repliMAT: *A* Guide to Reproducible MATLAB

David Wilby

3 May, 2024

Table of contents

Welcome! **5**
How to use this guide . . . 5
Contributing . . . 5
Issues . . . 6
Contributing code . . . 6
Contributors . . . 6

1 Introduction **7**
1.1 What is research reproducibility? . . . 7
1.2 Open Research & Reproducibility . . . 7
1.3 Is it worth the effort? . . . 8
1.4 Why MATLAB? . . . 8

I Writing Cleaner Code **10**

2 Introduction **11**

3 Comments **12**
3.1 For code readability . . . 12
3.2 Commenting out code . . . 13
3.3 Comments for controlling execution - don't do it! . . . 14

4 Variables **15**

5 Functions **16**

6 Project Organisation **17**
6.1 Directory Structure . . . 17
6.1.1 Templates . . . 18

7 Documentation **19**

II Reproducibility **20**

8 Introduction **21**

9 Dependencies **22**

10 Versioning **23**

11 Projects **24**

III Exercises **25**

12 Introduction **26**
12.1 How to work with these exercises 26

13 Getting Started **27**
13.1 Technical setup 27
13.1.1 MATLAB versions 27
13.2 How to use these materials 27
13.2.1 Where to start 27
13.2.2 Prerequisites 27

IV Extra Credit **28**

14 Testing **29**

15 Version Control **30**

Appendices **31**

Code Style Cheat Sheet {.appendix, .unnumbered} **31**

Naming Conventions **32**
Variables 32
Constants 32
Structures 33
Functions 33
General 33

Files and Organization **34**
M-Files 34
Input/Output 34

Statements **35**
Variables and constants 35
Global Variables 35

Loops . . . 35
Conditionals . . . 36
General . . . 36

Layout, Comments, and Documentation **37**
Layout . . . 37
White Space . . . 37
Comments . . . 37
Documentation . . . 38

References **39**

License **40**
License text . . . 40

Privacy Policy **48**

Welcome!

Welcome to **repliMAT**!

:construction: **This book is very much under development** :construction:

This is a resource for learning and teaching about developing reproducible and sustainable code in the MATLAB programming language.

Here you will find content, exercises and videos intended to either be followed alone or with a group, or taught as part of a workshop.

The materials are primarily aimed at researchers and their specific demands, but should be applicable to all uses of MATLAB.

How to use this guide

This guide is split into two main parts: a reference guide and a set of exercises.

The intention is for the reference guide (begin here) to act as a place to learn about the reasoning behind concepts in reproducible MATLAB project design, as well as somewhere to come back to as a reference when working on your own project.

The exercises are a worked-through set of examples that can either be taught as part of a workshop or followed on your own.

The third part of these two main parts (:roll_eyes:) is the extra credit section, containing concepts that aren't *absolutely* essential for reproducibility, but are considered good/excellent practice in programming.

Contributing

Contributions to the **repliMAT** materials are welcomed! Please follow the guidance below prior to making contributions to ensure that your kind efforts do not go to waste.

The project's source code and development is managed at its GitHub repository. There are a few ways to contribute, depending on whether you want to make changes to the source code or not.

In all interactions, please abide by the code of conduct

Issues

Open a new issue to describe a bug, error or to request changes.

Contributing code

If contributing source code changes to the project please follow the following workflow:

1. Make a fork of the repository on GitHub.
2. Clone your fork to your local machine and make a new branch with a name relevant to the task you're working on.
3. Make some changes and ensure that the pages render as expected by following the instructions in the README to render the materials.
4. Commit those changes with meaningful commit messages.
5. Push your branch to GitHub and open a pull request against the upstream repository's `main` branch.
6. In the pull request description, please reference the issue that you are resolving.
7. Someone will review your pull request and hopefully it will be merged! :tada:

Contributors

David Wilby

Dan Cummins

Michael Tso

Patrick J. Roddy

Gaurav Bhalerao

1 Introduction

1.1 What is research reproducibility?

According to The Turing Way's definitions[1], the term *reproducibility* refers to performing the *same* analysis on the *same* data for the *same* result. Other terms such as replicability and generalisability are used to refer to using different analyses or different data. This may not be your definition, but it's the one meant here and derived from the research done by the authors of The Turing Way (an exemplary guide to reproducible research software).

