1. **Names and email ids:** Krithika Malayarasan (krithikm@umich.edu), Coy Catrett (ccatrett@umich.edu)

2. **Overview and justification. A report on the project you selected, summarizing the relevant characteristics you considered when making your selection. Beyond whatever additional information you collect in your research, include at least a name, a website link, and a brief description of the project (what it does, who uses it, etc.). Explain the criteria you used in selecting it over any others, referencing the collected information from your overview. You may contrast it to other projects you considered but rejected, if applicable. (Approximately 2 paragraphs.)**

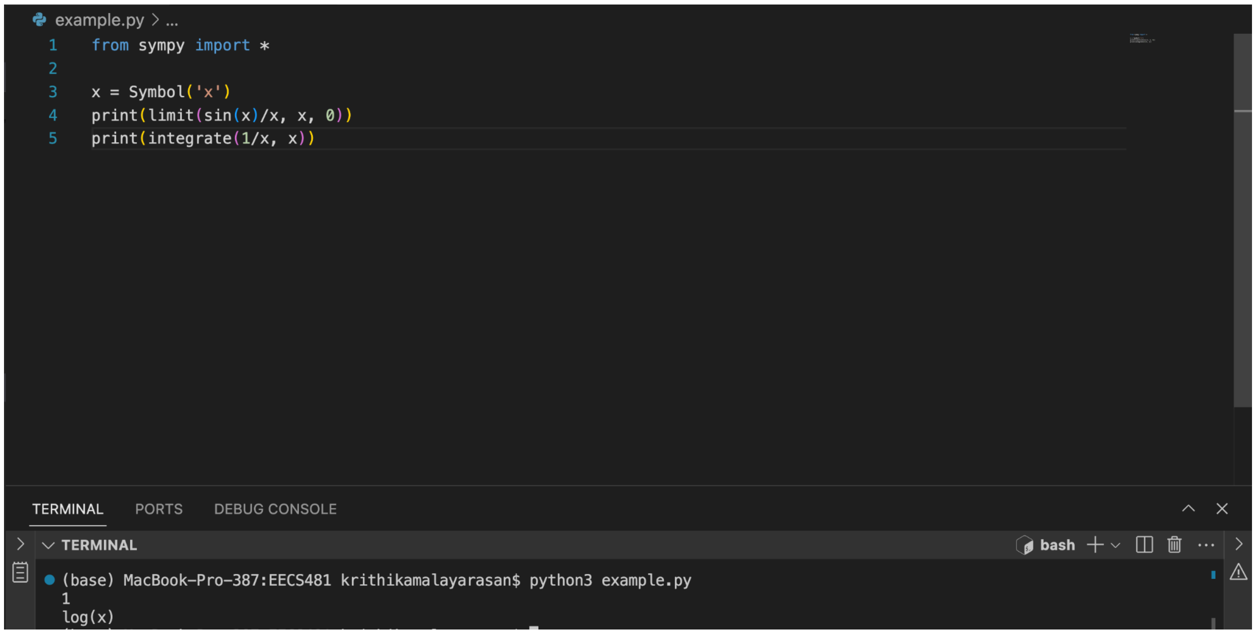
   Overview:
   The project we choose is SymPy. Here is a github link for the project: https://github.com/sympy/sympy and the website's link: https://www.sympy.org/en/index.html. It is a python library for symbolic mathematics. SymPy is entirely written in Python and they try to keep their code as simple as possible in order to be comprehensible and easily extensible.Their long-term goal is to become a full-featured computer algebra system (CAS) that's capable of performing a variety of mathematical operations. It offers functionalities for basic arithmetic, algebra, calculus, combinatorics, psychics, and more. This makes it an ideal tool for people who are in this space and are interested in science, engineering, and mathematics. They are able to use it in both educational and professional applications.

   Justification:
   The decision to choose SymPy over other open-source projects was dependent on several factors. Firstly, we wanted to choose an open-source project that we believed had a good impact on a community of learners and professionals. And we think Sympy does this well since they believe in providing easily accessible educational tools for people in the STEM field to benefit from. Next, we also noticed that this project had an active community which we thought would be a welcoming environment for new contributions since this is our first time contributing to an open-source project. Lastly, we also wanted to choose a project which used languages that we were mostly familiar with and since we both have had experience with Python in the past, we thought this project would be a good fit for the both of us.

