.

7.4.7 Code summary (*SangerRead*, ab1)

(1) Preparing *SangerRead* AB1 input

```
inputFilesPath <- system.file("extdata/", package = "sangeranalyseR")
A_chloroticaFFN <- file.path(inputFilesPath,
                             "Allolobophora_chlorotica",
                             "ACHLO",
                             "Achl_ACHLO006-09_1_F.ab1")
```

(2) Creating *SangerRead* instance from AB1

```
# using `constructor` function to create SangerRead instance
sangerReadF <- SangerRead(readFeature           = "Forward Read",
                          readFileName          = A_chloroticaFFN)

# using `new` method to create SangerRead instance
sangerReadF <- new("SangerRead",
                   readFeature           = "Forward Read",
                   readFileName          = A_chloroticaFFN)
```

Following is the R shell output that you will get.

(3) Visualizing *SangerRead* trimmed read

```
qualityBasePlot(sangerReadF)
```

(4) Writing *SangerRead* FASTA file (AB1)

```
writeFasta(sangerReadF)
```

Following is the R shell output that you will get.

And you will get one FASTA file:

(1) `Achl_ACHLO006-09_1_F.fa`

(5) Generating *SangerRead* report (AB1)

```
generateReport(sangerReadF)
```

You can check the html report of this SangerRead example (ABIF).

7.5 Advanced User Guide - *SangerContig* (AB1)

SangerContig is in the intermediate level of sangeranalyseR (*Figure_1*), and each *SangerContig* instance corresponds to a contig in a Sanger sequencing experiment. Among its slots, there are two lists, forward and reverse read list, storing *SangerRead* in the corresponding direction.

In this section, we are going to go through details about a reproducible *SangerContig* analysis example with the **AB1** file input in sangeranalyseR. By running the following example codes, you will get an end-to-end *SangerContig* analysis result.

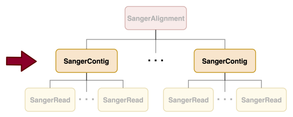


Fig. 11: Figure 1. Hierarchy of classes in sangeranalyseR, *SangerContig* level.

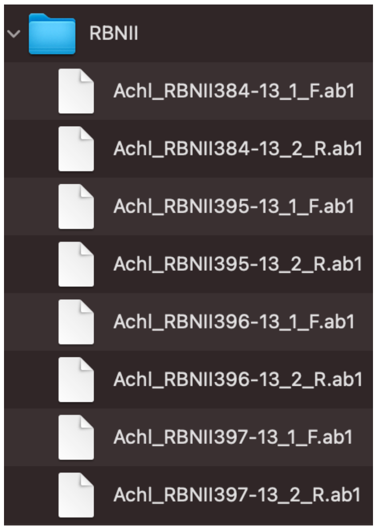
7.5.1 Preparing *SangerContig* AB1 inputs

The main input file format to create *SangerContig* instance is **AB1**. Before starting the analysis, users need to prepare one directory containing all **AB1** files, and all of them must be in the first layer of that directory. In other words, there should be no subdirectories. In this example, the data are in the sangeranalyseR package; thus, you can simply get its path by running the following codes:

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
parentDir <- file.path(rawDataDir, "Allolobophora_chlorotica", "RBNII")
```

The value of `parentDir` is where all **AB1** files are placed. If your operating system is macOS, then its value should look like this:

And we showed the files under `parentDir` in *Figure_2*:



RBNII
Achl_RBNII384-13_1_F.ab1
Achl_RBNII384-13_2_R.ab1
Achl_RBNII395-13_1_F.ab1
Achl_RBNII395-13_2_R.ab1
Achl_RBNII396-13_1_F.ab1
Achl_RBNII396-13_2_R.ab1
Achl_RBNII397-13_1_F.ab1
Achl_RBNII397-13_2_R.ab1

Fig. 12: Figure 2. *SangerContig* filename regulation.

Figure_2 shows the file-naming regulation and hierarchy. In this example, `RBNII` is the parent directory, and all **AB1** files must be under its first layer. There are two ways for users to group their **AB1** files which are **"regular expression matching"** and **"CSV file matching"**, and following are instructions of how to prepare and name your **AB1** input files.

(1) "regular expression matching" *SangerContig* inputs (AB1)

For regular expression matching method, sangeranalyseR will group **AB1** files based on their contig names and read directions in their filenames automatically; therefore, users have to follow the file-naming regulations below:

Note:

- All input files must have **.ab1** as its file extension
- All input files must have the same contig name in their filenames.
- Forward or reverse direction has to be specified in the filename.

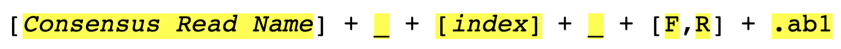
There are four parameters, `ABIF_Directory`, `contigName`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that define the grouping rule to let sangeranalyseR automatically match correct **AB1** files and divide them into forward and reverse directions.

Note:

- `ABIF_Directory`: this is the directory that contains all **AB1** files, and it can be either an absolute or relative path. We suggest users to put only target **AB1** files inside this directory and do not include any other unrelated files.
- `contigName`: this is a regular expression that matches filenames that are going to be included in the *SangerContig* analysis. `grepl` function in R is used.
- `REGEX_SuffixForward`: this is a regular expression that matches all filenames in forward direction. `grepl` function in R is used.
- `REGEX_SuffixReverse`: this is a regular expression that matches all filenames in reverse direction. `grepl` function in R is used.

If you don't know what regular expression is, don't panic - it's just a way of recognising text. Please refer to *What is a regular expression?* for more details. Here is an example of how it works in sangeranalseR:

So how sangeranalyseR works is that it first matches the `contigName` to exclude unrelated files and then separate the forward and reverse reads by matching `REGEX_SuffixForward` and `REGEX_SuffixReverse`. Therefore, it is important to make sure that all target **AB1** files share the same `contigName` and carefully select your `REGEX_SuffixForward` and `REGEX_SuffixReverse`. The bad file-naming and wrong regex matching might accidentally include reverse reads into the forward read list or vice versa, which will make the program generate wrong results. Therefore, it is important to have a consistent naming strategy. So, how should we systematically name **AB1** files? We suggest users to follow the file-naming regulation in *Figure_3*.



```
[Consensus Read Name] + _ + [index] + _ + [F,R] + .ab1
```

Fig. 13: Figure 3. Suggested **AB1** file-naming regulation - *SangerContig*.

As you can see, the first part of the regulation is a consensus read name (or contig name), which narrows down the scope of **AB1** files to those we are going to examine. The second part of the regulation is an index. Since there might be more than one read that is in the forward or reverse direction, we recommend you to number your reads in the same contig group. The third part is a direction which is either 'F' (forward) or 'R' (reverse). Last but not least, files have to end with **.ab1** file extension.

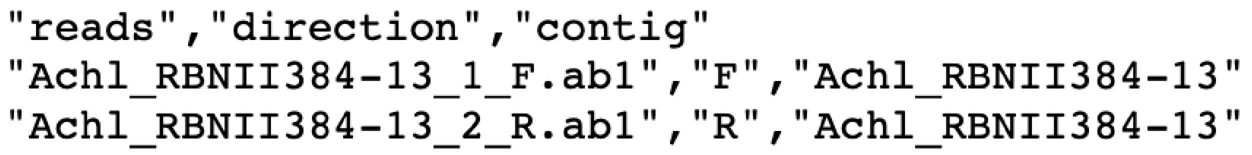
To make it more specific, let's go back to the true example. In *Figure_2*, there are a lot of **AB1** files from different contigs in `RBNII` (`ABIF_Directory`). First, we set `contigName` to `"Achl_RBNII384-13"` to reduce candidates from eight to two **AB1** files, `Achl_RBNII384-13_1_F.ab1` and `Achl_RBNII384-13_2_R.ab1`. Then, we set `REGEX_SuffixForward` to `"_[0-9]*_F.ab1$"` and `REGEX_SuffixReverse` to `"_[0-9]*_R.ab1$"` to let sangeranalyseR match and group forward and reverse reads automatically. By the regular expression rule, `Achl_RBNII384-13_1_F.ab1` and `Achl_RBNII384-13_2_R.ab1` will be categorized into "forward read list" and "reverse read list" respectively. The reason why we strongly recommend you to follow this file-naming regulation is that by doing so, you can directly adopt the example regular expression matching values, `"_[0-9]*_F.ab1$"` and `"_[0-9]*_R.ab1$"`, to group reads and reduce chances of error.

After understanding how parameters work, please refer to *Creating SangerContig instance from AB1* below to see how sangeranalseR creates 'Achl_RBNII384-13' *SangerContig* instance.

(2) "CSV file matching" *SangerContig* inputs (AB1)

For those who are not familiar with regular expression, we provide a second grouping approach, CSV file matching method. sangeranalyseR will group **AB1** files based on the information in a CSV file automatically; therefore, users have to follow the regulations below:

Note: Here is an `example CSV file` (*Figure_4*)



```
"reads","direction","contig"
"Achl_RBNII384-13_1_F.ab1","F","Achl_RBNII384-13"
"Achl_RBNII384-13_2_R.ab1","R","Achl_RBNII384-13"
```

Fig. 14: Figure 4. Example CSV file for *SangerContig* instance creation.

- There must be three columns, "**reads**", "**direction**", and "**contig**", in the CSV file.
- The "**reads**" column stores the filename of **AB1** files that are going to be included in the analysis.
- The "**direction**" column stores the direction of the reads. It must be "F" (forward) or "R" (reverse).
- The "**contig**" column stores the contig name that each read blongs. Reads in the same contig have to have the same contig name, and they will be grouped into the same *SangerContig* instance.

There are three parameters, `ABIF_Directory`, `contigName`, and `CSV_NamesConversion`,that define the grouping rule to help sangeranalseR to automatically match correct **AB1** files and divide them into forward and reverse directions.

Note:

- `ABIF_Directory`: this is the directory that contains all **AB1** files, and it can be either an absolute or relative path. We suggest users to put only target AB1 files inside this directory and do not include any other unrelated files.
- `contigName`: this is a regular expression that matches filenames that are going to be included in the *SangerContig* analysis. `grepl` function in R is used.
- `CSV_NamesConversion`: this is the path to the CSV file. It can be either an absolute or relative path.

The main difference between "CSV file matching" and "regular expression matching" is where the grouping rule is written. For "regular expression matching", rules are writtein in filenames, and thus more naming requirements are required. In contrast, rules of "CSV file matching" are written in an additional CSV file so it is more flexible on **AB1** file-naming.

So how sangeranalyseR works is that it first reads in the CSV file (with *"reads"*, *"direction"*, and *"contig"* columns), filter out rows whose *"contig"* is not the value of `contigName` parameter, find the names of **AB1** files listed in *"reads"*, and assign directions to them based on *"direction"*.

To make it more specific, let's go back to the true example. First, we prepare a `CSV file` (`CSV_NamesConversion`) and a file directory like *Figure_2* (`ABIF_Directory`) with some **AB1** files from different contigs. In the CSV file, both rows have the contig name `"Achl_RBNII384-13"`, which is what we need to assign to the `contigName` parameter. sangeranalyseR then checks and matches *"reads"* of these two rows, `"Achl_RBNII384-13_1_F.ab1"` and `"Achl_RBNII384-13_2_R.ab1"`, in `RBNII` directory and reduce candidates from eight to two **AB1** files. Last, these two reads are assigned into "forward read list" and "reverse read list" respectively by the *"direction"* column.

After understanding how parameters work, please refer to *Creating SangerContig instance from AB1* below to see how sangeranalseR creates 'Achl_RBNII384-13' *SangerContig* instance.

7.5.2 Creating *SangerContig* instance from AB1

After preparing the input directory, we can create a *SangerContig* instance by running `SangerContig` constructor function or `new` method. The constructor function is a wrapper for `new` method and it makes instance creation more intuitive. Their input parameters are same, and all of them have their default values. For more details about *SangerContig* inputs and slots definition, please refer to sangeranalyseR reference manual. We will explain two *SangerContig* instance creation methods, "regular expression matching" and "CSV file matching".

(1) "regular expression matching" *SangerContig* creation (AB1)

The consturctor function and `new` method below contain four parameters, `ABIF_Directory`, `contigName`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that we mentioned in the previous section. It also includes important parameters like quality trimming, chromatogram visualization, consensus alignment, and so on. Run the following code and create `my_sangerContig` instance.

```
# using `constructor` function to create SangerContig instance
my_sangerContig <- SangerContig(inputSource          = "ABIF",
                                processMethod        = "REGEX",
                                ABIF_Directory       = parentDir,
                                contigName           = "Achl_RBNII384-13",
                                REGEX_SuffixForward  = "_[0-9]*_F.ab1$",
                                REGEX_SuffixReverse  = "_[0-9]*_R.ab1$",
                                TrimmingMethod       = "M1",
                                M1TrimmingCutoff     = 0.0001,
                                M2CutoffQualityScore = NULL,
                                M2SlidingWindowSize  = NULL,
                                baseNumPerRow        = 100,
                                heightPerRow         = 200,
                                signalRatioCutoff    = 0.33,
                                showTrimmed          = TRUE,
                                refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                minReadsNum          = 2,
                                minReadLength        = 20,
                                minFractionCall      = 0.5,
                                maxFractionLost      = 0.5,
                                geneticCode          = GENETIC_CODE,
                                acceptStopCodons     = TRUE,
                                readingFrame         = 1,
```

(continues on next page)

(continued from previous page)

```
                                  processorsNum         = 1)

# using `new` method to create SangerContig instance
my_sangerContig <- new("SangerContig",
                        inputSource            = "ABIF",
                        processMethod          = "REGEX",
                        ABIF_Directory         = parentDir,
                        contigName             = "Achl_RBNII384-13",
                        REGEX_SuffixForward    = "_[0-9]*_F.ab1$",
                        REGEX_SuffixReverse    = "_[0-9]*_R.ab1$",
                        TrimmingMethod         = "M1",
                        M1TrimmingCutoff       = 0.0001,
                        M2CutoffQualityScore   = NULL,
                        M2SlidingWindowSize    = NULL,
                        baseNumPerRow          = 100,
                        heightPerRow           = 200,
                        signalRatioCutoff      = 0.33,
                        showTrimmed            = TRUE,
                        refAminoAcidSeq        =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                        minReadsNum            = 2,
                        minReadLength          = 20,
                        minFractionCall        = 0.5,
                        maxFractionLost        = 0.5,
                        geneticCode            = GENETIC_CODE,
                        acceptStopCodons       = TRUE,
                        readingFrame           = 1,
                        processorsNum          = 1)
```

In this example, `contigName` is set to `Achl_RBNII384-13`, so only `Achl_RBNII384-13_1_F.ab1` and `Achl_RBNII384-13_2_R.ab1` are selected. Moreover, by regular expression pattern matching, `Achl_RBNII384-13_1_F.ab1` is categorized into the forward list, and `Achl_RBNII384-13_2_R.ab1` is categorized into the reverse read. Both reads are aligned into a contig, `my_sangerContig`, and it will be used as the input for the following functions.

Inside the R shell, you can run `my_sangerContig` to get basic information of the instance or run `my_sangerContig@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerContig` is successfully created.

Here is the output of `my_sangerContig`:

```
SangerContig S4 instance
         Input Source :  ABIF
         Process Method :  REGEX
         ABIF Directory :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/Allolobophora_chlorotica/RBNII
   REGEX Suffix Forward :  _[0-9]*_F.ab1$
   REGEX Suffix Reverse :  _[0-9]*_R.ab1$
            Contig Name :  Achl_RBNII384-13
         'minReadsNum' :  2
      'minReadLength' :  20
      'minFractionCall' :  0.5
      'maxFractionLost' :  0.5
   'acceptStopCodons' :  TRUE
         'readingFrame' :  1
      Contig Sequence : ␣
↪AGCAGGATAGTAGGGGCTGGTATAAGACTCCTAATTCGAATTGAGCTAAGACAGCCGGGAGCATTTCTAGG[illegible]ATAACACTATTGTA
```

(continues on next page)

(continued from previous page)

```
Forward reads in the contig >>  1
Reverse reads in the contig >>  1
SUCCESS [2021-12-07 17:01:18] 'Achl_RBNII384-13' is successfully created!
```

Here is the output of `my_sangerContig@objectResults@readResultTable`:

```
                  readName creationResult errorType errorMessage inputSource   ␣
↪direction
1 Achl_RBNII384-13_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
2 Achl_RBNII384-13_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
```

(2) "CSV file matching" *SangerContig* creation (AB1)

The consturctor function and `new` method below contain three parameters, `ABIF_Directory`, `contigName`, and `CSV_NamesConversion`, that we mentioned in the previous section. It also includes important parameters like quality trimming, chromatogram visualization, consensus alignment, and so on. Run the following code and create `my_sangerContig` instance.

```
csv_namesConversion <- file.path(rawDataDir, "ab1", "SangerContig", "names_conversion_
↪2.csv")

# using `constructor` function to create SangerContig instance
my_sangerContig <- SangerContig(inputSource             = "ABIF",
                                processMethod           = "CSV",
                                ABIF_Directory          = parentDir,
                                contigName              = "Achl_RBNII384-13",
                                CSV_NamesConversion     = csv_namesConversion,
                                TrimmingMethod          = "M1",
                                M1TrimmingCutoff        = 0.0001,
                                M2CutoffQualityScore    = NULL,
                                M2SlidingWindowSize     = NULL,
                                baseNumPerRow           = 100,
                                heightPerRow            = 200,
                                signalRatioCutoff       = 0.33,
                                showTrimmed             = TRUE,
                                refAminoAcidSeq         =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                minReadsNum             = 2,
                                minReadLength           = 20,
                                minFractionCall         = 0.5,
                                maxFractionLost         = 0.5,
                                geneticCode             = GENETIC_CODE,
                                acceptStopCodons        = TRUE,
                                readingFrame            = 1,
                                processorsNum           = 1)


# using `new` method to create SangerContig instance
my_sangerContig <- new("SangerContig",
                       inputSource             = "ABIF",
                       processMethod           = "CSV",
                       ABIF_Directory          = parentDir,
```

(continues on next page)

(continued from previous page)

```
                           contigName              = "Achl_RBNII384-13",
                           CSV_NamesConversion     = csv_namesConversion,
                           TrimmingMethod           = "M1",
                           M1TrimmingCutoff         = 0.0001,
                           M2CutoffQualityScore     = NULL,
                           M2SlidingWindowSize      = NULL,
                           baseNumPerRow            = 100,
                           heightPerRow             = 200,
                           signalRatioCutoff        = 0.33,
                           showTrimmed              = TRUE,
                           refAminoAcidSeq =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                           minReadsNum              = 2,
                           minReadLength            = 20,
                           minFractionCall          = 0.5,
                           maxFractionLost          = 0.5,
                           geneticCode              = GENETIC_CODE,
                           acceptStopCodons         = TRUE,
                           readingFrame             = 1,
                           processorsNum            = 1)
```

First, you need to load the CSV file into the R environment. If you are still don't know how to prepare it, please check *(2) "CSV file matching" SangerContig inputs (AB1)*. Then, it will follow rules in the CSV file and create `my_sangerContig`. After it's created, inside the R shell, you can run `my_sangerContig` to get basic information of the instance or run `my_sangerContig@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerContig` is successfully created.

Here is the output of `my_sangerContig`:

```
SangerContig S4 instance
         Input Source :  ABIF
         Process Method :  CSV
         ABIF Directory :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/Allolobophora_chlorotica/RBNII
   CSV Names Conversion :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/ab1/SangerContig/names_conversion_2.csv
            Contig Name :  Achl_RBNII384-13
         'minReadsNum' :  2
      'minReadLength' :  20
      'minFractionCall' :  0.5
      'maxFractionLost' :  0.5
   'acceptStopCodons' :  TRUE
         'readingFrame' :  1
      Contig Sequence : ␣
↪AGCAGGATAGTAGGGGCTGGTATAAGACTCCTAATTCGAATTGAGCTAAGACAGCCGGGAGCATTTCTAGGAAGGGATCAACTCTATAACACTATTGTA
Forward reads in the contig >>  1
Reverse reads in the contig >>  1
SUCCESS [2021-12-07 17:11:48] 'Achl_RBNII384-13' is successfully created!
```

Here is the output of `my_sangerContig@objectResults@readResultTable`:

```
                  readName creationResult errorType errorMessage inputSource    ␣
↪direction
1 Achl_RBNII384-13_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
2 Achl_RBNII384-13_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
```

(continues on next page)

(continued from previous page)

7.5.3 Updating *SangerContig* quality trimming parameters

In the previous *Creating SangerContig instance from AB1* part, the constructor function will apply the quality trimming parameters to all reads. After creating a *SangerContig* instance, users can change the trimming parameters by running `updateQualityParam` function which will update all reads with the new trimming parameters and redo reads alignment. If users want to do quality trimming read by read instead of all at once, please move on to the next section, *Launching SangerContig Shiny app* page.

```
newSangerContig <- updateQualityParam(my_sangerContig,
                                      TrimmingMethod       = "M2",
                                      M1TrimmingCutoff     = NULL,
                                      M2CutoffQualityScore = 20,
                                      M2SlidingWindowSize  = 15)
```

7.5.4 Launching *SangerContig* Shiny app

We create an interactive local Shiny app for users to go into each *SangerRead* in *SangerContig* instance. Users only need to run one function, `launchApp`, with previously created instance as input and the *SangerContig* Shiny app will pop up. Here, we will go through *SangerRead* and *SangerContig* pages.

```
launchApp(my_sangerContig)
```

SangerContig page (SC app)

SangerContig page is the initial page of *SangerContig* Shiny app. *Figure 5* shows the overview page of the contig. Notice that there is a red "Re-calculate Contig" button. Users need to click the button after changing the quality trimming parameters in order to get the updated information. In SangerContig page, there are two expendable tabs, "Forward Reads" and "Reverse Reads" storing the corresponding reads on the left-hand side navigation panel in *Figure 5*. See *SangerRead page (SC app)* for more details of the subpage.

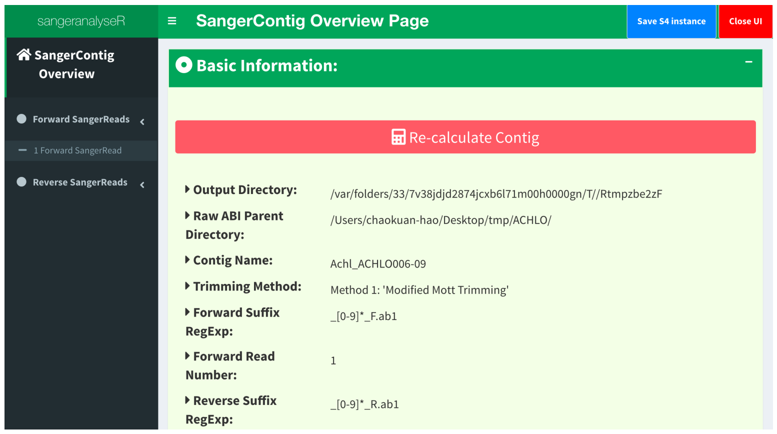
The information provided in this page are input parameters and contig results including "genetic code table", "reference amino acid sequence", "reads alignment", "difference data frame", "dendrogram", "sample distance heatmap", "indels data frame", and "stop codons data frame".

Figure 6 shows reads alignment result and difference data frame. The alignment is generated by `AlignSeqs` or `AlignTranslation` function in DECIPHER package.

Figure 7 shows dendrogram result in both plot and in data frame. The results are generated by `IdClusters` function in DECIPHER package.

Figure 8 shows distance between **AB1** files. The results are generated by `DistanceMatrix` function in DECIPHER package. The heatmap is generated by `plot_ly` function in plotly package.

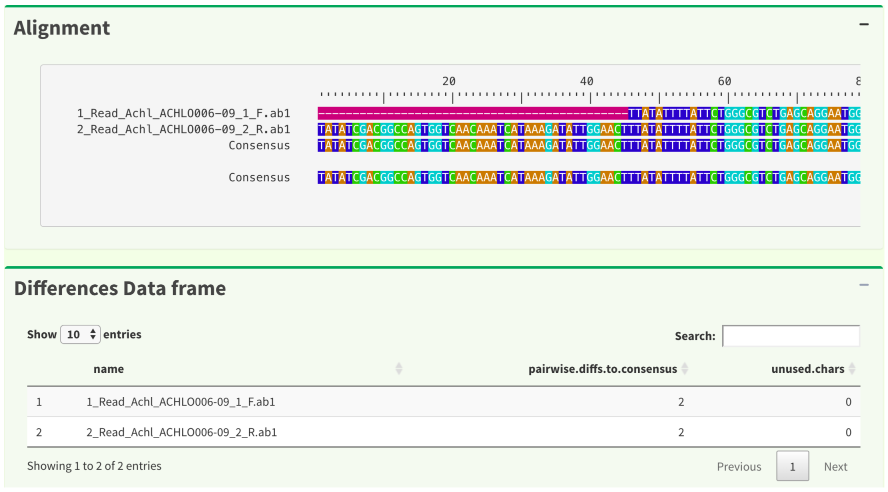
Figure 9 shows insertions, deletions and stop codons data frame.



Basic Information:

Output Directory:	/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//Rtmpzbe2zF
Raw ABI Parent Directory:	/Users/chaokuan-hao/Desktop/tmp/ACHLO/
Contig Name:	Achl_ACHLO006-09
Trimming Method:	Method 1: 'Modified Mott Trimming'
Forward Suffix RegExp:	_[0-9]*_F.ab1
Forward Read Number:	1
Reverse Suffix RegExp:	_[0-9]*_R.ab1

Fig. 15: Figure 5. *SangerContig* Shiny app initial page - *SangerContig* page.



Alignment

```
                                          20        40        60        8
1_Read_Achl_ACHLO006-09_1_F.ab1  ---------------------------------------------TTATATTTTATTCTGGGCGTCTGAGCAGGAATGG
2_Read_Achl_ACHLO006-09_2_R.ab1  TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGAATGG
                      Consensus  TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGAATGG

                      Consensus  TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGAATGG
```

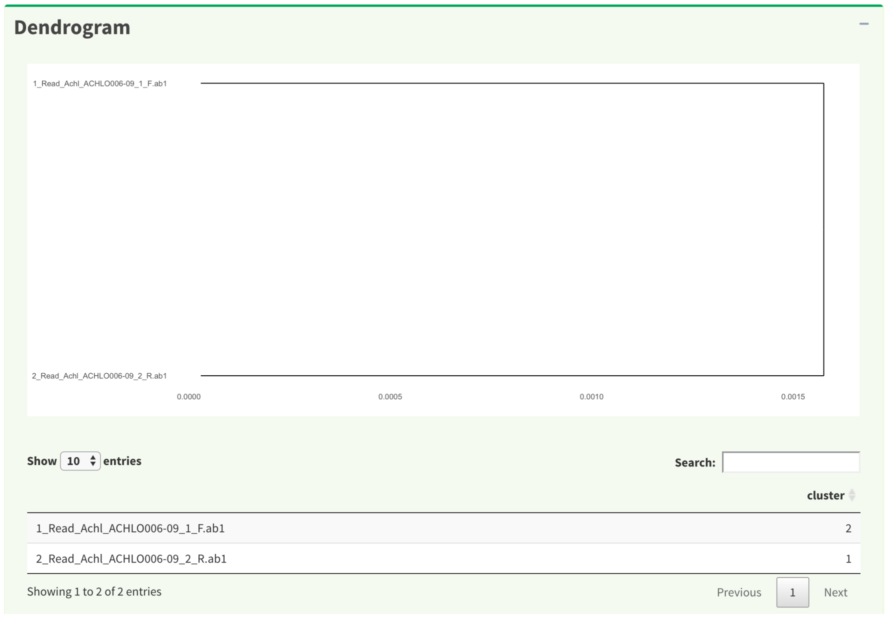
Differences Data frame

Show 10 entries Search:

	name	pairwise.diffs.to.consensus	unused.chars
1	1_Read_Achl_ACHLO006-09_1_F.ab1	2	0
2	2_Read_Achl_ACHLO006-09_2_R.ab1	2	0

Showing 1 to 2 of 2 entries

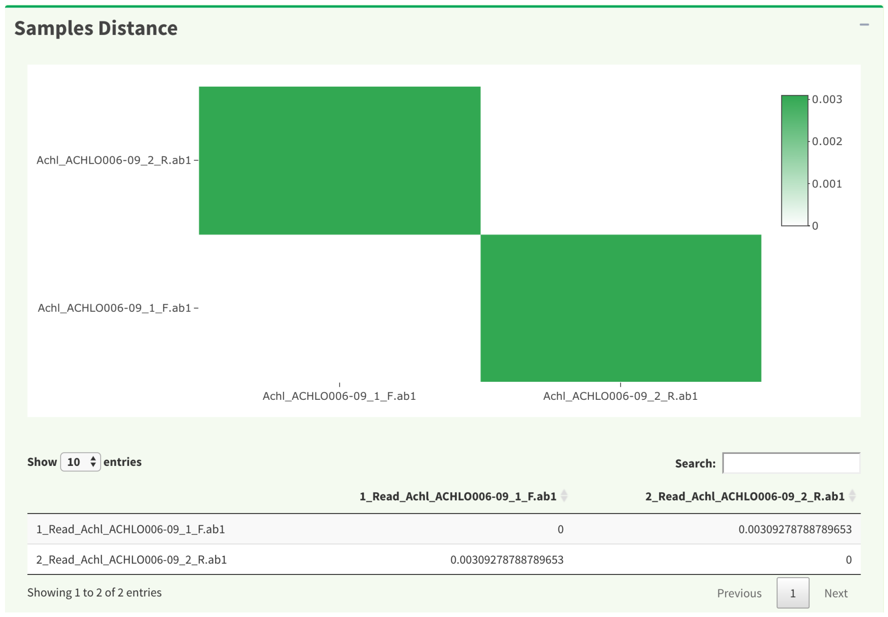
Fig. 16: Figure 6. *SangerContig* page - reads alignment and difference data frame.



	cluster
1_Read_Achl_ACHLO006-09_1_F.ab1	2
2_Read_Achl_ACHLO006-09_2_R.ab1	1

Showing 1 to 2 of 2 entries

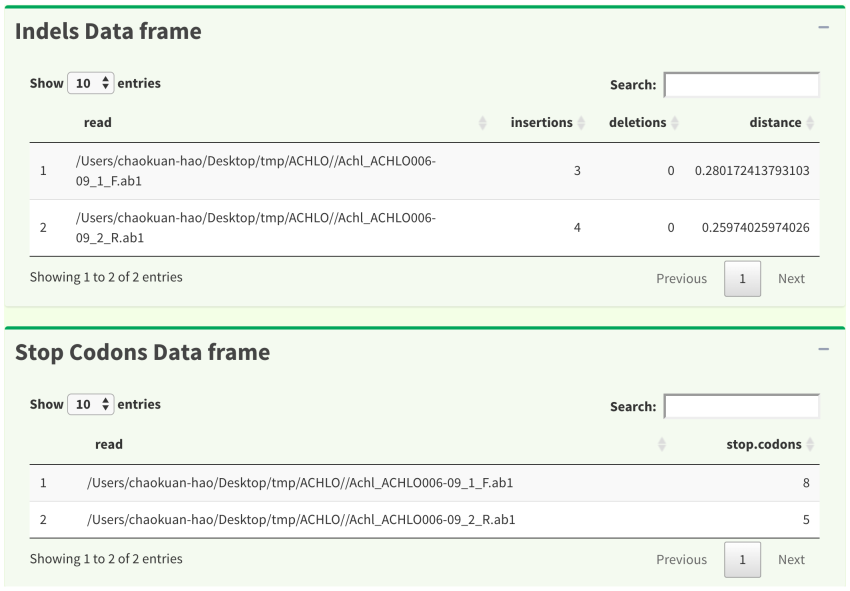
Fig. 17: Figure 7. *SangerContig* page - dendrogram.



	1_Read_Achl_ACHLO006-09_1_F.ab1	2_Read_Achl_ACHLO006-09_2_R.ab1
1_Read_Achl_ACHLO006-09_1_F.ab1	0	0.00309278788789653
2_Read_Achl_ACHLO006-09_2_R.ab1	0.00309278788789653	0

Showing 1 to 2 of 2 entries

Fig. 18: Figure 8. *SangerContig* page - samples distance.



Indels Data frame

Show 10 entries Search:

	read	insertions	deletions	distance
1	/Users/chaokuan-hao/Desktop/tmp/ACHLO//Achl_ACHLO006-09_1_F.ab1	3	0	0.280172413793103
2	/Users/chaokuan-hao/Desktop/tmp/ACHLO//Achl_ACHLO006-09_2_R.ab1	4	0	0.25974025974026

Showing 1 to 2 of 2 entries

Stop Codons Data frame

Show 10 entries Search:

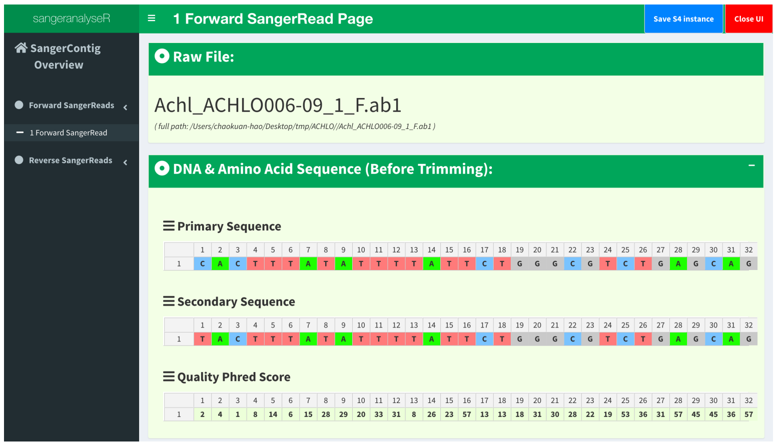
	read	stop.codons
1	/Users/chaokuan-hao/Desktop/tmp/ACHLO//Achl_ACHLO006-09_1_F.ab1	8
2	/Users/chaokuan-hao/Desktop/tmp/ACHLO//Achl_ACHLO006-09_2_R.ab1	5

Showing 1 to 2 of 2 entries

Fig. 19: Figure 9. *SangerContig* page - indels and stop codons data frame.

SangerRead page (SC app)

Now, let's go to the next level which is also the lowest level, *SangerRead* page. *SangerRead* page contains all details of a read including its trimming and chromatogram inputs and results. All reads are in "forward" or "reverse" direction. In this example, there is one read in each direction and *Figure 10* shows "1 Forward Read" page. This page provides basic information, quality trimming inputs, chromatogram plotting inputs etc. Primary/secondary sequences and quality Phred scores table in this figure are dynamic based on the `signalRatioCutoff` value for base calling and the length of them are always same. Another thing to mention is that primary/secondary sequences and the sequences in the chromatogram in *Figure 15* below will always be same after trimming and their color codings for A/T/C/G are same as well.



Raw File:

Achl_ACHLO006-09_1_F.ab1

(full path: /Users/chaokuan-hao/Desktop/tmp/ACHLO//Achl_ACHLO006-09_1_F.ab1)

DNA & Amino Acid Sequence (Before Trimming):

Primary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	C	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C	T	G	A	G	C	A	G

Secondary Sequence

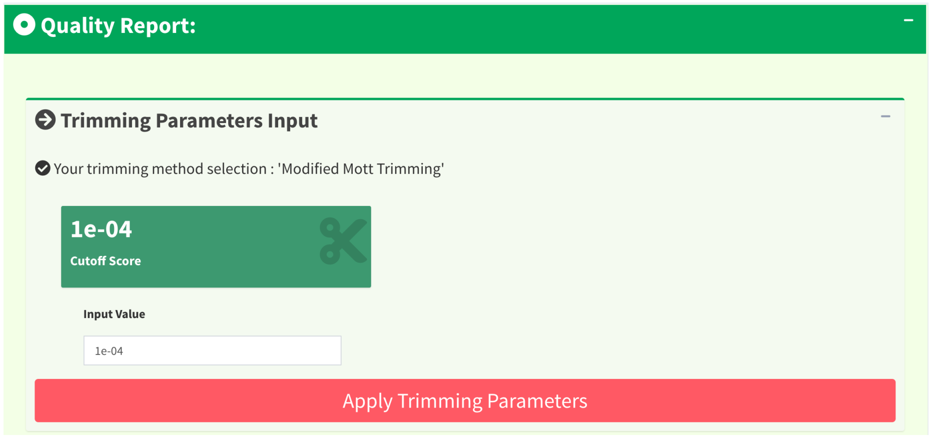
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	T	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C	T	G	A	G	C	A	G

Quality Phred Score

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	4	1	8	14	6	15	28	29	20	33	31	8	26	23	57	13	13	18	31	30	28	22	19	53	36	31	57	45	45	36	57

Fig. 20: Figure 10. *SangerContig* Shiny app - *SangerRead* page

In quality trimming steps, we removes fragment at both ends of sequencing reads with low quality score. It is important because trimmed reads will improves alignment results. *Figure 11* shows the UI for Trimming Method 1 (M1): 'Modified Mott Trimming'. This method is implemented in Phred. Users can change the cutoff score and click "Apply Trimming Parameters" button to update the UI. The value of input must be between 0 and 1. If the input is invalid, the cutoff score will be set to default 0.0001.



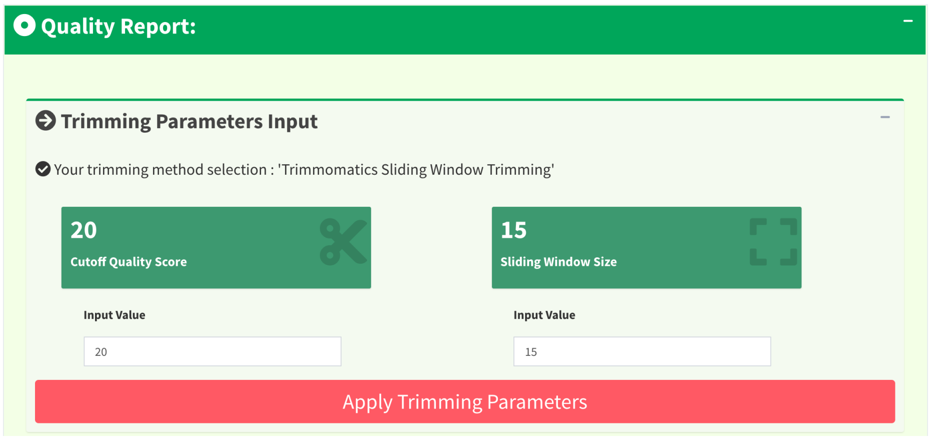
Your trimming method selection : 'Modified Mott Trimming'

1e-04
Cutoff Score

Input Value: 1e-04

Fig. 21: Figure 11. *SangerRead* page - Trimming Method 1 (M1): 'Modified Mott Trimming' UI.

Figure 12 shows another quality trimming method for users to choose from, Trimming Method 2 (M2): 'Trimmomatics Sliding Window Trimming'. This method is implemented in Trimmomatics. Users can change the cutoff quality score as well as sliding window size and click "Apply Trimming Parameters" button to update the UI. The value of cutoff quality score must be between 0 and 60 (default 20); the value of sliding window size must be between 0 and 40 (default 10). If the inputs are invalid, their values will be set to default.



Your trimming method selection : 'Trimmomatics Sliding Window Trimming'

20
Cutoff Quality Score

Input Value: 20

15
Sliding Window Size

Input Value: 15

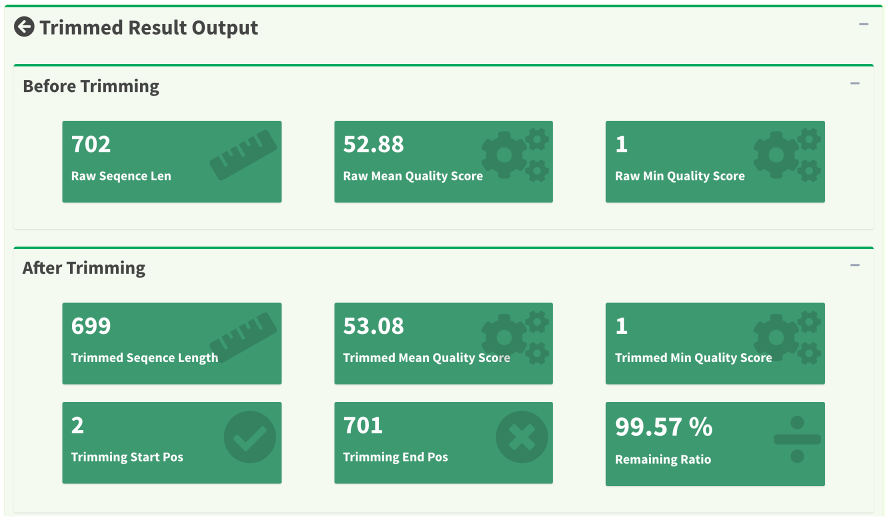
Fig. 22: Figure 12. *SangerRead* page - Trimming Method 2 (M2): 'Trimmomatics Sliding Window Trimming' UI.

Figure 13 shows the quality report before and after trimming. After clicking the "Apply Trimming Parameters" button in *Figure 11* or *Figure 12*, the values of these information boxes will be updated to the latest values.

In *Figure 14*, the x-axis is the index of the base pairs; the y-axis is the Phred quality score. The green horizontal bar at the top of the plot is the raw read region and the orange horizontal bar represents the remaining read region. Both *Figure 14* trimming plot and *Figure 15* chromatogram will be updated once users change the quality trimming parameters and click the "Apply Trimming Parameters" button in *Figure 15*.

If we only see primary and secondary sequences in the table, we will loose some variations. Chromatogram is very helpful to check the peak resolution. *Figure 15* shows the panel of plotting chromatogram. Users can change four parameters: `Base Number Per Row`, `Height Per Row`, `Signal Ratio Cutoff`, and `Show Trimmed Region`. Among them, `Signal Ratio Cutoff` is a key parameter. If its value is default value 0.33, it indicates that the lower peak should be at least 1/3rd as high as the higher peak for it count as a secondary peak.

Here is an example of applying new chromatogram parameters. We click "Show Trimmed Region" to set its value from `FALSE` to `TRUE` and click the "Apply Chromatogram Parameters" button. *Figure 16* shows the loading notification



Trimmed Result Output

Before Trimming

702
Raw Seqence Len

52.88
Raw Mean Quality Score

1
Raw Min Quality Score

After Trimming

699
Trimmed Seqence Length

53.08
Trimmed Mean Quality Score

1
Trimmed Min Quality Score

2
Trimming Start Pos

701
Trimming End Pos

99.57 %
Remaining Ratio

Fig. 23: Figure 13. *SangerRead* page - read quality report before / after trimming.

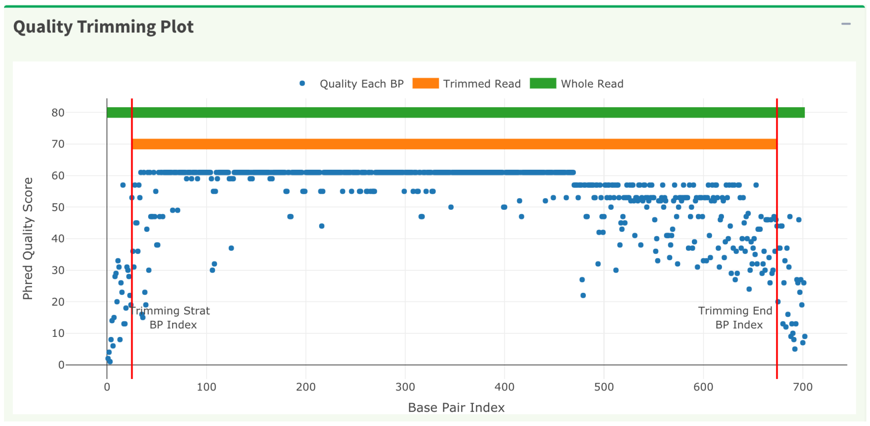


Fig. 24: Figure 14. *SangerContig* page - quality trimming plot.

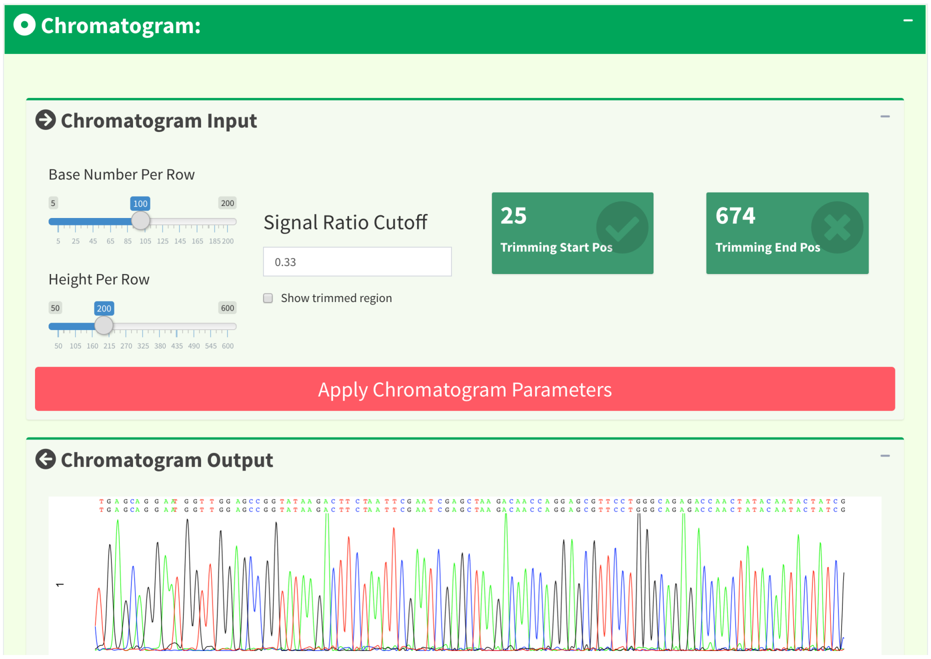


Fig. 25: Figure 15. *SangerContig* page - chromatogram panel.

popup during base calling and chromatogram plotting.

After replotting the chromatogram, we can see that trimmed region is showed in red striped region. *Figure 17* shows part of the the chromatogram (1 bp ~ 240 bp). Moreover, chromatogram will be replotted when trimmed positions or chromatogram parameters are updated.

To let users browse the trimmed primary/secondary sequences without finding “Trimming Start Point” and “Trimming End Point” by themselves, we provide the final trimmed primary/secondary sequences that will be used for reads alignment with quality scores in table format in *Figure 18*. Frameshift amino acid sequences are also provided.

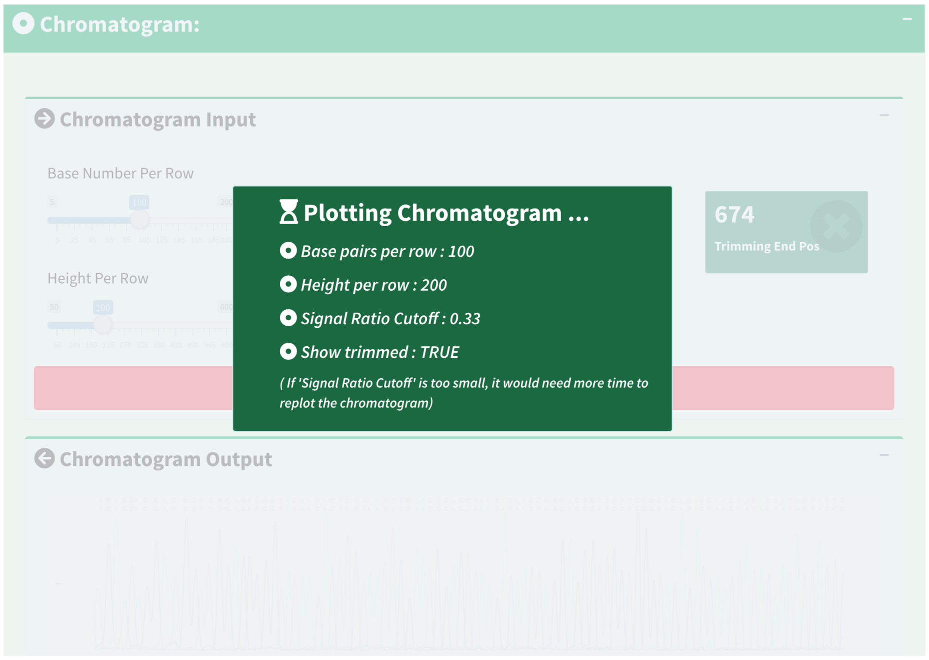
We have updated the trimming and chromatogram parameters for each read. Now, we need to click “Re-calculate contig” button to do alignment again. Last but not least, we can save all data into a new ‘SangerContig’ S4 instance by clicking “Save S4 Instance button”. New S4 instance will be saved in **Rda** format. Users can run `readRDS` function to load it into current R environment. *Figure 19* shows some hints in the save notification popup.

7.5.5 Writing *SangerContig* FASTA files (AB1)

Users can write the *SangerContig* instance, `my_sangerContig`, to **FASTA** files. There are four options for users to choose from in `selection` parameter.

- `reads_unalignment`: Writing reads into a single **FASTA** file (only trimmed without alignment).
- `reads_alignment`: Writing reads alignment and contig read to a single **FASTA** file.
- `contig`: Writing the contig to a single **FASTA** file.
- `all`: Writing reads, reads alignment, and the contig into three different files.

Below is the oneliner for writing out **FASTA** files. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.



Plotting Chromatogram ...

- Base pairs per row : 100
- Height per row : 200
- Signal Ratio Cutoff : 0.33
- Show trimmed : TRUE

(If 'Signal Ratio Cutoff' is too small, it would need more time to replot the chromatogram)

Fig. 26: Figure 16. *SangerContig* page - loading notification popup during replotting chromatogram.

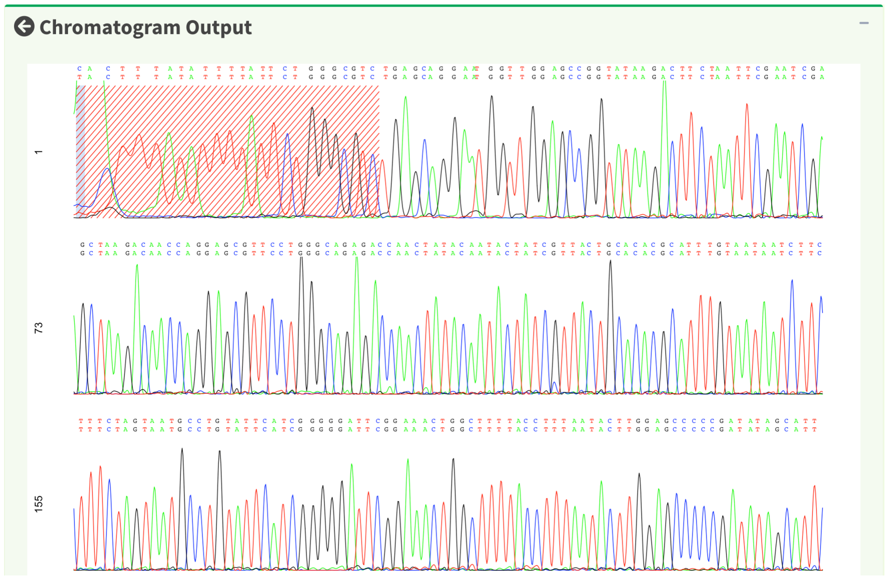
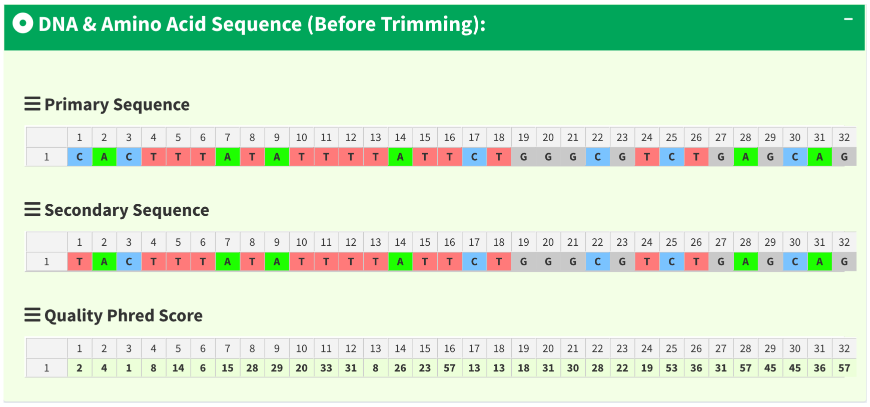


Fig. 27: Figure 17. *SangerContig* page - chromatogram with trimmed region showed.



DNA & Amino Acid Sequence (Before Trimming):

Primary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	C	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C	T	G	A	G	C	A	G

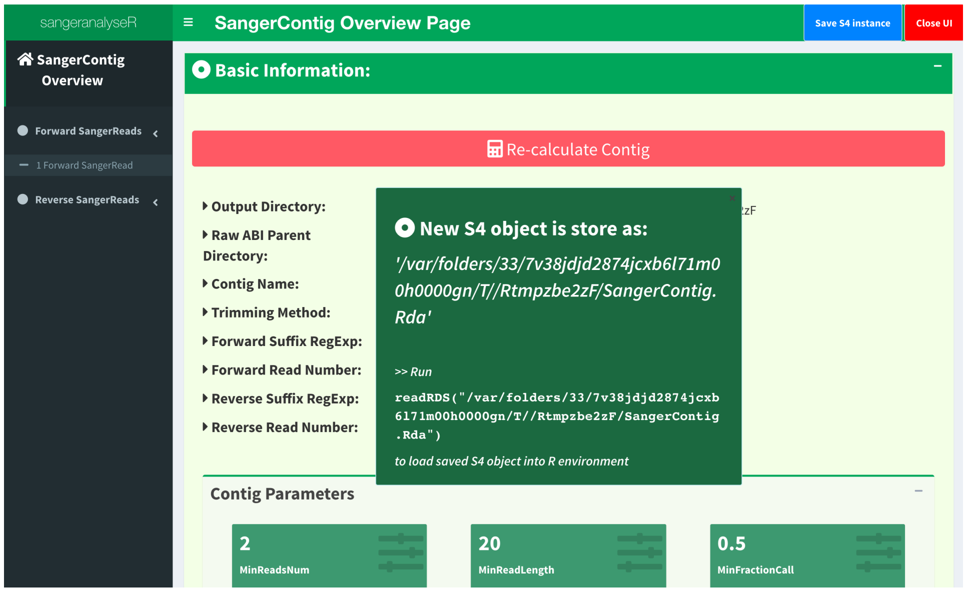
Secondary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	T	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C	T	G	A	G	C	A	G

Quality Phred Score

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	2	4	1	8	14	6	15	28	29	20	33	31	8	26	23	57	13	13	18	31	30	28	22	19	53	36	31	57	45	45	36	57

Fig. 28: Figure 18. *SangerContig* page - trimmed primary/secondary sequences and Phred quality score in table format.



Basic Information:

- Output Directory:
- Raw ABI Parent Directory:
- Contig Name:
- Trimming Method:
- Forward Suffix RegExp:
- Forward Read Number:
- Reverse Suffix RegExp:
- Reverse Read Number:

New S4 object is store as:

'/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//Rtmpzbe2zF/SangerContig.Rda'

>> Run

```
readRDS("/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//Rtmpzbe2zF/SangerContig.Rda")
```

to load saved S4 object into R environment

Fig. 29: Figure 19. *SangerContig* page - saving notification popup.

```
writeFasta(my_sangerContig,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA,
           selection         = "all")
```

Users can download the output FASTA file of this example through the following three links:

(1) `Achl_RBNII384-13_reads_unalignment.fa`

(2) `Achl_RBNII384-13_reads_alignment.fa`

(3) `Achl_RBNII384-13_contig.fa`

7.5.6 Generating *SangerContig* report (AB1)

Last but not least, users can save *SangerContig* instance, `my_sangerContig`, into a report after the analysis. The report will be generated in **HTML** by knitting **Rmd** files.

Users can set `includeSangerRead` parameter to decide to which level the *SangerContig* report will go. Moreover, after the reports are generated, users can easily navigate through reports in different levels within the **HTML** file.

One thing to pay attention to is that if users have many reads, it will take quite a long time to write out all reports. If users only want to generate the contig result, remember to set `includeSangerRead` to `FALSE` in order to save time.