Research that is reproducible has many benefits, it:

- is easier to validate (perhaps even *possible* to validate),
- has more long-term validity,
- is more extensible,
- reduces repetition,
- decreases likelihood of losing methodology,

among many others.

Code is great for research reproducibility in lots of ways. Code describes a proceduralised sequence of operations to some data, with (arguably) zero ambiguity - great! That's just what we need for research. Where appropriate, code is an excellent solution to capturing and reproducing the steps taken to go from some raw data/input to some research conclusions.

However, in practice it isn't always as easy as that. So this guide aims to provide researchers who code with the tools they need to make their MATLAB-based research (more) reproducible.

1.2 Open Research & Reproducibility

Open research is the idea that the entire research lifecycle should be transparent for all to see. As an approach, open research continues to grow and many funders now stipulate that the research that they fund must follow open principles including the open availability of publications, data and code. How does this fit in with reproducibility? I would argue that if you are required to make your code available, whether that's for a publication, thesis or just to share it with a colleague, it would be a good thing for the code to actually work, and for it

to be relatively easy to make it do so. It's commonplace in research to obtain some code and spend a significant period of time attempting to run it successfully, let alone validating that it produces something accurate. Therefore reproducibility is an important component of open research, though it need not be complicated.

1.3 Is it worth the effort?

There's no denying that learning and implementing the approaches required to enable reproducible research is *yet another* thing to do. As researchers we already have so many different skills to master: domain expertise, writing, graphic design, experimental design, public speaking, statistics. The list goes on. So why should we *voluntarily* make our programming practices even more involved than they already are. Many of us don't even *like* programming, so can't we just get the job done?

Yes and no. As mentioned above, many funders and publishers require open research practices, so as far as I'm concerned, the code should actually work when we share it. Understandably, people are often resistant to making their practices even more complicated. The culture still hasn't really shifted to expecting fully reproducible research code. However, I would give most researchers the benefit of the doubt and assume that most want their work to be verifiably correct.

Moreso than just being "the right thing to do" - making one's research more reproducible has all sorts of benefits. Let's expand on the benefits mentioned above. If we want to check that our work is correct, being able to send it to a colleague who can then actually make it run is a huge advantage. I'm sure we've all been in the situation where someone sends a zip file of code with no instructions and we spend ages trying to make it work. Reproducible research saves us and our colleagues and collaborators time by simplifying this process, allowing us to actually get on with the research we're interested in. If we publish some work and people are able to make it work, they're more likely to build upon it and cite our work. This accelerates research and gets us the recognition we deserve. Importantly, when we come back to our work in the future, we may actually understand what we did and be able to build upon it ourselves. Imagine that!

1.4 Why MATLAB?

Why is MATLAB a tool that we should care about when it comes to reproducible practices?

Because MATLAB is a popular language in research.

That's it.

Whatever your technical opinion of a language, or whether it is proprietary or open source, for all sorts of reasons, MATLAB is used by a lot of researchers. It has a relatively long history as being a tool with a lot of useful mathematical and analytical features, is relatively user friendly and a large number of universities have a license.

But, possibly because it's a proprietary language, most of the guidance and documentation comes from the organisation that develops it, MathWorks.

In comparison to other programming languages currently popular in research such as Python & R, the availability of guidance around reproducibility is relatively limited.

So that's why this guide has been developed, to allow those researchers who currently use MATLAB to make their research more reproducible and easier to share.

Not because I think MATLAB is the best, or the worst. I just think that all research should aim to be as reproducible as possible and that you should use the best tool for the job, even if that's just the one that you currently know.

Many researchers using MATLAB have said to me:

> I *know* I should rewrite this in python so that I can share it.

But realistically, the likelihood is that you'll just move on to your next project. The demands and incentives of the research world mean that investigating a new thing carries much more value than refining an existing project to a higher standard.

