RESEARCH SUMMARY

sangeranalyseR: making Sanger workflows reproducible in R

By Kuan-Hao Chao

Mar 1, 2021 · 7 min read · Updated Jul 1, 2026

Abstract

sangeranalyseR packages Sanger sequencing into a reproducible R workflow: grouping reads, trimming and inspecting chromatograms, assembling contigs, aligning results, generating quality-control views, and exporting FASTA files and interactive reports. The post traces the project from Robert Lanfear's ANU mentorship and the Bioconductor ecosystem to the object model that lets users automate analysis while still inspecting chromatogram-level evidence.

Chain-termination sequencing (Sanger et al., 1977) established a practical way to read DNA, and Sanger sequencing remains widely used for individual DNA fragments and for validating results from newer sequencing platforms. A Sanger run produces a chromatogram — the familiar four-colour trace of peaks — together with a called nucleotide sequence and per-base quality information.

The analysis ecosystem was less convenient. Many capable Sanger tools were commercial, restrictively licensed, or difficult to place inside a reproducible pipeline, while newer sequencing platforms increasingly had scriptable and interoperable tooling.

sangeranalyseR (Chao et al., 2021) was our answer: a free, open-source R package that makes Sanger processing a reproducible workflow rather than a sequence of manual steps.

Why the old way was painful

When I was at the Australian National University, near the start of my research career, getting from a folder of `.ab1` chromatogram files to a clean set of consensus contigs meant leaving the environment where the rest of the analysis often happened. Desktop sequence editors and trace-finishing tools were widely used, but the paper's point was practical: they were often expensive, restrictively licensed, cumbersome, or hard to integrate with other packages.

The project itself began with Robert Lanfear, my mentor at ANU. I was fortunate that he introduced me to it early in my research career, and that we could work through the engineering details of the new version together: how the Sanger workflow should be represented in R, how much should be automatic, where users needed control, and how the Shiny interface should preserve reproducibility rather than become a separate manual path.

There was already useful Bioconductor infrastructure (Huber et al., 2015). In particular, `sangerseqR` (Hill and Demarest) handled individual Sanger reads and provided the foundation we built on. But the missing layer was the full multi-read workflow: group reads into samples, trim them, assemble forward and reverse reads into contigs, align those contigs, inspect the results, and export the outputs for downstream analysis.

The workflow algorithm

sangeranalyseR's contribution is the workflow algorithm around Sanger data. It takes reads in ABIF or FASTA format, determines how they should be grouped into contigs, processes each read and contig with adjustable defaults, and keeps the full result inside R objects that can be inspected, revised, reported, and exported.

