1 Why sangeranalyseR
2 Main features
3 What sangeranalyseR doesn’t do
4 User Manual
5 User support
6 Key contributors
7 Documentation

7.1 Installation
7.1.1 System requirements
7.1.2 Install from Bioconductor
7.1.3 Install the development version
7.1.4 Where to go from here?
7.2 Quick Start Guide
7.2.1 Super-Quick Start (3 lines of code)
7.2.2 Step 1: Prepare your input files
7.2.3 Step 2: Load and analyse your data
7.2.4 Step 3 (optional): Explore your data
7.2.5 Step 4: Output your aligned contigs
7.2.6 Step 5 (optional): Generate an interactive report
7.3 Beginners Guide
7.3.1 Step 1: Preparing your input files
7.3.2 Step 2: Loading and analysing your data
7.3.3 Step 3: Exploring your data with the Shiny app
7.3.4 Step 4: Outputting your aligned contigs
7.3.5 Step 5: Generating an interactive report
7.3.6 What’s next?
7.4 Advanced User Guide - SangerRead (AB1)
7.4.1 Preparing SangerRead AB1 input
7.4.2 Creating SangerRead instance from AB1
7.4.3 Visualizing SangerRead trimmed read
7.4.4 Updating SangerRead quality trimming parameters
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.12</td>
<td>Q: What is the difference between two different trimming methods?</td>
<td>64</td>
</tr>
<tr>
<td>7.13</td>
<td>Conclusion</td>
<td>64</td>
</tr>
<tr>
<td>7.14</td>
<td>License</td>
<td>64</td>
</tr>
<tr>
<td>7.15</td>
<td>Contact</td>
<td>65</td>
</tr>
<tr>
<td>7.16</td>
<td>Help</td>
<td>65</td>
</tr>
<tr>
<td>7.16.1</td>
<td>Inside help test</td>
<td>65</td>
</tr>
<tr>
<td>7.5</td>
<td>Advanced User Guide - SangerContig (ABI)</td>
<td>23</td>
</tr>
<tr>
<td>7.5.1</td>
<td>Preparing SangerContig ABI input</td>
<td>24</td>
</tr>
<tr>
<td>7.5.2</td>
<td>Creating SangerContig instance from ABI</td>
<td>25</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Updating SangerContig quality trimming parameters</td>
<td>26</td>
</tr>
<tr>
<td>7.5.4</td>
<td>Launching SangerContig Shiny app</td>
<td>26</td>
</tr>
<tr>
<td>7.5.5</td>
<td>Writing SangerContig FASTA files (ABI)</td>
<td>32</td>
</tr>
<tr>
<td>7.5.6</td>
<td>Generating SangerContig report (ABI)</td>
<td>35</td>
</tr>
<tr>
<td>7.6</td>
<td>Advanced User Guide - SangerAlignment (ABI)</td>
<td>35</td>
</tr>
<tr>
<td>7.6.1</td>
<td>Preparing SangerAlignment ABI input</td>
<td>36</td>
</tr>
<tr>
<td>7.6.2</td>
<td>Creating SangerAlignment instance from ABI</td>
<td>38</td>
</tr>
<tr>
<td>7.6.3</td>
<td>Updating SangerAlignment quality trimming parameters</td>
<td>38</td>
</tr>
<tr>
<td>7.6.4</td>
<td>Launching SangerAlignment Shiny app</td>
<td>39</td>
</tr>
<tr>
<td>7.6.5</td>
<td>Writing SangerAlignment FASTA files (ABI)</td>
<td>49</td>
</tr>
<tr>
<td>7.6.6</td>
<td>Generating SangerAlignment report (ABI)</td>
<td>49</td>
</tr>
<tr>
<td>7.7</td>
<td>Advanced User Guide - SangerRead (FASTA)</td>
<td>50</td>
</tr>
<tr>
<td>7.7.1</td>
<td>Preparing SangerRead FASTA input</td>
<td>50</td>
</tr>
<tr>
<td>7.7.2</td>
<td>Creating SangerRead instance from FASTA</td>
<td>50</td>
</tr>
<tr>
<td>7.7.3</td>
<td>Writing SangerRead FASTA files (FASTA)</td>
<td>51</td>
</tr>
<tr>
<td>7.7.4</td>
<td>Generating SangerRead report (FASTA)</td>
<td>51</td>
</tr>
<tr>
<td>7.8</td>
<td>Advanced User Guide - SangerContig (FASTA)</td>
<td>51</td>
</tr>
<tr>
<td>7.8.1</td>
<td>Preparing SangerContig FASTA input</td>
<td>52</td>
</tr>
<tr>
<td>7.8.2</td>
<td>Creating SangerContig instance from FASTA</td>
<td>53</td>
</tr>
<tr>
<td>7.8.3</td>
<td>Writing SangerContig FASTA files (FASTA)</td>
<td>54</td>
</tr>
<tr>
<td>7.8.4</td>
<td>Generating SangerContig report (FASTA)</td>
<td>54</td>
</tr>
<tr>
<td>7.9</td>
<td>Advanced User Guide - SangerAlignment (FASTA)</td>
<td>55</td>
</tr>
<tr>
<td>7.9.1</td>
<td>Preparing SangerAlignment FASTA input</td>
<td>55</td>
</tr>
<tr>
<td>7.9.2</td>
<td>Creating SangerAlignment instance from FASTA</td>
<td>57</td>
</tr>
<tr>
<td>7.9.3</td>
<td>Writing SangerAlignment FASTA files (FASTA)</td>
<td>57</td>
</tr>
<tr>
<td>7.9.4</td>
<td>Generating SangerAlignment report (FASTA)</td>
<td>58</td>
</tr>
<tr>
<td>7.10</td>
<td>How to</td>
<td>59</td>
</tr>
<tr>
<td>7.10.1</td>
<td>How to deal with secondary peaks</td>
<td>59</td>
</tr>
<tr>
<td>7.10.2</td>
<td>How to work with FASTA files for input</td>
<td>59</td>
</tr>
<tr>
<td>7.11</td>
<td>User Manual (functions)</td>
<td>59</td>
</tr>
<tr>
<td>7.11.1</td>
<td>SangerRead Constructor Parameters</td>
<td>59</td>
</tr>
<tr>
<td>7.11.2</td>
<td>SangerContig Constructor Parameters</td>
<td>60</td>
</tr>
<tr>
<td>7.11.3</td>
<td>SangerAlignment Constructor Parameters</td>
<td>62</td>
</tr>
</tbody>
</table>
sangeranalyseR is an R package that provides fast, flexible, and reproducible workflows for assembling your sanger sequencing 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.
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 sequences in 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.
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.
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!)
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:

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

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

BiocManager::install("sangeranalyseR")
```
7.1.3 Install the development version

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

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

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

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

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

```r
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.

```r
my_aligned_contigs <- SangerAlignment(parentDirectory = "./my_data/",
                                  suffixForwardRegExp = "_[0-9]+_F+",
                                  suffixReverseRegExp = "_[0-9]+_R +")

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

```r
my_aligned_contigs <- SangerAlignment(parentDirectory = "./my_data/",
                                   suffixForwardRegExp = "_[0-9]+_F+",
                                   suffixReverseRegExp = "_[0-9]+_R +")
```

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

```r
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

```r
writeFasta(my_aligned_contigs)
```

This will save your aligned contigs as a FASTA file.

7.2.6 Step 5 (optional): Generate an interactive report

```r
generateReport(my_aligned_contigs)
```

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

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 files. If you are starting with FASTA 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 parentDirectory 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.

![Figure 1](image)

When using the filenames to group the reads, you’ll need to specify three parameters: parentDirectory, suffixForwardRegExp, and suffixReverseRegExp:

- **parentDirectory**: 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 parentDirectory should be /path/to/tmp/. In your case, it should be the absolute path to the folder that contains your reads.
- **suffixForwardRegExp**: 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 useful way to do it is __[0-9]+_F_. 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’. 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’.
• suffixReverseRegExp: 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. 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 prepare an input .csv file with three columns:

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

### 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:

```r
my_aligned_contigs <- SangerAlignment(
  parentDirectory = "/path/to/tmp/",
  suffixForwardRegExp = "\_\([0-9]+\)\_F",
  suffixReverseRegExp = "\_\([0-9]+\)\_R"
)
```

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

```r
my_aligned_contigs <- SangerAlignment(
  parentDirectory = "/path/to/tmp/",
  namesConversionCSV = "/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)

![SangerAlignment Overview Page](image)

**Fig. 3:** Figure 2. *SangerAlignment* Shiny app user interface.

*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).

### 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 as a FASTA file. 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)
```
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 the *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 the lowest level in sangeranalyseR showed in *Figure 1* which corresponds to a single read (one *AB1* file) in Sanger sequencing. It extends *sangerseq S4* class from *sangerseqR* package and contains quality trimming as well as chromatogram input parameters and results. In this section, we are going to go through detailed sangeranalyseR data analysis steps in *SangerRead level* with *AB1* file input.

Fig. 6: 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. The only hard regulation of the filename 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](image)

Fig. 7: 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. 8: Figure 3. Suggested **AB1** file naming regulation - *SangerRead*.

### 7.4.2 Creating *SangerRead* instance from **AB1**

After preparing the *SangerRead* input **AB1** file, the next step is to create the *SangerRead* S4 instance by running *SangerRead* constructor function or **new** method. The constructor function is a wrapper for **new** method which makes instance creation more intuitive. The inputs include **Basic Parameters**, **Trimming Parameters** and **Chromatogram Parameters** and most of them have their own default values. In the constructor below, we list important parameters.

```r
sangerReadF <- SangerRead(inputSource = "ABIF",
                          readFeature = "Forward Read",
                          readFileName = "Achl_RBNII397-13_1_F.ab1",
                          geneticCode = GENETIC_CODE,
                          (continues on next page)
```

7.4. Advanced User Guide - *SangerRead* (AB1) 21
The inputs of SangerRead constructor function and new method are same. For more details about SangerRead inputs and slots definition, please refer to sangeranalyseR reference manual (need update). The created SangerRead instance, sangerRead, is used as the input for the following functions.

### 7.4.3 Visualizing SangerRead trimmed read

Before going to Writing SangerRead FASTA files (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.

![Quality Base Plot](image)

Fig. 9: Figure 4. SangerRead trimmed read visualization.

```r
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. After creating the SangerRead S4 instance, users can change the trimming parameters by running 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 files (AB1)

Users can write the SangerRead instance to FASTA files. The trimmed read sequence will be written into a FASTA file. Below is the one-line function that users need to run. This function mainly depends on writeXStringSet function in Biostrings R package. Users can set the compression level through writeFasta function.