So let's make the projects we're working on **now** as good as they can be.

Part I

Writing Cleaner Code

2 Introduction

Whilst this is primarily a guide to writing more *reproducible* MATLAB, I would argue that writing *cleaner* code is an important step. It can make your code easier to read and follow (for you as well as others) - making it simpler to spot errors and mistakes, as well as for others to make contributions and improve or build upon your code.

We'll explain this in several steps:

- variables
- functions
- codebase organisation (it's not just the programming that helps)
- how to document your code.

One significant reference here is the book "The Elements of MATLAB Style" by Richard Johnson[2] which acts as the *de facto* community standard for style when it comes to programming in MATLAB. As well as the book itself, there are several resources based on it that have the core elements of its teachings on code style in MATLAB:

- cheat sheet - available from MATLAB file exchange - much better than reading the actual book
- updates to the book - also on file exchange

The cheat sheet is also reproduced on this site here for ease of searching and for quick reference.

With any programming language, it pays to disavow yourself of any *opinions* that you may have and save yourself the time of thinking about how to write your code by just following the standard practices of the community. Your code may **run** perfectly well. You may think that a certain way of function-naming is slightly better. But if we all stick to the same standards, we all have an easier time of reading each other's code and don't have to waste time arguing (yes, this is something that people argue about.)

3 Comments

3.1 For code readability

One element of reproducible, or more specifically, sustainable code is its *readability*. In an ideal world, all code would be implicitly readable: its variable, function and class names would all explain themselves and a reader would immediately understand the authour's intent (always remember that more often than not, reader and authour are the same person!). However, in reality, it's often necessary to clarify the intent of some particular logic in line with the code, and this is where comments come in handy.

```
% Comments are lines of code that do not get executed.
% In MATLAB, comment lines begin with a percent (%) symbol.
a = 42; % They can occur on the same line as executable code.
% Everything after the % is a comment.


disp(a)


%{
Multiple lines of
code can be
commented out
as a block.
Using curly braces.
%}


% Comments can also be used to stop lines of code from executing.
% b = 84;
```

To make this quick and easy, in the MATLAB editor you can use the following key combinations to **add** or **remove** comments, respectively:

- Ctrl+R and Ctrl+Shift+R on Windows

- Cmd +/ and Cmd +Shift+/ on MacOS
- Ctrl+/ and Ctrl+Shift+/ on Linux

Note: This depends on your MATLAB keyboard settings and can be changed to suit your preference.

Comments are one element of good coding practice that I find most developers (that's people who write code, i.e. you!) are familiar with, so this shouldn't come as too much of a shock. However, they aren't a substitute for the generally good coding practices described throughout this guide.

Large sections of comments should be avoided and if you find that your code needs them, it may be an indication that restructuring your code or using documentation (described later) instead may result in something easier to understand.

3.2 Commenting out code

As mentioned above, comments can be used to prevent a line from executing, which is a useful tool during development. This allows us to test different parameter options and approaches and can be helpful when debugging.

This being said, when we come to share our code, be this for publishing, dissemination or sending to colleagues, commented-out lines of code are a huge cause for confusion and potential errors in execution.

Reproducibility is a difficult challenge, hence this guide needing to exist, so when someone receives some code it will often not execute without errors (that's the problem we're trying to solve here). This leads the user to then look for sources of error and commented-out lines of code create ambiguity. Should this line be executed? Did my colleague intend to run this line or not?

Therefore, in most cases it's a good idea to avoid commenting out actual code when it will be shared.

In practice, this isn't always practical, so try to add other comments justifying why a line would be commented out at all. And the more packaged up and professional the codebase, the less you should have commented-out lines of executable code. I wouldn't expect to see this in a finished toolbox or package, for instance.

3.3 Comments for controlling execution - don't do it!

Following on from this, it's common during development to use comments for controlling execution, setting options and changing parameters. Whilst this will make sense to the person writing the code (at the time) - other users will rarely be able to understand or have confidence in this approach.

