

Guidelines for Tragula: help for authors, editors, and reviewers

Robin K. S. Hankin 
University of Stirling

Veet Voojagig 
University of Maximegalon

Dan Streetmentoner
Independent Scholar

Abstract

This document is part of `guidelines.zip`, available on the journal website, <https://ojs.stir.ac.uk/tragula>. It outlines the academic and typesetting conventions for Tragula, and gives some guidelines for reviewers. Authors should download `guidelines.zip` and use file `guidelines.Rnw` therein as a starting point for a Tragula submission; gradually replace the content with your own work. The abstract goes here, not more than a few sentences. You can follow ISO 214:1976 if you wish, no obligation. Do not include citations here. Editorial policy in a nutshell: Tragula is a peer-reviewed journal that showcases novel mathematical applications of R.

Keywords: key word one, key word two, squares, R.

1. Author guidelines

Good submissions to Tragula showcase novel software that demonstrates or otherwise enhances mathematical knowledge; “mathematics” is interpreted in the broadest sense. The emphasis is on quality open-source software, written in R. Submissions are expected to include a software component (typically in the form of an R package), and this will be reviewed alongside the manuscript and considered jointly. Packages are expected to pass R CMD check cleanly and to follow the tenets laid out in the most recent version of *Writing R extensions*, see <https://cran.r-project.org/doc/manuals/r-release/R-exts.html>.

Tragula is fully open-access, meaning articles are free to read on the journal website. There are no fees for authors at any stage of the publication process.

There is some overlap between Tragula and both *The R Journal* and the *Journal of Statistical Software*. However, Tragula emphasises shorter communications that offer succinct insight; also there is less emphasis on statistics and more on mathematics. The sample issue of Tragula contains some manuscripts that were rejected by one or both of these journals but would be

welcomed here.

Tragula is an inclusive publication and welcomes submissions that add value to the R community through mathematics, or conversely add value to the mathematical community through use of R (or preferably both). R is a programming environment that includes a general-purpose language and can be used in the fields of topology, recreational mathematics, combinatorics, finite geometry, mechanics, etc. Thus, submissions may link with any branch of mathematics, or indeed the wider scholarly community, and uses of R in areas such as linguistics, art, or aesthetics are welcome. Criteria for acceptance are wide and would include beauty, elegance, computational slickness, sheer awesomeness, curiosity, pedagogical value, etc, etc. Only English submissions will be considered.

Submitted software and indeed the manuscript must be available under an open-source license such as GPL or BSD, and preferably maintained under a version control system such as `git`. Reproducibility is required, and using `Sweave` on `.Rnw` files facilitates this (Leisch 2002).

Manuscripts are expected to be carefully prepared, and clearly linked with the associated software. There is no upper or lower limit on the length of a submission but writing is expected to be terse.

Submissions will be fully and rigorously peer-reviewed as for any scholarly publication. As such, authors are expected to incorporate a literature review in which the submission is motivated and compared with similar existing work, and the benefits of the current approach outlined (one benefit might be that the submission is open-source and existing software is proprietary).

A good *Tragula* submission bears many similarities to an R package vignette. Hadley Wickham, writing at <http://r-pkgs.had.co.nz/vignettes.html>, phrases it well:

A vignette is a long-form guide to your package. Function documentation is great if you know the name of the function you need, but it is useless otherwise. A vignette is like a book chapter or an academic paper: it can describe the problem that your package is designed to solve, and then show the reader how to solve it. A vignette should divide functions into useful categories, and demonstrate how to coordinate multiple functions to solve problems. Vignettes are also useful if you want to explain the details of your package. For example, if you have implemented a complex statistical algorithm, you might want to describe all the details in a vignette so that users of your package can understand what is going on under the hood, and be confident that you have implemented the algorithm correctly.

(Although note carefully that it is not necessary to “solve a problem”: a manuscript might “verify a theorem”, “demonstrate a theory”, or even “popularize an approach”). Please do not supply a long list of command arguments that reproduce the Rd documentation: the ideal is a structured, heuristic narrative. If your package already contains a vignette, then converting it to a *Tragula* submission is easy, and should take less than 60 minutes.

