Walkthrough for using the *sangerseqR* package

Jonathon T. Hill, PhD

January 20, 2015

Contents

1 Introduction

The *sangerseqR* package provides basic functions for importing and working with sanger sequencing data files. It currently functions with Scf and ABIF files. The Scf file specification is an open source, although somewhat limited, data file type. Several tools designed to view and or edit chromatogram data can convert file types to Scf. The ABIF file specification is a proprietary data storage file specification for sequencing data generated by Applied Biosystems machines. More information on each filetype can be found at the following sites:

ABIF: http://home.appliedbiosystems.com/support/software_community/ABIF_File_Format.pdf

SCF: http://staden.sourceforge.net/manual/formats_unix_2.html

The objects and functions included in this package were developed as part of the Poly Peak Parser web application (http://yost.genetics.utah.edu/software.php), which automates the process of seperating ambiguous double peaks from Sanger sequencing individuals containing heterozygous indels. However, because they have broader utility than is available in the web application, they are provided in this package for general use in R.

This vignette will walk you through a typical workflow using two sequence files: 1) homozygous.scf and 2) heterozygous.ab1. As their names indicate, the first example contains results typical of sequencing from a PCR product of a homozygous individual or from a plasmid. The second example contains results from sequencing the same region in an individual with a small indel.

2 Loading Data

The first step of most workflows will be to upload data from a sequencing results file. This can be done using one of three included functions: `read.abif`, `read.scf` and `readsangerseq`. The first two functions directly import all of the fields into *abif* and *scf* class objects, respectively. These classes are meant as intermediate classes and exist to allow the user to inspect the file contents, as file contents may vary between basecallers and sequencing machines. Users should generally use the `readsangerseq` function. This function automatically detects and reads in the file type and then extracts the fields necessary to create a *sangerseq* class object, which is used by all of the other functions in this package.

2.1 read.abif

`read.abif` takes a single argument for the filename of the abif file to be read. The resulting object contains three major parts. The header, containing information on the file structure, the directory, containing information on each of the data fields included in the file, and the data fields. Here is an example:

```
hetab1 <- read.abif(system.file("extdata", "heterozygous.ab1", package = "sangerseqR"))
str(hetab1, list.len = 20)
```

```
## Formal class 'abif' [package "sangerseqR"] with 3 slots
##   ..@ header   :Formal class 'abifHeader' [package "sangerseqR"] with 9 slots
##   .. .. ..@ abif       : chr "ABIF"
##   .. .. ..@ version    : int 101
##   .. .. ..@ name       : raw [1:4] 74 64 69 72
##   .. .. ..@ number     : int 1
##   .. .. ..@ elementtype: int 1023
##   .. .. ..@ elementsize: int 28
##   .. .. ..@ numelements: int 130
##   .. .. ..@ dataoffset : int 323971
##   .. .. ..@ datahandle : int 0
##   ..@ directory:Formal class 'abifDirectory' [package "sangerseqR"] with 7 slots
##   .. .. ..@ name       : chr [1:130] "AEPt" "AEPt" "APFN" "APXV" ...
##   .. .. ..@ tagnumber  : int [1:130] 1 2 2 1 1 1 1 1 1 1 ...
##   .. .. ..@ elementtype: int [1:130] 4 4 18 19 19 19 2 5 4 4 ...
##   .. .. ..@ elementsize: int [1:130] 2 2 1 1 1 1 1 4 2 2 ...
##   .. .. ..@ numelements: int [1:130] 1 1 6 2 6 2 4503 1 1 1 ...
##   .. .. ..@ datasize   : int [1:130] 2 2 6 2 6 2 4503 4 2 2 ...
##   .. .. ..@ dataoffset : int [1:130] 1113325568 1113325568 173231 838860800 163360 9563
## 01312 163366 0 65536 145752064 ...
##   ..@ data     :List of 130
##   .. ..$ AEPt.1 : int 16988
##   .. ..$ AEPt.2 : int 16988
##   .. ..$ APFN.2 : chr "Seq_A"
##   .. ..$ APXV.1 : chr "2"
##   .. ..$ APrN.1 : chr "Seq_A"
##   .. ..$ APrV.1 : chr "9"
##   .. ..$ APrX.1 : chr "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?><An
## alysisProtocolContainer doAnalysis=\"true\" name=\"Seq_A\" guid"| __truncated__
##   .. ..$ ARTN.1 : int 0
##   .. ..$ ASPF.1 : int 1
##   .. ..$ ASPt.1 : int 2224
##   .. ..$ ASPt.2 : int 2224
##   .. ..$ AUDT.1 : int [1:1370] 64 126 65 54 55 79 81 183 49 123 ...
##   .. ..$ B1Pt.1 : int 2223
##   .. ..$ B1Pt.2 : int 2223
```

```
##   .. ..$ BCTS.1 : chr "2013-06-13 17:26:28 -06:00"
##   .. ..$ BufT.1 : int [1:1596] -27 -27 -27 -27 -27 -27 -27 -27 -27 -27 ...
##   .. ..$ CMNT.1 : chr "<ID:119209><WELL:G02>"
##   .. ..$ CTID.1 : chr "bdt1735"
##   .. ..$ CTNM.1 : chr "bdt1735"
##   .. ..$ CTOw.1 : chr "aadamson"
##   .. .. [list output truncated]
```