If you're sharing code and telling the user to "just comment out this line" to get the result they want - this is an indication that you should either split the job of this code into separate functions, or use conditional statements to control the execution.

4 Variables

:construction: *Nothing here yet! Check back later.* :construction:

5 Functions

:construction: *Nothing here yet! Check back later.* :construction:

6 Project Organisation

6.1 Directory Structure

One thing that makes a project difficult to reproduce is a disorganised project folder. Adhering to a consistent structure convention mitigates others from opening the project folder to find a big heap of confusingly named scripts.

Use folders/directories to organise your project, so that it is obvious (to you and anyone else who uses your code) where to find what is needed.

At the very top level of your project, why not have something like the following structure?

```
source/           # source code
    @MyClass/     # a class directory
    a_module/     # a directory containing a group of functions or classes with some commo
    utils/        # a directory containing utility functions, for example
data/             # directory to contain your data
    raw/          # all your raw, unprocessed data
    processed/    # any data that you have altered
output/           # directory to contain your output
    figures/      # keep the figures that you produce from your code here
    reports/      # a clear folder for your papers or reports for the project
docs/             # documentation
tests/            # software tests for the project
README            # readme file! (essential)
LICENSE           # license file (essential)
```

This structure follows several conventions that are common across languages, such as keeping a `README` file in the top-level of your project's directory where users expect to find it, keeping your source code in a directory named `source` and your data and outputs separate from the code.

> **Note**
>
> This type of structure works well for relatively straightforward projects such as a piece of analysis or simulation, but if you're building a package, toolbox or similar, there will be more appropriate and conventional structures to use.

6.1.1 Templates

One way of saving time is to make use of a project template, we have made one available on GitHub here: matlab-project-template which contains this directory structure as a template with placeholder information, designed specifically for MATLAB projects.

If you're not familiar with Git, you can download this repository as a ZIP file, or just make it yourself in your project by creating the relevant directories.

7 Documentation

:construction: *Nothing here yet! Check back later.* :construction:

Part II

Reproducibility

8 Introduction

:construction: *Nothing here yet! Check back later.* :construction:

9 Dependencies

:construction: *Nothing here yet! Check back later.* :construction:

10 Versioning

:construction: *Nothing here yet! Check back later.* :construction:

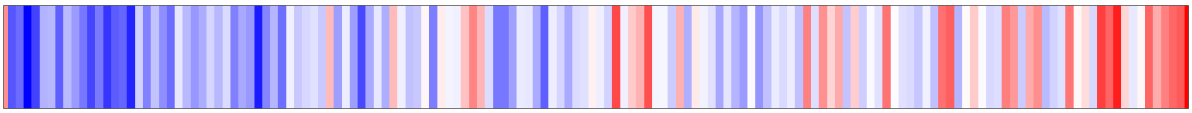
11 Projects

:construction: *Nothing here yet! Check back later.* :construction:

Part III

Exercises

12 Introduction



In this series of exercises, we're going to work with some research code which may be typical of what many of us would write and walk through how to transform this into a significantly more reproducible project.

We'll work with some publicly available data on surface temperatures in the USA from the late 19th to the early 21st century and create some interesting visualisations.

To start with, the project will hopefully look similar to what many of us would develop in our work, but it may not be as reproducible as it could be. By the end, we should have a project that we can easily share with anyone who can run it and generate the same results, and even use the code with different data.

12.1 How to work with these exercises

You'll either need a recent copy of MATLAB installed on your computer, or you can follow the links alongside the exercises to open the examples in MATLAB online.

To get hold of the code and use it on your own computer, you can use `git` to "clone" the code from GitHub, or just download it. Instructions are in the "Getting started" section.

13 Getting Started

13.1 Technical setup

You'll need an installation of MATLAB, preferably a recent version (i.e. in the last year or so). Where a specific version is needed, or doesn't work with an example, it should be indicated alongside the exercise. See Section 13.1.1 below for more info on versions.

Follow the instructions on MathWorks.com to install.

No specific toolboxes are required.

Note There is no guarantee that anything in these materials will work with Octave. In fact it probably won't. But let us know if you try!