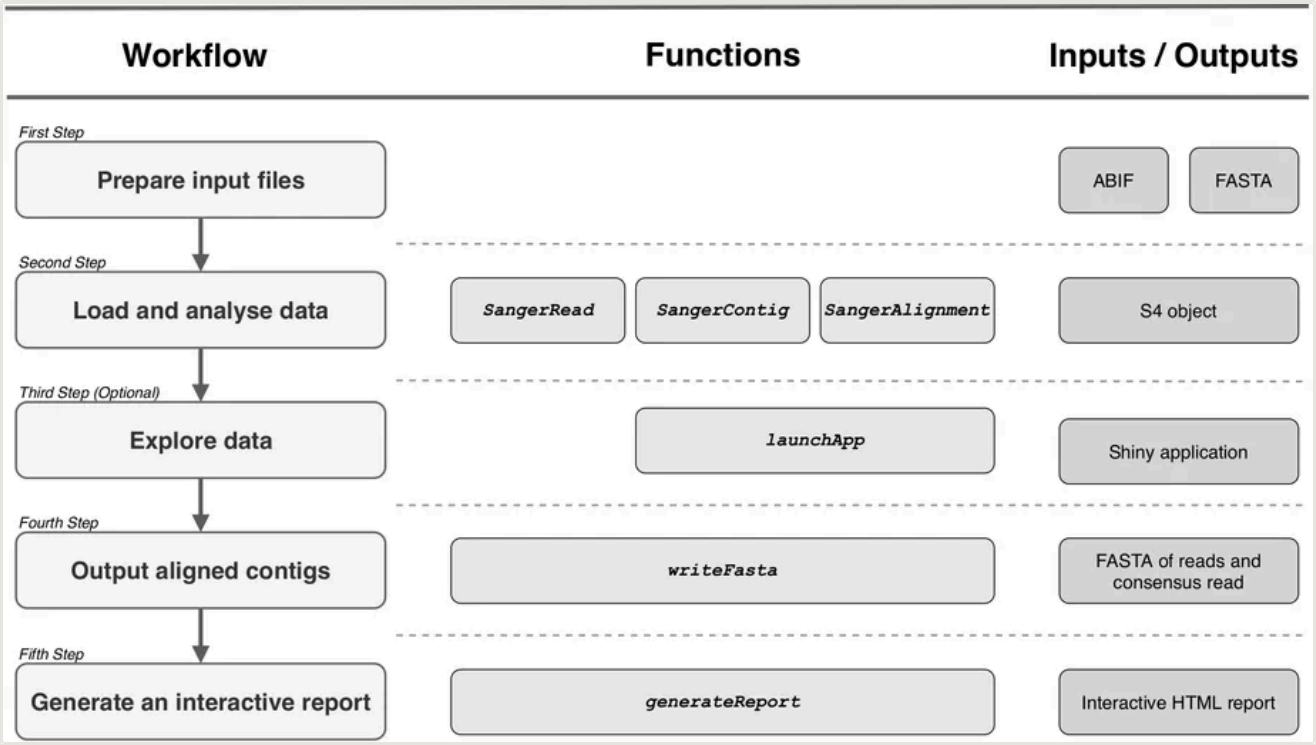

Figure 1. The sangeranalyseR workflow. Five steps take you from input files to a finished report: prepare `.ab1`/FASTA inputs; load and analyse them through three nested objects (`SangerRead` → `SangerContig` → `SangerAlignment`); optionally explore the data in an interactive Shiny app; write out aligned consensus contigs as FASTA; and generate a self-contained interactive HTML report.

The object model is the key design. `SangerRead` stores one read, its chromatogram, and trimming information. `SangerContig` stores the reads that form one contig, the consensus sequence created with DECIPHER (Wright, 2016), a dendrogram (a tree grouping the reads by similarity), and optional indel (insertion/deletion) or stop-codon information. `SangerAlignment` stores the collection of contigs, the contig alignment, and a quality-control tree. For downstream phylogenetics, the alignment and tree should be treated as QC summaries rather than final analyses.

The paper's small example is intentionally simple. A complete analysis of the example data can be driven in a few lines of R:

```r
library(sangeranalyseR)

# one call: group reads, trim them, assemble contigs, and align the contigs
aln <- SangerAlignment(inputSource = "ABIF",
                       parentDirectory = "./Allolobophora_chlorotica")

writeFasta(aln)       # write the aligned consensus contigs to FASTA
generateReport(aln)   # write a full, interactive HTML report
```

What it does

A few lines, but explicit underneath. That single `SangerAlignment` call does a sequence of concrete operations. It reads ABIF or FASTA inputs, groups reads into contigs by filename patterns or a CSV file, trims reads with either a modified Mott quality algorithm associated with Phred (per-base sequencing quality scores) or a configurable sliding-window approach, assembles forward and reverse reads into consensus contigs, and aligns the contigs together. The paper demonstrates this on eight annelid (Allolobophora chlorotica) samples, with one forward and one reverse ABIF read per sample.

Figure 2. A complete analysis of eight samples. **(A)** The few lines of R that drive it. **(B)** The ABIF input files, organized in folders and named so the package can infer contig groups and read direction. **(C)** The optional Shiny overview page. **(D)** The resulting multiple sequence alignment of the eight consensus contigs, and **(E)** a tree used as a quality-control view.