As you can see, the file is very long and contains a lot of Data fields (130 in this example). However, most of these contain run information and only a few are directly relevant to data analysis:

DATA.9–DATA.12 Vectors containing the signal intensities for each channel.

FWO_.1 A string containing the base corresponding to each channel. For example, if it is "ACGT", then DATA.9 = A, DATA.10 = C, DATA.11 = G and DATA.12 = T.

PLOC.2 Peak locations as an index of the trace vectors.

PBAS.1, PBAS.2 Primary basecalls. PBAS.1 may contain bases edited in the original basecaller, while PBAS.2 always contains the basecaller's calls.

P1AM.1 Amplitude of primary basecall peaks.

P2BA.1 (optional) Contains the secondary basecalls.

P2AM.1 (optional) Amplitude of the secondary basecall peaks.

2.2 read.scf

Like `read.abif`, `read.scf` takes a single argument with the filename. However, the data structure of the resulting *scf* object is far less complicated, containing only a header with file structure information, a matrix of the trace data (`@sample_points`), a matrix of relative probabilities of each base at each position (`@sequence_probs`), basecall positions (`@basecall_positions`), basecalls (`@basecalls`) and optionally a comments sections with the run data (`@comments`). The last slot (`@private`) is rarely used and impossible to interpret without knowing how it was created.

```
homoscf <- read.scf(system.file("extdata", "homozygous.scf", package = "sangerseqR"))
str(homoscf)
```

```
## Formal class 'scf' [package "sangerseqR"] with 7 slots
##   ..@ header            :Formal class 'scfHeader' [package "sangerseqR"] with 14 slots
##   .. .. ..@ scf             : chr ".scf"
##   .. .. ..@ samples         : int 16275
##   .. .. ..@ samples_offset  : int 128
##   .. .. ..@ bases           : int 722
##   .. .. ..@ bases_left_clip : int 0
##   .. .. ..@ bases_right_clip: int 0
##   .. .. ..@ bases_offset    : int 130328
##   .. .. ..@ comments_size   : int 1731
##   .. .. ..@ comments_offset : int 138992
##   .. .. ..@ version         : num 3
##   .. .. ..@ sample_size     : int 2
##   .. .. ..@ code_set        : int 2
##   .. .. ..@ private_size    : int 0
##   .. .. ..@ private_offset  : int 140723
##   ..@ sample_points     : num [1:16275, 1:4] 187 190 199 220 255 304 354 389 404 402 ..
```

```
## .
##   ..@ sequence_probs    : int [1:722, 1:4] 0 0 0 0 0 0 0 0 0 0 ...
##   ..@ basecall_positions: int [1:722] 2 18 25 39 45 56 63 68 85 94 ...
##   ..@ basecalls         : chr "ARGKRAMMYWACTATAGGGCGGAATTGAATTTAGCGGCCGCGAATTCGCCCTTTGG
## CAAGAGAGCGACAGTCAGTCGGACTTACGAGTTGTTTTTACAGGCGCAATTCTTTTTTTAGAATATTATAC"| __truncated__
##   ..@ comments          : chr "STRT=6/21/201318:04:27STOP=6/21/201320:02:45SIGN=G=124,A
## =134,T=204,C=159AEPt=16308AEPt=16308APFN=Seq_AAPXV=APrN=Seq_AAPrV=ARTN="| __truncated__
##   ..@ private           : raw [1:2] 00 31
```

2.3 readsangerseq

The `readsangerseq` function is a convenience function equivalent to `sangerseq(read.abif(file))` or `sangerseq(read.scf(file))`. It should generally be used when the contents of the file do not need to be directly accessed because it returns a *sangerseq* object, described below.