13.1.1 MATLAB versions

A new version of MATLAB is released twice a year. The version numbers are comprised of the letter *'R'* followed by the calendar year and *'a'* if it's the first release in the year and *'b'* for the second. e.g. *R2022b*. Each version has improvements from the last and makes changes. It's a good idea to be using the most recent version of MATLAB in most cases.

13.2 How to use these materials

13.2.1 Where to start

Anywhere you like! Hopefully the structure of the exercises included here means that you can dip in to any point that takes your fancy.

13.2.2 Prerequisites

These materials assume that you're already familiar with the basics of programming in MATLAB. Variables, arrays, loops, reading in data and making plots *etc.*

Part IV

Extra Credit

14 Testing

:construction: *Nothing here yet! Check back later.* :construction:

15 Version Control

:construction: *Nothing here yet! Check back later.* :construction:

Code Style Cheat Sheet {.appendix, .unnumbered}

This is a reproduction of the MATLAB Style Guidelines Cheat Sheet compiled by Jason Nicholson and Edited by Richard Johnson from the book *The Elements of MATLAB Style*[2].

Matlab Style Guidelines Cheat Sheet (https://www.mathworks.com/matlabcentral/fileexchange/45047-matlab-style-guidelines-cheat-sheet), MATLAB Central File Exchange. Retrieved February 5, 2024.

Reproduced here within this book for ease of use by its readers.

Naming Conventions

Variables

- Variable names should be mixed case starting with lower case: `velocity`, `angularAcceleration`.
- Variables with a large scope should have meaningful names. Variables with a small scope can have short names:
  - Small scope: `x`, `y`, `z`
  - Large scope: `velocity`, `acceleration`
- The prefix `n` should be used for variables representing the number of objects: `nFiles`, `nCars`, `nLines`
- Use a convention on pluralization consistently: `point`, `pointArray`
- Variables representing a single entity number can be suffixed by `No`: `tableNo`, `employeeNo`
- Iterator variables should be named or prefixed with `i`, `j`, `k` etc. e.g. `iFiles`, `jColumns`
- For nested loops, the iterator should be alphabetical order and helpful names e.g.

```
for iFiles = 1:nFiles
    for jPositions = 1:nPositions
        ...
    end
end
```

- Avoid negated boolean variable names: ~~`isNotFound`~~ instead use `isFound`
- Acronyms, even if normally uppercase, should be mixed or lower case. Use: `html`, `isUsaSpecific`
- Avoid using a keyword or special value name. Just don't do it.

Constants

- Named constants should be all uppercase using underscore to separate words: `MAX_ITERATIONS`, `COLOR_RED`
- Constants can be prefixed by a common type name: `COLOR_RED`, `COLOR_GREEN`, `COLOR_BLUE`

Structures

- Structure names should be mixed case and begin with a capital letter: `Car`, `DumpTruck`
- Do not include the name of the structure in the field name. Use `Segment.length`. Avoid `Segment.segmentLength`

Functions

- The names of functions should document their use.
- Names of functions should be written in lower or mixed case: `width()`, `computeTotalWidth()`
- Functions should have meaningful names. Use `computeTotalWidth`. Avoid `compwid`.
- Functions with single output can be named for the output: `shearStress()`, `standardError()`
- Functions with no output argument or which only return a handle should be named after what they do: `plotfft()`
- Reserve the prefix `get/set` for accessing an object or property: `getobj()`, `setappdata()`
- Reserve the prefix `compute` for methods where something is computed: `computeSumOfResiduals()`, `computeSpread()`
- Reserve the prefix `find` for methods where something is looked up: `findOldestRecord()`
- Reserve the prefix `initialize` for instantiating an object or concept: `initializeProblemState()`
- Reserve the prefix `is` for boolean functions: `isCrazy`, `isNuts`, `isOffHisRocker`
- Use complement names for complement operations: `get/set`, `add/remove`, `create/destroy`, `start/stop`, `insert/delete`, `increment/decrement`, `old/new`, `begin/end`, `first/last`, `up/down`, `min/max`, `next/previous`, `open/close`, `show/hide`, `suspend/resume`, *etc.*
- Avoid unintentional shadowing of function names. Use the `which -all` or `exist` tools to check for shadowing.