```r
writeFasta(newSangerRead,
  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 SangerRead instance into a report after the analysis. The report will be generated in HTML by knitting Rmd files. The results in the report are static.

```r
generateReport(newSangerRead,
  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.

7.5 Advanced User Guide - SangerContig (AB1)

SangerContig is the second level in sangeranalyseR showed in Figure 1 which corresponds to a contig in Sanger sequencing. Among slots inside it, 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 sangeranalyseR data analysis in SangerContig level from AB1 file input.

Fig. 10: Figure 1. Hierarchy of classes in sangeranalyseR, SangerContig level.
7.5.1 Preparing SangerContig AB1 input

The main input file format to create SangerRead instance is AB1. Before starting the analysis, users need to prepare all AB1 files inside one directory. It is the parent directory and all AB1 files must be in the first layer of it; in other words, there should not be any directory containing any AB1 files inside the parent directory. Because sangeranalyseR will group AB1 files based on their direction automatically, users have to follow the file naming regulations below:

Note:

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

There are four parameters, parentDirectory, contigName, suffixForwardRegExp and suffixReverseRegExp, that users need to provide so that program can automatically match correct AB1 files and divide them into forward and reverse direction.

Note:

- parentDirectory: The root directory that contains all the AB1 files. It can be absolute or relative path. We suggest users to put only target AB1 files inside this directory without other unrelated files.
- contigName: The value of this parameter is a regular expression that matches filenames that are going to be included in the SangerContig level analysis. grepl function in R is used.
- suffixForwardRegExp: The value of this parameter is a regular expression that matches all filenames in forward direction. grepl function in R is used to select forward reads from all AB1 files.
- suffixReverseRegExp: The value of this parameter is a regular expression that matches all filenames in reverse direction. grepl function in R is used to select reverse reads from all AB1 files.

Here, we have an example:

Figure 2 shows the file naming regulation and hierarchy. In this example, ACHLO is the parent directory that contains all AB1 files. They must be in the first layer of the directory.

sangeranalyseR will first match the contigName to exclude unrelated files and then separate the forward and reverse reads by matching suffixForwardRegExp and suffixReverseRegExp. Therefore, it is important to make sure all target AB1 files share the same contigName and carefully select suffixForwardRegExp and suffixReverseRegExp. 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 totally wrong results. Therefore, users should have a consistent naming strategy. In this example, "_[0-9]+_F", "_[0-9]+_R" for matching forward and reverse reads are highly suggested and are used as default. Moreover, it is a good habit to index your reads in the same contig group because there might be more than one read that are in the forward or reverse direction.

Figure 3 shows the suggested AB1 file naming regulation. Users are strongly recommended to follow this file naming regulation and use the default suffixForwardRegExp: "_[0-9]+_F" and suffixReverseRegExp: "_[0-9]+_R" to reduce any chance of error.
7.5.2 Creating `SangerContig` instance from AB1

After preparing the input directory, we can create the `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. Most parameters in the constructor have their own default values. In the constructor below, we list important parameters.

```r
sangerContig <- SangerContig(inputSource = "ABIF", 
                            parentDirectory = ".:/tmp/", 
                            contigName = "Achl_ACHLO006-09", 
                            suffixForwardRegExp = "[0-9]_F.ab1", 
                            suffixReverseRegExp = "[0-9]_R.ab1", 
                            TrimmingMethod = "M1", 
                            M1TrimmingCutoff = 0.0001, 
                            M2CutoffQualityScore = NULL, 
                            M2SlidingWindowSize = NULL, 
                            baseNumPerRow = 100, 
                            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)
```

In this example, `contigName` is set to "Achl_ACHLO006-09", so only "Achl_ACHLO006-09_1_F.ab1" and "Achl_ACHLO006-09_2_R.ab1" will be selected to align to a contig.
The inputs of SangerContig constructor function and new method are same. For more details about SangerContig inputs and slots definition, please refer to sangeranalyseR reference manual (need update). The created SangerContig instance, sangerContig, is used as the input for the following functions.

### 7.5.3 Updating SangerContig quality trimming parameters

In the previous Creating SangerContig instance from ABI part, the constructor function will apply the quality trimming parameters to all reads. After creating the SangerContig 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. If users want to do quality trimming read by read instead of all at once, please read Launching SangerContig Shiny app page.

```r
newSangerContig <- updateQualityParam(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 pages in the two levels, SangerRead and SangerContig pages.

```r
launchApp(newSangerContig)
```

**SangerContig page (SC app)**

SangerContig page is the initial page of SangerContig Shiny app. Figure 4 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 4. 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 5 shows reads alignment result and difference data frame. The alignment is generated by AlignSeqs or AlignTranslation function in DECIPHER package.

Figure 6 shows dendrogram result in both plot and in data frame. The results are generated by IdClusters function in DECIPHER package.

Figure 7 shows distance between ABI files. The results are generated by DistanceMatrix function in DECIPHER package. The heatmap is generated by plot_ly function in plotly package.
Fig. 13: Figure 4. *SangerContig* Shiny app initial page - *SangerContig* page.

Fig. 14: Figure 5. *SangerContig* page - reads alignment and difference data frame.
Fig. 15: Figure 6. *SangerContig* page - dendrogram.

Fig. 16: Figure 7. *SangerContig* page - samples distance.
Figure 8 shows insertions, deletions and stop codons data frame.

![Indels Data frame](image)

![Stop Codons Data frame](image)

Fig. 17: Figure 8. 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 9 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 14 below will always be same after trimming and their color codings for A/T/C/G are same as well.

![1 Forward SangerRead Page](image)

Fig. 18: Figure 9. 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 10 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:](image)

**Fig. 19:** Figure 10. *SangerRead* page - Trimming Method 1 (M1): ‘Modified Mott Trimming’ UI.

*Figure 11* 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.

![Quality Report:](image)

**Fig. 20:** Figure 11. *SangerRead* page - Trimming Method 2 (M2): ‘Trimmomatics Sliding Window Trimming’ UI.

*Figure 12* shows the quality report before and after trimming. After clicking the “Apply Trimming Parameters” button in *Figure 10* or *Figure 11*, the values of these information boxes will be updated to the latest values.

In *Figure 13*, 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 13* trimming plot and *Figure 14* chromatogram will be updated once users change the quality trimming parameters and click the “Apply Trimming Parameters” button in *Figure 14*.

If we only see primary and secondary sequences in the table, we will lose some variations. Chromatogram is very helpful to check the peak resolution. *Figure 14* 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
Fig. 21: Figure 12. *SangerRead* page - read quality report before / after trimming.