```
generateReport(my_sangerContig,
               outputDir           = tempdir(),
               includeSangerRead   = TRUE)
```

Here is the generated SangerContig html report of this example (ABIF). Users can access to '*Basic Information*', '*SangerContig Input Parameters*', '*Contig Sequence*' and '*Contig Results*' sections inside it. Furthermore, users can also navigate through html reports of all forward and reverse *SangerRead* in this *SangerContig* report.

7.5.7 Code summary (*SangerContig*, AB1)

(1) Preparing *SangerContig* AB1 inputs

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
parentDir <- file.path(rawDataDir, "Allolobophora_chlorotica", "RBNII")
```

(2) Creating *SangerContig* instance from AB1

(2.1) "Regular Expression Method" *SangerContig* creation (AB1)

```
# using `constructor` function to create SangerContig instance
my_sangerContig <- SangerContig(inputSource           = "ABIF",
                                processMethod         = "REGEX",
                                ABIF_Directory        = parentDir,
                                contigName            = "Achl_RBNII384-13",
                                REGEX_SuffixForward   = "_[0-9]*_F.ab1$",
                                REGEX_SuffixReverse   = "_[0-9]*_R.ab1$",
                                refAminoAcidSeq =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪")

# using `new` method to create SangerContig instance
my_sangerContig <- new("SangerContig",
                       inputSource           = "ABIF",
                       processMethod         = "REGEX",
                       ABIF_Directory        = parentDir,
                       contigName            = "Achl_RBNII384-13",
                       REGEX_SuffixForward   = "_[0-9]*_F.ab1$",
                       REGEX_SuffixReverse   = "_[0-9]*_R.ab1$",
                       refAminoAcidSeq =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪")
```

Following is the R shell output that you will get.

(2.2) "CSV file matching" *SangerContig* creation (AB1)

```
csv_namesConversion <- file.path(rawDataDir, "ab1", "SangerContig", "names_conversion_
↪2.csv")

# using `constructor` function to create SangerContig instance
my_sangerContig <- SangerContig(inputSource           = "ABIF",
                                processMethod         = "CSV",
                                ABIF_Directory        = parentDir,
                                contigName            = "Achl_RBNII384-13",
                                CSV_NamesConversion   = csv_namesConversion,
                                refAminoAcidSeq =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪")


# using `new` method to create SangerContig instance
my_sangerContig <- new("SangerContig",
                       inputSource           = "ABIF",
                       processMethod         = "CSV",
                       ABIF_Directory        = parentDir,
                       contigName            = "Achl_RBNII384-13",
```

(continues on next page)

(continued from previous page)

```
                               CSV_NamesConversion    = csv_namesConversion,
                               refAminoAcidSeq =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

Following is the R shell output that you will get.

(3) Updating *SangerContig* quality trimming parameters

```
newSangerContig <- updateQualityParam(my_sangerContig,
                                      TrimmingMethod        = "M2",
                                      M1TrimmingCutoff      = NULL,
                                      M2CutoffQualityScore = 20,
                                      M2SlidingWindowSize  = 15)
```

(4) Launching *SangerContig* Shiny app

```
launchApp(my_sangerContig)
```

(5) Writing *SangerContig* FASTA files $_{\text{(AB1)}}$

```
writeFasta(my_sangerContig)
```

Following is the R shell output that you will get.

You will get three FASTA files:

(1) `Achl_RBNII384-13_reads_unalignment.fa`

(2) `Achl_RBNII384-13_reads_alignment.fa`

(3) `Achl_RBNII384-13_contig.fa`

(6) Generating *SangerContig* report (AB1)

```
generateReport(my_sangerContig)
```

You can check the html report of this SangerContig example (ABIF).

7.6 Advanced User Guide - *SangerAlignment* (AB1)

SangerAlignment is in the toppest level of sangeranalyseR (*Figure_1*), and each **SangerAlignment** instance corresponds to an alignment of contigs in a Sanger sequencing experiment. Among its slots, there is a *SangerContig* list which will be aligned into a consensus contig. Users can access to each *SangerContig* and *SangerRead* inside a *SangerAlignment* instance.

In this section, we are going to go through details about a reproducible *SangerAlignment* analysis example with the **AB1** file input in sangeranalyseR. By running the following example codes, you will get an end-to-end *SangerAlignment* analysis result.

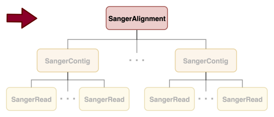


Fig. 30: Figure 1. Classes hierarchy in sangeranalyseR, *SangerAlignment* level.

7.6.1 Preparing *SangerAlignment* AB1 input

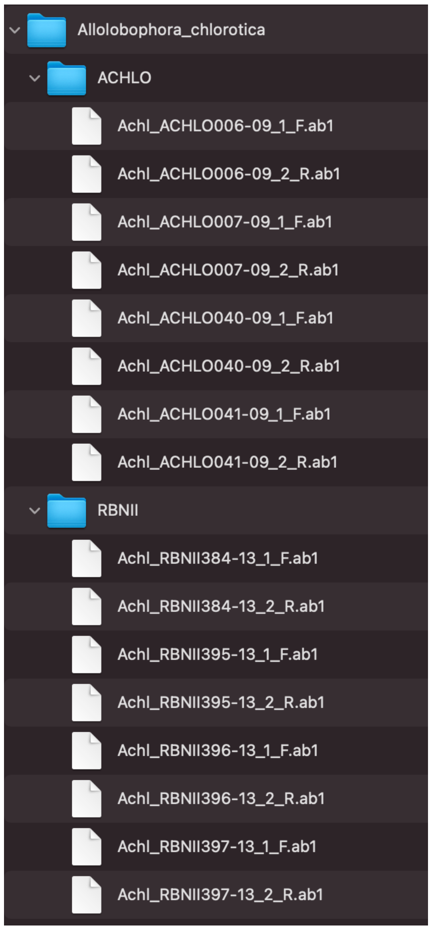
The main input file format to create *SangerAlignment* instance is **AB1**. Before starting the analysis, users need to prepare one directory containing all **AB1** files, and they can be either all placed in the first layer of that directory or be distributed in different subdirectories. In this example, the data are in the sangeranalyseR package; thus, you can simply get its path by running the following codes:

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
parentDir <- file.path(rawDataDir, 'Allolobophora_chlorotica')
```

The value of `parentDir` is where all **AB1** files are placed. If your operating system is macOS, then its value should look like this:

And we showed the files under `parentDir` in *Figure_2*:

Figure_2 shows the file-naming regulation and hierarchy. In this example, `Allolobophora_chlorotica` is the parent directory, and **AB1** files are separated into `ACHLO` and `RBNII` directories. There are two ways for users to group their **AB1** files which are **"regular expression matching"** and **"CSV file matching"**, and following are instructions of how to prepare and name your **AB1** input files.



```
Allolobophora_chlorotica
    ACHLO
        Achl_ACHLO006-09_1_F.ab1
        Achl_ACHLO006-09_2_R.ab1
        Achl_ACHLO007-09_1_F.ab1
        Achl_ACHLO007-09_2_R.ab1
        Achl_ACHLO040-09_1_F.ab1
        Achl_ACHLO040-09_2_R.ab1
        Achl_ACHLO041-09_1_F.ab1
        Achl_ACHLO041-09_2_R.ab1
    RBNII
        Achl_RBNII384-13_1_F.ab1
        Achl_RBNII384-13_2_R.ab1
        Achl_RBNII395-13_1_F.ab1
        Achl_RBNII395-13_2_R.ab1
        Achl_RBNII396-13_1_F.ab1
        Achl_RBNII396-13_2_R.ab1
        Achl_RBNII397-13_1_F.ab1
        Achl_RBNII397-13_2_R.ab1
```

Fig. 31: Figure 2. *SangerAlignment* filename regulation.

(1) "regular expression matching" *SangerAlignment* inputs (AB1)

For regular expression matching method, sangeranalyseR will group **AB1** files based on their contig names and read directions in their filenames automatically; therefore, users have to follow the file-naming regulations below:

Note:

- All input files must have **.ab1** as its file extension.
- Input files that are in the same contig group must have the same contig name in their filenames.
- Forward or reverse direction has to be specified in the filename.

There are three parameters, `ABIF_Directory`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that define the grouping rule to let sangeranalyseR automatically match correct **AB1** files and divide them into forward and reverse directions.

Note:

- `ABIF_Directory`: this is the directory that contains all **AB1** files, and it can be either an absolute or relative path. We suggest users to put only target **AB1** files inside this directory and do not include any other unrelated files.
- `REGEX_SuffixForward`: this is a regular expression that matches all filenames in forward direction. `grepl` function in R is used.
- `REGEX_SuffixReverse`: this is a regular expression that matches all filenames in reverse direction. `grepl` function in R is used.

If you don't know what regular expression is, don't panic - it's just a way of recognising text. Please refer to *What is a regular expression?* for more details. Here is an example of how it works in sangeranalseR:

So how sangeranalyseR works is that it first matches the forward and reverse reads by matching `REGEX_SuffixForward` and `REGEX_SuffixReverse`. Then, sangeranalyseR uses the `str_split` function to split and vectorize their filenames into "contig name" and "direction-suffix" two parts. For those having the same "contig name" will be grouped into the same contig.

Therefore, it is important to have a consistent naming strategy. You need to make sure that **AB1** files in the same contig group share the same contig name and carefully select your `REGEX_SuffixForward` and `REGEX_SuffixReverse`. The bad file-naming and wrong regex matching might accidentally include reverse reads into the forward read list or vice versa, which will make the program generate wrong results. So, how should we systematically name **AB1** files? We suggest users to follow the file-naming regulation in *Figure_3*.

```
[Consensus Read Name] + _ + [index] + _ + [F,R] + .ab1
```

Fig. 32: Figure 3. Suggested **AB1** file-naming regulation - *SangerContig*.

As you can see, the first part of the regulation is a consensus read name (or contig name), which helps sangeranalseR to identify which reads should be grouped into the same contig automatically. The second part of the regulation is an index; since there might be more than one read that is in the forward or reverse direction, we recommend you to number your reads in the same contig group. The third part is a direction which is either 'F' (forward) or 'R' (reverse). Last but not least, files have to end with **.ab1** file extension.

To make it more specific, let's go back to the true example. In *Figure_2*, there are two subdirectories, `ACHLO` and `RBNII`, containing lots of **AB1** files from different contigs in the root directory, `Allolobophora_chlorotica` (`ABIF_Directory`).

First, we set `REGEX_SuffixForward` to `"_[0-9]*_F.ab1$"` and `REGEX_SuffixReverse` to `"_[0-9]*_R.ab1$"` to let sangeranalyseR match and group forward and reverse reads automatically. By the regular expression rule, `Achl_ACHLO006-09_1_F.ab1`, `Achl_ACHLO007-09_1_F.ab1`, `Achl_ACHLO040-09_1_F.ab1`, `Achl_ACHLO041-09_1_F.ab1`, `Achl_RBNII384-13_1_F.ab1`, `Achl_RBNII395-13_1_F.ab1`, `Achl_RBNII396-13_1_F.ab1`, and `Achl_RBNII397-13_1_F.ab1` are categorized into forward reads, and `Achl_ACHLO006-09_1_R.ab1`, `Achl_ACHLO007-09_1_R.ab1`, `Achl_ACHLO040-09_1_R.ab1`, `Achl_ACHLO041-09_1_R.ab1`, `Achl_RBNII384-13_1_R.ab1`, `Achl_RBNII395-13_1_R.ab1`, `Achl_RBNII396-13_1_R.ab1`, and `Achl_RBNII397-13_1_R.ab1` are categorized into reverse reads. Then, `str_split` function is used to split each filename above into "contig name" and "direction-suffix". Eight contig names are detected in this example which are `Achl_ACHLO006-09`, `Achl_ACHLO007-09`, `Achl_ACHLO040-09`, `Achl_ACHLO041-09`, `Achl_RBNII384-13`, `Achl_RBNII395-13`, `Achl_RBNII396-13`, and `Achl_RBNII397-13`. Last, a loop iterates through all contigs, and sangeranalseR creates each of them into a *SangerContig* instance. You can check *Advanced User Guide - SangerContig (AB1)* to see how sangeranalseR creates a *SangerContig* instance.

The reason why we strongly recommend you to follow this file-naming regulation is that by doing so, you can directly adopt the example regular expression matching values, `"_[0-9]*_F.ab1$"` and `"_[0-9]*_R.ab1$"`, to group reads and reduce chances of error. Everything mentioned above will be done automatically.

After understanding how parameters work, please refer to *Creating SangerAlignment instance from AB1* below to see how sangeranalseR creates *SangerAlignment* instance.

(2) "CSV file matching" *SangerAlignment* inputs (AB1)

For those who are not familiar with regular expression, we provide a second grouping approach, CSV file matching method. sangeranalyseR will group **AB1** files based on the information in a CSV file automatically. The note below shows the regulations:

Note: Here is an `example CSV file` (*Figure 4*)

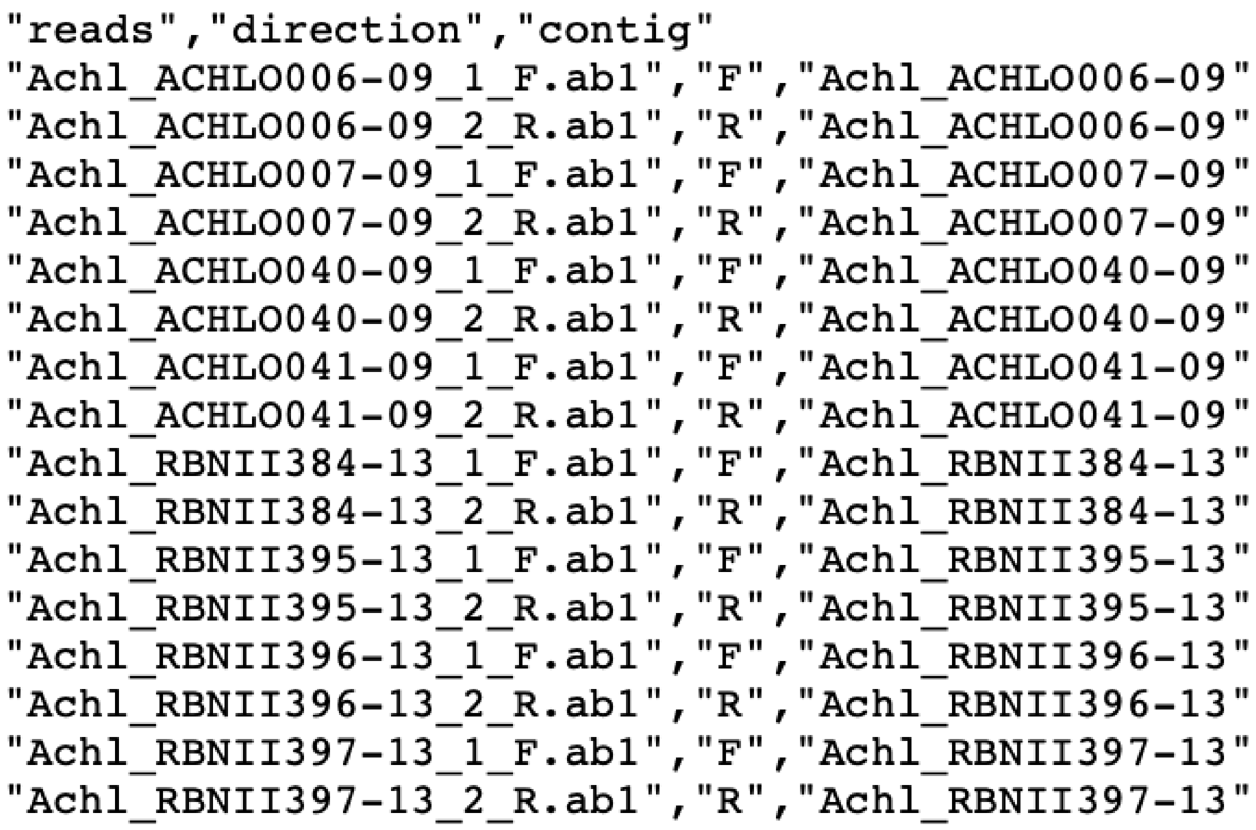
- There must be three columns, "**reads**", "**direction**", and "**contig**", in the CSV file.
- The "**reads**" column stores the filename of **AB1** files that are going to be included in the analysis.
- The "**direction**" column stores the direction of the reads. It must be "F" (forward) or "R" (reverse).
- The "**contig**" column stores the contig name that each read blongs. Reads in the same contig have to have the same contig name, and they will be grouped into the same contig.

There are two parameters, `ABIF_Directory` and `CSV_NamesConversion`,that define the grouping rule to help sangeranalseR to automatically match correct **AB1** files and divide them into forward and reverse directions.

Note:

- `ABIF_Directory`: this is the directory that contains all **AB1** files, and it can be either an absolute or relative path. We suggest users to put only target AB1 files inside this directory and do not include any other unrelated files.
- `CSV_NamesConversion`: this is the path to the CSV file. It can be either an absolute or relative path.

The main difference between "CSV file matching" and "regular expression matching" is where the grouping rule is written. For "regular expression matching", rules are writtein in filenames, and thus more naming requirements are required. In contrast, rules of "CSV file matching" are written in an additional CSV file so it is more flexible on **AB1** file-naming.



```
"reads","direction","contig"
"Achl_ACHLO006-09_1_F.ab1","F","Achl_ACHLO006-09"
"Achl_ACHLO006-09_2_R.ab1","R","Achl_ACHLO006-09"
"Achl_ACHLO007-09_1_F.ab1","F","Achl_ACHLO007-09"
"Achl_ACHLO007-09_2_R.ab1","R","Achl_ACHLO007-09"
"Achl_ACHLO040-09_1_F.ab1","F","Achl_ACHLO040-09"
"Achl_ACHLO040-09_2_R.ab1","R","Achl_ACHLO040-09"
"Achl_ACHLO041-09_1_F.ab1","F","Achl_ACHLO041-09"
"Achl_ACHLO041-09_2_R.ab1","R","Achl_ACHLO041-09"
"Achl_RBNII384-13_1_F.ab1","F","Achl_RBNII384-13"
"Achl_RBNII384-13_2_R.ab1","R","Achl_RBNII384-13"
"Achl_RBNII395-13_1_F.ab1","F","Achl_RBNII395-13"
"Achl_RBNII395-13_2_R.ab1","R","Achl_RBNII395-13"
"Achl_RBNII396-13_1_F.ab1","F","Achl_RBNII396-13"
"Achl_RBNII396-13_2_R.ab1","R","Achl_RBNII396-13"
"Achl_RBNII397-13_1_F.ab1","F","Achl_RBNII397-13"
"Achl_RBNII397-13_2_R.ab1","R","Achl_RBNII397-13"
```

Fig. 33: Figure 4. Example CSV file for *SangerAlignment* instance creation.

So how sangeranalyseR works is that it first reads in the CSV file (with *"reads"*, *"direction"*, and *"contig"* columns), find the names of **AB1** files listed in *"reads"*, group them based on *"contig"*, and assign directions to them based on *"direction"*.

To make it more specific, let's go back to the true example. First, we prepare a `CSV file` (`CSV_NamesConversion`) and a file directory like *Figure_2* (`ABIF_Directory`) with **AB1** files from different contigs. In the CSV file, there are 16 rows and 8 distinct contig names. sangeranalyseR matches *"reads"* of these 16 rows to filenames in `Allolobophora_chlorotica` directory. Then sangeranalyseR groups all matched reads, `Achl_ACHLO006-09_1_F.ab1`, `Achl_ACHLO007-09_1_F.ab1`, `Achl_ACHLO040-09_1_F.ab1`, `Achl_ACHLO041-09_1_F.ab1`, `Achl_RBNII384-13_1_F.ab1`, `Achl_RBNII395-13_1_F.ab1`, `Achl_RBNII396-13_1_F.ab1`, `Achl_RBNII397-13_1_F.ab1`, `Achl_ACHLO006-09_1_R.ab1`, `Achl_ACHLO007-09_1_R.ab1`, `Achl_ACHLO040-09_1_R.ab1`, `Achl_ACHLO041-09_1_R.ab1`, `Achl_RBNII384-13_1_R.ab1`, `Achl_RBNII395-13_1_R.ab1`, `Achl_RBNII396-13_1_R.ab1`, and `Achl_RBNII397-13_1_R.ab1`, into 8 distinct contig names which are `Achl_ACHLO006-09`, `Achl_ACHLO007-09`, `Achl_ACHLO040-09`, `Achl_ACHLO041-09`, `Achl_RBNII384-13`, `Achl_RBNII395-13`, `Achl_RBNII396-13`, and `Achl_RBNII397-13`, by the *"contig"* column. Last, the directions of reads in each contig are assigned by the *"direction"* column. Take `Achl_ACHLO041-09` contig as an example. Its "forward read list" will include `Achl_ACHLO041-09_1_F.ab1`, and its "reverse read list" will include `Achl_ACHLO041-09_1_R.ab1`.

After understanding how parameters work, please refer to *Creating SangerAlignment instance from AB1* below to see how sangeranalseR creates *SangerAlignment* instance.

7.6.2 Creating *SangerAlignment* instance from AB1

After preparing the input directory, we can create a *SangerAlignment* instance by running `SangerAlignment` constructor function or `new` method. The constructor function is a wrapper for `new` method and it makes instance creation more intuitive. Their input parameters are same, and all of them have their default values. For more details about *SangerAlignment* inputs and slots definition, please refer to sangeranalyseR reference manual. We will explain two *SangerAlignment* instance creation methods, "regular expression matching" and "CSV file matching".

(1) "regular expression matching" *SangerAlignment* creation (AB1)

The consturctor function and `new` method below contain three parameters, `ABIF_Directory`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that we mentioned in the previous section. It also includes important parameters like quality trimming, chromatogram visualization, consensus alignment, contigs alignment, and so on. Run the following code and create `my_sangerAlignment` instance.

```
# using `constructor` function to create SangerAlignment instance
my_sangerAlignment <- SangerAlignment(inputSource          = "ABIF",
                                      processMethod        = "REGEX",
                                      ABIF_Directory       = parentDir,
                                      REGEX_SuffixForward  = "_[0-9]*_F.ab1$",
                                      REGEX_SuffixReverse  = "_[0-9]*_R.ab1$",
                                      TrimmingMethod       = "M1",
                                      M1TrimmingCutoff     = 0.0001,
                                      M2CutoffQualityScore = NULL,
                                      M2SlidingWindowSize  = NULL,
                                      baseNumPerRow        = 100,
                                      heightPerRow         = 200,
```

(continues on next page)

(continued from previous page)

```
                                    signalRatioCutoff    = 0.33,
                                    showTrimmed          = TRUE,
                                    refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                                    minReadsNum          = 2,
                                    minReadLength        = 20,
                                    minFractionCall      = 0.5,
                                    maxFractionLost      = 0.5,
                                    geneticCode          = GENETIC_CODE,
                                    acceptStopCodons     = TRUE,
                                    readingFrame         = 1,
                                    processorsNum        = 2)


# using `new` method to create SangerAlignment instance
my_sangerAlignment <- new("SangerAlignment",
                          inputSource            = "ABIF",
                          processMethod          = "REGEX",
                          ABIF_Directory         = parentDir,
                          REGEX_SuffixForward    = "_[0-9]*_F.ab1$",
                          REGEX_SuffixReverse    = "_[0-9]*_R.ab1$",
                          TrimmingMethod         = "M1",
                          M1TrimmingCutoff       = 0.0001,
                          M2CutoffQualityScore   = NULL,
                          M2SlidingWindowSize    = NULL,
                          baseNumPerRow          = 100,
                          heightPerRow           = 200,
                          signalRatioCutoff      = 0.33,
                          showTrimmed            = TRUE,
                          refAminoAcidSeq        =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                          minReadsNum            = 2,
                          minReadLength          = 20,
                          minFractionCall        = 0.5,
                          maxFractionLost        = 0.5,
                          geneticCode            = GENETIC_CODE,
                          acceptStopCodons       = TRUE,
                          readingFrame           = 1,
                          processorsNum          = 2)
```

In this example, 16 reads are detected and 8 distinct *SangerContig* instances are created. These *SangerContig* instances are stored in a "contig list" in `my_sangerAlignment`, which will be used as the input for the following functions.

Inside the R shell, you can run `my_sangerAlignment` to get basic information of the instance or run `my_sangerAlignment@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerAlignment` is successfully created.

Here is the output of `my_sangerAlignment`:

```
SangerAlignment S4 instance
         Input Source :  ABIF
         Process Method :  REGEX
         ABIF Directory :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/Allolobophora_chlorotica
   REGEX Suffix Forward :  _[0-9]*_F.ab1$
```

(continues on next page)

(continued from previous page)

```
   REGEX Suffix Reverse :  _[0-9]*_R.ab1$
      Contigs Consensus : ␣
↪TTATAYTTTATTYTRGGCGTCTGAAGCAGGATAGTAGGAGCYGGTATAAGACTCCTAATTCGAATTGAGCTAAGACARCCGGGAGCATTCCTAGGAAGE
SUCCESS [2021-13-07 23:16:16] 'SangerAlignment' is successfully created!
```

Here is the output of `my_sangerAlignment@objectResults@readResultTable`:

```
                  readName creationResult errorType errorMessage inputSource   ␣
↪direction
1  Achl_ACHLO006-09_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
2  Achl_ACHLO006-09_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
3  Achl_ACHLO007-09_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
4  Achl_ACHLO007-09_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
5  Achl_ACHLO040-09_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
6  Achl_ACHLO040-09_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
7  Achl_ACHLO041-09_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
8  Achl_ACHLO041-09_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
9  Achl_RBNII384-13_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
10 Achl_RBNII384-13_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
11 Achl_RBNII395-13_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
12 Achl_RBNII395-13_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
13 Achl_RBNII396-13_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
14 Achl_RBNII396-13_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
15 Achl_RBNII397-13_1_F.ab1           TRUE       None          None         ABIF Forward␣
↪Read
16 Achl_RBNII397-13_2_R.ab1           TRUE       None          None         ABIF Reverse␣
↪Read
```

(2) "CSV file matching" *SangerAlignment* creation (AB1)

The consturctor function and `new` method below contain two parameters, `ABIF_Directory`, and `CSV_NamesConversion`, that we mentioned in the previous section. It also includes important parameters like quality trimming, chromatogram visualization, consensus alignment, contigs alignment, and so on. Run the following code and create `my_sangerAlignment` instance.

```
csv_namesConversion <- file.path(rawDataDir, "ab1", "SangerAlignment", "names_
↪conversion_all.csv")

# using `constructor` function to create SangerAlignment instance
my_sangerAlignment <- SangerAlignment(inputSource          = "ABIF",
                                      processMethod        = "CSV",
```

(continues on next page)

(continued from previous page)

```
                                      ABIF_Directory       = parentDir,
                                      CSV_NamesConversion  = csv_namesConversion,
                                      TrimmingMethod       = "M1",
                                      M1TrimmingCutoff     = 0.0001,
                                      M2CutoffQualityScore = NULL,
                                      M2SlidingWindowSize  = NULL,
                                      baseNumPerRow        = 100,
                                      heightPerRow         = 200,
                                      signalRatioCutoff    = 0.33,
                                      showTrimmed          = TRUE,
                                      refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                      minReadsNum          = 2,
                                      minReadLength        = 20,
                                      minFractionCall      = 0.5,
                                      maxFractionLost      = 0.5,
                                      geneticCode          = GENETIC_CODE,
                                      acceptStopCodons     = TRUE,
                                      readingFrame         = 1,
                                      processorsNum        = 1)


# using `new` method to create SangerAlignment instance
my_sangerAlignment <- new("SangerAlignment",
                          processMethod        = "CSV",
                          ABIF_Directory       = parentDir,
                          CSV_NamesConversion  = csv_namesConversion,
                          TrimmingMethod       = "M1",
                          M1TrimmingCutoff     = 0.0001,
                          M2CutoffQualityScore = NULL,
                          M2SlidingWindowSize  = NULL,
                          baseNumPerRow        = 100,
                          heightPerRow         = 200,
                          signalRatioCutoff    = 0.33,
                          showTrimmed          = TRUE,
                          refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                          minReadsNum          = 2,
                          minReadLength        = 20,
                          minFractionCall      = 0.5,
                          maxFractionLost      = 0.5,
                          geneticCode          = GENETIC_CODE,
                          acceptStopCodons     = TRUE,
                          readingFrame         = 1,
                          processorsNum        = 1)
```

First, you need to load the CSV file into the R environment. If you are still don't know how to prepare it, please check *(2) "CSV file matching" SangerAlignment inputs (AB1)*. Then, it will follow rules in the CSV file and create `my_sangerAlignment`. After it's created, inside the R shell, you can run `my_sangerAlignment` to get basic information of the instance or run `my_sangerAlignment@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerAlignment` is successfully created.