General

- Abbreviations in names should be avoided. Use `computeArrivalTime`. Avoid `comparr`.
- Consider making names pronounceable.
- All names should be written in English.

Files and Organization

M-Files

- Modularize code. Use small well designed pieces to make the whole.
- Write functions that are easy to test.
- Make interaction clear. Use inputs and outputs rather than global variables.
- Replace long lists of arguments with structures.
- Partitioning. All sub-functions and most functions should do one thing very well.
- Use existing functions rather than custom coded functions when possible.
- Move blocks of code used in multiple m-files to functions.
- Use sub-functions when a function is only called by one other function.
- Write test scripts for every function.

Input/Output

- Make input and output modules for large functions.
- Format output for easy use. For humans, make it human readable. For machines, make it parsable.

Statements

Variables and constants

- Variables should not be reused unless required by memory limitations.
- Related variables of the same type can be declared in a common statement. Unrelated variables should not be declared in the same statement:

```
persistent x, y, z
```

- Document important variables in comments near the start of the file.
- Document constants with end of line comments:

```
THRESHOLD = 10; % Max noise level
```

Global Variables

- Minimize use of global variables and constants.
- Consider using a function instead of a global constant.

Loops

- Variables used in loops should be initialized immediately before the loop.

```
result = zeros(nDays,1);
for iDay = 1:nDays
    result(iDay)= foo(iDay);
end
```

- Minimize the use of `break` and `continue` in loops.
- The end lines in nested loops can have comments to clarify the code block.

```
for index=1:2
    if index==1
        dosomething(index);
        ...
    end % End if
end % End for
```

Conditionals

- Avoid complicated conditional expressions. Use temporary logical variables instead.

```
isValid = (v >= lowerLimit) & (v <= upperLimit);
isNew = ismember(v, valueArray);
```

- Avoid the conditional expression `if 0`.
- An `if-else` sequence should include the else condition.
- The usual case should be put in the `if`-part and the exception in the `else`-part of an `if-else` statement.
- A `switch` statement should include the `otherwise` condition.
- Use a `switch` sequence if the variable is a string.
- Use a `switch` statement in place of many `if-elseif-else` statements when possible.

General

- Avoid cryptic code. You should be able to look at it a month from now and know what it does.
- Use parentheses for clarity even if not need because of operator precedence.
- Minimize the use of numbers in expressions. Use a named constant instead.
- Always use a zero before the decimal point: `THRESHOLD = 0.5`
- Make floating point comparisons with caution.

Layout, Comments, and Documentation

Layout

- Contents should be kept within the first 80 columns.
- Lines should be split after commas, spaces, and operators.
- Align a continued line with the beginning of the expression on the previous line:

```
totalSum = a + b + c ...
    d + e;
```

- Basic indentation should be 4 spaces.
- In general, a line of code should contain only one executable statement.
- Short single statement `if`, `for`, or `while` statements can be written on one line:

```
if(condition), statement; end
```

White Space

- Surround `=`, `&`, and `|` by spaces.
- Follow commas by a space.
- Keywords should be followed by a space.
- Blocks of code should be separated by three blank lines or a section break.
- Use code alignment wherever it enhances readability.

Comments

- Comments cannot justify poorly written code.
- Comments should agree with the code but not restate the code.
- Comments should have the same indentation as the statement(s) referenced.
- Traditional function header comments should support `help` and `lookfor`:
  - `help` prints the first continuous block of comments.
  - `lookfor` searches the 1st comment line of all m-files on the path.

- Function headers should discuss any special requirements for the input/output argument and describe any side effects of the function.
- Write the function name using correct case in the function header comments.

```
function runEverything
% runEverything runs all mfiles in its folder
```

- Put any copyright lines and change history after the function header with a blank line in between.
- All comments should be in English.