- Strip out the boilerplate: remove standard setup code or basic installation instructions (not needed for *Tragula*.)
- Focus the narrative: highlight the motivation for the package, the design patterns, and the underlying data structures, and possibly the salient parts of the underlying R or C++ implementation

- Formatting: ensure headings and code snippets match **Tragula** typesetting guidelines.

1.1. Types of submissions

Currently, **Tragula** publishes two classes of submissions: articles and snippets. These have separate sections in issues. Both have the same style file and the same overall criteria for acceptance. However, snippets are short-form documents, expressing a single new idea in succinct and concise form. Four or five pages is maybe an upper limit.

The snippet format is particularly suitable in cases where an R package implements an algorithm that already has a peer-reviewed article. One might hold that writing a **Tragula** article is thus superfluous. However, this mindset treats software as a passive by-product of a published article. **Tragula** takes the view that value lies in the gap between a theory paper and the R package code. A theoretical paper explains the mathematics to statisticians—but a **Tragula** article explains the software architecture to R developers, and gives a way to highlight and showcase their software.

1.2. Layout

Authors are expected to provide germane R idiom that showcases their software with succinct and informative examples, as in:

```
> (1:10)^2    # short inline comments are fine

[1]    1    4    9   16   25   36   49   64   81  100
```

Readers should be able to execute these commands in an R session with minimal preparation (perhaps after loading a package).

Figures can enhance a submission very much. It is possible to use `\includegraphics` but this can make reproducibility difficult. It is better to generate PDFs on the fly, as in figure 1.

1.3. Typesetting

Submissions must be in clean seven-bit ascii and follow the house style. Currently only **article** styles are supported; start the file with `\documentclass[article]{tragula}`.

Generally, a submission will be an `.Rnw` file which may be processed using **Sweave** to produce a `.tex` file; then use `pdfLATEX` to create a PDF document. These steps may be performed more straightforwardly using **RStudio**, available at <https://www.rstudio.com/>. It is a good idea to use this file (`guidelines.Rnw`) as a starting point, and gradually replace the content with your own work.

The L^AT_EX 2_ε document class **tragula** is a (slight, cosmetic) modification of the **jss** document class, which is used for publications in the Journal of Statistical Software (JSS, <http://www.jstatsoft.org/>). There, the “instructions to authors” tab and the `jss.pdf` style guide give technical advice. I would like to thank Achim Zeileis, author of `jss.sty` and `jss.bst`, for his help and encouragement.

The **Tragula** style file imports three PDF figures: `tragula_word.pdf`, `tragula_knot.pdf`, and `ORCID_id.pdf`.

```
> plot((1:10)^2) # echo the commands used if practicable
```