3. **Successful build. Evidence that you can build and run the software (e.g., a screenshot or text output from a successful build, a screenshot of the running program). Getting an open-source project to build and run can be a huge effort, and we want to mitigate this risk. (Many students have been unpleasantly surprised in HW6b at being unable to compile and run the program they are working on.)**

Here is the documentation and tutorial page that I followed to install and run a sample program in order to learn more about how to use the library:
https://github.com/sympy/sympy/wiki/release-notes-for-1.12.1
https://certik.github.io/scipy-2013-tutorial/html/tutorial/index.html#tutorial

Below is a screenshot to show that the project builds and gives us the expected output to this example program.
Krithika:

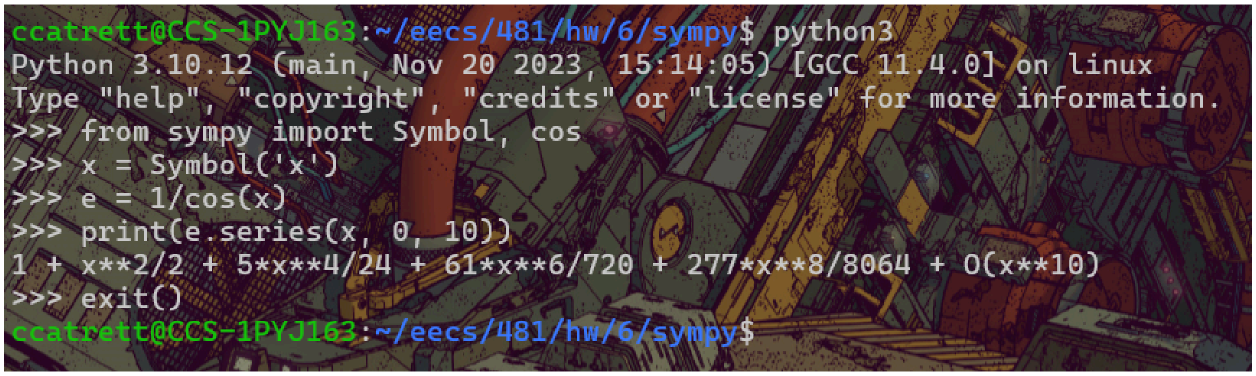


```python
from sympy import *

x = Symbol('x')
print(limit(sin(x)/x, x, 0))
print(integrate(1/x, x))
```

```
(base) MacBook-Pro-387:EECS481 krithikamalayarasan$ python3 example.py
1
log(x)
```

Coy:



```
ccatrett@CCS-1PYJ163:~/eecs/481/hw/6/sympy$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from sympy import Symbol, cos
>>> x = Symbol('x')
>>> e = 1/cos(x)
>>> print(e.series(x, 0, 10))
1 + x**2/2 + 5*x**4/24 + 61*x**6/720 + 277*x**8/8064 + O(x**10)
>>> exit()
ccatrett@CCS-1PYJ163:~/eecs/481/hw/6/sympy$
```

4. **Task(s) description. A brief textual description of your proposed task(s). In the likely event that you are proposing to tackle several tasks, list each one and give a priority order. Depending on how difficult the issues end up being, you may not necessarily have to implement all of them. In the final report submission HW6b for you will have a chance to indicate how your actual activity deviated from this prediction. (About, or up to, 2 paragraphs per task.)**

Below are the listed tasks we choose from highest priority to lowest priority.

Task 1: FCodePrinter and Integer arguments #20435
This task involves addressing an issue within the Sympy library, specifically related to the FCodePrinter functionality. The problem is with expressions involving integer arguments. Specifically, it is how it incorrectly converts the integers to floats which leads to type mismatch errors. When thinking about how we can solve this issue, we would need to look into FCodePrinter's mechanism for processing and printing argument types within function calls. One of the comments in the discussion was also talking about how we could teach the printer about argument types used in Fortran in order to have the capability of having users pass in a printer options to inform the printer of argument types of external functions. Another discussion post was also discussing implementing a new interface to specify argument types for user_functions. So we think these can be some starting points for us to either correct this default behavior or introduce a mechanism to allow users to specify the argument type for the external functions.