3 Sangerseq Class Objects

The *sangerseq* class is the backbone of the sangerseqR package and contains the chromatogram data necesary to perform all other functions. It can be created in two ways: from an *abif* or *scf* object using the `sangerseq` method or directly from an abif or scf file using `readsangerseq`.

```
# from a sequence file object
homosangerseq <- sangerseq(homoscf)

# directly from the file
hetsangerseq <- readsangerseq(system.file("extdata", "heterozygous.ab1", package = "sangerseqR"))
str(hetsangerseq)
```

```
## Formal class 'sangerseq' [package "sangerseqR"] with 7 slots
##   ..@ primarySeqID  : chr "From ab1 file"
##   ..@ primarySeq    :Formal class 'DNAString' [package "Biostrings"] with 5 slots
##   .. .. ..@ shared         :Formal class 'SharedRaw' [package "XVector"] with 2 slots
##   .. .. .. .. ..@ xp                    :<externalptr>
##   .. .. .. .. ..@ .link_to_cached_object:<environment: 0x0471843c>
##   .. .. ..@ offset         : int 0
##   .. .. ..@ length         : int 605
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ secondarySeqID: chr "From ab1 file"
##   ..@ secondarySeq  :Formal class 'DNAString' [package "Biostrings"] with 5 slots
##   .. .. ..@ shared         :Formal class 'SharedRaw' [package "XVector"] with 2 slots
##   .. .. .. .. ..@ xp                    :<externalptr>
##   .. .. .. .. ..@ .link_to_cached_object:<environment: 0x0471843c>
##   .. .. ..@ offset         : int 0
##   .. .. ..@ length         : int 605
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ traceMatrix   : int [1:16215, 1:4] 0 0 0 1 2 4 4 2 1 0 ...
##   ..@ peakPosMatrix : num [1:605, 1:4] 4 13 21 31 43 58 64 73 83 98 ...
##   ..@ peakAmpMatrix : int [1:605, 1:4] 380 694 836 934 1367 1063 2072 1502 1234 539 ...
```

The slots are as follows:

primarySeqID	Identification of the primary Basecalls.
primarySeq	The primary Basecalls formatted as a DNAString object.
secondarySeqID	Identification of the secondary Basecalls.
secondarySeq	The secondary Basecalls formatted as a DNAString object.
traceMatrix	A numerical matrix containing 4 columns corresponding to the normalized signal values for the chromatogram traces. Column order = A,C,G,T.
peakPosMatrix	A numerical matrix containing the position of the maximum peak values for each base within each Basecall window. If no peak was detected for a given base in a given window, then "NA". Column order = A,C,G,T.
peakAmpMatrix	A numerical matrix containing the maximum peak amplitudes for each base within each Basecall window. If no peak was detected for a given base in a given window, then 0. Column order = A,C,G,T.

Accessor functions also exist for each slot in the sangerseq object. Most of the accessors return the data in its native format, but the `primarySeq` and `secondarySeq` accessors can optionally return the data as a character string or a *DNAString* class object from the *Biostrings* package by setting `string=TRUE` or `string=FALSE`, respectively. The *DNAString* class contains several convenient functions for manipulating the sequence, including generating the reverse compliment and performing alignments. The *Biostrings* package is automatically loaded with the *sangerseq* package, so all methods should be available.

```
# default is to return a DNAString object
Seq1 <- primarySeq(homosangerseq)
reverseComplement(Seq1)
```

```
##   722-letter "DNAString" instance
## seq: TTAACCCTCACTAAAAGGGAATTAGTCCTGCAGGTTTAAAC...GCGGCCGCTAAATTCAATTCCGCCCTATAGTWRKKTYM
## CYT
```

```
# can return as string
primarySeq(homosangerseq, string = TRUE)
```

```
## [1] "ARGKRAMMYWACTATAGGGCGGAATTGAATTTAGCGGCCGCGAATTCGCCCTTTGGCAAGAGAGCGACAGTCAGTCGGACTT
## ACGAGTTGTTTTTACAGGCGCAATTCTTTTTTTAGAATATTATACATTCATCTGGCTTTTTGGGTGCACCGATGAGAGATCCAGTTTTCA
## CAGCGAACGCTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCCATGTCAGCTTTCCTTGCGGCGGCTCAACCTTCGT
## TCTTTCCAGCGCTCACTTTACCAGTAAACCGCTGGCGGATCATGCGCTCTCCGGTGCGGCTGAAGCTGGTTTACACGCGGCGCTTGGACA
## TCACCACCAGGCGGCTCATCTGCGCTCTTTCAAGGGTCTCGAGCCAGAGGAGGATGTTGAGGACGATCCTAAAGTTACATTAGAAGCTAA
## GGAGCTTTGGGATCAATTCCACAAAATTGGAACAGAAATGGTCATCACTAAATCAGGAAGGTAAGGTCTTTACATTATTTAACCTATTGA
## ATGCTGCATAGGGTGATGTTATTATATTACTCCGCGAAGAGTTGGGTCTATTTTATCGTAAAATATACTTTACATTATAAAATATTGCTC
## GGTTAAAATTCAGATGTACTGGATGCTGACATAGCATCGAAGCCTCTAARGGCGAATTCGTTTAAACCTGCAGGACTAATTCCCTTTTAG
## TGAGGGTTAA"
```