Fig. 22: Figure 13. *SangerContig* page - quality trimming plot.
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 15 shows the loading notification popup during base calling and chromatogram plotting.

After replotting the chromatogram, we can see that trimmed region is showed in red striped region. Figure 16 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 17. 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 18 shows some hints in the save notification popup.

7.5.5 Writing SangerContig FASTA files (AB1)

Users can write the SangerContig instance 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.
Fig. 24: Figure 15. *SangerContig* page - loading notification popup during replotting chromatogram.

Fig. 25: Figure 16. *SangerContig* page - chromatogram with trimmed region showed.
Fig. 26: Figure 17. *SangerContig* page - trimmed primary/secondary sequences and Phred quality score in table format.

<table>
<thead>
<tr>
<th>Primary Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Sequence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Secondary Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
<tr>
<td>Sequence</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Quality Phred Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
</tr>
</tbody>
</table>

Fig. 27: Figure 18. *SangerContig* page - saving notification popup.
• **all**: Writing reads, reads alignment, and the contig into three different files.

Below is the one-line function that users need to run. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

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

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

- reads_unalignment FASTA file
- reads_alignment FASTA file
- contig FASTA file

### 7.5.6 Generating SangerContig report (AB1)

Last but not least, users can save SangerContig instance 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.

```r
generateReport(newSangerContig,
               outputDir = tempdir(),
               includeSangerRead = TRUE)
```

Users can access to ‘Basic Information’, ‘SangerContig Input Parameters’, ‘Contig Sequence’ and ‘Contig Results’ sections inside the generated SangerContig html report of this example. Furthermore, users can also navigate through html reports of all forward and reverse SangerRead in this SangerContig report.

### 7.6 Advanced User Guide - SangerAlignment (AB1)

*SangerAlignment* is the highest class level in sangeranalyseR showed in Figure 1. It contains SangerContig list and the contigs alignment result. Users can access to SangerContig and SangerRead instance inside SangerAlignment instance. In this section, we are going to go through detailed sangeranalyseR data analysis steps in SangerAlignment level from AB1 file input.

Fig. 28: 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 a directory which contains all the **AB1** files. Here are some filename regulations:

**Note:**

- All the input files must have **.ab1** as its file extension.
- The reads that belong to the same contig must have the same contig name in its filename.
- Forward or reverse direction also has to be specified in the filename.

There are three parameters, `parentDirectory`, `suffixForwardRegExp`, and `suffixReverseRegExp`, that users need to provide so that program can automatically group all **AB1** files.

**Note:**

- `parentDirectory`: The root directory that contains all the **AB1** files. It can be absolute or relative path. We suggest users to put only target **AB1** files inside this directory without other unrelated files.
- `suffixForwardRegExp`: The value of this parameter is a regular expression that matches all filenames in forward direction. `grepl` function in R is used to select forward reads from all **AB1** files.
- `suffixReverseRegExp`: The value of this parameter is a regular expression that matches all filenames in reverse direction. `grepl` function in R is used to select reverse reads from all **AB1** files.

For basic input files preparation example, please go to *Beginners Guide*. Here, we have another more complicated example. *Figure 2* shows the file naming regulation and directory hierarchy. In this example, the parent directory is `extdata` and the directories in first layer are `Allolobophora_chlorotica` and `Drosophila_melanogaster`. All target **AB1** files need to be inside parent directory but it is not necessary to put them in the same level. sangeranalyseR will recursively search all files with **.ab1** file extension and automatically group reads with the same contig name. The direction of reads in each contig will be grouped by matching `suffixForwardRegExp` and `suffixReverseRegExp` with filenames. Therefore, it is important to carefully select `suffixForwardRegExp` and `suffixReverseRegExp`. The bad file naming regulation and wrong regex matching might accidentally include reverse reads into the forward read list or vice versa, which will make the program generate totally wrong results. Therefore, users should have a consistent naming strategy. In this example, “^[0-9]+_F”, “^[0-9]+_R” for matching forward and reverse reads are highly suggested and are used as default. It is a good habit to index your reads in the same contig group because there might be more than one read that are in the forward or reverse direction.

*Figure 3* shows the suggested **AB1** file naming regulation. Users are strongly recommended to follow this file naming regulation and use the default `suffixForwardRegExp` : “^[0-9]+_F” and `suffixReverseRegExp` : “^[0-9]+_R” to reduce any chance of error.
Fig. 29: Figure 2. Input ab1 files inside the parent directory, ./tmp/.

Fig. 30: Figure 3. Suggested AB1 file naming regulation - SangerAlignment.

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

Fig. 30: Figure 3. Suggested AB1 file naming regulation - SangerAlignment.
7.6.2 Creating SangerAlignment instance from AB1

After preparing the input directory, we can create the SangerAlignment S4 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. Most parameters in the constructor have their own default values. In the constructor below, we list important parameters. For a simpler command, please go to Quick Start Guide.

```r
sangerAlignment <- SangerAlignment(inputSource = "ABIF",
parentDirectory = ".\tmp/",
suffixForwardRegExp = "\[0-9]+_F",
suffixReverseRegExp = "\[0-9]+_R",
refAminoAcidSeq = "SRQWLSTNHKDIGTLYFIFGAWAGMVGSILISILISAELGHPALIGDDQIYNVIVTAHAIFIMIFFMVMPIMGGFGNLVPLMGAPDMAFPRMN",
TrimmingMethod = "M1",
M1TrimmingCutoff = 0.0001,
M2CutoffQualityScore = NULL,
M2SlidingWindowSize = NULL,
baseNumPerRow = 100,
heightPerRow = 200,
signalRatioCutoff = 0.33,
showTrimmed = TRUE,
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)
```

The inputs of SangerAlignment constructor function and new method are same. For more details about SangerAlignment inputs and slots definition, please refer to sangeranalyseR reference manual (need update).