Interactive when you need inspection. The workflow is automated, but the paper is clear that users sometimes need to inspect reads and tune parameters. The Shiny applications expose that inspection layer inside R: users can view read alignments, Hamming-distance heatmaps (counting the positions at which two reads differ) between reads in a contig, trimmed primary and secondary sequences, per-base Phred quality scores (Ewing and Green, 1998), secondary peaks, trimming positions, and chromatograms with the 5′ and 3′ trimmed regions highlighted. If an amino-acid reference is supplied, the app also reports indels and stop codons in individual reads.

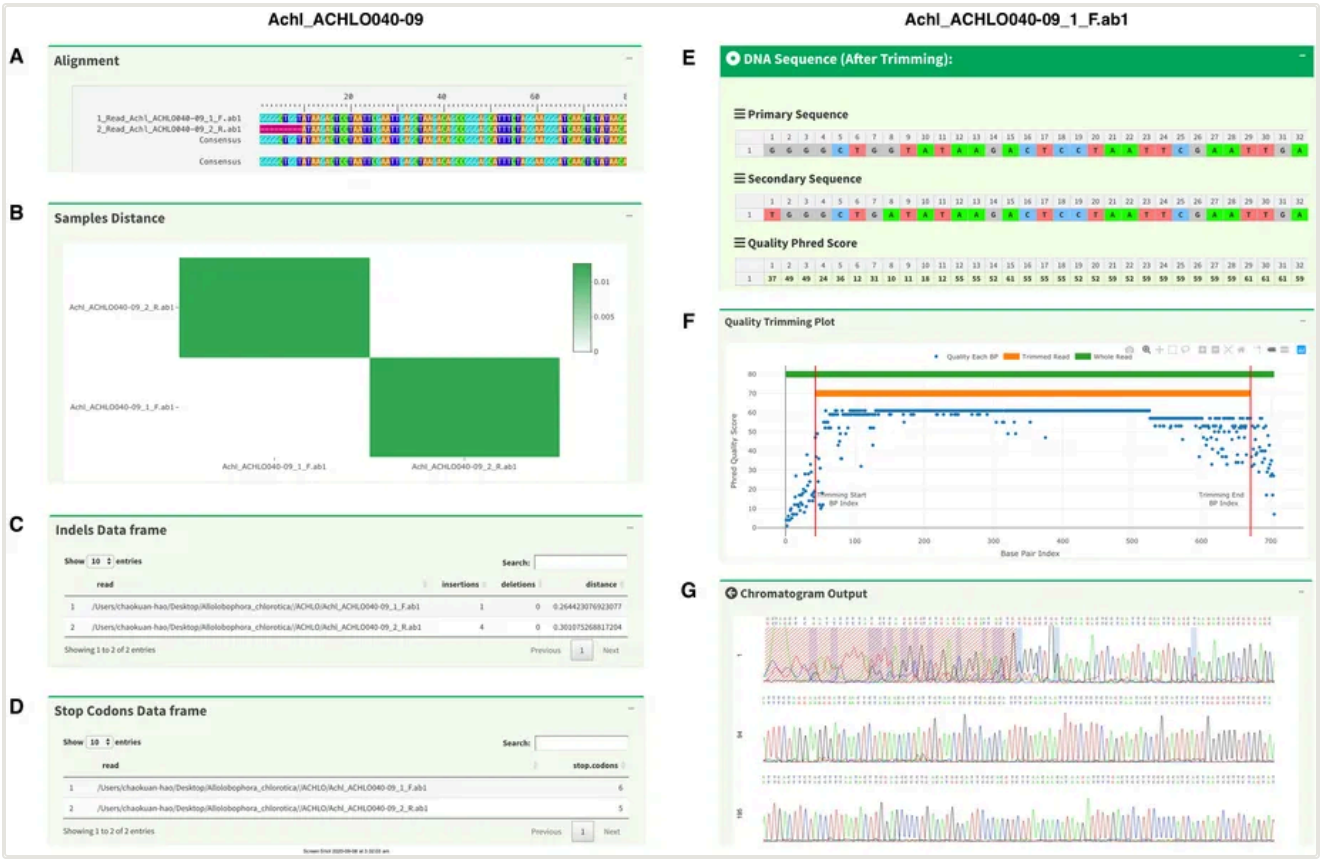

Figure 3. Inspecting the details in the Shiny GUI. The left column works at the contig level — the read alignment **(A)**, a Hamming-distance heatmap **(B)**, and reference-dependent indel **(C)** and stop-codon **(D)** tables. The right column drills into a single read — the trimmed primary and secondary sequences with quality scores **(E)**, an interactive trimming plot **(F)**, and the raw chromatogram with trimmed ends highlighted **(G)**.

The important implementation detail is reproducibility. When users adjust parameters in Shiny and save the updated S4 object, those parameter values are stored back in the object rather than being left as undocumented manual edits. Results can then be written as FASTA files — trimmed reads, consensus contigs, aligned consensus contigs, or all of them — and as interactive HTML reports arranged around reads, contigs, and alignments.

There is also a real input-format boundary. ABIF files contain the raw trace information needed for base calling, chromatogram plotting, and trace-based trimming. FASTA input is useful for interoperability,

including edited reads from other programs, but FASTA does not contain the raw chromatogram information, so those ABIF-dependent features are unavailable.

And it lives in R. The final outputs are standard files for other tools and structured R objects for further analysis. The package is open source, distributed through Bioconductor and GitHub, documented through ReadTheDocs, and tested as an R package, matching the paper's emphasis on a reproducible tool rather than a one-off script.