Here is the output of `my_sangerAlignment`:

```
SangerAlignment S4 instance
         Input Source :  ABIF
         Process Method :  CSV
         ABIF Directory :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/Allolobophora_chlorotica
   CSV Names Conversion :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/ab1/SangerAlignment/names_conversion_all.csv
      Contigs Consensus : ␣
↪TTATAYTTTATTYTRGGCGTCTGAAGCAGGATAGTAGGAGCYGGTATAAGACTCCTAATTCGAATTGAGCTAAGACARCCGGGAGCATTCCTAGGAAGI
SUCCESS [2021-14-07 01:48:28] 'SangerAlignment' is successfully created!
```

Here is the output of `my_sangerAlignment@objectResults@readResultTable`:

```
                  readName creationResult errorType errorMessage inputSource   ␣
↪direction
1  Achl_ACHLO006-09_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
2  Achl_ACHLO006-09_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
3  Achl_ACHLO007-09_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
4  Achl_ACHLO007-09_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
5  Achl_ACHLO040-09_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
6  Achl_ACHLO040-09_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
7  Achl_ACHLO041-09_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
8  Achl_ACHLO041-09_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
9  Achl_RBNII384-13_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
10 Achl_RBNII384-13_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
11 Achl_RBNII395-13_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
12 Achl_RBNII395-13_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
13 Achl_RBNII396-13_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
14 Achl_RBNII396-13_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
15 Achl_RBNII397-13_1_F.ab1           TRUE      None         None        ABIF Forward␣
↪Read
16 Achl_RBNII397-13_2_R.ab1           TRUE      None         None        ABIF Reverse␣
↪Read
```

7.6.3 Updating *SangerAlignment* quality trimming parameters

In the previous *Creating SangerAlignment instance from AB1* part, the constructor function will apply the quality trimming parameters to all reads. After creating a *SangerAlignment* S4 instance, users can change the trimming parameters by running `updateQualityParam` function which will update all reads with the new trimming parameters and redo reads alignment in *SangerContig* and contigs alignment in *SangerAlignment*. If users want to do quality trimming read by read instead all at once, please read *Launching SangerAlignment Shiny app*.

```
newSangerAlignment <- updateQualityParam(my_sangerAlignment,
                                         TrimmingMethod       = "M2",
                                         M1TrimmingCutoff     = NULL,
                                         M2CutoffQualityScore = 29,
                                         M2SlidingWindowSize  = 15)
```

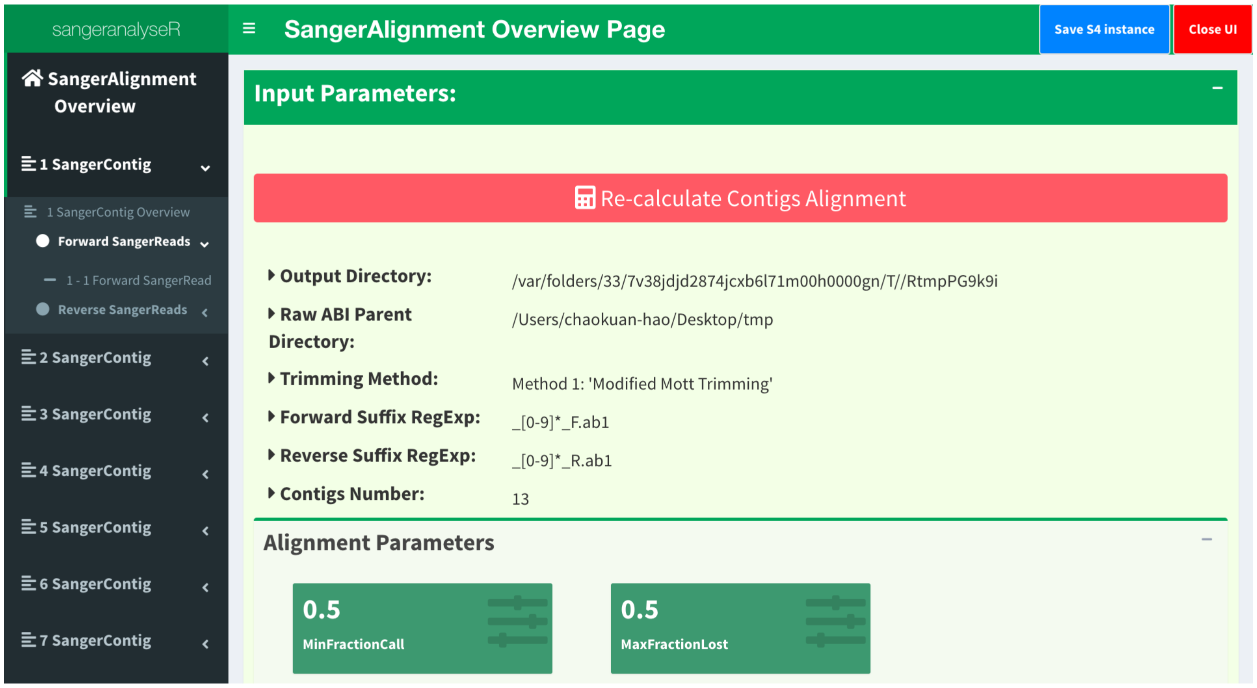
7.6.4 Launching *SangerAlignment* Shiny app

We create an interactive local Shiny app for users to go into each *SangerRead* and *SangerContig* in *SangerAlignment* instance. Users only need to run one function with previously created instance as input, `my_sangerAlignment`, and the *SangerAlignment* Shiny app will pop up. Here, we will go through pages in the three levels.

```
launchApp(my_sangerAlignment)
```

SangerAlignment page (SA app)

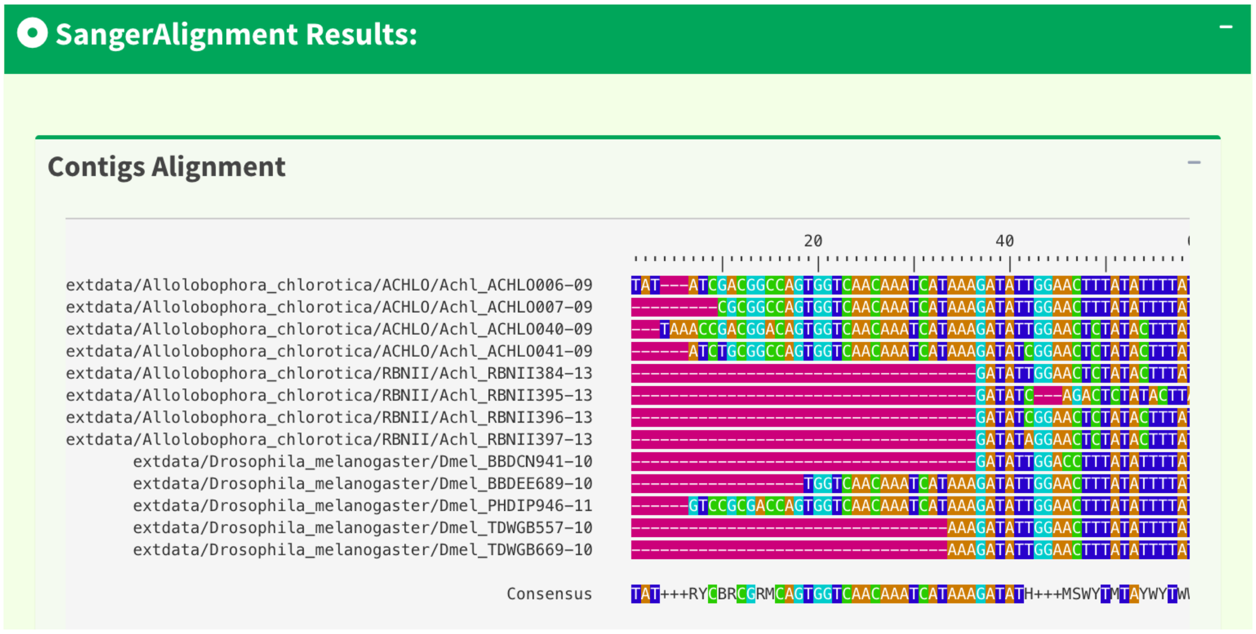
Figure 5 is the initial page and the toppest layer of *SangerAlignment* App. It provides basic parameters in *SangerAlignment* instance, contigs alignment result and phylogenetic tree etc. Before checking the results, users need to click "Re-calculate Contigs Alignment" button to do contigs alignment in order to get the updated results. From the left-hand side panel, we can clearly see the hierarchy of the *SangerAlignment* S4 instance and easily access to all reads and contigs in it.



Output Directory: /var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpPG9k9i
Raw ABI Parent Directory: /Users/chaokuan-hao/Desktop/tmp
Trimming Method: Method 1: 'Modified Mott Trimming'
Forward Suffix RegExp: _[0-9]*_F.ab1
Reverse Suffix RegExp: _[0-9]*_R.ab1
Contigs Number: 13
0.5 MinFractionCall
0.5 MaxFractionLost

Fig. 34: Figure 5. *SangerAlignment* Shiny app initial page - *SangerAlignment* Page.

Scroll down a bit, users can see the contigs alignment result generated by DECIPHER R package embedded in *SangerAlignment* page. *Figure 6* shows the contigs alignment result.



```
                                                            20                  40
extdata/Allolobophora_chlorotica/ACHLO/Achl_ACHLO006-09   TAT---ATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTA
extdata/Allolobophora_chlorotica/ACHLO/Achl_ACHLO007-09   ---------CGCGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTA
extdata/Allolobophora_chlorotica/ACHLO/Achl_ACHLO040-09   ---TAAACCGACGGACAGTGGTCAACAAATCATAAAGATATTGGAACTCTATACTTTA
extdata/Allolobophora_chlorotica/ACHLO/Achl_ACHLO041-09   ------ATCTGCGGCCAGTGGTCAACAAATCATAAAGATATCGGAACTCTATACTTTA
extdata/Allolobophora_chlorotica/RBNII/Achl_RBNII384-13   ------------------------------------GATATTGGAACTCTATACTTTA
extdata/Allolobophora_chlorotica/RBNII/Achl_RBNII395-13   ------------------------------------GATATC---AGACTCTATACTT
extdata/Allolobophora_chlorotica/RBNII/Achl_RBNII396-13   ------------------------------------GATATCGGAACTCTATACTTTA
extdata/Allolobophora_chlorotica/RBNII/Achl_RBNII397-13   ------------------------------------GATATAGGAACTCTATACTTTA
          extdata/Drosophila_melanogaster/Dmel_BBDCN941-10   ------------------------------------GATATTGGACCTTTATATTTTA
          extdata/Drosophila_melanogaster/Dmel_BBDEE689-10   ------------------TGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTA
          extdata/Drosophila_melanogaster/Dmel_PHDIP946-11   ------GTCCGCGACCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTA
          extdata/Drosophila_melanogaster/Dmel_TDWGB557-10   ---------------------------------AAAGATATTGGAACTTTATATTTTA
          extdata/Drosophila_melanogaster/Dmel_TDWGB669-10   ---------------------------------AAAGATATTGGAACTTTATATTTTA

                                                Consensus   TAT+++RYCBRCGRMCAGTGGTCAACAAATCATAAAGATATH+++MSWYTMTAYWYTW
```

Fig. 35: Figure 6. *SangerAlignment* Page - contigs alignment result.

In *SangerAlignment* page, the phylogenetic tree result is provided as well (*Figure 7*). The tree is generated by ape R package which uses neighbor-joining algorithm.

SangerContig page (SA app)

Now, let's go to the page in the next level, *SangerContig* page. Users can click into all contigs and check their results. *Figure 8* shows the overview page of Contig 1. Notice that there is a red "Re-calculate Contig" button. After changing the quality trimming parameters, users need to click the button before checking the results below in order to get the updated information.

The information provided in this page includes : "input parameters", "genetic code table", "reference amino acid sequence", "reads alignment", "difference data frame", "dendrogram", "sample distance heatmap", "indels data frame", "stop codons data frame". *Figure 9* and *Figure 10* show part of the results in the *SangerContig* page. The results are dynamic based on the trimming parameters from user inputs.

SangerRead page (SA app)

Now, let's go to the page in the lowest level, *SangerRead* page. *SangerRead* page contains all details of a read including its trimming and chromatogram inputs and results. All reads are in "forward" or "reverse" direction. Under "Contig Overview" tab (*SangerContig* page), there are two expendable tabs, "Forward Reads" and "Reverse Reads" storing corresponding reads on the left-hand side navigation panel in *Figure 11*. In this example, there are one read in each tab and *Figure 11* shows the "1 - 1 Forward Read" page. It provides basic information, quality trimming inputs, chromatogram plotting inputs etc. Primary/secondary sequences in this figure are dynamic based on the `signalRatioCutoff` value for base calling and the length of them are always same. Another thing to mention is that primary/secondary sequences and the sequences in the chromatogram in *Figure 16* below will always be same after trimming and their color codings for A/T/C/G are same as well.

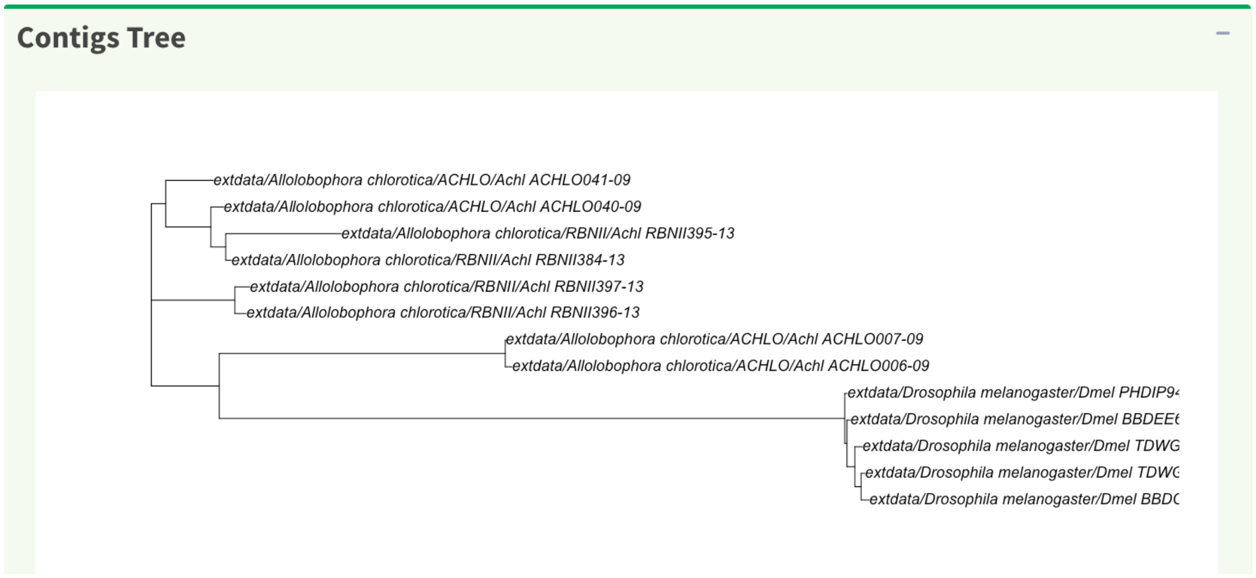
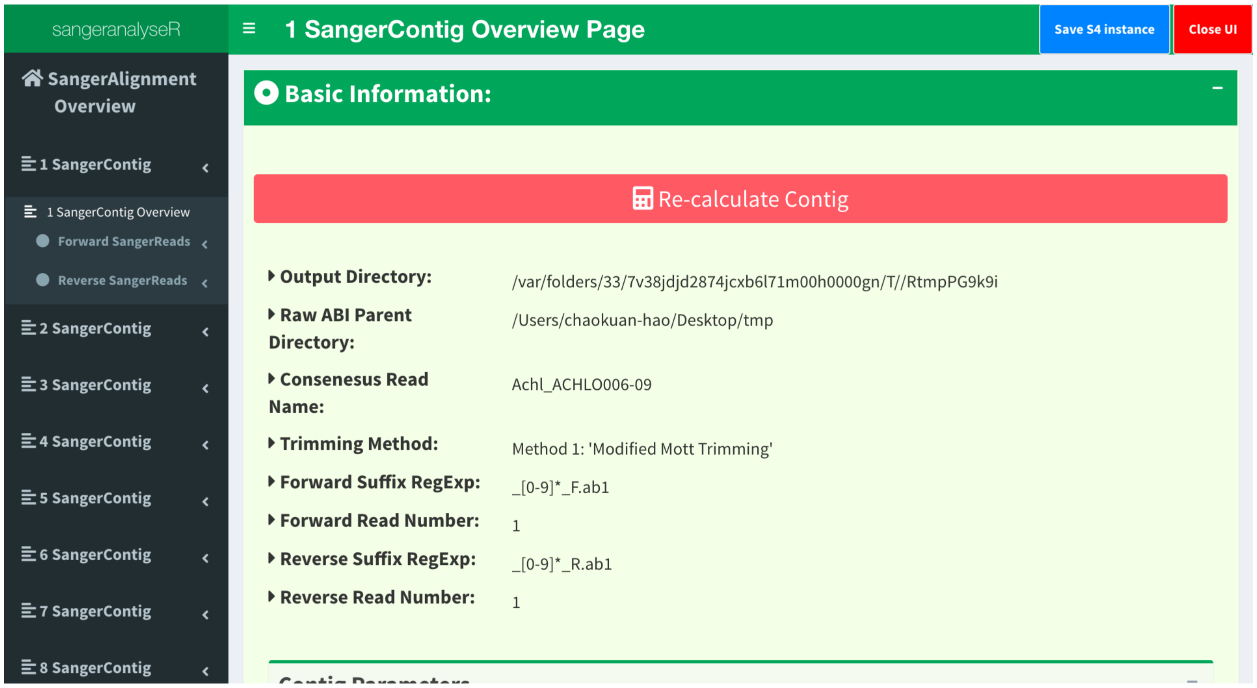
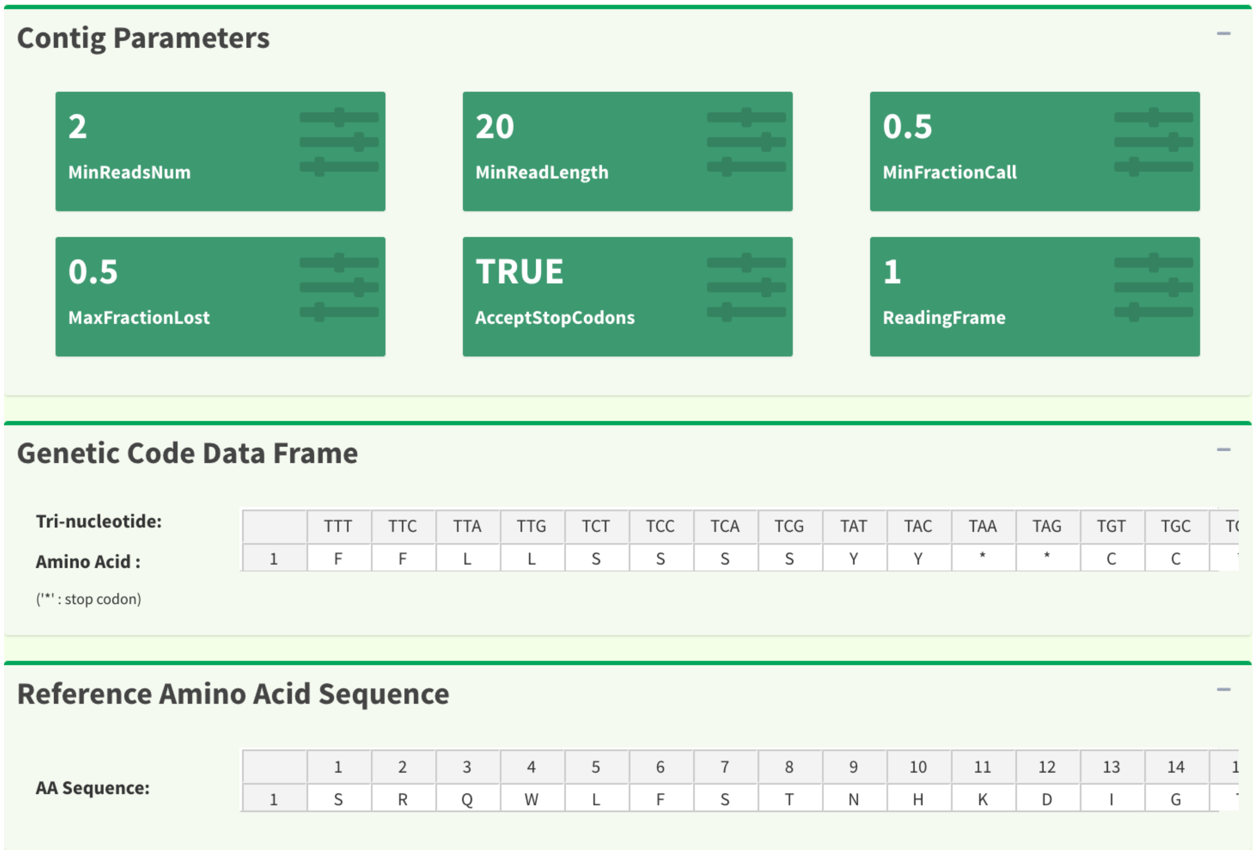


Fig. 36: Figure 7. *SangerAlignment* Page - phylogenetic tree result.



Output Directory:	/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpPG9k9i
Raw ABI Parent Directory:	/Users/chaokuan-hao/Desktop/tmp
Consenesus Read Name:	Achl_ACHLO006-09
Trimming Method:	Method 1: 'Modified Mott Trimming'
Forward Suffix RegExp:	_[0-9]*_F.ab1
Forward Read Number:	1
Reverse Suffix RegExp:	_[0-9]*_R.ab1
Reverse Read Number:	1

Fig. 37: Figure 8. *SangerAlignment* Shiny app - *SangerContig* page.



Contig Parameters

2 MinReadsNum	20 MinReadLength	0.5 MinFractionCall
0.5 MaxFractionLost	TRUE AcceptStopCodons	1 ReadingFrame

Genetic Code Data Frame

Tri-nucleotide:

Amino Acid :

('*' : stop codon)

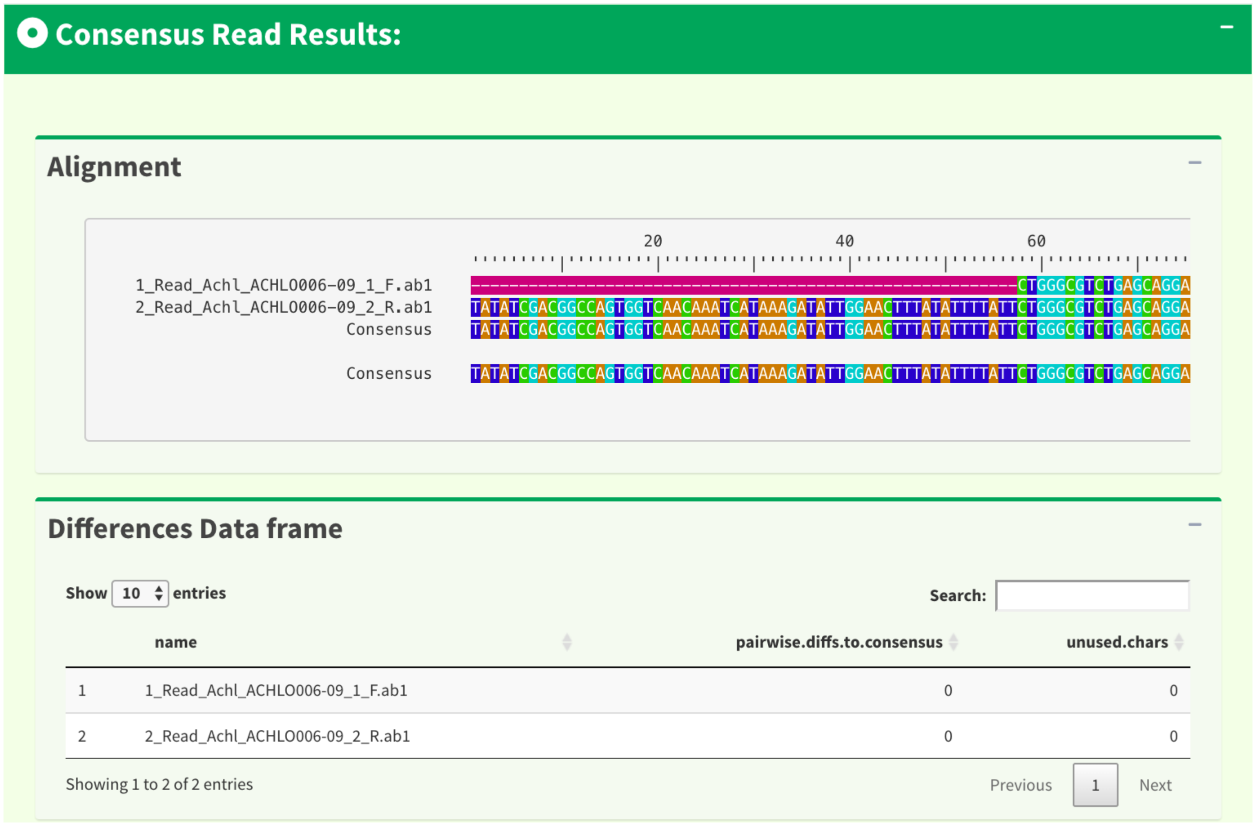
	TTT	TTC	TTA	TTG	TCT	TCC	TCA	TCG	TAT	TAC	TAA	TAG	TGT	TGC
1	F	F	L	L	S	S	S	S	Y	Y	*	*	C	C

Reference Amino Acid Sequence

AA Sequence:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	S	R	Q	W	L	F	S	T	N	H	K	D	I	G

Fig. 38: Figure 9. *SangerContig* page - contig-related parameters, genetic code and reference amino acid sequence.



Consensus Read Results:

Alignment

```
                                   20        40        60
1_Read_Achl_ACHLO006-09_1_F.ab1    ---------------------------------------------------------CTGGGCGTCTGAGCAGGA
2_Read_Achl_ACHLO006-09_2_R.ab1    TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGA
                      Consensus    TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGA

                      Consensus    TATATCGACGGCCAGTGGTCAACAAATCATAAAGATATTGGAACTTTATATTTTATTCTGGGCGTCTGAGCAGGA
```

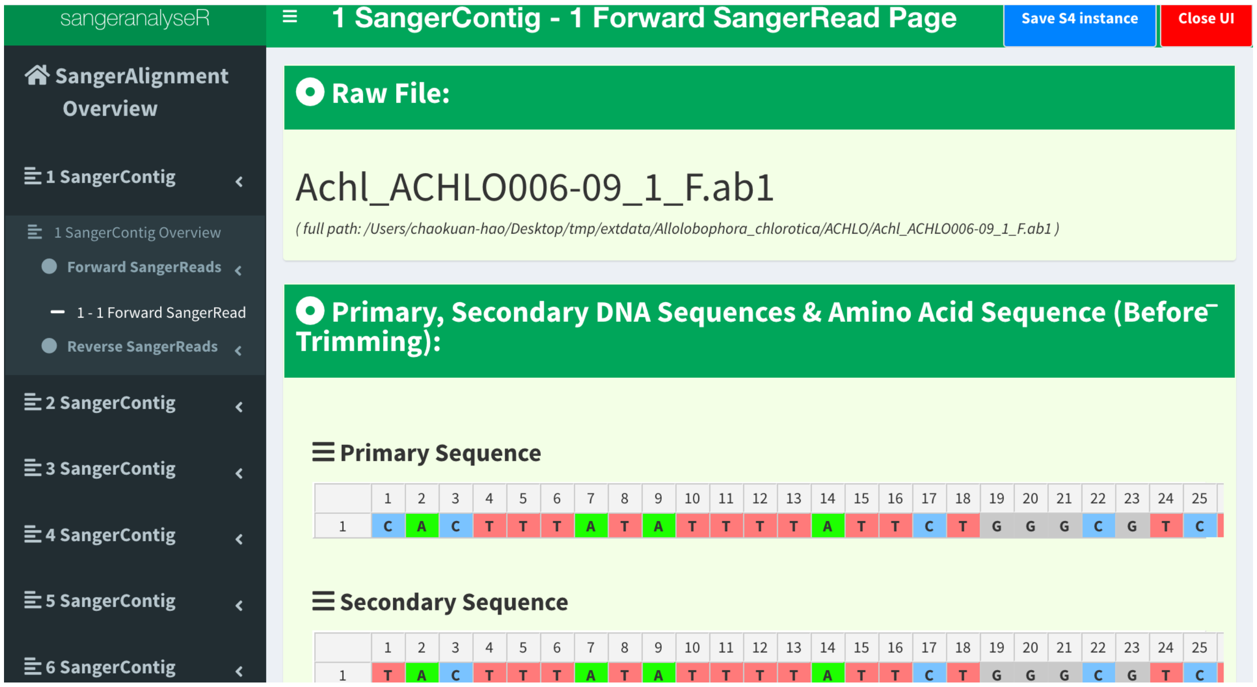
Differences Data frame

Show 10 entries Search:

	name	pairwise.diffs.to.consensus	unused.chars
1	1_Read_Achl_ACHLO006-09_1_F.ab1	0	0
2	2_Read_Achl_ACHLO006-09_2_R.ab1	0	0

Showing 1 to 2 of 2 entries Previous 1 Next

Fig. 39: Figure 10. *SangerContig* page - reads alignment and difference data frame.