7.6.3 Updating SangerAlignment quality trimming parameters

In the previous Creating SangerAlignment instance from ABI part, the constructor function will apply the quality trimming parameters to all reads. After creating the 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.

```r
newSangerAlignment <- updateQualityParam(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 and the *SangerAlignment* Shiny app will pop up. Here, we will go through pages in the three levels.

```r
launchApp(sangerAlignment)
```

*SangerAlignment* page (SA app)

*Figure 4* 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.

![SangerAlignment Overview Page](image)

*Fig. 31: Figure 4. 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 5* shows the contigs alignment result.

In *SangerAlignment* page, the phylogenetic tree result is provided as well (*Figure 6*). 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 7* 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.
**SangerAlignment Results:**

**Contigs Alignment**

![Contigs Alignment Result]

Fig. 32: Figure 5. *SangerAlignment* Page - contigs alignment result.

**Contigs Tree**

![Contigs Tree Result]

Fig. 33: Figure 6. *SangerAlignment* Page - phylogenetic tree result.
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 8 and Figure 9 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 10. In this example, there are one read in each tab and Figure 10 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 15 below will always be same after trimming and their color codings for A/T/C/G are same as well.

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.

Figure 12 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.

Figure 13 shows the quality report before and after trimming. After clicking the “Apply Trimming Parameters” button,
Fig. 35: Figure 8. *SangerContig* page - contig-related parameters, genetic code and reference amino acid sequence.
Fig. 36: Figure 9. *SangerContig* page - reads alignment and difference data frame.

Fig. 37: Figure 10. *SangerAlignment* Shiny app - *SangerRead* page.
Fig. 38: Figure 11. SangerRead page - Trimming Method 1 (M1): ‘Modified Mott Trimming’ UI.

Fig. 39: Figure 12. SangerRead page - Trimming Method 2 (M2): ‘Trimmomatics Sliding Window Trimming’ UI.
the values of these information boxes will be updated to the latest values.

---

**Trimmed Result Output**

<table>
<thead>
<tr>
<th>Before Trimming</th>
<th>After Trimming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>702</strong></td>
<td><strong>461</strong></td>
</tr>
<tr>
<td>Raw Sequence Len</td>
<td>Trimmed Sequence Length</td>
</tr>
<tr>
<td><strong>52.88</strong></td>
<td><strong>58.21</strong></td>
</tr>
<tr>
<td>Raw Mean Quality Score</td>
<td>Trimmed Mean Quality Score</td>
</tr>
<tr>
<td><strong>1</strong></td>
<td><strong>13</strong></td>
</tr>
<tr>
<td>Raw Min Quality Score</td>
<td>Trimmed Min Quality Score</td>
</tr>
<tr>
<td><strong>16</strong></td>
<td><strong>477</strong></td>
</tr>
<tr>
<td>Trimming Start Pos</td>
<td>Trimming End Pos</td>
</tr>
<tr>
<td><strong>65.67 %</strong></td>
<td><strong>65.67 %</strong></td>
</tr>
<tr>
<td>Remaining Ratio</td>
<td>Remaining Ratio</td>
</tr>
</tbody>
</table>

---

*Fig. 40: Figure 13. SangerRead page - read quality report before / after trimming.*

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 trimmed 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 13* 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 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.

Here is an example of applying new chromatogram parameters. We click “Show Trimmed Region” to set its value from FALSE to TRUE. *Figure 16* shows the loading notification popup during base calling and chromatogram plotting.

After replotting the chromatogram, 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 in table format with quality scores 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.
Fig. 41: Figure 14. *SangerRead* page - quality trimming plot.

Fig. 42: Figure 15. *SangerRead* page - chromatogram panel.
Fig. 43: Figure 16. *SangerRead* page - loading notification popup during replotting chromatogram.

Fig. 44: Figure 17. *SangerRead* page - chromatogram with trimmed region showed.
Fig. 45: Figure 18. *SangerRead* page - trimmed primary/secondary sequences and Phred quality score in table format.

Fig. 46: Figure 19. *SangerRead* page - saving notification popup.
7.6.5 Writing SangerAlignment FASTA files (AB1)

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

- contigs_unalignment: Writing contigs into a single FASTA file.
- 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 one-line function that users need to run. This function mainly depends on writeXStringSet function in Biostrings R package. Users can set the compression level through writeFasta function.

```r
writeFasta(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:

- contigs_unalignment FASTA file
- contigs_alignment FASTA file
- all_reads FASTA file

7.6.6 Generating SangerAlignment report (AB1)

Last but not least, users can save SangerAlignment instance into a report after the analysis. The report will be generated in HTML by knitting Rmd files. There are two parameters, includeSangerContig and includeSangerRead, for users to decide 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.

- includeSangerContig: Whether users want to generate the report of each SangerContig in SangerAlignment.
- includeSangerRead: If includeSangerContig is TRUE, then users can set this value to decide whether they want to include SangerRead reports in each SangerContig.

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 contigs alignment, remember to set includeSangerContig and includeSangerRead to FALSE in order to save time.

```r
generateReport(sangerAlignment,
               outputDir = tempdir(),
               includeSangerContig = TRUE,
               includeSangerRead = TRUE)
```
Users can access to ‘Basic Information’, ‘Contigs Consensus’, ‘Contigs Alignment’ and ‘Contigs Tree’ sections inside the generated SangerContig html report of this example. Furthermore, users can also navigate through html reports of all forward and reverse SangerRead in this SangerContig report.

### 7.7 Advanced User Guide - SangerRead (FASTA)

*SangerRead* is the lowest level in sangeranalyseR showed in *Figure 1* which 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* from FASTA file input.