Task 2: sympy.TableForm produces invalid math-mode latex #20063
This task involves addressing a discrepancy in the TableForm object's LaTeX representation. The problem arises since the expected behavior for handling TableForm is to return math-mode LaTeX but it instead returns text mode LaTeX. Specifically this error occurs when we are trying to encapsulate a TableForm object within LaTeX commands which requires math-mode content but this leads to invalid output since it returns text mode instead of math-mode content. In order to address this issue we need to look into how SymPy renders expressions in LaTeX. One solution that was proposed in the discussions was redefining the _latex method for the TableForm class in order to ensure that it returns math mode LaTeX. We think this can be our starting point for this task.

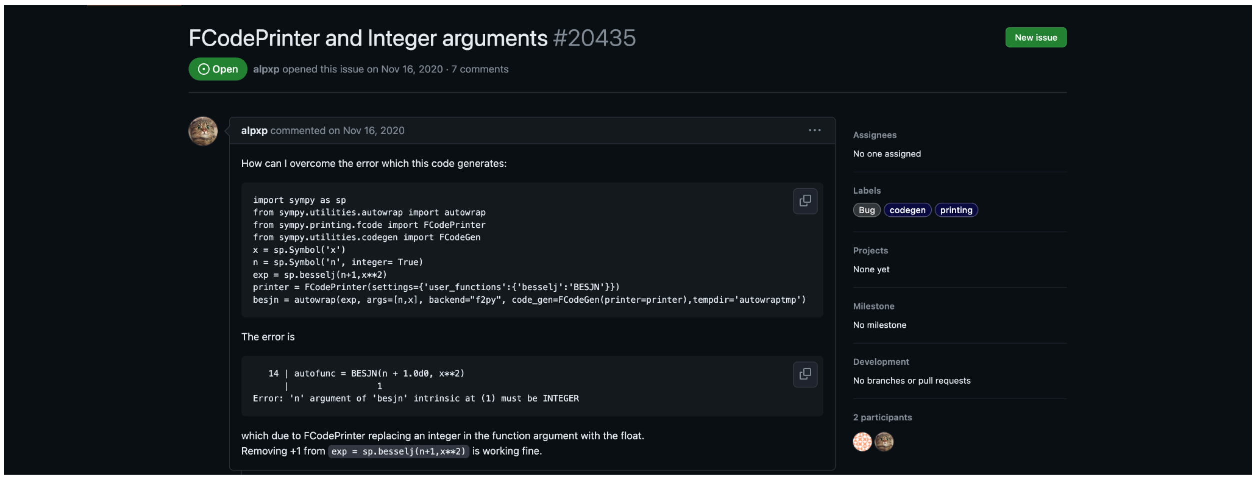
Task 3: Extra \cdot in LaTeX output for Mul between a numeric power and a polynomial with numeric coefficient since SymPy 1.10 #26430
This issue involves an extra cdot appearing in the LaTeX representation of the multiplication operations between a numeric power and a polynomial with numeric coefficient. This issue was introduced in the Sympy 1.10 version which is different than the output we were seeing with Sympy version 1.9. Our approach for this issue is that we first want to look into the Mul and Pow operations within the LaTeX printing system to see why an extra cdot is being introduced here. We would probably need to add in some extra logic to identify where exactly a cdot is mathematically relevant and where we would not need it.

Task 4: sympy.floor returns wrong result #26368
This issue is with an anomaly in the floor function's output which is on both 1.12 and 1.13 dev. The user shows this by giving these two examples:
sympy.floor((sympy.log(2**31-1,2)+1)) returning 32 instead of 31 and
int((sympy.log(2**360-1,2)+1)) returning 361 instead of 360 which is the correct output.