1 SangerContig - 1 Forward SangerRead Page

Raw File:

Achl_ACHLO006-09_1_F.ab1

(full path: /Users/chaokuan-hao/Desktop/tmp/extdata/Allolobophora_chlorotica/ACHLO/Achl_ACHLO006-09_1_F.ab1)

Primary, Secondary DNA Sequences & Amino Acid Sequence (Before Trimming):

Primary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	C	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C

Secondary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	T	A	C	T	T	T	A	T	A	T	T	T	T	A	T	T	C	T	G	G	G	C	G	T	C

Fig. 40: Figure 11. *SangerAlignment* Shiny app - *SangerRead* page.

In quality trimming steps, we removes fragment at both ends of sequencing reads with low quality score. It is important because trimmed reads will improves alignment results. *Figure 12* shows the UI for Trimming Method 1 (M1): 'Modified Mott Trimming'. This method is implemented in Phred. Users can change the cutoff score and click "Apply Trimming Parameters" button to update the UI. The value of input must be between 0 and 1. If the input is invalid, the cutoff score will be set to default 0.0001.

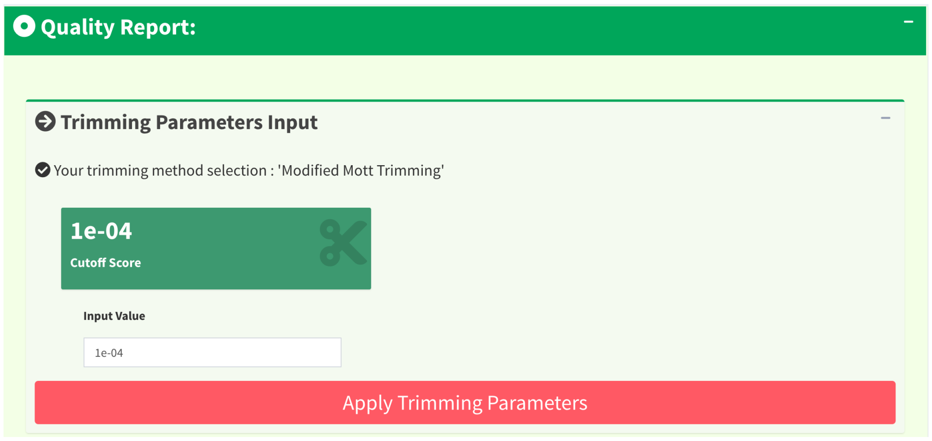

Quality Report:
Trimming Parameters Input
Your trimming method selection : 'Modified Mott Trimming'
1e-04
Cutoff Score
Input Value
1e-04
Apply Trimming Parameters

Fig. 41: Figure 12. *SangerRead* page - Trimming Method 1 (M1): 'Modified Mott Trimming' UI.

Figure 13 shows another quality trimming methods for users to choose from, Trimming Method 2 (M2): 'Trimmomatics Sliding Window Trimming'. This method is implemented in Trimmomatics. Users can change the cutoff quality score as well as sliding window size and click "Apply Trimming Parameters" button to update the UI. The value of cutoff quality score must be between 0 and 60 (default 20); the value of sliding window size must be between 0 and 40 (default 10). If the inputs are invalid, their values will be set to default.

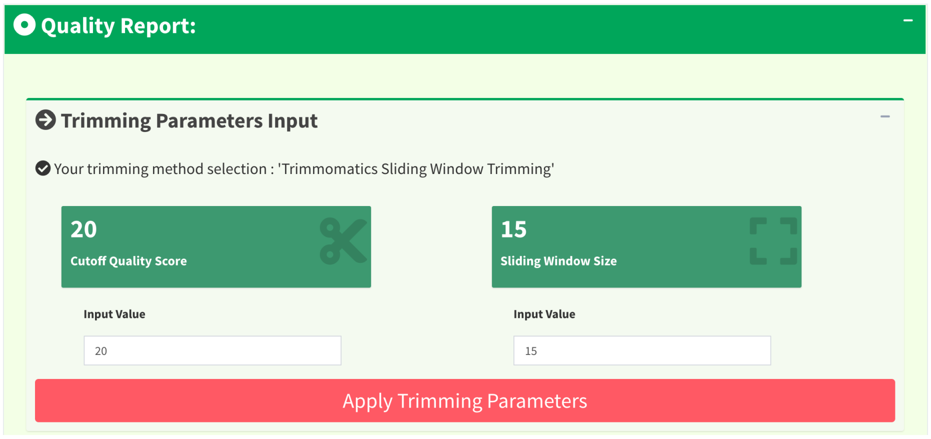
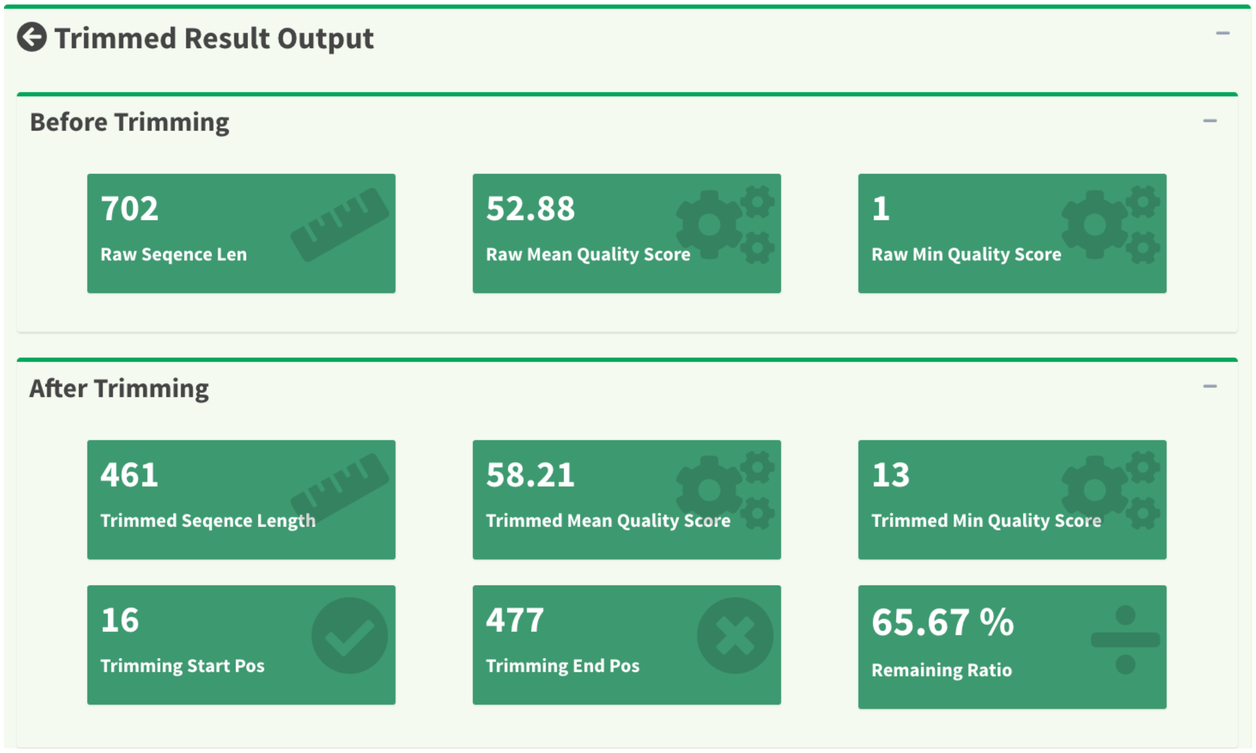

Quality Report:
Trimming Parameters Input
Your trimming method selection : 'Trimmomatics Sliding Window Trimming'
20
Cutoff Quality Score
15
Sliding Window Size
Input Value
20
Input Value
15
Apply Trimming Parameters

Fig. 42: Figure 13. *SangerRead* page - Trimming Method 2 (M2): 'Trimmomatics Sliding Window Trimming' UI.

Figure 14 shows the quality report before and after trimming. After clicking the "Apply Trimming Parameters" button, the values of these information boxes will be updated to the latest values.

In *Figure 15*, the x-axis is the index of the base pairs; the y-axis is the Phred quality score. The green horizontal bar at the top of the plot is the raw read region and the orange horizontal bar represents the trimmed read region. Both *Figure 15* trimming plot and *Figure 16* chromatogram will be updated once users change the quality trimming parameters and click the "Apply Trimming Parameters" button in *Figure 16*.

If we only see primary and secondary sequences in the table, we will loose some variations. Chromatogram is very helpful to check the peak resolution. *Figure 16* shows the panel of plotting chromatogram. Users can change four



Trimmed Result Output

Before Trimming

702	Raw Seqence Len
52.88	Raw Mean Quality Score
1	Raw Min Quality Score

After Trimming

461	Trimmed Seqence Length
58.21	Trimmed Mean Quality Score
13	Trimmed Min Quality Score
16	Trimming Start Pos
477	Trimming End Pos
65.67 %	Remaining Ratio

Fig. 43: Figure 14. *SangerRead* page - read quality report before / after trimming.

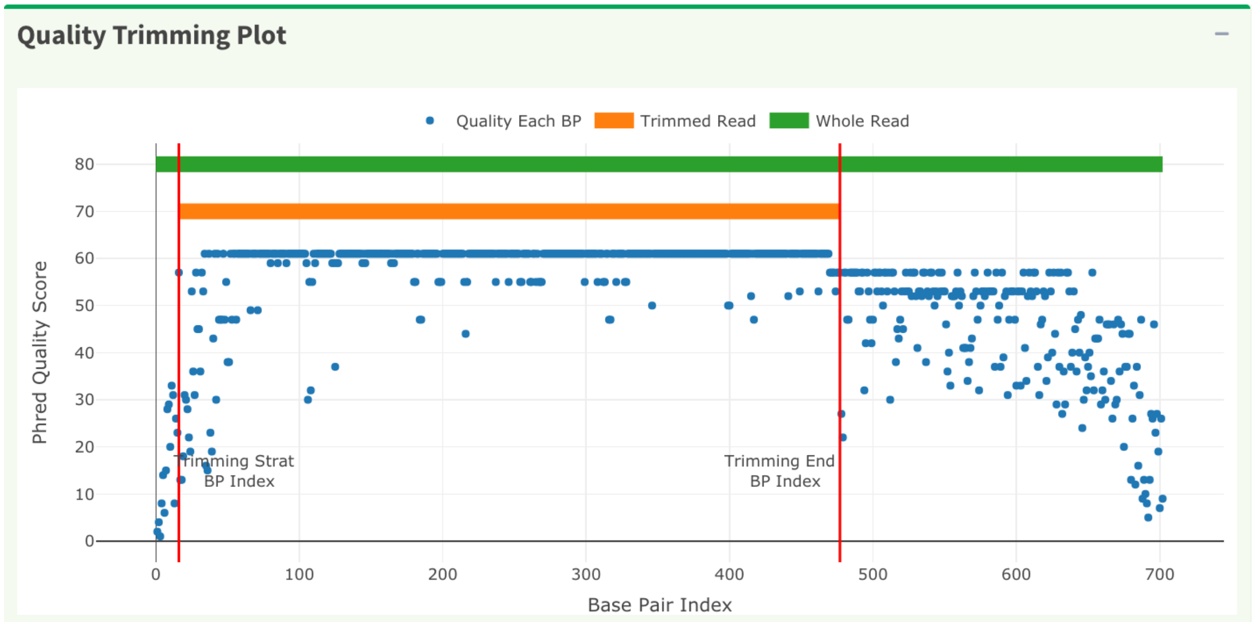
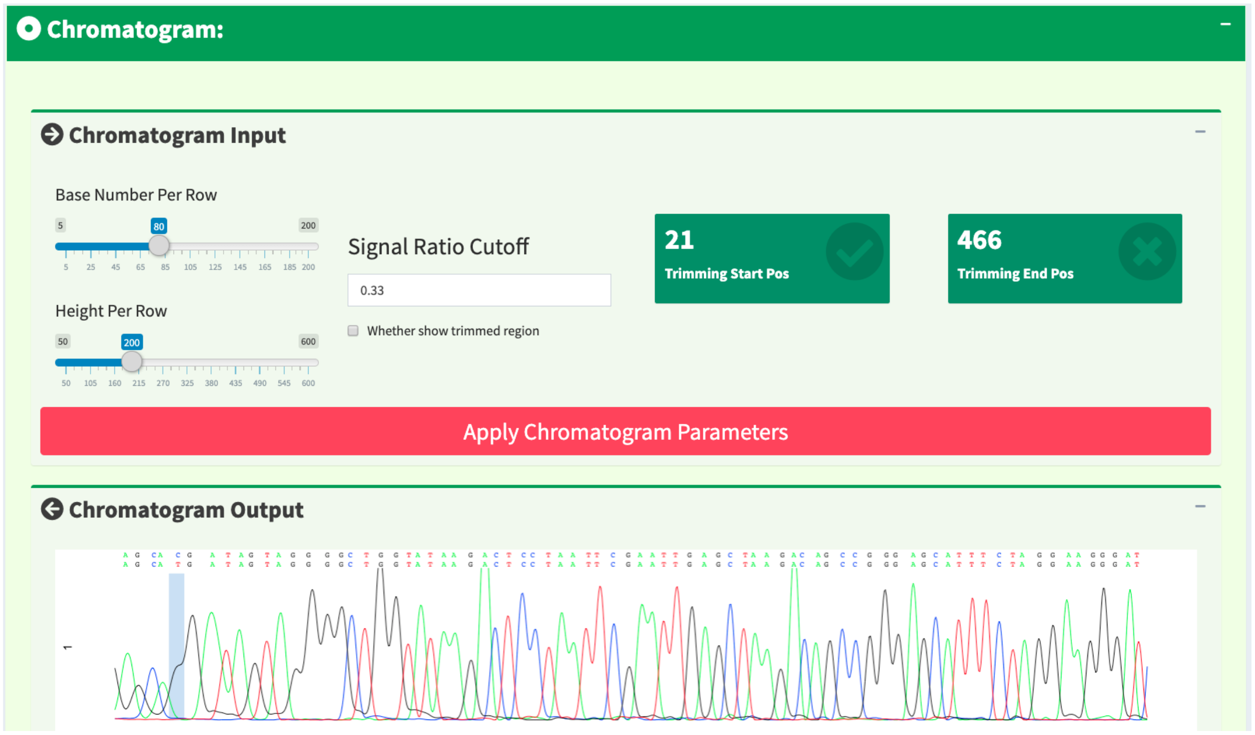


Fig. 44: Figure 15. *SangerRead* page - quality trimming plot.

parameters: `Base Number Per Row`, `Height Per Row`, `Signal Ratio Cutoff`, and `Show Trimmed Region`. Among them, `Signal Ratio Cutoff` is the key parameter. If its value is default value 0.33, it indicates that the lower peak should be at least 1/3rd as high as the higher peak for it count as a secondary peak.



Base Number Per Row: 80

Height Per Row: 200

Signal Ratio Cutoff: 0.33

Whether show trimmed region

21 Trimming Start Pos

466 Trimming End Pos

Fig. 45: Figure 16. *SangerRead* page - chromatogram panel.

Here is an example of applying new chromatogram parameters. We click "Show Trimmed Region" to set its value from FALSE to TRUE. *Figure 17* shows the loading notification popup during base calling and chromatogram plotting.

After replotting the chromatogram, trimmed region is showed in red striped region. *Figure 18* shows part of the the chromatogram (1 bp ~ 240 bp). Moreover, chromatogram will be replotted when trimmed positions or chromatogram parameters are updated.

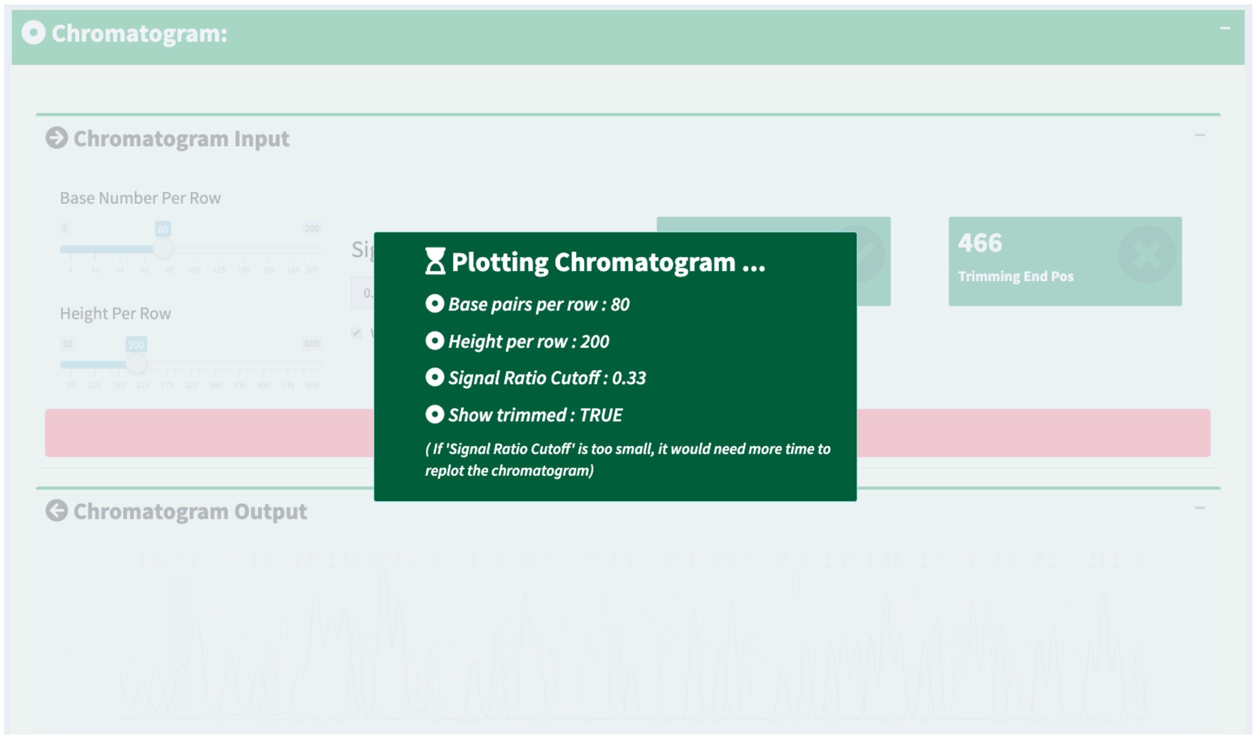
To let users browse the trimmed primary/secondary sequences without finding "Trimming Start Point" and "Trimming End Point" by themselves, we provide the final trimmed primary/secondary sequences that will be used for reads alignment in table format with quality scores in *Figure 19*. Frameshift amino acid sequences are also provided.

We have updated the trimming and chromatogram parameters for each read. Now, we need to click "Re-calculate contig" button to do alignment again. Last but not least, we can save all data into a new 'SangerContig' S4 instance by clicking "Save S4 instance button". New S4 instance will be saved in **Rda** format. Users can run `readRDS` function to load it into current R environment. *Figure 20* shows some hints in the save notification popup.

7.6.5 Writing *SangerAlignment* FASTA files $_{\text{(AB1)}}$

Users can write the *SangerAlignment* instance, `my_sangerAlignment`, to **FASTA** files. There are four options for users to choose from in `selection` parameter.

- `contigs_unalignment`: Writing contigs into a single **FASTA** file.



Plotting Chromatogram ...

- Base pairs per row : 80
- Height per row : 200
- Signal Ratio Cutoff : 0.33
- Show trimmed : TRUE

(If 'Signal Ratio Cutoff' is too small, it would need more time to replot the chromatogram)

Fig. 46: Figure 17. *SangerRead* page - loading notification popup during replotting chromatogram.

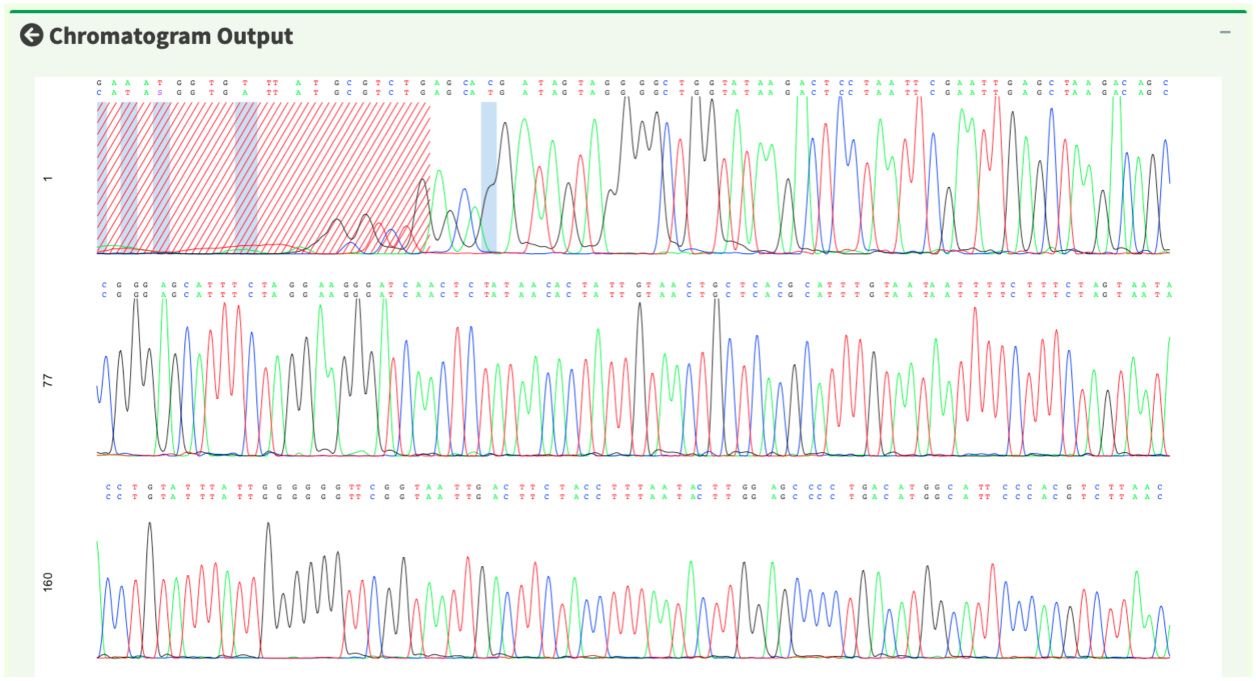
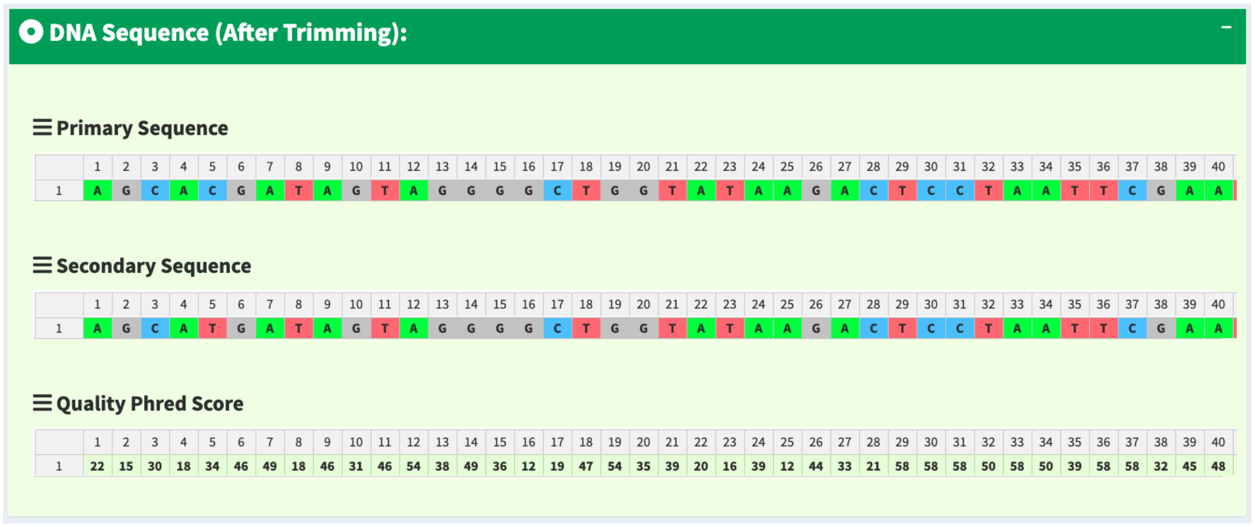


Fig. 47: Figure 18. *SangerRead* page - chromatogram with trimmed region showed.



DNA Sequence (After Trimming):

Primary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	A	G	C	A	C	G	A	T	A	G	T	A	G	G	G	G	C	T	G	G	T	A	T	A	A	G	A	C	T	C	C	T	A	A	T	T	C	G	A	A

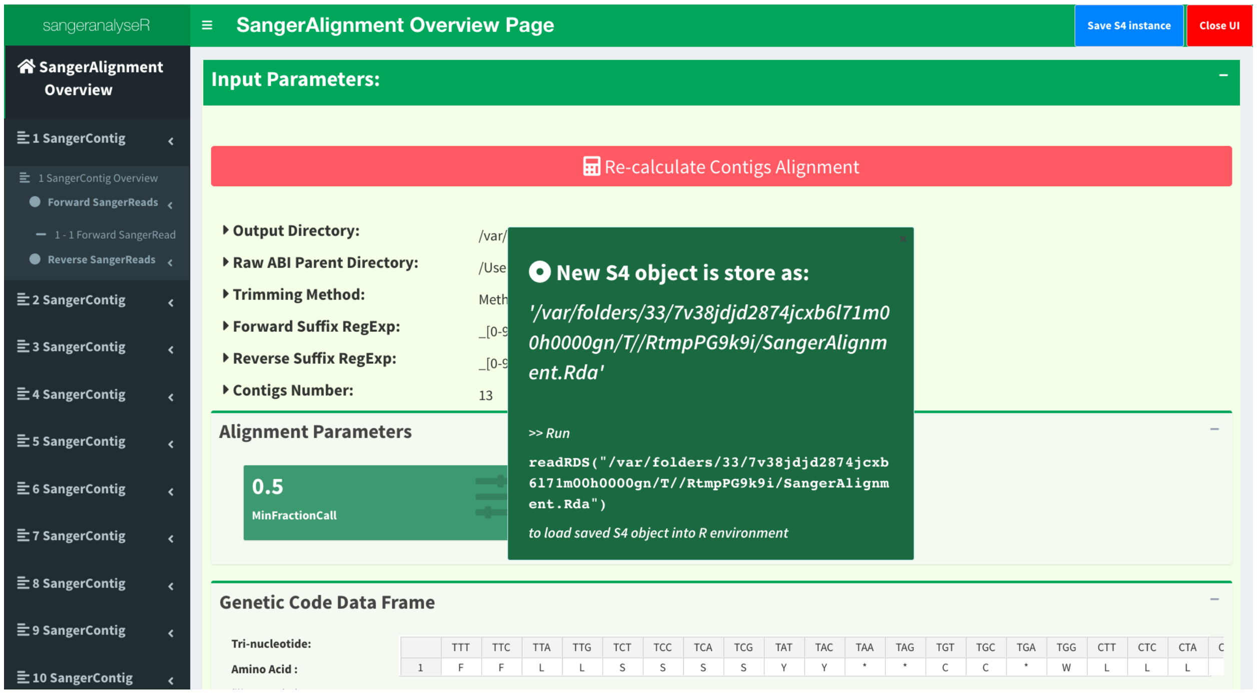
Secondary Sequence

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	A	G	C	A	T	G	A	T	A	G	T	A	G	G	G	G	C	T	G	G	T	A	T	A	A	G	A	C	T	C	C	T	A	A	T	T	C	G	A	A

Quality Phred Score

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	22	15	30	18	34	46	49	18	46	31	46	54	38	49	36	12	19	47	54	35	39	20	16	39	12	44	33	21	58	58	58	50	58	50	39	58	58	32	45	48

Fig. 48: Figure 19. *SangerRead* page - trimmed primary/secondary sequences and Phred quality score in table format.