4 Creating Chromatograms

Basic chromatogram plots can be made using the `chromatogram` function. These plots are optimized for printing, so they contain several rows to plot all of the data simultaneously. The downside of this is that it can give an error if the graphics device dimensions are not large enough. If this occurs, we suggest you provide a filename in the command to save it to a pdf automatically sized to fit everything. Several parameters can also be set to affect how the plot appears. These are documented in the chromatogram help file.

```
chromatogram(hetsangerseq, width = 200, height = 2, trim5 = 50, trim3 = 100, showcalls = "both",
    filename = "chromatogram.pdf")
```

```
## Chromatogram saved to chromatogram.pdf in the current working directory
```

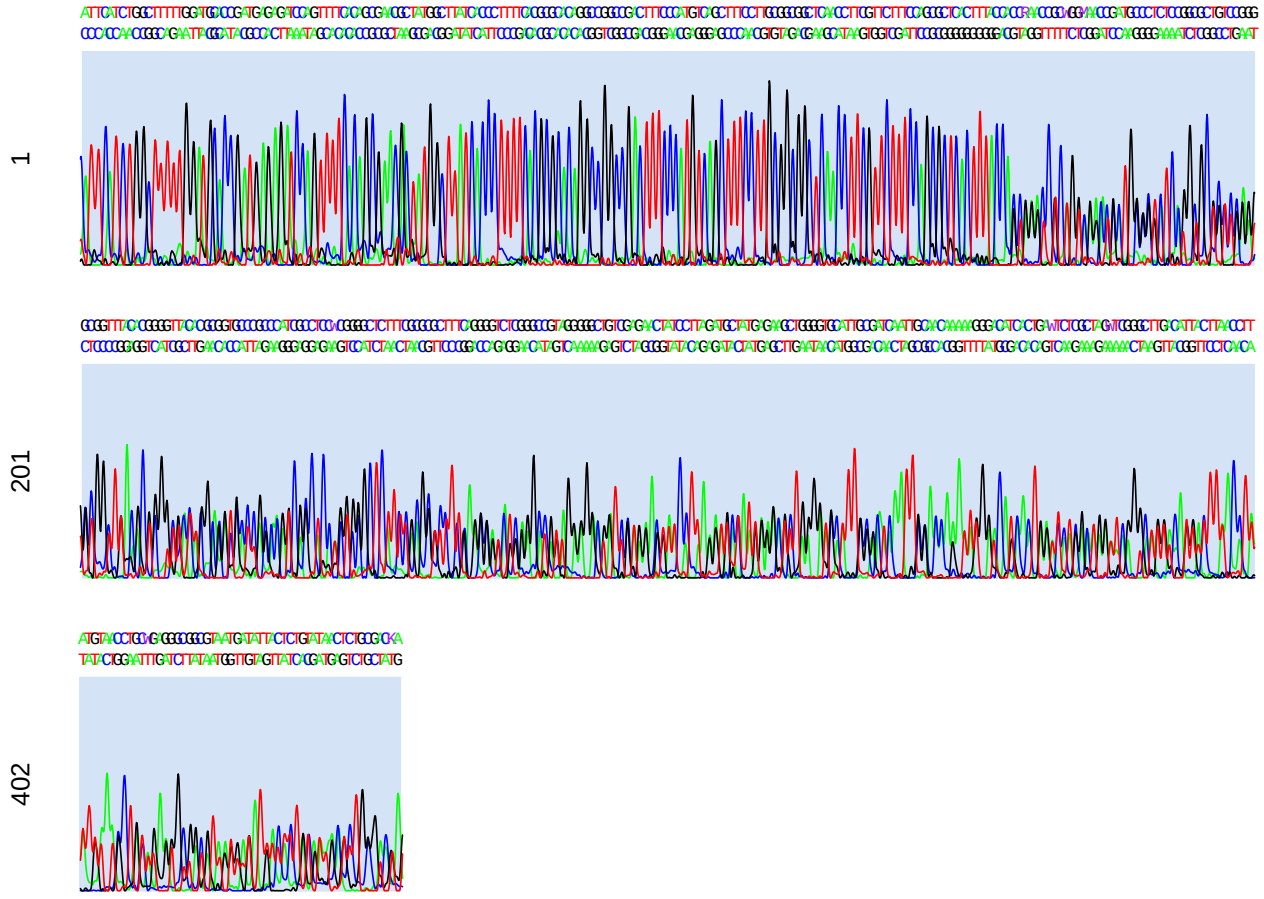

Figure 1: **Example Chromatogram.** Chromatogram of heterozygous sequencing results. Notice the double peak region caused by an indel.

5 Making Basecalls

As shown in the chromatogram, secondary basecalls are sometimes provided in ab1 files (Scf files are unable to show them). However, the exact nature of these calls is inconsistent. In the heterozygous.ab1 file used here, it is any peak near the primary peak, no matter how small. For example, base 100 (first base on the second line) has a primary call of "C" and a secondary call of "A", even though the A peak is very small and likely noise. In homozygous sequencing results, these calls should simply be ignored and are hidden in the chromatogram by default (`showcalls="primary"`). When heterozygous regions of the sequence are present, the `makeBaseCalls` can be used to determine whether a particular peak is homozygous or heterozygous and call the appropriate bases.