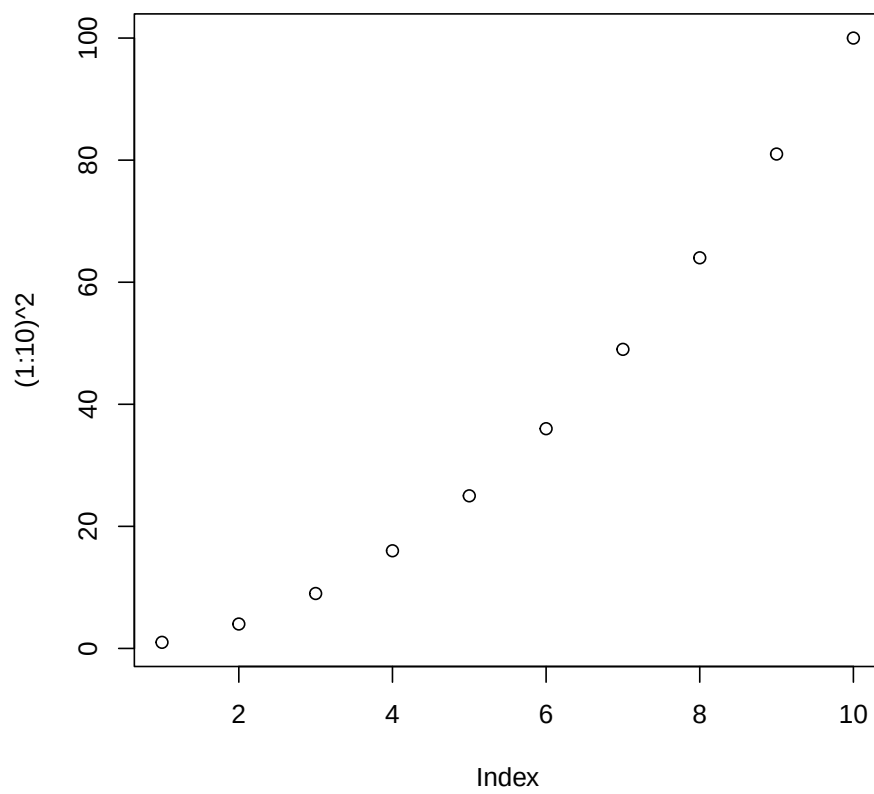


Figure 1: The squares of integers 1-10

- File `tragula_word.pdf` uses syncopate bold at 48pt (all capitals); the source document is available at

https://docs.google.com/document/d/e/2PACX-1vR6y-EVoIo4EOFY6P45AwN_-KCSH808KM8SBXsIn710UVk4_cJxImV8ilH4FBF_BCwN9_71BQ29458b/pub

- File `tragula_knot.pdf` is a five-fold rotationally symmetric ornamental knot created by the **knotR** package with name `ornamental20`. It is created in an R session by

```
> library("knotR")
> pdf(file="tragula_knot.pdf")
> knotplot(ornamental20, gap=2)
> dev.off()
```

(the production file has been cropped so that image fits on the page better).

- File `ORCID_id.pdf` is used under author affiliations and is available at

<https://orcid.org/trademark-and-id-display-guidelines>.

For *Tragula* the typesetting conventions are generally those of JSS, with one exception: do not capitalize unless absolutely necessary (that is, use upper case only for the first word of a sentence, proper nouns, and abbreviations); use sentence case for headings.

New sentences are indicated with *two* spaces; use `\colon` instead of “:” to indicate mappings. Commands such as `\author` and `\title` are unchanged from JSS; they are largely self-evident but the JSS style guide gives some more technical information.

Citation management must be via BibTeX; the style file uses **natbib**. Most of the time `\citet` or `\citep` will be appropriate, but macros `\citealt`, `\citealp`, `\citeauthor`, and `\citeyear` give greater control over punctuation. A good citation might be: “see [Lowry, Rosebrough, Farr, and Randall \(1951\)](#) for an example”. Subsequent citations of many-author (≥ 3) articles should be abbreviated as in “[Lowry et al. \(1951\)](#) provide details”; this is done automatically by the style file. Packages (whether or not on CRAN) should be indicated using the `\pkg` macro, as in “the **mvp** package ([Hankin 2018](#))”. Citing of software can be problematic but try to follow the citation guidelines provided by the software itself (in the case of R packages, `citation("pkgname")` gives the preferred BibTeX entry).

Inline equations should use dollars, as in $y=x^2$; but if an equation is too high (e.g., a fraction or complicated integral), use an equation environment. All displayed equations must be numbered and labelled, for ease of review, and use the `equation` environment, as in:

```
\begin{equation}\label{complex_gaussian}
\int_{z\in\mathbb{C}}\frac{\exp(-z^*\Gamma z)}{|\pi\Gamma|}dz=\frac{1}{|\Gamma|}.
\end{equation}
```

resulting in

$$\int_{z\in\mathbb{C}} \frac{\exp(-z^*\Gamma z)}{|\pi\Gamma|} dz = 1. \quad (1)$$

Sometimes a code segment includes an R variable `x` which refers to algebraic concept x , as in “if `x <- runif(1)`, then $0 \leq x \leq 1$ ”. Sometimes it makes sense to typeset the equation as “ $0 \leq x \leq 1$ ”, that is, using the same font for the symbol in the equation and the code segment. Decide this on a case-by-case basis, but aim for consistency if possible.

2. Reviewer guidelines

Reviewing can look very good on one’s CV as it is a valuable contribution to the scholarly ecology. Because of the editorial policy of *Tragula*, we expect reviewers to be drawn from a wider circle than usual for academic journals, so here we offer some guidance to reviewers who may not have been part of the formal peer-review process before.