![Figure 1. Hierarchy of classes in sangeranalyseR, SangerRead level.](image)

#### 7.7.1 Preparing SangerRead FASTA input

We design the FASTA file input for those who do not want to do quality trimming and base calling for their Sanger-Read; therefore, it does not contain quality trimming and chromatogram input parameters and results in its slots. Before starting the analysis, users need to prepare one target FASTA file. The only hard regulation of the filename is that file extension must be `.fasta` or `.fa`.

#### 7.7.2 Creating SangerRead instance from FASTA

After preparing the SangerRead input FASTA file, the next step is to create the SangerRead S4 instance by running SangerRead constructor function or new method. The constructor function is a wrapper for new method which makes instance creation more intuitive. Most of the input parameters have their own default values. In the constructor below, we list important parameters. The filename is assigned to readFileName. Inside FASTA file, the string in the first line after “>” is the name of the read. Users need to assign the read name to fastaReadName which is used to match the target read in FASTA input file. *Figure 2* is a valid FASTA file and the value of fastaReadName is Achl_ACHLO006-09_1_Forward.

```r
sangerReadFfa <- new("SangerRead",
                      inputSource = "FASTA",
                      readFeature = "Forward Read",
                      readFileName = "ACHLO006-09[LCO1490_t1,HCO2198_t1]_1_F.fa",
                      fastaReadName = "Achl_ACHLO006-09_1_Forward",
                      geneticCode = GENETIC_CODE)
```

The inputs of SangerRead constructor function and new method are same. For more details about SangerRead inputs and slots definition, please refer to sangeranalyseR reference manual (need update after upload function manul).
7.7.3 Writing SangerRead FASTA files (FASTA)

Users can write the SangerRead instance to FASTA files. Because the FASTA input does not support quality trimming and base calling, in this example, the sequence of the written FASTA file will be same as the input FASTA file. Moreover, users can set the compression level through the one-line function writeFasta which mainly depends on writeXStringSet function in Biostrings R package.

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

Users can download the output FASTA file of this example.

7.7.4 Generating SangerRead report (FASTA)

Last but not least, users can save SangerRead instance into a report after the analysis. The report will be generated in HTML by knitting Rmd files. The results in the report are static.

```r
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.

7.8 Advanced User Guide - SangerContig (FASTA)

SangerContig is the second level in sangeranalyseR showed in Figure_1 which corresponds to a contig in Sanger sequencing. Among slots inside it, 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 sangeranalyseR data analysis in SangerContig level with FASTA file input.
7.8.1 Preparing SangerContig FASTA input

We design the FASTA file input for those who do not want to do quality trimming and base calling for each SangerRead in SangerContig; therefore, it does not contain quality trimming and chromatogram input parameters and results in SangerRead slots. Before starting the analysis, users need to prepare one FASTA file containing sequence of all reads. Inside the FASTA file, the strings starting with “>” before each read are the read names. Because sangeranalyseR will automatically group reads into “Forward Read List” and “Reverse Read List”, users have to follow the naming regulations. Below are some regulations:

Note:

- The same contig name must be included in all read names.
- Forward or reverse direction also has to be specified in the read names.

There are four parameters, fastaFileName, contigName, suffixForwardRegExp and suffixReverseRegExp, that users need to provide so that program can automatically match correct reads in FASTA file and divide them into forward and reverse direction.

Note:

- fastaFileName: The path of FASTA file that contains sequence of all reads. The read names have to follow the naming regulation.
- contigName: The value of this parameter is a regular expression that matches read names that are going to be included in the SangerContig level analysis. grepl function in R is used.
- suffixForwardRegExp: The value of this parameter is a regular expression that matches all read names in forward direction. grepl function in R is used to select forward reads from all read names in FASTA files.
- suffixReverseRegExp: The value of this parameter is a regular expression that matches all read names in reverse direction. grepl function in R is used to select reverse reads from all read names in FASTA files.

No doubt read names in the original FASTA file will not follow the naming regulation; however, it is highly not recommended to change the name directly in the raw FASTA file. Therefore, we provide a feature to let users do read names mapping conversion by a CSV file showed in Figure 2. The first column is “original_read_name” which are the read names in the raw FASTA file, and the second column is “analysis_read_name” which are the read names that follow the naming regulation. The read names will be mapped onto the names in “original_read_name” without changing the raw FASTA file. namesConversionCSV is the parameter that stores the path to this CSV file.

```
"original_read_name","analysis_read_name"
"Achl_ACHL0006-09_1 Forward","Achl_ACHL0006-09_1_F"
"Achl_ACHL0006-09_2 Reverse","Achl_ACHL0006-09_2_R"
```

Fig. 50: Figure 2. SangerContig CSV file - read names conversion.

Here, we have an example:
sangeranalyseR

Fig. 51: Figure 3. *SangerContig* FASTA input file.

*Figure 3* shows the FASTA input file and the read names in it will be mapped onto the CSV file showed in *Figure 2*. sangeranalyseR will first match the contigName to exclude unrelated reads and then separate the forward and reverse reads by matching suffixForwardRegExp and suffixReverseRegExp. Therefore, it is important to make sure all target reads share the same contigName and carefully select suffixForwardRegExp and suffixReverseRegExp. 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 totally wrong results. Therefore, users should have a consistent naming strategy. In this example, “\[0-9\]+_F”, “\[0-9\]+_R” for matching forward and reverse reads are highly suggested. It is a good habit to index your reads in the same contig group because there might be more than one read that are in the forward or reverse direction.

\[
\text{Consensus Read Name} + \_ + \text{[index]} + \_ + [F, R]
\]

Fig. 52: Figure 4. Suggested read naming regulation in FASTA file - *SangerContig*.

*Figure 4* shows the suggested read naming regulation which is used in the “analysis_read_name” column in CSV file (*Figure 2*). Users are strongly recommended to follow this file naming regulation and use suffixForwardRegExp: “\[0-9\]+_F” and suffixReverseRegExp: “\[0-9\]+_R” to reduce any chance of error.

### 7.8.2 Creating *SangerContig* instance from FASTA

After preparing the input directory, we can create the *SangerContig* S4 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. Most parameters in the constructor have their own default values. In the constructor below, we list important parameters.