Let's use the chromatogram we created in the previous section as an example. The chromatogram contains a homozygous region from bases 1 to approximately 160, but then breaks down into a series of double peaks for the remainder of the chromatogram. This is due to an indel in one allele of the sequenced region. `makeBaseCalls` can be used to show this more clearly or to add the secondary basecalls if the data file does not contain them. The function essentially divides the sequence into a series of basecall windows and identifies the tallest peak for each fluorescence channel within the window. These peaks are converted to signal ratio to the tallest peak. A cutoff ratio is then applied to determine if a peak is signal or noise. Peaks below this ratio are ignored. Remaining peaks in each window are used to make primary and secondary basecalls.

```
hetcalls <- makeBaseCalls(hetsangerseq, ratio = 0.33)
hetcalls
```

```
## Number of datapoints: 16215
## Number of basecalls: 604
##
## Primary Basecalls: AGGCGCTGGAGTGGGTTTGACGGCGCATTCTTTTTTTTAGAGTATTATACATTCATCTGGCTTTTTGGA
## TGCACCGATGAGAGATCCAGTTTTCACAGCGAACGCTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCCATGTCAGC
## TTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCACTTTACCACCGAACCTCAGGCGAACGCTGGGCTCTCAGGCGCGCTCCAG
## GCCGGCTGACGCGGGGCGACTGGACGTGCCCGCAAGGCGCCTCCAGGGGGCTCTTCCGCGCTCCTTCAGGGATATCAGGCCGTAGGGGGC
## GATCCTGAACTATCCTTAAATGCCATTAAGCGCTGGGGTGCTTTGCGATCAATTGCACCAAATTGGGACATCACTGAATCTCGCTGGATC
## GGGCTGGACATGACTTAACCTATTTAAACCTGTAGAAGGCGGCGTAATGAGATTACTCTGCATAACTCTGGGACGAGTTGATCCTATATT
## ATCCTTTACATTACATAACATTGTAAAATATTGATTCAGATGTATTGGGAGGTGCCGTANCCTCACGATACCTATAGAAAGCCTCT
##
## Secondary Basecalls: AGGGGCMGGGGTGAATTTGAAGGCGCATTCTTTTTTTTAGACTATTATACATTCATCTGGCTTTTTG
## GATGCACCGATGAGAGATCCAGTTTTCACAGCGAACACTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCCATGTCA
## GCTTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCACTTTACCAGTAGGTCGCTGTAAGCTCATGCCGGATCCTGTGCTGCTG
## GATGTGGTTTAAACTCGTTTCTACGCGACCATTAGCCATCAGCACAKCTCCGCTCATTTAAGGGTTTCGAACCGGCGGGAGATAGTGAAG
## AATGTTGAAGAGGTACATAAGGATATAAGGGAATTTAAGAACAATTCGACTAAATTCGAAAAGAAATGATCAGAAATAGTCAAGAAAAAT
## AAAGTAATTTAAGTTTTTTACATTATGTATGCTACTTAGTGTTATATTGGTTTATGTTATTATGATGAGTTGCGTATATTTTGGTGTATT
## ATATAGTATACTATATTATATATTACTCGGTAAAACTCGGTTAAAACTCAGTTCTAATATANGATRGCTAGCCATCTAGAAAACCTCT
```