If you have been asked to review a *Tragula* article, this means that the editor considers you to be an authority on the subject in question. However, note that being “an authority” in the context of *Tragula* editing can have many manifestations, and you should not feel that you need expertise in all of them to offer a competent review. A typical submission will showcase novel mathematical software and the editor might solicit your opinion as, for example,

- A **software** expert (perhaps you have written similar or overlapping software)
- A **mathematical** expert (perhaps your area of research is close to the subject of the submission)
- An **educational** expert (perhaps you teach material discussed in the submission)
- An **industry** expert (perhaps you use similar software in your work)

Having competence in any single criterion of the above list would mean that your opinion is valuable to us. You may wish to differ though, and before you accept or decline an invitation to review, consider the following questions:

- Does the submission match your area of expertise? Editors often make mistakes. Only accept if you feel you can give a constructive review.
- Is there either a real or perceived conflict of interest? If so, disclose this to the editor when you reply.
- Do you have time? Reviewing can take a long time: before you commit yourself, make sure you can meet the deadline (although extensions can be granted).
- Are you excited by the prospect of the submission? Are you likely to use the associated software? Do you have strong opinions about the subject matter? If so, your views are valuable to us.

If you need to find out more about reviewing and the peer review process, please feel free to ask for more guidance.

Please respond to the invitation as soon as you can (even you decide to decline); a delay slows down the review process and means more waiting for the author. If you do decline the invitation, it would be very much appreciated if you could suggest alternative reviewers.

2.1. Nice or nasty?

Reviewers are enormously influential in the scholarly world. Editors depend crucially on their reviews and the tone of a review can result in submissions being accepted or rejected; and authors often find that succinct and well-placed feedback from reviewers can catalyse much improved writing and programming.

Some reviewers tend to view submissions very favourably, then gradually add more and more negative elements to their review as they find them. Conversely, some reviewers start with a negative view and gradually become convinced that the article is high quality. Either approach is fine. Some questions to consider might include the following:

- Is the material new? Publication requires novelty, and a contribution to knowledge. Is the novel aspect of the submission clearly described?
- Does the abstract provide a fair summary?
- Is existing work clearly described, and is the new contribution clearly placed in this context?
- Is the software well-written? maintainable? comprehensible? Linked to published methodologies? Does it re-invent the wheel?
- Are limitations discussed fairly?

If the theoretical methodology for the submission has been published previously, that is OK. The value of the submission lies in the implementation and how it operates in the wider ecology of the R world.

You should feel free to say *good* things about a submission. If you find some aspect of a submission to be particularly interesting or insightful or otherwise impressive, please feel free to say so. Authors like this, of course, but so do editors as your comments help them to get a feel for the quality of the submission. Reviews should be critical in the sense that they involve careful informed judgement.

Criticism in the sense of passing unfavourable feedback or fault-finding is a necessary part of the review process, so do not feel the need to be overly polite or to protect the feelings of the authors. In particular, if you do not understand some part of the submission, do not feel embarrassed to say so: remember that you are an expert in the field, and editors will find such candid comments valuable. One important component of the peer review process is anonymity: your identity will not be revealed to the authors (or anyone else) so you can speak freely. Editors will mediate between authors and reviewers if necessary.

However, please do not be unkind in your reviews. Negative feedback is important: authors should know why their work is being rejected. Comments should be strictly confined to the submission, not the authors. Before sending a review, go through it and check for unnecessarily harsh, cruel, or severe statements. Check carefully for unclear, unfair or biased comments. Be polite, be firm, but be fair and be kind. If you recommend rejection of the manuscript, do so gently. If you recommend revision, try to motivate the authors to do the required work, as motivated workers often produce higher quality writing and software. Add a compliment or compliments if at all possible. Authors will be unhappy with a bad review (it is the editor's

job to deal with complaints) but should feel that their work was taken seriously and if possible should have a good idea of what would be required for an acceptable submission.

Try to be constructive if possible. Quite often reviewers will have suggestions that substantially add to the clarity and quality of a submission. Reviewers often make the mistake of not making explicit suggestions because they are “obvious”; but a direct and unambiguous suggestion is often a helpful spur to improvement. It also helps editors decide whether the reviews have been properly engaged with. Remember that the authors have right of reply and are free to decline to carry out a suggestion (but the editor may well take a dim view of this strategy, unless the authors give clear reasons for not doing so).