Output Directory: /var/

Raw ABI Parent Directory: /Use

Trimming Method: Meth

Forward Suffix RegExp: _[0-9

Reverse Suffix RegExp: _[0-9

Contigs Number: 13

Alignment Parameters

0.5 MinFractionCall

New S4 object is store as:

'/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpPG9k9i/SangerAlignment.Rda'

>> Run

```
readRDS("/var/folders/33/7v38jdjd2874jcxb6l71m00h0000gn/T//RtmpPG9k9i/SangerAlignment.Rda")
```

to load saved S4 object into R environment

Genetic Code Data Frame

Tri-nucleotide:		TTT	TTC	TTA	TTG	TCT	TCC	TCA	TCG	TAT	TAC	TAA	TAG	TGT	TGC	TGA	TGG	CTT	CTC	CTA	C
Amino Acid :	1	F	F	L	L	S	S	S	S	Y	Y	*	*	C	C	*	W	L	L	L	

Fig. 49: Figure 20. *SangerRead* page - saving notification popup.

- `contigs_alignment`: Writing contigs alignment and contigs consensus read to a single **FASTA** file.
- `all_reads`: Writing all reads to a single **FASTA** file.
- `all`: Writing contigs, contigs alignment, and all reads into three different files.

Below is the oneliner for writing out **FASTA** files. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

```
writeFasta(my_sangerAlignment,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA,
           selection         = "all")
```

Users can download the output FASTA file of this example through the following three links:

(1) `Sanger_contigs_unalignment.fa`

(2) `Sanger_contigs_alignment.fa`

(3) `Sanger_all_trimmed_reads.fa`

7.6.6 Generating *SangerAlignment* report (AB1)

Last but not least, users can save *SangerAlignment* instance, `my_sangerAlignment`, into a report after the analysis. The report will be generated in **HTML** by knitting **Rmd** files.

Users can set `includeSangerContig` and `includeSangerRead` parameters to decide to which level the *SangerAlignment* report will go. Moreover, after the reports are generated, users can easily navigate through reports in different levels within the **HTML** file.

One thing to pay attention to is that if users have many reads, it will take quite a long time to write out all reports. If users only want to generate the contig result, remember to set `includeSangerRead` and `includeSangerContig` to `FALSE` in order to save time.

```
generateReport(my_sangerAlignment,
               outputDir           = tempdir(),
               includeSangerRead   = FALSE,
               includeSangerContig = FALSE)
```

Here is the generated SangerAlignment html report of this example (ABIF). Users can access to '*Basic Information*', '*Contigs Consensus*', '*Contigs Alignment*', '*Contigs Tree*', and '*Contig Reports*' sections inside it. Furthermore, users can also navigate through html reports of all contigs and forward and reverse *SangerRead* in this *SangerAlignment* report.

7.6.7 Code summary (*SangerAlignment*, AB1)

(1) Preparing *SangerAlignment* AB1 inputs

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
parentDir <- file.path(rawDataDir, 'Allolobophora_chlorotica')
```

(2) Creating *SangerAlignment* instance from AB1

(2.1) "Regular Expression Method" *SangerAlignment* creation (AB1)

```
# using `constructor` function to create SangerAlignment instance
my_sangerAlignment <- SangerAlignment(inputSource          = "ABIF",
                                      processMethod        = "REGEX",
                                      ABIF_Directory       = parentDir,
                                      REGEX_SuffixForward  = "_[0-9]*_F.ab1$",
                                      REGEX_SuffixReverse  = "_[0-9]*_R.ab1$",
                                      refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")


# using `new` method to create SangerAlignment instance
my_sangerAlignment <- new("SangerAlignment",
                          inputSource          = "ABIF",
                          processMethod        = "REGEX",
                          ABIF_Directory       = parentDir,
                          REGEX_SuffixForward  = "_[0-9]*_F.ab1$",
                          REGEX_SuffixReverse  = "_[0-9]*_R.ab1$",
                          refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

Following is the R shell output that you will get.

(2.2) "CSV file matching" *SangerAlignment* creation (AB1)

```
csv_namesConversion <- file.path(rawDataDir, "ab1", "SangerAlignment", "names_
↪conversion_all.csv")

# using `constructor` function to create SangerAlignment instance
my_sangerAlignment <- SangerAlignment(inputSource          = "ABIF",
                                      processMethod        = "CSV",
                                      ABIF_Directory       = parentDir,
                                      CSV_NamesConversion  = csv_namesConversion,
```

(continues on next page)

(continued from previous page)

```
                                     refAminoAcidSeq       =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")


# using `new` method to create SangerAlignment instance
my_sangerAlignment <- new("SangerAlignment",
                          processMethod         = "CSV",
                          ABIF_Directory        = parentDir,
                          CSV_NamesConversion   = csv_namesConversion,
                          refAminoAcidSeq       =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

Following is the R shell output that you will get.

(3) Updating *SangerAlignment* quality trimming parameters (AB1)

```
newSangerAlignment <- updateQualityParam(my_sangerAlignment,
                                         TrimmingMethod         = "M2",
                                         M1TrimmingCutoff       = NULL,
                                         M2CutoffQualityScore = 29,
                                         M2SlidingWindowSize  = 15)
```

(4) Launching *SangerAlignment* Shiny app (AB1)

```
launchApp(my_sangerAlignment)
```

(5) Writing *SangerAlignment* FASTA files (AB1)

```
writeFasta(my_sangerAlignment)
```

Following is the R shell output that you will get.

You will get three FASTA files:

(1) `Sanger_contigs_unalignment.fa`

(2) `Sanger_contigs_alignment.fa`

(3) `Sanger_all_trimmed_reads.fa`

(6) Generating *SangerAlignment* report (AB1)

```
generateReport(my_sangerAlignment)
```

You can check the html report of this SangerAlignment example (ABIF).

7.7 Advanced User Guide - *SangerRead* (FASTA)

SangerRead is in the bottommost level of sangeranalyseR (*Figure_1*), and each *SangerRead* object corresponds to a single read in Sanger sequencing. In this section, we are going to go through detailed sangeranalyseR data analysis steps in *SangerRead level* with **FASTA** file input.

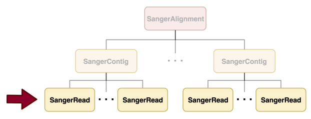


Fig. 50: Figure 1. Hierarchy of classes in sangeranalyseR, *SangerRead* level.

7.7.1 Preparing *SangerRead* FASTA input

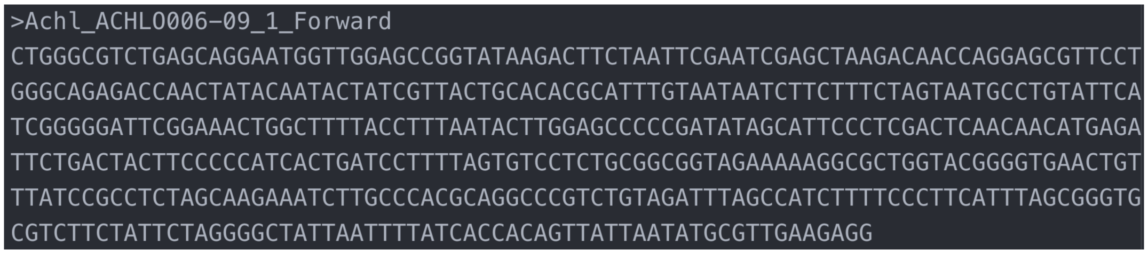
The **FASTA** input method is designed for those who do not want to do quality trimming and base calling on their Sanger sequencing data; therefore, no quality trimming and chromatogram input parameters are needed. Before starting the analysis, users need to prepare a **FASTA** file, and in this example, it is in the sangeranalyseR package; thus, you can simply get its path by running the following codes:

```
inputFilesPath <- system.file("extdata/", package = "sangeranalyseR")
A_chloroticaFFNfa <- file.path(inputFilesPath,
                               "fasta",
                               "SangerRead",
                               "Achl_ACHLO006-09_1_F.fa")
```

The only hard regulation of the filename, `Achl_ACHLO006-09_1_F.fa` in this example, is that file extension must be **.fasta** or **.fa**.

7.7.2 Creating *SangerRead* instance from FASTA

After preparing an input **FASTA** file, the next step is to create a *SangerRead* instance by running `SangerRead` constructor function or `new` method. The constructor function is a wrapper for `new` method which makes instance creation more intuitive. All of the input parameters have their default values. We list important parameters in the two *SangerRead* creation methods below. `readFileName` stores the **FASTA** filename, and inside it, the string in the first line after ">" is the name of the read. Users need to assign the name of the read to `fastaReadName` which is used for read-matching. *Figure 2* is a valid **FASTA** file, `Achl_ACHLO006-09_1_F.fa` (`example FASTA file`), and the value of `fastaReadName` is `Achl_ACHLO006-09_1_F`.



```
>Achl_ACHLO006-09_1_Forward
CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCT
GGGCAGAGACCAACTATACAATACTATCGTTACTGCACACGCATTTGTAATAATCTTCTTTCTAGTAATGCCTGTATTCA
TCGGGGGATTCGGAAACTGGCTTTTACCTTTAATACTTGGAGCCCCCGATATAGCATTCCCTCGACTCAACAACATGAGA
TTCTGACTACTTCCCCCATCACTGATCCTTTTAGTGTCCTCTGCGGCGGTAGAAAAAGGCGCTGGTACGGGGTGAACTGT
TTATCCGCCTCTAGCAAGAAATCTTGCCCACGCAGGCCCGTCTGTAGATTTAGCCATCTTTTCCCTTCATTTAGCGGGTG
CGTCTTCTATTCTAGGGGCTATTAATTTTATCACCACAGTTATTAATATGCGTTGAAGAGG
```

Fig. 51: Figure 2. *SangerRead* **FASTA** input file.

```
# using `constructor` function to create SangerRead instance
sangerReadFfa <- SangerRead(inputSource          = "FASTA",
                            readFeature          = "Forward Read",
                            readFileName         = A_chloroticaFFNfa,
                            fastaReadName        = "Achl_ACHLO006-09_1_F",
                            geneticCode          = GENETIC_CODE)


# using `new` method to create SangerRead instance
sangerReadFfa <- new("SangerRead",
                     inputSource          = "FASTA",
                     readFeature          = "Forward Read",
                     readFileName         = A_chloroticaFFNfa,
                     fastaReadName        = "Achl_ACHLO006-09_1_F",
                     geneticCode          = GENETIC_CODE)
```

The inputs of `SangerRead` constructor function and `new` method are the same. For more details about *SangerRead* inputs and slots definition, please refer to sangeranalyseR reference manual.

Inside the R shell, you can run `sangerReadFfa` to get basic information of the instance or run `sangerReadFfa@objectResults@readResultTable` to check the creation result of every Sanger read after `sangerReadFfa` is successfully created.

Here is the output of `sangerReadFfa`:

```
SangerRead S4 instance
         Input Source :  FASTA
         Read Feature :  Forward Read
         Read FileName :  Achl_ACHLO006-09_1_F.fa
      Fasta Read Name :  Achl_ACHLO006-09_1_F
      Primary Sequence : ␣
↪CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCTGGGCAGAGACCAACTATA
SUCCESS [2021-12-07 23:37:43] 'Achl_ACHLO006-09_1_F.fa' is successfully created!
```

Here is the output of `sangerReadFfa@objectResults@readResultTable`:

```
            readName creationResult errorType errorMessage inputSource    direction
1 Achl_ACHLO006-09_1_F           TRUE      None         None       FASTA Forward Read
```

7.7.3 Writing *SangerRead* FASTA files (FASTA)

Users can write `sangerReadFfa` to a **FASTA** file. Because the **FASTA** input method does not support quality trimming or base calling, in this example, the sequence of the output **FASTA** file will be the same as the input **FASTA** file. Moreover, users can set the compression level through the one-liner, `writeFasta`, which mainly depends on `writeXStringSet` function in Biostrings R package.

```
writeFasta(sangerReadFfa,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA)
```

Users can download the `Achl_ACHLO006-09_1_F.fa` of this example.

7.7.4 Generating *SangerRead* report (FASTA)

Last but not least, users can save `sangerReadFfa` into a static **HTML** report by knitting **Rmd** files. In this example, `tempdir` function will generate a random path.

```
generateReport(sangerReadFfa,
               outputDir = tempdir())
```

SangerRead_Report_fasta.html is the generated *SangerRead* report html of this example. Users can access to '*Basic Information*', '*DNA Sequence*' and '*Amino Acids Sequence*' sections inside this report.

7.7.5 Code summary (*SangerRead*, fasta)

(1) Preparing *SangerRead* FASTA input

```
inputFilesPath <- system.file("extdata/", package = "sangeranalyseR")
A_chloroticaFFNfa <- file.path(inputFilesPath,
                               "fasta",
                               "SangerRead",
                               "Achl_ACHLO006-09_1_F.fa")
```

(2) Creating *SangerRead* instance from FASTA

```
# using `constructor` function to create SangerRead instance
sangerReadFfa <- SangerRead(inputSource         = "FASTA",
                            readFeature         = "Forward Read",
                            readFileName        = A_chloroticaFFNfa,
                            fastaReadName       = "Achl_ACHLO006-09_1_F")

# using `new` method to create SangerRead instance
sangerReadFfa <- new("SangerRead",
                     inputSource         = "FASTA",
                     readFeature         = "Forward Read",
                     readFileName        = A_chloroticaFFNfa,
                     fastaReadName       = "Achl_ACHLO006-09_1_F")
```

Following is the R shell output that you will get.

(3) Writing *SangerRead* FASTA files (FASTA)

```
writeFasta(sangerReadFfa)
```

Following is the R shell output that you will get.

And you will get one FASTA file:

(1) `Achl_ACHLO006-09_1_F.fa`

(4) Generating *SangerRead* report (FASTA)

```
generateReport(sangerReadFfa)
```

You can check the html report of this SangerRead example (FASTA).

7.8 Advanced User Guide - *SangerContig* (FASTA)

SangerContig is in the intermediate level of sangeranalyseR (*Figure_1*), and each *SangerContig* instance corresponds to a contig in a Sanger sequencing experiment. Among its slots, there are two lists, forward and reverse read list, storing *SangerRead* in the corresponding direction.

In this section, we are going to go through details about a reproducible *SangerContig* analysis example with the **FASTA** file input in sangeranalyseR. By running the following example codes, you will get an end-to-end SangerContig analysis result.

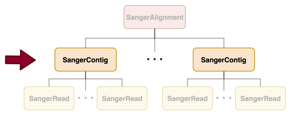


Fig. 52: Figure 1. Hierarchy of classes in sangeranalyseR, *SangerContig* level.

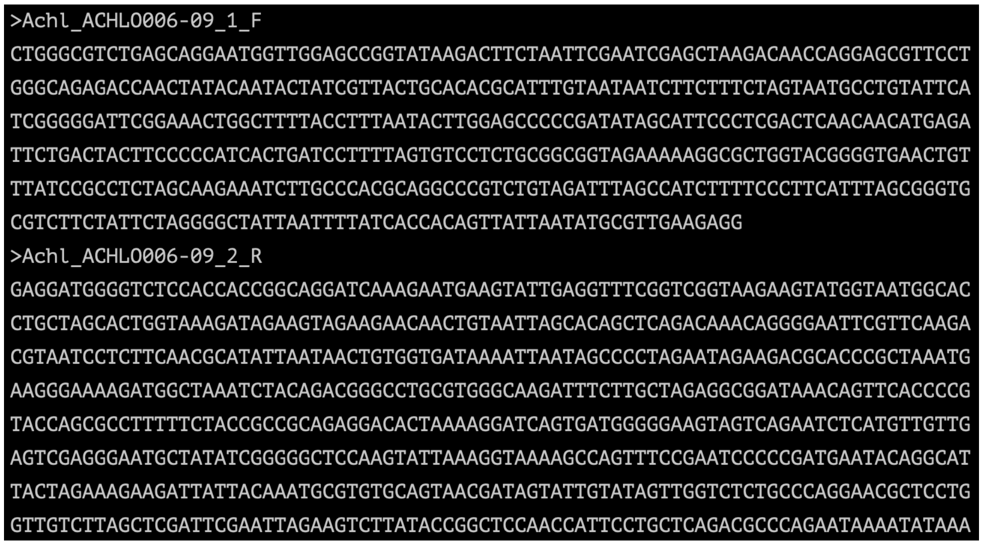
7.8.1 Preparing *SangerContig* FASTA input

In *Advanced User Guide - SangerContig (AB1)*, we demonstrated how to use **AB1** input files to create *SangerContig* instance. Here, we explain another input format - the **FASTA** input. Before starting the analysis, users need to prepare one **FASTA** file, which must end with **.fa** or **.fasta**, containing sequences of all reads. In this example, the **FASTA** file is in the sangeranalyseR package, and you can simply get its path by running the following codes:

```r
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
fastaFN <- file.path(rawDataDir, "fasta", "SangerContig", "Achl_ACHL0006-09.fa")
```

The value of `fastaFN` is where the **FASTA** file is placed. If your operating system is macOS, then its value should look like this:

And we showed the reads in `fastaFN` in Figure_2 (`example FASTA file`):



```
>Achl_ACHL0006-09_1_F
CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCT
GGGCAGAGACCAACTATACAATACTATCGTTACTGCACACGCATTTGTAATAATCTTCTTTCTAGTAATGCCTGTATTCA
TCGGGGGATTCGGAAACTGGCTTTTACCTTTAATACTTGGAGCCCCCGATATAGCATTCCCTCGACTCAACAACATGAGA
TTCTGACTACTTCCCCCATCACTGATCCTTTTAGTGTCCTCTGCGGCGGTAGAAAAAGGCGCTGGTACGGGGTGAACTGT
TTATCCGCCTCTAGCAAGAAATCTTGCCCACGCAGGCCCGTCTGTAGATTTAGCCATCTTTTCCCTTCATTTAGCGGGTG
CGTCTTCTATTCTAGGGGCTATTAATTTTATCACCACAGTTATTAATATGCGTTGAAGAGG
>Achl_ACHL0006-09_2_R
GAGGATGGGGTCTCCACCACCGGCAGGATCAAAGAATGAAGTATTGAGGTTTCGGTCGGTAAGAAGTATGGTAATGGCAC
CTGCTAGCACTGGTAAAGATAGAAGTAGAAGAACAACTGTAATTAGCACAGCTCAGACAAACAGGGGAATTCGTTCAAGA
CGTAATCCTCTTCAACGCATATTAATAACTGTGGTGATAAAATTAATAGCCCCTAGAATAGAAGACGCACCCGCTAAATG
AAGGGAAAAGATGGCTAAATCTACAGACGGGCCTGCGTGGGCAAGATTTCTTGCTAGAGGCGGATAAACAGTTCACCCCG
TACCAGCGCCTTTTTCTACCGCCGCAGAGGACACTAAAAGGATCAGTGATGGGGGAAGTAGTCAGAATCTCATGTTGTTG
AGTCGAGGGAATGCTATATCGGGGGCTCCAAGTATTAAAGGTAAAAGCCAGTTTCCGAATCCCCCGATGAATACAGGCAT
TACTAGAAAGAAGATTATTACAAATGCGTGTGCAGTAACGATAGTATTGTATAGTTGGTCTCTGCCCAGGAACGCTCCTG
GTTGTCTTAGCTCGATTCGAATTAGAAGTCTTATACCGGCTCCAACCATTCCTGCTCAGACGCCCAGAATAAAATATAAA
```

Fig. 53: Figure 2. *SangerContig* **FASTA** input file.

Inside the **FASTA** file (*Figure_2*; `Achl_ACHLO006-09.fa`), the strings starting with ">" before each read are the read names. There are two ways of grouping reads which are **"regular expression matching"** and **"CSV file matching"**, and following are instructions of how to prepare your **FASTA** input file.

(1) "regular expression matching" *SangerContig* inputs (FASTA)

For regular expression matching method, sangeranalyseR will group reads based on their contig name and read direction in their names automatically; therefore, users have to follow the read-naming regulations below:

Note:

- All reads in the same contig group must include the same contig name in their read names.
- Forward or reverse direction also has to be specified in their read names.

There are four parameters, `FASTA_File`, `contigName`, `REGEX_SuffixForward` and `REGEX_SuffixReverse`, that define the grouping rule to let sangeranalyseR automatically match correct reads in **FASTA** file and divide them into forward and reverse directions.

Note:

- `FASTA_File`: this is the path to **FASTA** file that contains all sequences of reads, and it can be either an absolute or relative path. We suggest users to include only target reads inside this **FASTA** file and do not include any other unrelated reads.
- `contigName`: this is a regular expression that matches read names that are going to be included in the *SangerContig* analysis. `grepl` function in R is used.
- `REGEX_SuffixForward`: this is a regular expression that matches all read names in forward direction. `grepl` function in R is used.
- `REGEX_SuffixReverse`: this is a regular expression that matches all read names in reverse direction. `grepl` function in R is used.

If you don't know what regular expression is, don't panic - it's just a way of recognising text. Please refer to *What is a regular expression?* for more details. Here is an example of how it works in sangeranalseR:

So how sangeranalyseR works is that it first matches the `contigName` to exclude unrelated files and then separate the forward and reverse reads by matching `REGEX_SuffixForward` and `REGEX_SuffixReverse`. Therefore, it is important to make sure that all target reads in the **FASTA** file share the same `contigName` and carefully select your `REGEX_SuffixForward` and `REGEX_SuffixReverse`. The bad file-naming and wrong regex matching might accidentally include reverse reads into the forward read list or vice versa, which will make the program generate wrong results. Therefore, it is important to have a consistent naming strategy. So, how should we systematically name the reads? We suggest users to follow the file-naming regulation in *Figure_3*.

```
[Consensus Read Name] + _ + [index] + _ + [F,R]
```

Fig. 54: Figure 3. Suggested read naming regulation in **FASTA** file - *SangerContig*.

As you can see, the first part of the regulation is a consensus read name (or contig name), which narrows down the scope of reads to those we are going to examine. The second part of the regulation is an index. Since there might be more than one read that is in the forward or reverse direction, we recommend you to number your reads in the same contig group. The last part is a direction which is either 'F' (forward) or 'R' (reverse).

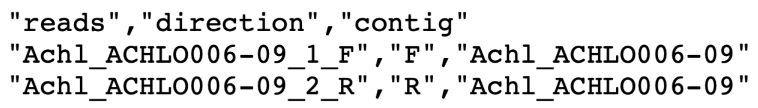
To make it more specific, let's go back to the true example. In *Figure_2*, there are two reads in the FASTA file (`fasta_FN`). First, we set `contigName` to `"Achl_ACHLO006-09"` to confirm that two of them, `Achl_ACHLO006-09_1_F` and `Achl_ACHLO006-09_2_R`, contain our target `contigName` and should be included. Then, we set `REGEX_SuffixForward` to `"_[0-9]*_F$"` and `REGEX_SuffixReverse` to `"_[0-9]*_R$"` to let sangeranalyseR match and group forward and reverse reads automatically. By the regular expression rule, `Achl_ACHLO006-09_1_F` and `Achl_ACHLO006-09_2_R` will be categorized into "forward read list" and "reverse read list" respectively. The reason why we strongly recommend you to follow this file-naming regulation is that by doing so, you can directly adopt the example regular expression matching values, `"_[0-9]*_F$"` and `"_[0-9]*_R$"`, to group reads and reduce chances of error.

After understanding how parameters work, please refer to *Creating SangerContig instance from FASTA* below to see how sangeranalseR creates 'Achl_ACHLO006-09' *SangerContig* instance.

(2) "CSV file matching" *SangerContig* inputs (FASTA)

No doubt that read names in the original **FASTA** file do not follow the naming regulation, and you do not want to change the original **FASTA** file; thus, we provide a second grouping approach, CSV file matching method. sangeranalyseR will group reads in the **FASTA** file based on the information in a CSV file automatically, and users do not need to alter the read names in the **FASTA** file; therefore, users have to follow the regulations below:

Note: Here is an `example CSV file` (*Figure_4*)



```
"reads","direction","contig"
"Achl_ACHLO006-09_1_F","F","Achl_ACHLO006-09"
"Achl_ACHLO006-09_2_R","R","Achl_ACHLO006-09"
```

Fig. 55: Figure 4. Example CSV file for *SangerContig* instance creation.

- There must be three columns, "**reads**", "**direction**", and "**contig**", in the CSV file.
- The "**reads**" column stores the read names in the **FASTA** file that are going to be included in the analysis.
- The "**direction**" column stores the direction of the reads. It must be "F" (forward) or "R" (reverse).
- The "**contig**" column stores the contig name that each read blongs. Reads in the same contig have to have the same contig name, and they will be grouped into the same *SangerContig* instance.

There are three parameters, `FASTA_File`, `contigName`, and `CSV_NamesConversion`,that define the grouping rule to help sangeranalseR to automatically match correct reads in a **FASTA** file and divide them into forward and reverse directions.

Note:

- `FASTA_File`: this is the path to **FASTA** file that contains all sequences of reads, and it can be either an absolute or relative path. We suggest users to include only target reads inside this **FASTA** file and do not include any other unrelated reads.
- `contigName`: this is a regular expression that matches read names that are going to be included in the *SangerContig* analysis. `grepl` function in R is used.
- `CSV_NamesConversion`: this is the path to the CSV file. It can be either an absolute or relative path.

The main difference between “CSV file matching” and “regular expression matching” is where the grouping rule is written. For “regular expression matching”, rules are writtein in read names, and thus more naming requirements are required. In contrast, rules of “CSV file matching” are written in an additional CSV file so it is more flexible on naming reads.

So how sangeranalyseR works is that it first reads in the CSV file (with *“reads”*, *“direction”*, and *“contig”* columns), filter out rows whose *“contig”* is not the value of `contigName` parameter, find the read names in the **FASTA** file listed in *“reads”*, and assign directions to them based on *“direction”*.

To make it more specific, let’s go back to the true example. First, we prepare a `CSV file` (`CSV_NamesConversion`) and a `FASTA file` (`FASTA_File`). In the CSV file, both rows have the contig name `"Achl_ACHLO006-09"`, which is what we need to assign to the `contigName` parameter. sangeranalyseR then checks and matches *“reads”* of these two rows, `"Achl_ACHLO006-09_1_F"` and `"Achl_ACHLO006-09_2_R"`. Last, these two reads are assigned into “forward read list” and “reverse read list” respectively by the *“direction”* column.

After understanding how parameters work, please refer to *Creating SangerContig instance from FASTA* below to see how sangeranalseR creates ‘Achl_ACHLO006-09’ *SangerContig* instance.

7.8.2 Creating *SangerContig* instance from FASTA

After preparing the input directory, we can create a *SangerContig* instance by running `SangerContig` constructor function or `new` method. The constructor function is a wrapper for `new` method and it makes instance creation more intuitive. Their input parameters are same, and all of them have their default values. For more details about *SangerContig* inputs and slots definition, please refer to sangeranalyseR reference manual. We will explain two *SangerContig* instance creation methods, “regular expression matching” and “CSV file matching”.

(1) “regular expression matching” *SangerContig* creation (FASTA)

The consturctor function and `new` method below contain four parameters, `FASTA_File`, `contigName`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that we mentioned in the previous section. In contrast to **AB1** input method, it does not include quality trimming and chromatogram visualization parameters. Run the following code and create `my_sangerContigFa` instance.

```
# using `constructor` function to create SangerRead instance
my_sangerContigFa <- SangerContig(inputSource          = "FASTA",
                                  processMethod        = "REGEX",
                                  FASTA_File           = fastaFN,
                                  contigName           =  "Achl_ACHLO006-09",
                                  REGEX_SuffixForward  = "_[0-9]*_F$",
                                  REGEX_SuffixReverse  = "_[0-9]*_R$",
                                  refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                  minReadsNum          = 2,
                                  minReadLength        = 20,
                                  minFractionCall      = 0.5,
                                  maxFractionLost      = 0.5,
                                  geneticCode          = GENETIC_CODE,
                                  acceptStopCodons     = TRUE,
                                  readingFrame         = 1,
```

(continues on next page)

(continued from previous page)

```
                                    processorsNum         = 1)

# using `new` method to create SangerRead instance
my_sangerContigFa <- new("SangerContig",
                         inputSource            = "FASTA",
                         processMethod          = "REGEX",
                         FASTA_File             = fastaFN,
                         contigName             = "Achl_ACHLO006-09",
                         REGEX_SuffixForward    = "_[0-9]*_F$",
                         REGEX_SuffixReverse    = "_[0-9]*_R$",
                         refAminoAcidSeq        =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                         minReadsNum            = 2,
                         minReadLength          = 20,
                         minFractionCall        = 0.5,
                         maxFractionLost        = 0.5,
                         geneticCode            = GENETIC_CODE,
                         acceptStopCodons       = TRUE,
                         readingFrame           = 1,
                         processorsNum          = 1)
```

In this example, `contigName` is set to `Achl_ACHLO006-09`, so `Achl_ACHLO006-09_1_F` and `Achl_ACHLO006-09_2_R` are matched and selected. Moreover, by regular expression pattern matching, `Achl_ACHLO006-09_1_F` is categorized into the forward list, and `Achl_ACHLO006-09_2_R` is categorized into the reverse read. Both reads are aligned into a contig, `my_sangerContigFa`, and it will be used as the input for the following functions.

Inside the R shell, you can run `my_sangerContigFa` to get basic information of the instance or run `my_sangerContigFa@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerContigFa` is successfully created.

Here is the output of `my_sangerContigFa`:

```
SangerContig S4 instance
         Input Source :  FASTA
         Process Method :  REGEX
      Fasta File Name :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerContig/Achl_ACHLO006-09.fa
   REGEX Suffix Forward :  _[0-9]*_F$
   REGEX Suffix Reverse :  _[0-9]*_R$
            Contig Name :  Achl_ACHLO006-09
         'minReadsNum' :  2
      'minReadLength' :  20
      'minFractionCall' :  0.5
      'maxFractionLost' :  0.5
   'acceptStopCodons' :  TRUE
         'readingFrame' :  1
      Contig Sequence : ␣
↪TTATATTTTATTCTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCTGGGCAGA
Forward reads in the contig >>  1
Reverse reads in the contig >>  1
SUCCESS [2021-13-07 11:52:40] 'Achl_ACHLO006-09' is successfully created!
```

Here is the output of `my_sangerContigFa@objectResults@readResultTable`:

```
            readName creationResult errorType errorMessage inputSource    direction
1 Achl_ACHLO006-09_1_F           TRUE      None         None       FASTA Forward Read
2 Achl_ACHLO006-09_2_R           TRUE      None         None       FASTA Reverse Read
```

(2) "CSV file matching" *SangerContig* creation (FASTA)

The consturctor function and `new` method below contain three parameters, `FASTA_File`, `contigName`, and `CSV_NamesConversion`, that we mentioned in the previous section. Run the following code and create `my_sangerContigFa` instance.