The resulting file now contains the maximum peak in the `@primarySeq` slot and the second tallest peak, if it is above the cutoff, in the `@secondarySeq` slot. If only one peak is above the cutoff ratio, then this call matches the primary basecall. If three peaks were above the cutoff ratio, then the peak with the maximum amplitude is the primary basecall and an ambiguous base code is used as the secondary basecall. The resulting chromatogram also shows this:

```
chromatogram(hetcalls, width = 100, height = 2, trim5 = 50, trim3 = 100, showcalls = "both",
    filename = "chromatogram2.pdf")
```

```
## Chromatogram saved to chromatogram2.pdf in the current working directory
```

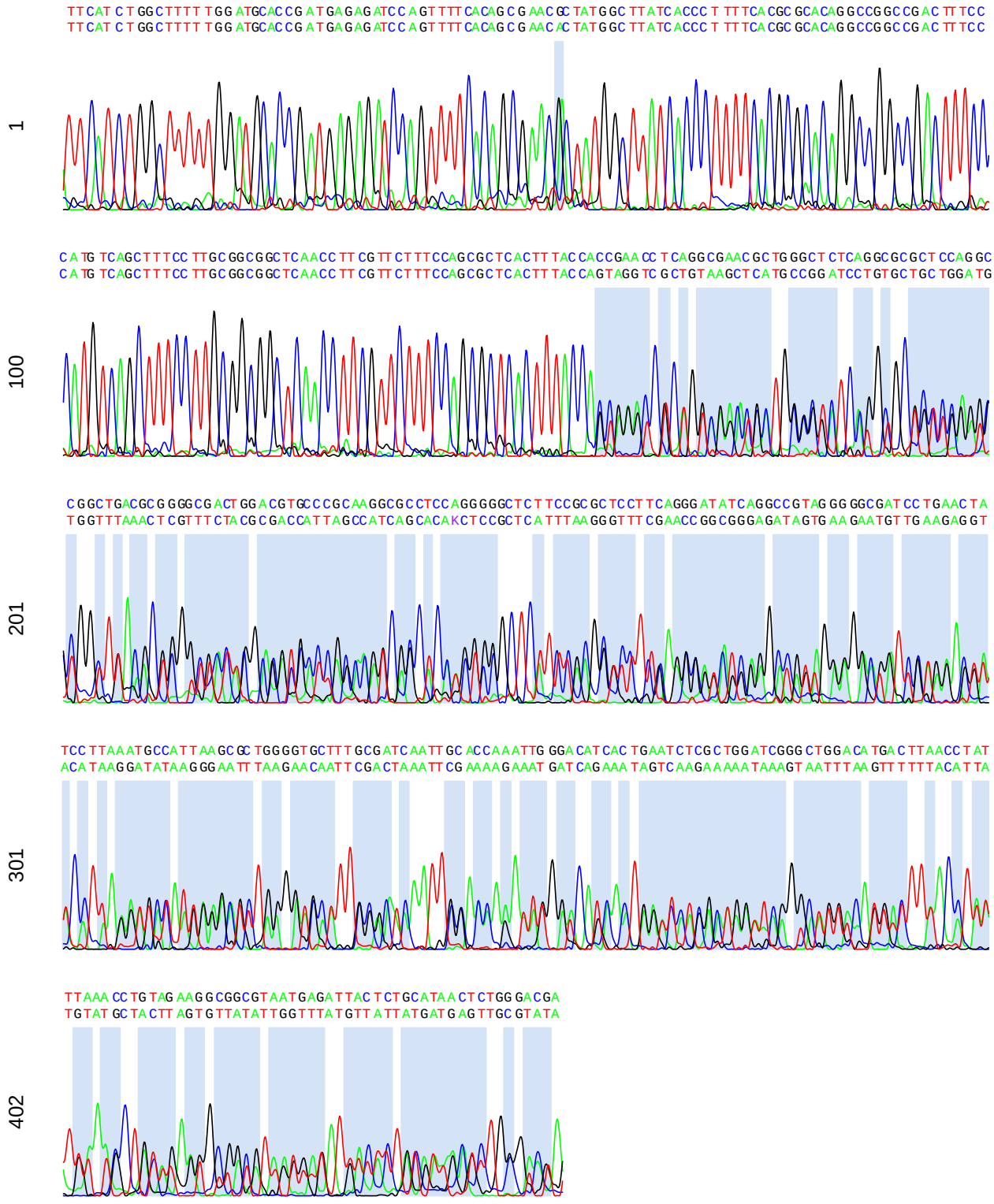

Figure 2: **Chromatogram from above after making basecalls.** Chromatogram of heterozygous sequencing results after making basecalls. Primary and secondary basecalls now match for homozygous peaks

6 Parsing Wildtype and Alternate Alleles

Although `makeBaseCalls` has fixed the primary and secondary peak calls. It still does not tell us anything about the nature of the mutation. For this, we need to set the allele phase using a reference base sequence from an online source or from another sequencing run on a homozygous sample. The examples used in this vignette are from heterozygous and homozygous siblings, so we will use the primary basecalls from the homozygous sibling (loaded earlier) as our reference. The beginnings and ends of these sequences do not need to match, but the reference should ideally encompass the sequenced region. `setAllelePhase` will then separate the primary and secondary basecalls into reference and non-reference bases at each position and set (`@primarySeq`) to the reference and `@secondarySeq` to the non-reference allele.