2.2. Review outcomes

A reviewer makes one of four recommendations:

Accept submission This recommendation is consistent with minor typographical errors which can be pointed out at this stage.

Request revisions This is indicated if the submission is high quality in the sense of making a respectable contribution to mathematical software but has flaws or omissions that may be addressed readily. The submission must be fundamentally well-written and structured; the software maintainable, clear, novel and interesting. Most ultimately successful submissions will come under this category following the first round of review.

Resubmit for review This means that the reviewers or the editor found value in the submission, but need to see substantial revisions before considering it again. Typically the submission is fundamentally flawed in a way that may or may not be rectifiable. The authors are invited to engage with the reviews and to submit revised work, although there is no guarantee that the submission will be accepted. Any resubmission will be given a new submission coding.

Decline submission This can be for one of two reasons: firstly the submission might be classified as an *editorial reject*, sometimes known as a *desk reject*. An editorial reject means that the submission is not in scope for the journal; it makes no statement about the quality of the work. The implication is that the authors should send the work elsewhere for consideration. Secondly, the submission might be in scope for the journal but be of low quality. Typically, this recommendation is for submissions with critical problems which have no clear fix; severely sub-standard work; patent nonsense; identical or similar software available and the current work presents no reasonable improvement.

The corresponding editor will come to a decision based on the available reviews, which may not all make the same recommendation.

3. Review and publication process

The submission and review process is managed via OJS but this may be modified in the future. Submissions are made in the first instance to the editor in chief, who will delegate (if in scope) to a member of the editorial board. Tragula review cannot be double-blind due to the nature

of open-source repositories. The software and manuscript will be reviewed jointly by at least two reviewers, one of whom can be the editor. As discussed above, referees' confidentiality is absolute and will be respected (although referees may, at their discretion, waive anonymity in the interests of a better publication). It is strongly suggested that software be presented as a working R package, which is expected to pass `R CMD check` cleanly. Decisions will be made in the light of reviews and rebuttals if appropriate. The typical flow is as follows:

1. Author submits manuscript and software to editor-in-chief.
2. If in scope, submission passed to corresponding editor.
3. Reviews solicited and collated.
4. Corresponding editor summarises the reviews and chooses one of the four review outcomes as representative of the reviews and the editor's own options.
5. Repeat point (4) as necessary, either with or without further input from referees at the editor's discretion.

Rejected manuscripts may not be resubmitted. Details of current editorial board membership can be found at <http://ojs.aut.ac.nz/tragula/>.

3.1. Rebuttal

The majority of submissions which are ultimately published go through at least one round of revision. One component of the process is the *rebuttal*. This is a point-by-point response to the reviews. Authors should consider the rebuttal as an opportunity to show the editors that any issues raised by the reviews have been dealt with. Editors might, at their discretion, ask for guidance from reviewers but generally it is expected that editors will be able to assess the rebuttal themselves. It is important that the whole review is included in the rebuttal so the editor can easily see the response to all the comments. Rebuttals can be provided in PDF form, or ascii. A typical rebuttal is a conversation between the reviewer and the author, who is responding to the points raised. One common convention is to typeset the rebuttal as an email conversation, marking reviewer's comments with a ">". A rebuttal might look like this:

```
> I found the paper to be carefully written and structured and an
> interesting read. The manuscript represents a respectable
> advancement of knowledge in computational foobology, and I would be
> happy to recommend publication.
```

Thank you.

```
> However, I have a few comments. Is equation 3 presented as new? It
> appears in Smith and Jones (1951). Is variable x real or complex?
```

Equation 3 is not presented as new, and we have added a citation to Smith and Jones as its source. Variable x is, as stated in the previous paragraph, real.

```
> Function subtract() contains a bug:
```

```
> > subtract(5,3)
> Error in add(x - y) : argument "y" is missing, with no default
```

Fixed:

<https://github.com/RobinHankin/add/issues/1>

```
> The documentation for function add() is confusing and incomplete,
> specifically it gives only one example and I could not understand
> how the function was intended to be used.
```

The vignette contains multiple informative use-cases.
Nevertheless we have included more documentation in `foo.Rd`:

<https://github.com/RobinHankin/add/commit/a984a20686b037877ae85b95e9e4d20ffb305869>

```
> The manuscript does not address Smithification which would be very
> desirable.
```

We feel that Smithification is beyond the scope of the submission, as this violates the assumption of Jonesness. However, we have included a brief discussion of Smithification in "Further work" at the end.