```
csv_namesConversion <- file.path(rawDataDir, "fasta", "SangerContig", "names_
↪conversion_1.csv")

# using `constructor` function to create SangerRead instance
my_sangerContigFa <- SangerContig(inputSource           = "FASTA",
                                  processMethod         = "CSV",
                                  FASTA_File            = fastaFN,
                                  contigName            = "Achl_ACHLO006-09",
                                  CSV_NamesConversion   = csv_namesConversion,
                                  refAminoAcidSeq       =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                  minReadsNum           = 2,
                                  minReadLength         = 20,
                                  minFractionCall       = 0.5,
                                  maxFractionLost       = 0.5,
                                  geneticCode           = GENETIC_CODE,
                                  acceptStopCodons      = TRUE,
                                  readingFrame          = 1,
                                  processorsNum         = 1)

# using `new` method to create SangerRead instance
my_sangerContigFa <- new("SangerContig",
                         inputSource           = "FASTA",
                         processMethod         = "CSV",
                         FASTA_File            = fastaFN,
                         contigName            = "Achl_ACHLO006-09",
                         CSV_NamesConversion   = csv_namesConversion,
                         refAminoAcidSeq       =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                         minReadsNum           = 2,
                         minReadLength         = 20,
                         minFractionCall       = 0.5,
                         maxFractionLost       = 0.5,
                         geneticCode           = GENETIC_CODE,
                         acceptStopCodons      = TRUE,
                         readingFrame          = 1,
                         processorsNum         = 1)
```

First, you need to load the CSV file into the R environment. If you are still don't know how to prepare it, please check *(2) "CSV file matching" SangerContig inputs (FASTA)*. Then, it will follow rules in the CSV file and create `my_sangerContigFa`. After it's created, inside the R shell, you can run `my_sangerContigFa` to get basic information of the instance or run `my_sangerContigFa@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerContigFa` is successfully created.

Here is the output of `my_sangerContigFa`:

```
SangerContig S4 instance
         Input Source :  FASTA
         Process Method :  CSV
      Fasta File Name :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerContig/Achl_ACHLO006-09.fa
   CSV Names Conversion :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerContig/names_conversion_1.csv
            Contig Name :  Achl_ACHLO006-09
         'minReadsNum' :  2
      'minReadLength' :  20
      'minFractionCall' :  0.5
      'maxFractionLost' :  0.5
   'acceptStopCodons' :  TRUE
         'readingFrame' :  1
      Contig Sequence : ␣
↪TTATATTTTATTCTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCTGGGCAGA
Forward reads in the contig >>  1
Reverse reads in the contig >>  1
SUCCESS [2021-13-07 12:01:57] 'Achl_ACHLO006-09' is successfully created!
```

Here is the output of `my_sangerContigFa@objectResults@readResultTable`:

```
            readName creationResult errorType errorMessage inputSource    direction
1 Achl_ACHLO006-09_1_F           TRUE      None         None       FASTA Forward Read
2 Achl_ACHLO006-09_2_R           TRUE      None         None       FASTA Reverse Read
```

7.8.3 Writing *SangerContig* FASTA files (FASTA)

Users can write the *SangerContig* instance, `my_sangerContigFa`, to **FASTA** files. There are four options for users to choose from in `selection` parameter.

- `reads_unalignment`: Writing reads into a single **FASTA** file (only trimmed without alignment).
- `reads_alignment`: Writing reads alignment and contig read to a single **FASTA** file.
- `contig`: Writing the contig to a single **FASTA** file.
- `all`: Writing reads, reads alignment, and the contig into three different files.

Below is the oneliner for writing out **FASTA** files. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

```
writeFasta(my_sangerContigFa,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA,
           selection         = "all")
```

Users can download the output FASTA file of this example through the following three links:

(1) `Achl_ACHLO006-09_reads_unalignment.fa`

(2) `Achl_ACHLO006-09_reads_alignment.fa`

(3) `Achl_ACHLO006-09_contig.fa`

7.8.4 Generating *SangerContig* report (FASTA)

Last but not least, users can save *SangerContig* instance, `my_sangerContigFa`, into a report after the analysis. The report will be generated in **HTML** by knitting **Rmd** files.

Users can set `includeSangerRead` parameter to decide to which level the *SangerContig* report will go. Moreover, after the reports are generated, users can easily navigate through reports in different levels within the **HTML** file.

One thing to pay attention to is that if users have many reads, it will take quite a long time to write out all reports. If users only want to generate the contig result, remember to set `includeSangerRead` to `FALSE` in order to save time.

```
generateReport(my_sangerContigFa,
               outputDir          = tempdir(),
               includeSangerRead  = TRUE)
```

Here is the generated SangerContig html report of this example (FASTA). Users can access to '*Basic Information*', '*SangerContig Input Parameters*', '*Contig Sequence*' and '*Contig Results*' sections inside it. Furthermore, users can also navigate through html reports of all forward and reverse *SangerRead* in this *SangerContig* report.

7.8.5 Code summary (*SangerContig*, FASTA)

1. Preparing *SangerContig* FASTA input

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
fastaFN <- file.path(rawDataDir, "fasta", "SangerContig", "Achl_ACHLO006-09.fa")
```

2. Creating *SangerContig* instance from FASTA

```
# using `constructor` function to create SangerRead instance
my_sangerContigFa <- SangerContig(inputSource          = "FASTA",
                                  processMethod        = "REGEX",
                                  FASTA_File           = fastaFN,
                                  contigName           =  "Achl_ACHLO006-09",
                                  REGEX_SuffixForward  = "_[0-9]*_F$",
                                  REGEX_SuffixReverse  = "_[0-9]*_R$",
                                  refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪")
```

(continues on next page)

(continued from previous page)

```
# using `new` method to create SangerRead instance
my_sangerContigFa <- new("SangerContig",
                         inputSource           = "FASTA",
                         processMethod         = "REGEX",
                         FASTA_File            = fastaFN,
                         contigName            = "Achl_ACHLO006-09",
                         REGEX_SuffixForward   = "_[0-9]*_F$",
                         REGEX_SuffixReverse   = "_[0-9]*_R$",
                         refAminoAcidSeq       =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

Following is the R shell output that you will get.

3. Writing *SangerContig* FASTA files (FASTA)

```
writeFasta(my_sangerContigFa)
```

Following is the R shell output that you will get.

And you will get three FASTA files:

(1) `Achl_ACHLO006-09_reads_unalignment.fa`

(2) `Achl_ACHLO006-09_reads_alignment.fa`

(3) `Achl_ACHLO006-09_contig.fa`

4. Generating *SangerContig* report (FASTA)

```
generateReport(my_sangerContigFa)
```

You can check the html report of this SangerContig example (FASTA).

7.9 Advanced User Guide - *SangerAlignment* (FASTA)

SangerAlignment is in the toppest level of sangeranalyseR (*Figure_1*), and each **SangerAlignment** instance corresponds to an alignment of contigs in a Sanger sequencing experiment. Among its slots, there is a *SangerContig* list which will be aligned into a consensus contig. Users can access to each *SangerContig* and *SangerRead* inside a *SangerAlignment* instance.

In this section, we are going to go through details about a reproducible *SangerAlignment* analysis example with the **FASTA** file input in sangeranalyseR. By running the following example codes, you will get an end-to-end *SangerAlignment* analysis result.

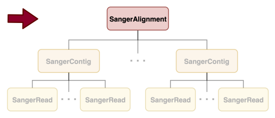


Fig. 56: Figure 1. Classes hierarchy in sangeranalyseR, *SangerAlignment* level.

7.9.1 Preparing *SangerAlignment* FASTA input

In *Advanced User Guide - SangerAlignment (AB1)*, we demonstrated how to use **AB1** input files to create *SangerAlignment* instance. Here, we explain another input format - the **FASTA** input. Before starting the analysis, users need to prepare one **FASTA** file, which must end with **.fa** or **.fasta**, containing sequences of all reads. In this example, the **FASTA** file is in the sangeranalyseR package, and you can simply get its path by running the following codes:

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
fastaFN <- file.path(rawDataDir, "fasta", "SangerAlignment", "Sanger_all_reads.fa")
```

The value of `fastaFN` is where the **FASTA** file is placed. If your operating system is macOS, then its value should look like this:

And we showed the reads in `fastaFN` in Figure_2 (`example FASTA file`):

Inside the **FASTA** file (*Figure_2*; `Sanger_all_reads.fa`), the strings starting with ">" before each read are the read names. There are two ways of grouping reads which are **"regular expression matching"** and **"CSV file matching"**, and following are instructions of how to prepare your **FASTA** input file.

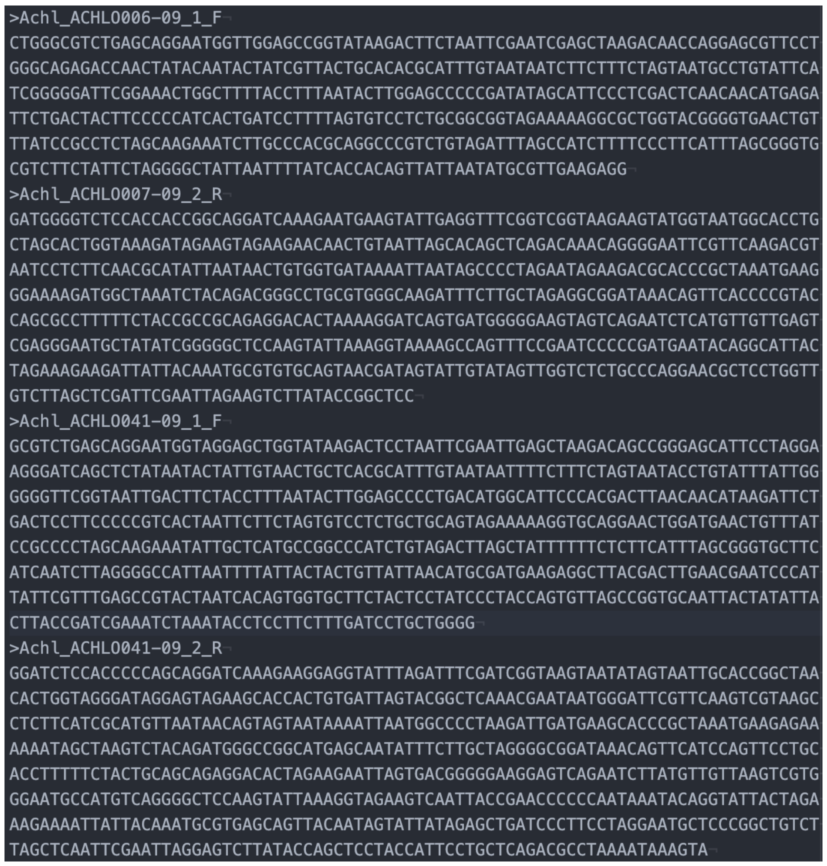
(1) "regular expression matching" *SangerAlignment* inputs (FASTA)

For regular expression matching method, sangeranalyseR will group reads based on their contig name and read direction in their read names automatically; therefore, users have to follow the read-naming regulations below:

Note:

- All reads in the same contig group must include the same contig name in their read names.
- Forward or reverse direction also has to be specified in their read names.

There are three parameters, `FASTA_File`, `REGEX_SuffixForward` and `REGEX_SuffixReverse`, that define the grouping rule to let sangeranalyseR automatically match correct reads in **FASTA** file and divide them into forward and reverse directions.



```
>Achl_ACHLO006-09_1_F
CTGGGCGTCTGAGCAGGAATGGTTGGAGCCGGTATAAGACTTCTAATTCGAATCGAGCTAAGACAACCAGGAGCGTTCCT
GGGCAGAGACCAACTATACAATACTATCGTTACTGCACACGCATTTGTAATAATCTTCTTTCTAGTAATGCCTGTATTCA
TCGGGGGATTCGGAAACTGGCTTTTACCTTTAATACTTGGAGCCCCCGATATAGCATTCCCTCGACTCAACAACATGAGA
TTCTGACTACTTCCCCCATCACTGATCCTTTTAGTGTCCTCTGCGGCGGTAGAAAAAGGCGCTGGTACGGGGTGAACTGT
TTATCCGCCTCTAGCAAGAAATCTTGCCCACGCAGGCCCGTCTGTAGATTTAGCCATCTTTTCCCTTCATTTAGCGGGTG
CGTCTTCTATTCTAGGGGCTATTAATTTTATCACCACAGTTATTAATATGCGTTGAAGAGG
>Achl_ACHLO007-09_2_R
GATGGGGTCTCCACCACCGGCAGGATCAAAGAATGAAGTATTGAGGTTTCGGTCGGTAAGAAGTATGGTAATGGCACCTG
CTAGCACTGGTAAAGATAGAAGTAGAAGAACAACTGTAATTAGCACAGCTCAGACAAACAGGGGAATTCGTTCAAGACGT
AATCCTCTTCAACGCATATTAATAACTGTGGTGATAAAATTAATAGCCCCTAGAATAGAAGACGCACCCGCTAAATGAAG
GGAAAAGATGGCTAAATCTACAGACGGGCCTGCGTGGGCAAGATTTCTTGCTAGAGGCGGATAAACAGTTCACCCCGTAC
CAGCGCCTTTTTCTACCGCCGCAGAGGACACTAAAAGGATCAGTGATGGGGGAAGTAGTCAGAATCTCATGTTGTTGAGT
CGAGGGAATGCTATATCGGGGGCTCCAAGTATTAAAGGTAAAAGCCAGTTTCCGAATCCCCCGATGAATACAGGCATTAC
TAGAAAGAAGATTATTACAAATGCGTGTGCAGTAACGATAGTATTGTATAGTTGGTCTCTGCCCAGGAACGCTCCTGGTT
GTCTTAGCTCGATTCGAATTAGAAGTCTTATACCGGCTCC
>Achl_ACHLO041-09_1_F
GCGTCTGAGCAGGAATGGTAGGAGCTGGTATAAGACTCCTAATTCGAATTGAGCTAAGACAGCCGGGAGCATTCCTAGGA
AGGGATCAGCTCTATAATACTATTGTAACTGCTCACGCATTTGTAATAATTTTCTTTCTAGTAATACCTGTATTTATTGG
GGGGTTCGGTAATTGACTTCTACCTTTAATACTTGGAGCCCCTGACATGGCATTCCCACGACTTAACAACATAAGATTCT
GACTCCTTCCCCCGTCACTAATTCTTCTAGTGTCCTCTGCTGCAGTAGAAAAAGGTGCAGGAACTGGATGAACTGTTTAT
CCGCCCCTAGCAAGAAATATTGCTCATGCCGGCCCATCTGTAGACTTAGCTATTTTTTCTCTTCATTTAGCGGGTGCTTC
ATCAATCTTAGGGGCCATTAATTTTATTACTACTGTTATTAACATGCGATGAAGAGGCTTACGACTTGAACGAATCCCAT
TATTCGTTTGAGCCGTACTAATCACAGTGGTGCTTCTACTCCTATCCCTACCAGTGTTAGCCGGTGCAATTACTATATTA
CTTACCGATCGAAATCTAAATACCTCCTTCTTTGATCCTGCTGGGG
>Achl_ACHLO041-09_2_R
GGATCTCCACCCCCAGCAGGATCAAAGAAGGAGGTATTTAGATTTCGATCGGTAAGTAATATAGTAATTGCACCGGCTAA
CACTGGTAGGGATAGGAGTAGAAGCACCACTGTGATTAGTACGGCTCAAACGAATAATGGGATTCGTTCAAGTCGTAAGC
CTCTTCATCGCATGTTAATAACAGTAGTAATAAAATTAATGGCCCCTAAGATTGATGAAGCACCCGCTAAATGAAGAGAA
AAAATAGCTAAGTCTACAGATGGGCCGGCATGAGCAATATTTCTTGCTAGGGGCGGATAAACAGTTCATCCAGTTCCTGC
ACCTTTTTCTACTGCAGCAGAGGACACTAGAAGAATTAGTGACGGGGGAAGGAGTCAGAATCTTATGTTGTTAAGTCGTG
GGAATGCCATGTCAGGGGCTCCAAGTATTAAAGGTAGAAGTCAATTACCGAACCCCCCAATAAATACAGGTATTACTAGA
AAGAAAATTATTACAAATGCGTGAGCAGTTACAATAGTATTATAGAGCTGATCCCTTCCTAGGAATGCTCCCGGCTGTCT
TAGCTCAATTCGAATTAGGAGTCTTATACCAGCTCCTACCATTCCTGCTCAGACGCCTAAAATAAAGTA
```

Fig. 57: Figure 2. *SangerAlignment* **FASTA** input file (4 out of 8 reads are showed).

Note:

- `FASTA_File`: this is the path to **FASTA** file that contains all sequences of reads, and it can be either an absolute or relative path. We suggest users to include only target reads inside this **FASTA** file and do not include any other unrelated reads.
- `REGEX_SuffixForward`: this is a regular expression that matches all read names in forward direction. `grepl` function in R is used.
- `REGEX_SuffixReverse`: this is a regular expression that matches all read names in reverse direction. `grepl` function in R is used.

If you don't know what regular expression is, don't panic - it's just a way of recognising text. Please refer to *What is a regular expression?* for more details. Here is an example of how it works in sangeranalseR:

So how sangeranalyseR works is that it first matches the forward and reverse reads by matching `REGEX_SuffixForward` and `REGEX_SuffixReverse`. Then, sangeranalyseR uses the `str_split` function to split and vectorize their read names into "contig name" and "direction-suffix" two parts. For those having the same "contig name" will be grouped into the same contig.

Therefore, it is important to have a consistent naming strategy. You need to make sure that reads in the **FASTA** file that are in the same contig group share the same contig name and carefully select your `REGEX_SuffixForward` and `REGEX_SuffixReverse`. The bad file-naming and wrong regex matching might accidentally include reverse reads into the forward read list or vice versa, which will make the program generate wrong results. So, how should we systematically name the reads? We suggest users to follow the file-naming regulation in *Figure_3*.

```
[Consensus Read Name] + _ + [index] + _ + [F,R]
```

Fig. 58: Figure 3. Suggested read naming regulation in **FASTA** file - *SangerAlignment*.

As you can see, the first part of the regulation is a consensus read name (or contig name), which helps sangeranalseR to identify which reads should be grouped into the same contig automatically. The second part of the regulation is an index; since there might be more than one read that is in the forward or reverse direction, we recommend you to number your reads in the same contig group. The Last part is a direction which is either 'F' (forward) or 'R' (reverse).

To make it more specific, let's go back to the true example. In *Figure_2*, there are eight reads in the FASTA file (`fasta_FN`; `Sanger_all_reads.fa`). First, we set `REGEX_SuffixForward` to `"_[0-9]*_F$"` and `REGEX_SuffixReverse` to `"_[0-9]*_R$"` to let sangeranalyseR match and group forward and reverse reads automatically. By the regular expression rule, `Achl_ACHLO006-09_1_F`, `Achl_ACHLO007-09_1_F`, `Achl_ACHLO040-09_1_F`, and `Achl_ACHLO041-09_1_F`, are categorized into forward reads, and `Achl_ACHLO006-09_1_R`, `Achl_ACHLO007-09_1_R`, `Achl_ACHLO040-09_1_R`, and `Achl_ACHLO041-09_1_R` are categorized into reverse reads. Then, `str_split` function is used to split each filename above into "contig name" and "direction-suffix". Four contig names are detected in this example which are `Achl_ACHLO006-09`, `Achl_ACHLO007-09`, `Achl_ACHLO040-09`, and `Achl_ACHLO041-09`. Last, a loop iterates through all contig names, and sangeranalseR creates each of them into a **SangerContig** instance. You can check *Advanced User Guide - SangerContig (FASTA)* to see how sangeranalyseR creates a *SangerContig* instance.

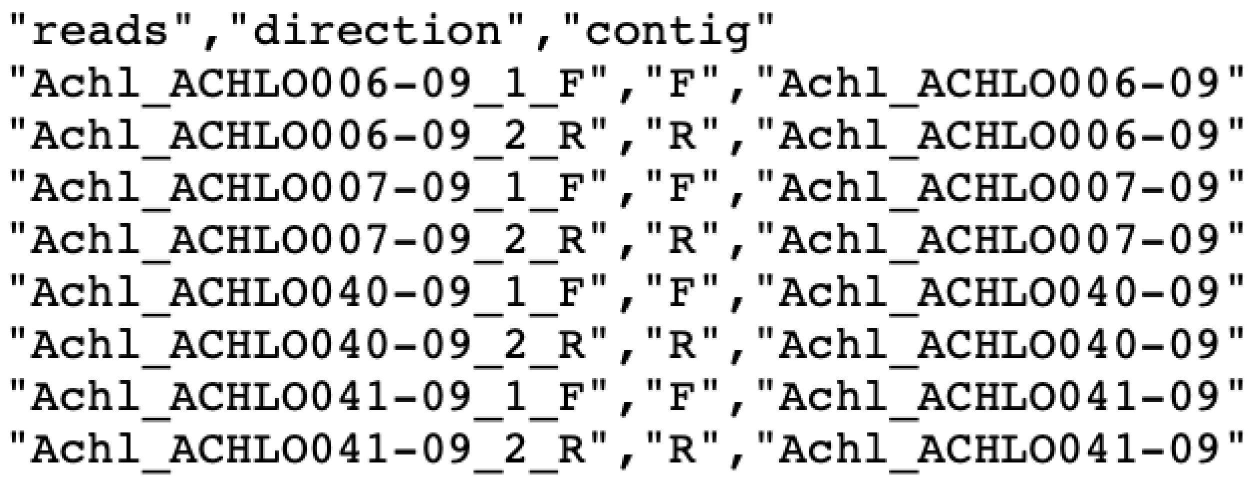
The reason why we strongly recommend you to follow this file-naming regulation is that by doing so, you can directly adopt the example regular expression matching values, `"_[0-9]*_F$"` and `"_[0-9]*_R$"`, to group reads and reduce chances of error. Everything mentioned above will be done automatically.

After understanding how parameters work, please refer to *Creating SangerAlignment instance from FASTA* below to see how sangeranalseR creates *SangerAlignment* instance.

(2) "CSV file matching" *SangerAlignment* inputs (FASTA)

No doubt that read names in the original **FASTA** file do not follow the naming regulation, and you do not want to change the original **FASTA** file; thus, we provide a second grouping approach, CSV file matching method. sangeranalyseR will group reads in the **FASTA** file based on the information in a CSV file automatically, and users do not need to alter the read names in the **FASTA** file. The note below shows the regulations:

Note: Here is an `example CSV file` (*Figure 4*)



```
"reads","direction","contig"
"Achl_ACHLO006-09_1_F","F","Achl_ACHLO006-09"
"Achl_ACHLO006-09_2_R","R","Achl_ACHLO006-09"
"Achl_ACHLO007-09_1_F","F","Achl_ACHLO007-09"
"Achl_ACHLO007-09_2_R","R","Achl_ACHLO007-09"
"Achl_ACHLO040-09_1_F","F","Achl_ACHLO040-09"
"Achl_ACHLO040-09_2_R","R","Achl_ACHLO040-09"
"Achl_ACHLO041-09_1_F","F","Achl_ACHLO041-09"
"Achl_ACHLO041-09_2_R","R","Achl_ACHLO041-09"
```

Fig. 59: Figure 4. Example CSV file for *SangerAlignment* instance creation.

- There must be three columns, "**reads**", "**direction**", and "**contig**", in the CSV file.
- The "**reads**" column stores the filename of **AB1** files that are going to be included in the analysis.
- The "**direction**" column stores the direction of the reads. It must be "F" (forward) or "R" (reverse).
- The "**contig**" column stores the contig name that each read blongs. Reads in the same contig have to have the same contig name, and they will be grouped into the same contig.

There are two parameters, `FASTA_File` and `CSV_NamesConversion`,that define the grouping rule to help sangeranalseR to automatically match correct reads in the **FASTA** file and divide them into forward and reverse directions.

Note:

- `FASTA_File`: this is the path to **FASTA** file that contains all sequences of reads, and it can be either an absolute or relative path. We suggest users to include only target reads inside this **FASTA** file and do not include any other unrelated reads.
- `CSV_NamesConversion`: this is the path to the CSV file. It can be either an absolute or relative path.

The main difference between "CSV file matching" and "regular expression matching" is where the grouping rule is written. For "regular expression matching", rules are writtein in read names, and thus more naming requirements are required. In contrast, rules of "CSV file matching" are written in an additional CSV file so it is more flexible on naming reads.

So how sangeranalyseR works is that it first reads in the CSV file (with *“reads”*, *“direction”*, and *“contig”* columns), find the read names in the **FASTA** file that are listed in *“reads”*, and assign directions to them based on *“direction”*.

To make it more specific, let’s go back to the true example. First, we prepare a `CSV file` (`CSV_NamesConversion`) and a `fasta file` (`FASTA_File`). In the CSV file, there are 8 rows and 4 distinct contig names. sangeranalyseR matches *“reads”* of these 8 rows to read names in the **FASTA** file. Then sangeranalyseR groups all matched reads, `Achl_ACHLO006-09_1_F`, `Achl_ACHLO007-09_1_F`, `Achl_ACHLO040-09_1_F`, `Achl_ACHLO041-09_1_F`, `Achl_ACHLO006-09_1_R`, `Achl_ACHLO007-09_1_R`, `Achl_ACHLO040-09_1_R`, and `Achl_ACHLO041-09_1_R`, into 4 distinct contigs which are `Achl_ACHLO006-09`, `Achl_ACHLO007-09`, `Achl_ACHLO040-09`, and `Achl_ACHLO041-09`, by the *“contig”* column. Last, the directions of reads in each contig are assigned by the *“direction”* column. Take `Achl_ACHLO041-09` contig as an example. Its “forward read list” will include `Achl_ACHLO041-09_1_F`, and its “reverse read list” will include `Achl_ACHLO041-09_1_R`.

After understanding how parameters work, please refer to *Creating SangerAlignment instance from FASTA* below to see how sangeranalseR creates *SangerAlignment* instance.

7.9.2 Creating *SangerAlignment* instance from FASTA

After preparing the input directory, we can create a *SangerAlignment* instance by running `SangerAlignment` constructor function or `new` method. The constructor function is a wrapper for `new` method and it makes instance creation more intuitive. Their input parameters are same, and all of them have their default values. For more details about *SangerAlignment* inputs and slots definition, please refer to sangeranalyseR reference manual. We will explain two *SangerAlignment* instance creation methods, “regular expression matching” and “CSV file matching”.