```
ref <- subseq(primarySeq(homosangerseq, string = TRUE), start = 30, width = 500)
hetseqalleles <- setAllelePhase(hetcalls, ref, trim5 = 50, trim3 = 300)
hetseqalleles
```

```
## Number of datapoints: 16215
## Number of basecalls: 604
##
## Primary Basecalls: AGGSGCHGGRGTGRRTTTGAMGGCGCATTCTTTTTTTAGASTATTATACATTCATCTGGCTTTTTGGA
## TGCACCGATGAGAGATCCAGTTTTCACAGCGAACGCTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCCATGTCAGC
## TTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCACTTTACCAGTAAACCGCTGGCGGATCATGCGCTCTCCGGTGCGGCTGAA
## GCTGGTTTACACGCGGCGCTTGGACATCACCACCAGGCGGCTCATCTGCGCTCTTTCAAGGGTCTCGAGCCAGAGGAGGATGTTGAGGAC
## GATCCTAAAGTTACATTAGAAGCTAAGGAGCTTTGGGATCAATTCCACTAAATTGGAACAGAAATGGTCATCACTAAATCAGGAAGGTAA
## GGTCTTTACATTATTTAACCTATTKWAWSCTRYWKARKGYKRYRTWRKKWKATKWYWYTRYRWWRMKYTGSGWMKAKTTKRKYSTATWWT
## ATMSTWTACWWTAYWWWAYATTRYWMRRTAWWRMTYSRKWWRWAYTSRGWKSTRMYRTANSMTVRCKAKMCMTMTAGAAARCCTCT
##
## Secondary Basecalls: AGGSGCHGGRGTGRRTTTGAMGGCGCATTCTTTTTTTAGASTATTATACATTCATCTGGCTTTTTG
## GATGCACCGATGAGAGATCCAGTTTTCACAGCGAACACTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCCATGTCA
## GCTTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCACTTTACCACCGGGTCTCAGTAAACCGCTGGCGGATCATGCGCTCTCC
## GGTGCGGCTGAAGCTGGTTTACACGCGGCGCTTGGACATCACCACCRGGCGGCTCATCTGCGCTCTTTCAAGGGTCTCGAGCCAGAGGAG
## GATGTTGAGGACGATCCTAAAGTTACATTAGAAGCTAAGGAGCTTTGGGATCAATTCCACAAAATTGGAACAGAAATGGTCATCACTAAA
## TCAGGAAGGTAAGGTCTTTACATTATKWAWSCTRYWKARKGYKRYRTWRKKWKATKWYWYTRYRWWRMKYTGSGWMKAKTTKRKYSTATW
## WTATMSTWTACWWTAYWWWAYATTRYWMRRTAWWRMTYSRKWWRWAYTSRGWKSTRMYRTANSMTVRCKAKMCMTMTAGAAARCCTCT
```