Documentation

- Write header comments with text markup to provide user documentation. Include sections that correspond to a help page: syntax, description, example, and see also.
- Consider writing the documentation first to better define inputs, outputs and functionality.
- Consider using a source control tool such as SVN or GIT. If you do not use a source control tool, document changes by adding change history comments after the function header or near the top of the script.

References

1. The Turing Way Community. The Turing Way: A handbook for reproducible, ethical and collaborative research. (2022) doi:10.5281/zenodo.3233853.
2. Johnson, R. K. *The elements of MATLAB style*. (Cambridge University Press, 2010).

License

License text

```
licensed material, or material used under an exception or
limitation to copyright. More considerations for licensors:
wiki.creativecommons.org/Considerations_for_licensors

Considerations for the public: By using one of our public
licenses, a licensor grants the public permission to use the
licensed material under specified terms and conditions. If
the licensor's permission is not necessary for any reason--for
example, because of any applicable exception or limitation to
copyright--then that use is not regulated by the license. Our
licenses grant only permissions under copyright and certain
other rights that a licensor has authority to grant. Use of
the licensed material may still be restricted for other
reasons, including because others have copyright or other
rights in the material. A licensor may make special requests,
such as asking that all changes be marked or described.
Although not required by our licenses, you are encouraged to
respect those requests where reasonable. More considerations
for the public:
wiki.creativecommons.org/Considerations_for_licensees
```

==========

Creative Commons Attribution-ShareAlike 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-ShareAlike 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. Adapted Material means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Adapter's License means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.

c. BY-SA Compatible License means a license listed at creativecommons.org/compatiblelicenses, approved by Creative Commons as essentially the equivalent of this Public License.

d. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

e. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

f. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

g. License Elements means the license attributes listed in the name of a Creative Commons Public License. The License Elements of this Public License are Attribution and ShareAlike.

h. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

i. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

j. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

k. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

l. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

m. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 – Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

   a. reproduce and Share the Licensed Material, in whole or in part; and

   b. produce, reproduce, and Share Adapted Material.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)

   (4) never produces Adapted Material.

5. Downstream recipients.

   a. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

   b. Additional offer from the Licensor – Adapted Material. Every recipient of Adapted Material from You automatically receives an offer from the Licensor to exercise the Licensed Rights in the Adapted Material under the conditions of the Adapter's License You apply.

   c. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
2. Patent and trademark rights are not licensed under this Public License.
3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material (including in modified form), You must:
    a. retain the following if it is supplied by the Licensor with the Licensed Material:
        i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);
        ii. a copyright notice;
        iii. a notice that refers to this Public License;
        iv. a notice that refers to the disclaimer of warranties;
        v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
    b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
    c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

b. ShareAlike.

In addition to the conditions in Section 3(a), if You Share Adapted Material You produce, the following conditions also apply.

1. The Adapter's License You apply must be a Creative Commons license with the same License Elements, this version or later, or a BY-SA Compatible License.
2. You must include the text of, or the URI or hyperlink to, the Adapter's License You apply. You may satisfy this condition in any reasonable manner based on the medium, means, and context in which You Share Adapted Material.
3. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, Adapted Material that restrict exercise of the rights granted under the Adapter's License You apply.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material, including for purposes of Section 3(b); and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR

OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

==========

Creative Commons is not a party to its public licenses. Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." The text of the Creative Commons public licenses is dedicated to the public domain under the CC0 Public Domain Dedication. Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at creativecommons.org.

Privacy Policy

This site collects no user data.

It does use Google analytics to aid in measuring the impact and usage of the materials but does not use cookies to do so.

Measuring usage of **repliMAT** allows us to potentially apply for future funding as well as finding out which sections are most popular, allowing us to use our efforts to the best effect.

Please see the source code to see that anonymity settings are enabled in the quarto config (`_quarto.yml`) - which is used to generate the site. Namely this section:

```
  google-analytics:
    . . .
    anonymize-ip: true
    storage: none
```

Online privacy is important to us. If you have any concerns about privacy when using **repliMAT** - please contact David Wilby on the email address found in the code of conduct.