(1) “regular expression matching” *SangerAlignment* creation (FASTA)

The consturctor function and `new` method below contain three parameters, `FASTA_File`, `REGEX_SuffixForward`, and `REGEX_SuffixReverse`, that we mentioned in the previous section. In contrast to **AB1** input method, it does not include quality trimming and chromatogram visualization parameters. Run the following code and create `my_sangerAlignmentFa` instance.

```
# using `constructor` function to create SangerAlignment instance
my_sangerAlignmentFa <- SangerAlignment(inputSource          = "FASTA",
                                        processMethod        = "REGEX",
                                        FASTA_File           = fastaFN,
                                        REGEX_SuffixForward  = "_[0-9]*_F$",
                                        REGEX_SuffixReverse  = "_[0-9]*_R$",
                                        refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                                        minReadsNum           = 2,
                                        minReadLength         = 20,
                                        minFractionCall       = 0.5,
                                        maxFractionLost       = 0.5,
                                        geneticCode           = GENETIC_CODE,
                                        acceptStopCodons      = TRUE,
                                        readingFrame          = 1,
                                        processorsNum         = 1)
```

(continues on next page)

(continued from previous page)

```
my_sangerAlignmentFa <- new("SangerAlignment",
                            inputSource          = "FASTA",
                            processMethod        = "REGEX",
                            FASTA_File           = fastaFN,
                            REGEX_SuffixForward  = "_[0-9]*_F$",
                            REGEX_SuffixReverse  = "_[0-9]*_R$",
                            refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪",
                            minReadsNum           = 2,
                            minReadLength         = 20,
                            minFractionCall       = 0.5,
                            maxFractionLost       = 0.5,
                            geneticCode           = GENETIC_CODE,
                            acceptStopCodons      = TRUE,
                            readingFrame          = 1,
                            processorsNum         = 1)
```

In this example, 8 reads are detected and 4 distinct *SangerContig* instances are created. These *SangerContig* instances are stored in a "contig list" in `my_sangerAlignmentFa`, which will be used as the input for the following functions.

Inside the R shell, you can run `my_sangerAlignmentFa` to get basic information of the instance or run `my_sangerAlignmentFa@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerAlignmentFa` is successfully created.

Here is the output of `my_sangerAlignmentFa`:

```
SangerAlignment S4 instance
         Input Source :  FASTA
         Process Method :  REGEX
      Fasta File Name :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerAlignment/Sanger_all_reads.fa
   REGEX Suffix Forward :  _[0-9]*_F$
   REGEX Suffix Reverse :  _[0-9]*_R$
      Contigs Consensus : ␣
↪TTATAYTTTATTYTRGGCGTCTGAGCAGGAATGGTTGGAGCYGGTATAAGACTYCTAATTCGAATYGAGCTAAGACARCCRGGAGCRTTCCTRGGMAG
SUCCESS [2021-14-07 04:33:57] 'SangerAlignment' is successfully created!
```

Here is the output of `my_sangerAlignmentFa@objectResults@readResultTable`:

```
             readName creationResult errorType errorMessage inputSource    direction
1 Achl_ACHLO006-09_1_F           TRUE      None         None       FASTA Forward Read
2 Achl_ACHLO006-09_2_R           TRUE      None         None       FASTA Reverse Read
3 Achl_ACHLO007-09_1_F           TRUE      None         None       FASTA Forward Read
4 Achl_ACHLO007-09_2_R           TRUE      None         None       FASTA Reverse Read
5 Achl_ACHLO040-09_1_F           TRUE      None         None       FASTA Forward Read
6 Achl_ACHLO040-09_2_R           TRUE      None         None       FASTA Reverse Read
7 Achl_ACHLO041-09_1_F           TRUE      None         None       FASTA Forward Read
8 Achl_ACHLO041-09_2_R           TRUE      None         None       FASTA Reverse Read
```

(2) "CSV file matching" *SangerAlignment* creation (FASTA)

The consturctor function and `new` method below contain two parameters, `FASTA_File`, and `CSV_NamesConversion`, that we mentioned in the previous section. Run the following code and create `my_sangerAlignmentFa` instance.

```
csv_namesConversion <- file.path(rawDataDir, "fasta", "SangerAlignment", "names_
↪conversion.csv")

# using `constructor` function to create SangerAlignment instance
my_sangerAlignmentFa <- SangerAlignment(inputSource          = "FASTA",
                                        processMethod        = "CSV",
                                        FASTA_File           = fastaFN,
                                        CSV_NamesConversion  = csv_namesConversion,
                                        refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                                        minReadsNum           = 2,
                                        minReadLength         = 20,
                                        minFractionCall       = 0.5,
                                        maxFractionLost       = 0.5,
                                        geneticCode           = GENETIC_CODE,
                                        acceptStopCodons      = TRUE,
                                        readingFrame          = 1,
                                        processorsNum         = 1)


my_sangerAlignmentFa <- new("SangerAlignment",
                            inputSource          = "FASTA",
                            processMethod        = "CSV",
                            FASTA_File           = fastaFN,
                            CSV_NamesConversion  = csv_namesConversion,
                            refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNM
↪",
                            minReadsNum           = 2,
                            minReadLength         = 20,
                            minFractionCall       = 0.5,
                            maxFractionLost       = 0.5,
                            geneticCode           = GENETIC_CODE,
                            acceptStopCodons      = TRUE,
                            readingFrame          = 1,
                            processorsNum         = 1)
```

First, you need to load the CSV file into the R environment. If you are still don't know how to prepare it, please check *(2) "CSV file matching" SangerAlignment inputs (FASTA)*. Then, it will follow rules in the CSV file and create `my_sangerAlignmentFa`. After it's created, inside the R shell, you can run `my_sangerAlignmentFa` to get basic information of the instance or run `my_sangerAlignmentFa@objectResults@readResultTable` to check the creation result of every Sanger read after `my_sangerAlignmentFa` is successfully created.

Here is the output of `my_sangerAlignmentFa`:

```
SangerAlignment S4 instance
         Input Source :  FASTA
       Process Method :  CSV
      Fasta File Name :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerAlignment/Sanger_all_reads.fa
   CSV Names Conversion :  /Library/Frameworks/R.framework/Versions/4.0/Resources/
↪library/sangeranalyseR/extdata/fasta/SangerAlignment/names_conversion.csv
      Contigs Consensus : ␣
↪TTATAYTTTATTYTRGGCGTCTGAGCAGGAATGGTTGGAGCYGGTATAAGACTYCTAATTCGAATYGAGCTAAGACARCCRGGAGCRTTCCTRGGMAGR
SUCCESS [2021-14-07 04:38:44] 'SangerAlignment' is successfully created!
```

Here is the output of `my_sangerAlignmentFa@objectResults@readResultTable`:

```
            readName creationResult errorType errorMessage inputSource    direction
1 Achl_ACHLO006-09_1_F           TRUE      None         None       FASTA Forward Read
2 Achl_ACHLO006-09_2_R           TRUE      None         None       FASTA Reverse Read
3 Achl_ACHLO007-09_1_F           TRUE      None         None       FASTA Forward Read
4 Achl_ACHLO007-09_2_R           TRUE      None         None       FASTA Reverse Read
5 Achl_ACHLO040-09_1_F           TRUE      None         None       FASTA Forward Read
6 Achl_ACHLO040-09_2_R           TRUE      None         None       FASTA Reverse Read
7 Achl_ACHLO041-09_1_F           TRUE      None         None       FASTA Forward Read
8 Achl_ACHLO041-09_2_R           TRUE      None         None       FASTA Reverse Read
```

7.9.3 Writing *SangerAlignment* FASTA files (FASTA)

Users can write the *SangerAlignment* instance, `my_sangerAlignmentFa`, to **FASTA** files. There are four options for users to choose from in `selection` parameter.

- `reads_unalignment`: Writing reads into a single **FASTA** file (only trimmed without alignment).
- `reads_alignment`: Writing reads alignment and contig read to a single **FASTA** file.
- `contig`: Writing the contig to a single **FASTA** file.
- `all`: Writing reads, reads alignment, and the contig into three different files.

Below is the oneliner for writing out **FASTA** files. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

```
writeFasta(my_sangerAlignmentFa,
           outputDir         = tempdir(),
           compress          = FALSE,
           compression_level = NA,
           selection         = "all")
```

Users can download the output FASTA file of this example through the following three links:

(1) `Sanger_contigs_unalignment.fa`

(2) `Sanger_contigs_alignment.fa`

(3) `Sanger_all_trimmed_reads.fa`

7.9.4 Generating *SangerAlignment* report (FASTA)

Last but not least, users can save *SangerAlignment* instance, `my_sangerAlignmentFa`, into a report after the analysis. The report will be generated in **HTML** by knitting **Rmd** files.

Users can set `includeSangerContig` and `includeSangerRead` parameters to decide to which level the *SangerAlignment* report will go. Moreover, after the reports are generated, users can easily navigate through reports in different levels within the **HTML** file.

One thing to pay attention to is that if users have many reads, it will take quite a long time to write out all reports. If users only want to generate the contig result, remember to set `includeSangerRead` and `includeSangerContig` to `FALSE` in order to save time.

```
generateReport(my_sangerAlignmentFa,
               outputDir           = tempdir(),
               includeSangerRead   = FALSE,
               includeSangerContig = FALSE)
```

Here is the generated SangerAlignment html report of this example (FASTA). Users can access to '*Basic Information*', '*Contigs Consensus*', '*Contigs Alignment*', '*Contigs Tree*', and '*Contig Reports*' sections inside it. Furthermore, users can also navigate through html reports of all contigs and forward and reverse *SangerRead* in this *SangerAlignment* report.

7.9.5 Code summary (*SangerAlignment*, FASTA)

(1) Preparing *SangerAlignment* FASTA inputs

```
rawDataDir <- system.file("extdata", package = "sangeranalyseR")
fastaFN <- file.path(rawDataDir, "fasta", "SangerAlignment", "Sanger_all_reads.fa")
```

(2) Creating *SangerAlignment* instance from FASTA

(2.1) "Regular Expression Method" *SangerAlignment* creation (FASTA)

```
# using `constructor` function to create SangerAlignment instance
my_sangerAlignmentFa <- SangerAlignment(inputSource          = "FASTA",
                                        processMethod        = "REGEX",
                                        FASTA_File           = fastaFN,
                                        REGEX_SuffixForward  = "_[0-9]*_F$",
                                        REGEX_SuffixReverse  = "_[0-9]*_R$",
                                        refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")


my_sangerAlignmentFa <- new("SangerAlignment",
                            inputSource          = "FASTA",
                            processMethod        = "REGEX",
                            FASTA_File           = fastaFN,
                            REGEX_SuffixForward  = "_[0-9]*_F$",
                            REGEX_SuffixReverse  = "_[0-9]*_R$",
                            refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

(continues on next page)

(continued from previous page)

```
```

Following is the R shell output that you will get.

(2.2) "CSV file matching" *SangerAlignment* creation (FASTA)

```
csv_namesConversion <- file.path(rawDataDir, "fasta", "SangerAlignment", "names_
↪conversion.csv")

# using `constructor` function to create SangerAlignment instance
my_sangerAlignmentFa <- SangerAlignment(inputSource          = "FASTA",
                                        processMethod        = "CSV",
                                        FASTA_File           = fastaFN,
                                        CSV_NamesConversion  = csv_namesConversion,
                                        refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")


my_sangerAlignmentFa <- new("SangerAlignment",
                            inputSource          = "FASTA",
                            processMethod        = "CSV",
                            FASTA_File           = fastaFN,
                            CSV_NamesConversion  = csv_namesConversion,
`refAminoAcidSeq      =
↪"SRQWLFSTNHKDIGTLYFIFGAWAGMVGTSLSILIRAELGHPGALIGDDQIYNVIVTAHAFIMIFFMVMPIMIGGFGNWLVPLMLGAPDMAFPRMNNN
↪")
```

Following is the R shell output that you will get.

(3) Writing *SangerAlignment* FASTA files (FASTA)

```
writeFasta(my_sangerAlignmentFa)
```

Following is the R shell output that you will get.

You will get three FASTA files:

(1) `Sanger_contigs_unalignment.fa`

(2) `Sanger_contigs_alignment.fa`

(3) `Sanger_all_trimmed_reads.fa`

(4) Generating *SangerAlignment* report (FASTA)

```
generateReport(my_sangerAlignmentFa)
```

You can check the html report of this SangerAlignment example (FASTA).

7.10 Q & A . . .

7.10.1 What is a regular expression?

A regular expression (sometimes shortened as regex or regexp) is a sequence of characters that define a sequence pattern matching rule, mainly used for searching and replacing. It is used in all programming languages like C++, Python, Javascript, and in our case, R.

7.10.2 How to deal with secondary peaks

7.10.3 How to work with FASTA files for input

7.11 User Manual (functions)

Following are input parameters for **SangerRead**, **SangerContig**, and **SangerAlignment** constructors. For more detials about other functions, please refer to the sangeranalyseR user manual.

7.11.1 SangerRead Constructor Parameters

```
SangerRead(inputSource = "ABIF",
           readFeature = "",
           readFileName = "",
           fastaReadName = "",
           geneticCode = GENETIC_CODE,
           TrimmingMethod = "M1",
           M1TrimmingCutoff = 0.0001,
           M2CutoffQualityScore = NULL,
           M2SlidingWindowSize = NULL,
           baseNumPerRow = 100,
           heightPerRow = 200,
           signalRatioCutoff = 0.33,
           showTrimmed = TRUE)
```

- **inputSource**: The input source of the raw file. It must be *“ABIF”* or *“FASTA”*. The default value is *“ABIF”*.
- **readFeature**: The direction of the Sanger read. The value must be *“Forward Read”* or *“Reverse Read”*.
- **readFileName**: The absolute filename of the target ABIF or FASTA file.
- **fastaReadName**: If *“inputSource”* is *“FASTA”*, then this value has to be the name of the read inside the FASTA file; if *“inputSource”* is *“ABIF”*, then this value is *“NULL”* by default.
- **geneticCode**: Named character vector in the same format as *“GENETIC_CODE”* (the default), which represents the standard genetic code. This is the code with which the function will attempt to translate your DNA sequences. You can get an appropriate vector with the *“getGeneticCode()”* function. The default is the standard code.
- **TrimmingMethod**: The read trimming method for the *SangerRead*. The value must be *“M1”* (the default) or *“M2”*, which represents *“method 1”* or *“method 2”* respectively. M1 is the modified Mott’s trimming algorithm that can also be found in Phred/Phrap and Biopython. M2 is like trimmomatic’s sliding window method.
- **M1TrimmingCutoff**: The cutoff for the trimming method 1. If *TrimmingMethod* is *“M1”*, then the default value is *“0.0001”*. Otherwise, the value must be *“NULL”*.
- **M2CutoffQualityScore**: The trimming cutoff quality score for the trimming method 2. If *TrimmingMethod* is *“M2”*, then the default value is *“20”*. Otherwise, the value must be *“NULL”*. This parameter works with *M2SlidingWindowSize*.
- **M2SlidingWindowSize**: The trimming sliding window size for the trimming method 2. If *TrimmingMethod* is *“M2”*, then the default value is *“10”*. Otherwise, the value must be *“NULL”*. This parameter works with *M2CutoffQualityScore*.
- **baseNumPerRow**: This parameter is related to chromatogram and defines maximum base pairs in each row. The default value is *“100”*.
- **heightPerRow**: This parameter is related to chromatogram and defines the height of each row in chromatogram. The default value is *“200”*.
- **signalRatioCutoff**: The ratio of the height of a secondary peak to a primary peak. Secondary peaks higher than this ratio are annotated. Those below the ratio are excluded. The default value is *“0.33”*. This parameter is related to chromatogram.
- **showTrimmed**: The logical value storing whether to show trimmed base pairs in chromatogram. The default value is *“TRUE”*.

7.11.2 SangerContig Constructor Parameters

```
SangerContig(inputSource            = "ABIF",
             fastaFileName          = "",
             namesConversionCSV     = NULL,
             parentDirectory        = "",
             contigName             = "",
             suffixForwardRegExp    = "_F.ab1",
             suffixReverseRegExp    = "_R.ab1",
             TrimmingMethod         = "M1",
             M1TrimmingCutoff       = 0.0001,
             M2CutoffQualityScore   = NULL,
             M2SlidingWindowSize    = NULL,
             baseNumPerRow          = 100,
```

(continues on next page)

(continued from previous page)

```
             heightPerRow           = 200,
             signalRatioCutoff      = 0.33,
             showTrimmed            = TRUE,
             refAminoAcidSeq        = "",
             minReadsNum            = 2,
             minReadLength          = 20,
             minFractionCall        = 0.5,
             maxFractionLost        = 0.5,
             geneticCode            = GENETIC_CODE,
             acceptStopCodons       = TRUE,
             readingFrame           = 1,
             processorsNum          = NULL)
```

- **inputSource**: The input source of the raw file. It must be *"ABIF"* or *"FASTA"*. The default value is *"ABIF"*.
- **fastaFileName**: If *"inputSource"* is *"FASTA"*, then this value has to be the name of the FASTA file; if *"inputSource"* is *"ABIF"*, then this value is *"NULL"* by default.
- **namesConversionCSV**: The absolute filename of CSV file that provides read names following the naming regulation. If *"inputSource"* is *"FASTA"*, then users need to prepare the csv file or make sure the original names inside FASTA file are valid; if *"inputSource"* is *"ABIF"*, then this value is *"NULL"* by default.
- **parentDirectory**: The parent directory of all of the reads contained in ABIF format you wish to analyse. In SangerContig, all reads must be in the first layer in this directory.
- **contigName**: The contig name of all the reads in *"parentDirectory"*.
- **suffixForwardRegExp**: The suffix of the filenames for forward reads in regular expression, i.e. reads that do not need to be reverse-complemented. For forward reads, it should be "_F.ab1".
- **suffixReverseRegExp**: The suffix of the filenames for reverse reads in regular expression, i.e. reads that need to be reverse-complemented. For revcerse reads, it should be "_R.ab1".
- **TrimmingMethod**: The read trimming method for the *SangerRead*. The value must be *"M1"* (the default) or *"M2"*, which represents *"method 1"* or *"method 2"* respectively. M1 is the modified Mott's trimming algorithm that can also be found in Phred/Phrap and Biopython. M2 is like trimmomatic's sliding window method.
- **M1TrimmingCutoff**: The cutoff for the trimming method 1. If *TrimmingMethod* is *"M1"*, then the default value is *"0.0001"*. Otherwise, the value must be *"NULL"*.
- **M2CutoffQualityScore**: The trimming cutoff quality score for the trimming method 2. If *TrimmingMethod* is *"M2"*, then the default value is *"20"*. Otherwise, the value must be *"NULL"*. This parameter works with *M2SlidingWindowSize*.
- **M2SlidingWindowSize**: The trimming sliding window size for the trimming method 2. If *TrimmingMethod* is *"M2"*, then the default value is *"10"*. Otherwise, the value must be *"NULL"*. This parameter works with *M2CutoffQualityScore*.
- **baseNumPerRow**: This parameter is related to chromatogram and defines maximum base pairs in each row. The default value is *"100"*.
- **heightPerRow**: This parameter is related to chromatogram and defines the height of each row in chromatogram. The default value is *"200"*.
- **signalRatioCutoff**: The ratio of the height of a secondary peak to a primary peak. Secondary peaks higher than this ratio are annotated. Those below the ratio are excluded. The default value is *"0.33"*. This parameter is related to chromatogram.
- **showTrimmed**: The logical value storing whether to show trimmed base pairs in chromatogram. The default value is *"TRUE"*.

- **refAminoAcidSeq**: An amino acid reference sequence supplied as a string or an AAString object. If your sequences are protein-coding DNA seuqences, and you want to have frameshifts automatically detected and corrected, supply a reference amino acid sequence via this argument. If this argument is supplied, the sequences are then kept in frame for the alignment step. Fwd sequences are assumed to come from the sense (i.e. coding, or "+") strand. The default value is "".
- **minReadsNum**: The minimum number of reads required to make a consensus sequence, must be 2 or more. The default value is *"2"*.
- **minReadLength**: Reads shorter than this will not be included in the readset. The default *"20"* means that all reads with length of 20 or more will be included. Note that this is the length of a read after it has been trimmed.
- **minFractionCall**: Minimum fraction of the sequences required to call a consensus sequence for SangerContig at any given position (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.75 implying that 3/4 of all reads must be present in order to call a consensus.
- **maxFractionLost**: Numeric giving the maximum fraction of sequence information that can be lost in the consensus sequence for SangerContig (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.5, implying that each consensus base can ignore at most 50 percent of the information at a given position.
- **geneticCode**: Named character vector in the same format as *"GENETIC_CODE"* (the default), which represents the standard genetic code. This is the code with which the function will attempt to translate your DNA sequences. You can get an appropriate vector with the getGeneticCode() function. The default is the standard code.
- **acceptStopCodons**: The logical value *"TRUE"* or *"FALSE"*. *"TRUE"* (the defualt): keep all reads, regardless of whether they have stop codons; *"FALSE"*: reject reads with stop codons. If *"FALSE"* is selected, then the number of stop codons is calculated after attempting to correct frameshift mutations (if applicable).
- **readingFrame**: *"1"*, *"2"*, or *"3"*. Only used if *"accept.stop.codons == FALSE"*. This specifies the reading frame that is used to determine stop codons. If you use a *"refAminoAcidSeq"*, then the frame should always be *"1"*, since all reads will be shifted to frame 1 during frameshift correction. Otherwise, you should select the appropriate reading frame.
- **processorsNum**: The number of processors to use, or NULL (the default) for all available processors.

7.11.3 SangerAlignment Constructor Parameters

```
SangerAlignment(inputSource              = "ABIF",
                fastaFileName            = "",
                namesConversionCSV       = NULL,
                parentDirectory          = "",
                suffixForwardRegExp      = "_F.ab1",
                suffixReverseRegExp      = "_R.ab1",
                TrimmingMethod           = "M1",
                M1TrimmingCutoff         = 0.0001,
                M2CutoffQualityScore     = NULL,
                M2SlidingWindowSize      = NULL,
                baseNumPerRow            = 100,
                heightPerRow             = 200,
                signalRatioCutoff        = 0.33,
                showTrimmed              = TRUE,
                refAminoAcidSeq          = "",
```

(continues on next page)

(continued from previous page)

```
                minReadsNum            = 2,
                minReadLength          = 20,
                minFractionCall        = 0.5,
                maxFractionLost        = 0.5,
                geneticCode            = GENETIC_CODE,
                acceptStopCodons       = TRUE,
                readingFrame           = 1,
                minFractionCallSA      = 0.5,
                maxFractionLostSA      = 0.5,
                processorsNum          = NULL)
```

- **inputSource**: The input source of the raw file. It must be *"ABIF"* or *"FASTA"*. The default value is *"ABIF"*.
- **fastaFileName**: If *"inputSource"* is *"FASTA"*, then this value has to be the name of the FASTA file; if *"inputSource"* is *"ABIF"*, then this value is *"NULL"* by default.
- **namesConversionCSV**: The file path to the CSV file that provides read names that follow the naming regulation. If *"inputSource"* is *"FASTA"*, then users need to prepare the csv file or make sure the original names inside FASTA file are valid; if *"inputSource"* is *"ABIF"*, then this value is *"NULL"* by default.
- **parentDirectory**: The parent directory of all of the reads contained in ABIF format you wish to analyse. In SangerContig, all reads must be in the first layer in this directory.
- **suffixForwardRegExp**: The suffix of the filenames for forward reads in regular expression, i.e. reads that do not need to be reverse-complemented. For forward reads, it should be "_F.ab1".
- **suffixReverseRegExp**: The suffix of the filenames for reverse reads in regular expression, i.e. reads that need to be reverse-complemented. For revcerse reads, it should be "_R.ab1".
- **TrimmingMethod**: The read trimming method for the *SangerRead*. The value must be *"M1"* (the default) or *"M2"*, which represents *"method 1"* or *"method 2"* respectively. M1 is the modified Mott's trimming algorithm that can also be found in Phred/Phrap and Biopython. M2 is like trimmomatic's sliding window method.
- **M1TrimmingCutoff**: The cutoff for the trimming method 1. If *TrimmingMethod* is *"M1"*, then the default value is *"0.0001"*. Otherwise, the value must be *"NULL"*.
- **M2CutoffQualityScore**: The trimming cutoff quality score for the trimming method 2. If *TrimmingMethod* is *"M2"*, then the default value is *"20"*. Otherwise, the value must be *"NULL"*. This parameter works with *M2SlidingWindowSize*.
- **M2SlidingWindowSize**: The trimming sliding window size for the trimming method 2. If *TrimmingMethod* is *"M2"*, then the default value is *"10"*. Otherwise, the value must be *"NULL"*. This parameter works with *M2CutoffQualityScore*.
- **baseNumPerRow**: This parameter is related to chromatogram and defines maximum base pairs in each row. The default value is *"100"*.
- **heightPerRow**: This parameter is related to chromatogram and defines the height of each row in chromatogram. The default value is *"200"*.
- **signalRatioCutoff**: The ratio of the height of a secondary peak to a primary peak. Secondary peaks higher than this ratio are annotated. Those below the ratio are excluded. The default value is *"0.33"*. This parameter is related to chromatogram.
- **showTrimmed**: The logical value storing whether to show trimmed base pairs in chromatogram. The default value is *"TRUE"*.
- **refAminoAcidSeq**: An amino acid reference sequence supplied as a string or an AAString object. If your sequences are protein-coding DNA seuqences, and you want to have frameshifts automatically detected and corrected, supply a reference amino acid sequence via this argument. If this argument is supplied, the sequences

are then kept in frame for the alignment step. Fwd sequences are assumed to come from the sense (i.e. coding, or "+") strand. The default value is "".

- **minReadsNum**: The minimum number of reads required to make a consensus sequence, must be 2 or more. The default value is *"2"*.
- **minReadLength**: Reads shorter than this will not be included in the readset. The default *"20"* means that all reads with length of 20 or more will be included. Note that this is the length of a read after it has been trimmed.
- **minFractionCall**: Minimum fraction of the sequences required to call a consensus sequence for SangerContig at any given position (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.75 implying that 3/4 of all reads must be present in order to call a consensus.
- **maxFractionLost**: Numeric giving the maximum fraction of sequence information that can be lost in the consensus sequence for SangerContig (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.5, implying that each consensus base can ignore at most 50 percent of the information at a given position.
- **geneticCode**: Named character vector in the same format as *"GENETIC_CODE"* (the default), which represents the standard genetic code. This is the code with which the function will attempt to translate your DNA sequences. You can get an appropriate vector with the getGeneticCode() function. The default is the standard code.
- **acceptStopCodons**: The logical value *"TRUE"* or *"FALSE"*. *"TRUE"* (the defualt): keep all reads, regardless of whether they have stop codons; *"FALSE"*: reject reads with stop codons. If *"FALSE"* is selected, then the number of stop codons is calculated after attempting to correct frameshift mutations (if applicable).
- **readingFrame**: *"1"*, *"2"*, or *"3"*. Only used if *"accept.stop.codons == FALSE"*. This specifies the reading frame that is used to determine stop codons. If you use a *"refAminoAcidSeq"*, then the frame should always be *"1"*, since all reads will be shifted to frame 1 during frameshift correction. Otherwise, you should select the appropriate reading frame.
- **minFractionCallSA**: Minimum fraction of the sequences required to call a consensus sequence for SangerAlignment at any given position (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.75 implying that 3/4 of all reads must be present in order to call a consensus.
- **maxFractionLostSA**: Numeric giving the maximum fraction of sequence information that can be lost in the consensus sequence for SangerAlignment (see the ConsensusSequence() function from DECIPHER for more information). Defaults to 0.5, implying that each consensus base can ignore at most 50 percent of the information at a given position.
- **processorsNum**: The number of processors to use, or NULL (the default) for all available processors.

7.12 Frequently Asked Questions

7.12.1 Q: What is the difference between two different trimming methods?

A: In sangeranalyseR, we provide two trimming methods, *"M1"* (the default) or *"M2"*, which represents *"method 1"* or *"method 2"* respectively. M1 is the modified Mott's trimming algorithm that can also be found in Phred/Phrap and Biopython. M2 is like trimmomatic's sliding window method. If you want to set M1 as your trimming method, you need to assign **"TrimmingMethod"** to **"M1"** and **"M1TrimmingCutoff"** as the threshold that you want. Its default value is *"0.0001"*. In contrast, you can assign **"TrimmingMethod"** to **"M2"** if you want to set M2 as your trimming method. **"M2CutoffQualityScore"** and **"M2SlidingWindowSize"** are two parameters that control M2 trimming and their default values are *"20"* and *"10"* respectively.

7.13 Conclusion

sangeranalyseR provides a new approach to do Sanger sequencing data analysis in R. The main features include well-structured S4 classes, automated data analysis, interactive Shiny apps, exporting reads to FASTA and the generation thorough report. We hope it will be helpful for R users and the bioinformatics community!

7.14 License

7.15 Contact

Contact here:

For now, please just use the issue tracker on GitHub for all contacts. That will help us keep up to date with things.

7.16 Help

If you need any help, feel free to contact me <kuanhao.chao@gmail.com>

7.16.1 Inside help test