The bigger picture

The algorithmic idea in sangeranalyseR is a carefully defined processing workflow with a data model that matches the experiment: reads become contigs, contigs become alignments, and each level can be inspected, adjusted, exported, and reported. That is what made the package useful: it turned a common Sanger-processing task into something scriptable, interactive, and reproducible inside the R/Bioconductor ecosystem.

The limitation is important too. sangeranalyseR does not support manual editing of individual bases directly inside the Shiny chromatogram view. The paper is explicit about that boundary. If users need base-by-base editing while viewing chromatograms, they can use other tools for that step and then bring edited reads back into sangeranalyseR as FASTA input.

Working on sangeranalyseR helped set a pattern I have followed since: make the workflow explicit, keep the claims testable, document the tool, and return results in forms that other scientists can reuse.

Read the paper in Genome Biology and Evolution, install it from Bioconductor, browse the code, or work through the documentation. sangeranalyseR was built with Kirston Barton, Sarah Palmer, and Robert Lanfear.

References

1. Sanger, F., Nicklen, S., and Coulson, A. R. DNA sequencing with chain-terminating inhibitors. PNAS (1977). doi:10.1073/pnas.74.12.5463
2. Chao, K.-H., Barton, K., Palmer, S., and Lanfear, R. sangeranalyseR: simple and interactive processing of Sanger sequencing data in R. Genome Biology and Evolution (2021). doi:10.1093/gbe/evab028
3. Chang, W. et al. shiny: Web Application Framework for R. https://shiny.posit.co/
4. Huber, W. et al. Orchestrating high-throughput genomic analysis with Bioconductor. Nature Methods (2015). doi:10.1038/nmeth.3252
5. Hill, J. T. and Demarest, B. sangerseqR: Tools for Sanger Sequencing Data in R. Bioconductor package. doi:10.18129/B9.bioc.sangerseqR

6. Wright, E. S. Using DECIPHER v2.0 to analyze big biological sequence data in R. The R Journal (2016). doi:10.32614/RJ-2016-025

7. Ewing, B. and Green, P. Base-calling of automated sequencer traces using phred. II. Error probabilities. Genome Research (1998). doi:10.1101/gr.8.3.186