```r
sangerContigFa <- SangerContig(inputSource = "FASTA",
                               fastaFileName = "Achl_ACHLO006-09.fa",
                               namesConversionCSV = "names_conversion_1.csv",
                               suffixForwardRegExp = "\[0-9\]+_F",
                               suffixReverseRegExp = "\[0-9\]+_R")
```

(continues on next page)
In this example, `contigName` is set to "Achl_ACHLO006-09", so only "Achl_ACHLO006-09_1_F.ab1" and "Achl_ACHLO006-09_2_R.ab1" reads will be selected from FASTA file to align into a contig.

The inputs of `SangerContig` constructor function and `new` method are same. For more details about `SangerContig` inputs and slots definition, please refer to sangeranalyseR reference manual (need update).

### 7.8.3 Writing `SangerContig` FASTA files (FASTA)

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

- **reads_unalignment**: Writing reads into a single FASTA file.
- **reads_alignment**: Writing reads alignment and the aligned contig to a single FASTA file.
- **contig**: Writing the contig to a single FASTA file.
- **all**: Executing the three options mentioned above and writing `SangerContig` instance into three different files.

Below is the one-line function that users need to run. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

```r
writeFasta(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:

- reads_unalignment FASTA file
- reads_alignment FASTA file
- contig FASTA file

### 7.8.4 Generating `SangerContig` report (FASTA)

Last but not least, users can save `SangerContig` instance 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 would 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.

```r
generateReport(sangerContigFa,
               outputDir = tempdir(),
               includeSangerRead = TRUE)
```

Users can access to ‘Basic Information’, ‘SangerContig Input Parameters’, ‘Contig Sequence’ and ‘Contig Results’ sections inside the generated SangerContig html report of this example. Furthermore, users can also navigate through html reports of all forward and reverse SangerRead in this SangerContig report.

### 7.9 Advanced User Guide - SangerAlignment (FASTA)

*SangerAlignment* is the highest class level in sangeranalyseR showed in *Figure_1*. It contains *SangerContig* list and the contigs alignment result. Users can access to *SangerContig* and *SangerRead* instance inside *SangerAlignment* instance. In this section, we are going to go through detailed sangeranalyseR data analysis steps in *SangerAlignment* level from FASTA file input.

![Figure 1. Classes hierarchy in sangeranalyseR, SangerAlignment level.](image)

#### 7.9.1 Preparing SangerAlignment FASTA input

We design the FASTA file input for those who do not want to do quality trimming and base calling for each *SangerRead* in *SangerAlignment*; therefore, it does not contain quality trimming and chromatogram input parameters and results in *SangerRead* slots.

Before starting the analysis, users need to prepare one FASTA file containing sequence of all reads. Inside the FASTA file, there are strings starting with “>” before each read which are the read names. Because sangeranalyseR will group reads into “Forward Read List” and “Reverse Read List”, users have to follow the naming regulations for the read names. Below are some regulations:

**Note:**

- Reads that are in the same contig have to share the same contig name.
- Forward or reverse direction also has to be specified in the read names.

There are three parameters, `fastaFileName`, `suffixForwardRegExp`, and `suffixReverseRegExp`, that users need to provide so that program can automatically group all FASTA files.

**Note:**
• **fastaFileName**: The FASTA file that contains sequence of all reads. The read names have to follow the naming regulation.

• **suffixForwardRegExp**: The value of this parameter is a regular expression that matches all filenames in forward direction. `grep` function in R is used to select forward reads from all FASTA files.

• **suffixReverseRegExp**: The value of this parameter is a regular expression that matches all filenames in reverse direction. `grep` function in R is used to select reverse reads from all FASTA files.

No doubt read names in the original FASTA file will not follow the naming regulation; however, it is highly not recommended to change the name directly in the raw FASTA file. Therefore, we provide a feature to let users do read names mapping conversion by a CSV file showed in Figure 2. The first column is “original_read_name” which are the read names in the raw FASTA file, and the second column is “analysis_read_name” which are the read names that follow the naming regulation. The read names will be mapped onto the names in “original_read_name” without changing the raw FASTA file. `namesConversionCSV` is the parameter that stores the path to this CSV file.

![Figure 2. SangerAlignment CSV file - read names conversion.](image)

Here, we have another more complicated example.

```plaintext
> Achl_ACHL0006-09_Achl_ACHL0006-09_1_F
GCGTCTGACGACAGTATGTTAGAGGCTGTTATATAAGACTCTAATTTGAATAGCTAAGACAGGCAGACATTCTAGAG
AGGGATATCGCTTCTAATATCATTAGAAATCTGTCTCAGCAGTATTGAAATATTCTCTAGATTATCTTATTG
GGTTGATGATATCTCCTCTCAGTAACTTTAATCTTGGAGCCTGCTGTTAACATGTACATCTGCTAGTAAAGATCGAGAATCTGATGTAATCTTTT
AGATCCTTTTGGCTCCATTCTATCTGTCGTTGATGATTATTATCTTACCTTTGATGAGCGGTCCTTCTTTTTATG
ATCATTGATTTGGCGGATTATTTTATAATATCAGTACGTGATGCTATTTTCTATCTTCTCTTACCAGTGTTACGCCTGATTACATATATTACTATT
ACGACAGGAATACAGCCTCTCCTTTGATGCTTGGGCC
> Achl_ACHL0006-09_Achl_ACHL0006-09_2_R
GCGTCTGACGACAGTATGTTAGAGGCTGTTATATAAGACTCTAATTTGAATAGCTAAGACAGGCAGACATTCTAGAG
AGGGATATCGCTTCTAATATCATTAGAAATCTGTCTCAGCAGTATTGAAATATTCTCTAGATTATCTTATTG
GGTTGATGATATCTCCTCTCAGTAACTTTAATCTTGGAGCCTGCTGTTAACATGTACATCTGCTAGTAAAGATCGAGAATCTGATGTAATCTTTT
AGATCCTTTTGGCTCCATTCTATCTGTCGTTGATGATTATTATCTTACCTTTGATGAGCGGTCCTTCTTTTTATG
ATCATTGATTTGGCGGATTATTTTATAATATCAGTACGTGATGCTATTTTCTATCTTCTCTTACCAGTGTTACGCCTGATTACATATATTACTATT
ACGACAGGAATACAGCCTCTCCTTTGATGCTTGGGCC
```