(the manuscript is imaginary but the links work). Observe how the reviewer makes comments ranging from individual typos right through to overarching views on the entirety of the manuscript. Authors should either accommodate the comments, or give a good reason why not. See how the rebuttal includes commit hashes which enable editors to evaluate any patches. If the manuscript itself is under version control, sometimes it makes sense to quote the commit hash, sometimes not. The github "issues" mechanism can be used too.

To help editors and reviewers identify revisions of the manuscript, authors should mark any text added or modified in response to review comments using blue text, implemented with the `\textcolor{blue}{...}` command from the `textcolor` L^AT_EX package. For example:

[Further work might include Smithification of results, although we note that this is not straightforward as it violates the Jonesness assumption.](#)

The blue highlighting will be removed for the final published version.

4. Post-publication maintenance

The accepted manuscript will be hosted online alongside the accompanying software; but it is inevitable that the software and documentation will be modified post-acceptance. The `\Repository` field appears on the final document and points readers to the accepted version and furnish easy access to subsequent modifications. Tragula papers are "static", that is, the accepted version does not change with time. However, the associated software will almost certainly be modified.

We strongly recommend that every R package associated with a Tragula article includes a

vignette. This vignette should be based on the *Tragula* article, although it might include extra material. A good way to do this is to use the JSS headers, so that R’s package management systems can find the correct style files (use the `nojss` option: “`\documentclass[nojss]{jss}`”). An example may be found at <https://github.com/RobinHankin/permutations> or <https://CRAN.R-project.org/package=permutations>). In the vignette, cite the *Tragula* article along with the other references. This device maintains a distinction between the static *Tragula* article and the mutable vignette; and lets readers know that the work has been rigorously peer-reviewed.

Post-acceptance, you should include pointers to the *Tragula* article in the regular R package documentation: include a reference to the article in the `\references{}` section of any appropriate `.Rd` files; and also include a `CITATION` file in the `inst/` directory, which will be given to you in a standard format when the article becomes live.

5. Notes on use of Open Journal Systems (OJS)

Tragula is hosted on an Open Journal Systems (OJS) platform; documentation is available [here](#). Currently we use OJS 3.5.0.3. The standard workflow flowchart given [here](#) does not explicitly include the process of galley production, which is managed separately via email correspondence with the author. Usually, authors should submit at least the following documents:

- The manuscript itself in PDF format (sent to reviewers)
- The resulting `tex` file and any supplementary files needed by pdfL^AT_EX. Typically these will include a `.bib` file and and PDFs needed for figures, produced by the `.Rnw` (not sent to reviewers)
- Any R code needed to establish reproducibility. This will usually be in the form of an R package or standalone R code (sent to reviewers)

This makes proof-reading and copyediting stages much easier, in particular because updated files can be designated as such at upload. The submission process presents a checklist that includes some technical requirements should be followed. Note that the source file, in `.Rnw` format, is not absolutely required—although it is a convenient and acceptable way to ensure reproducibility.

References

- Hankin RKS (2018). *mvp: fast symbolic multivariate polynomials*. R package version 1.0-0, URL <https://github.com/RobinHankin/mvp.git>.
- Leisch F (2002). “Sweave, part I: mixing R and L^AT_EX.” *R News*, **2/3**, 28–31.
- Lowry OH, Rosebrough NJ, Farr AL, Randall RJ (1951). “Protein measurement with the folin phenol reagent.” *Journal of Biological Chemistry*, **193**, 265–275.

Affiliation:

Robin K. S. Hankin (corresponding author)
University of Stirling
Scotland
email: tragula@stir.ac.uk

Veet Voojagig
Faculty of Divinity and Water Polo
University of Maximegalon
email: vvoojagig@maxi.ac.ursa

Dan Streetmentioner



<https://ojs.stir.ac.uk/tragula>

Published by the University of Stirling
Volume 0, Issue 0, 0000

<https://stir.ac.uk/>

Submitted: 0000-00-00
Accepted: 0000-00-00