At this point, we could plot the chromatogram again, but it is more informative to align the resulting sequences to see how the alleles differ. Since *sangerseqR* depends on *Biostrings*, `pairwiseAlignment` can be used.

```
pa <- pairwiseAlignment(primarySeq(hetseqalleles)[1:400], secondarySeq(hetseqalleles)[1:400],
    type = "global-local")
writePairwiseAlignments(pa)
```

```
## ########################################
## # Program: Biostrings (version 2.34.1), a Bioconductor package
## # Rundate: Tue Jan 20 21:21:30 2015
## ########################################
## #======================================
## #
## # Aligned_sequences: 2
## # 1: P1
## # 2: S1
## # Matrix: NA
## # Gap_penalty: 14.0
## # Extend_penalty: 4.0
## #
## # Length: 410
```

```
## # Identity:     387/410 (94.4%)
## # Similarity:    NA/410 (NA%)
## # Gaps:          20/410 (4.9%)
## # Score: 649.2401
## #
## #
## #=====================================
##
## P1                1 AGGSGCHGGRGTGRRTTTGAMGGCGCATTCTTTTTTTAGASTATTATACA     50
##                     ||||||||||||||||||||||||||||||||||||||||||||||||||
## S1                1 AGGSGCHGGRGTGRRTTTGAMGGCGCATTCTTTTTTTAGASTATTATACA     50
##
## P1               51 TTCATCTGGCTTTTTGGATGCACCGATGAGAGATCCAGTTTTCACAGCGA    100
##                     ||||||||||||||||||||||||||||||||||||||||||||||||||
## S1               51 TTCATCTGGCTTTTTGGATGCACCGATGAGAGATCCAGTTTTCACAGCGA    100
##
## P1              101 ACGCTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCC    150
##                     || ||||||||||||||||||||||||||||||||||||||||||||||||
## S1              101 ACACTATGGCTTATCACCCTTTTCACGCGCACAGGCCGGCCGACTTTCCC    150
##
## P1              151 ATGTCAGCTTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCAC    200
##                     ||||||||||||||||||||||||||||||||||||||||||||||||||
## S1              151 ATGTCAGCTTTCCTTGCGGCGGCTCAACCTTCGTTCTTTCCAGCGCTCAC    200
##
## P1              201 TTTACCA----------GTAAACCGCTGGCGGATCATGCGCTCTCCGGTG    240
##                     |||||||          ||||||||||||||||||||||||||||||||||
## S1              201 TTTACCACCGGGTCTCAGTAAACCGCTGGCGGATCATGCGCTCTCCGGTG    250
##
## P1              241 CGGCTGAAGCTGGTTTACACGCGGCGCTTGGACATCACCACCAGGCGGCT    290
##                     ||||||||||||||||||||||||||||||||||||||||||| |||||||
## S1              251 CGGCTGAAGCTGGTTTACACGCGGCGCTTGGACATCACCACCRGGCGGCT    300
##
## P1              291 CATCTGCGCTCTTTCAAGGGTCTCGAGCCAGAGGAGGATGTTGAGGACGA    340
##                     ||||||||||||||||||||||||||||||||||||||||||||||||||
## S1              301 CATCTGCGCTCTTTCAAGGGTCTCGAGCCAGAGGAGGATGTTGAGGACGA    350
##
## P1              341 TCCTAAAGTTACATTAGAAGCTAAGGAGCTTTGGGATCAATTCCACTAAA    390
##                     |||||||||||||||||||||||||||||||||||||||||||||| ||
## S1              351 TCCTAAAGTTACATTAGAAGCTAAGGAGCTTTGGGATCAATTCCACAAA-    399
##
## P1              391 TTGGAACAGA    400
##                             |
## S1              400 ---------A    400
##
##
## #---------------------------------------
## #---------------------------------------
```

7 Conclusion

In this vignette, we have walked you through the basic functions in the *sangerseqR* package. This work is a work in progress and we hope to improve its functionality. For example, improving the base calling algorithm and adding an interactive chromatogram function. If you have any suggestions or requested features, please email Jonathon Hill at jhill@genetics.utah.edu.