![Figure 3. SangerAlignment FASTA input file.](image)

*Figure 3* shows the FASTA input file and the read names in it will be mapping conversed by CSV file.
showed in Figure 2 (Only two reads are showed). 

sangeranalyseR will first match the `contigName` to exclude unrelated reads. The direction of reads in each contig will be grouped by matching `suffixForwardRegExp` and `suffixReverseRegExp` with read names. Therefore, it is important to carefully select `suffixForwardRegExp` and `suffixReverseRegExp`. 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 totally wrong results. Therefore, users should have a consistent naming strategy. In this example, "\_[0-9]\+_F", "\_[0-9]\+_R" for matching forward and reverse reads are highly suggested and are used as default. It is a good habit to index your reads in the same contig group because there might be more than one read that are in the forward or reverse direction.

\[
\text{[Consensus Read Name]} + [\text{index}] + [\text{F}, \text{R}]
\]

Fig. 56: Figure 4. Suggested read naming regulation in FASTA file - SangerAlignment.

Figure 4 shows the suggested reads naming regulation. Users are strongly recommended to follow this reads naming regulation and use the default `suffixForwardRegExp` : "\_[0-9]\+_F" and `suffixReverseRegExp`: "\_[0-9]\+_R" to reduce any chance of error.

### 7.9.2 Creating SangerAlignment instance from FASTA

After preparing the input directory, we can create the SangerAlignment S4 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. Most parameters in the constructor have their own default values. In the constructor below, we list important parameters.

```r
sangerAlignmentFa <- SangerAlignment(inputSource = "FASTA",
                                       fastaFileName = "Sanger_all_reads.fa",
                                       namesConversionCSV = "names_conversion.csv",
                                       suffixForwardRegExp = "\_[0-9]+_F",
                                       suffixReverseRegExp = "\_[0-9]+_R",
                                       refAminoAcidSeq = "SRQWLFSNTHKDIGTYFIFGAWAGMVGSILIRAILGHGPALIGDDQIYNYVITAHIFIMIFFMVPIMGFGNWLVPLMGAPDCAFPRMN...
                                       )
```

The inputs of SangerAlignment constructor function and `new` method are same. For more details about SangerAlignment inputs and slots definition, please refer to sangeranalyseR reference manual (need update).

### 7.9.3 Writing SangerAlignment FASTA files (FASTA)

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

- `contigs_unalignment`: Writing contigs into a single FASTA file.
- `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 one-line function that users need to run. This function mainly depends on `writeXStringSet` function in Biostrings R package. Users can set the compression level through `writeFasta` function.

```
writeFasta(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:

- **contigs_unalignment** FASTA file
- **contigs_alignment** FASTA file
- **all_reads** FASTA file

### 7.9.4 Generating **SangerAlignment** report (FASTA)

Last but not least, users can save SangerAlignment instance into a report after the analysis. The report will be generated in HTML by knitting Rmd files. There are two parameters, `includeSangerContig` and `includeSangerRead`, for users 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.

- **includeSangerContig**: Whether users want to generate the report of each SangerContig in SangerAlignment.

- **includeSangerRead**: If `includeSangerContig` is TRUE, then users can set this value to decide whether they want to include SangerRead reports in each SangerContig.

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 SangerAlignment level (contigs alignment), remember to set `includeSangerContig` and `includeSangerRead` to FALSE in order to save time.

```
generateReport(sangerAlignmentFa, 
  outputDir = tempdir(), 
  includeSangerContig = TRUE, 
  includeSangerRead = TRUE)
```

Users can access to `Basic Information`, `Contigs Consensus`, `Contigs Alignment` and `Contigs Tree` sections inside the generated SangerContig html report of this example. Furthermore, users can also navigate through html reports of all forward and reverse SangerRead in this SangerContig report.
7.10 How to . . .

7.10.1 How to deal with secondary peaks

7.10.2 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 details about other functions, please refer to the sangeranalyseR user manual.

7.11.1 SangerRead Constructor Parameters

```r
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

```r
SangerContig(inputSource = "ABIF",
             fastaFileName = "",
             namesConversionCSV = NULL,
             parentDirectory = 
             contigName = 
             suffixForwardRegExp = 
             suffixReverseRegExp = 
             TrimmingMethod = "M1",
             M1TrimmingCutoff = 0.0001,
             M2CutoffQualityScore = NULL,
             M2SlidingWindowSize = NULL,
             baseNumPerRow = 100,
             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 reverse 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 sequences, 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 default): 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

```r
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 = "",
                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 reverse 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 sequences, 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 default): 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).
sangeranalyseR

- **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 Sanger-Alignment 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 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 liketrimmomatic’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

MIT License

Copyright (c) 2019 Kuan-Hao Chao

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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