The discussion was mentioning that this behavior could be the result of a precision error within Sympy's logarithmic and flooring computations since the error only happens in some scenarios. And they further talk about how this could be because of assumptions made within Sympy's evaluation process about the size of the expressions (assumed_size = 30). The discussion also talks about a temporary fix using the 'N()' function or the '.evalf()' function but this doesn't seem to fix the full issue. So our approach needs to be a fix that ensures accurate evaluation of the floor function in all scenarios. We would first need to look into how Sympy's log expressions are evaluated with floors which could help us see what's leading to these inaccuracies.

5. **Task link(s). Evidence that the task(s) is/are requested by the community (a screenshot or issue tracking link suffices).**

Task 1: https://github.com/sympy/sympy/issues/20435



FCodePrinter and Integer arguments #20435

alpxp opened this issue on Nov 16, 2020 · 7 comments

alpxp commented on Nov 16, 2020

How can I overcome the error which this code generates:

```
import sympy as sp
from sympy.utilities.autowrap import autowrap
from sympy.printing.fcode import FCodePrinter
from sympy.utilities.codegen import FCodeGen
x = sp.Symbol('x')
n = sp.Symbol('n', integer= True)
exp = sp.besselj(n+1,x**2)
printer = FCodePrinter(settings={'user_functions':{'besselj':'BESJN'}})
besjn = autowrap(exp, args=[n,x], backend="f2py", code_gen=FCodeGen(printer=printer),tempdir='autowraptmp')
```

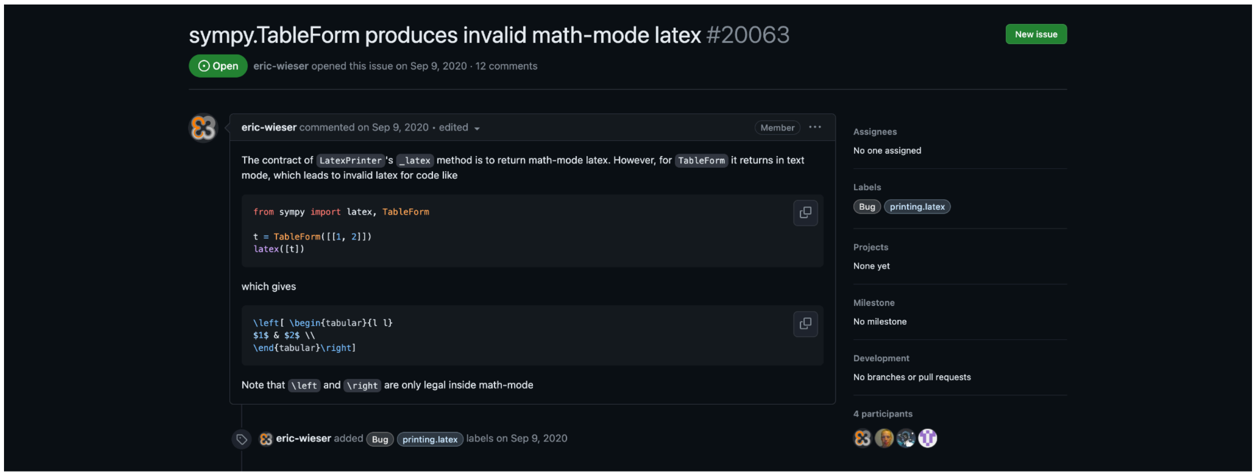
The error is

```
   14 | autofunc = BESJN(n + 1.0d0, x**2)
      |                  1
Error: 'n' argument of 'besjn' intrinsic at (1) must be INTEGER
```

which due to FCodePrinter replacing an integer in the function argument with the float.
Removing +1 from `exp = sp.besselj(n+1,x**2)` is working fine.

Requirements: For this task, we plan to modify the current behavior of the besselj function so that the argument types are preserved instead of evaluating all constant function arguments as floats. This will require unit and regression testing in order to make sure that our modifications do not change anything else. This will likely involve some research about the Bessel functions, a special class of differential equations as well as verification of our modifications via an alternative implementation of the besselj function, possibly on Wolfram Alpha.

Task 2: https://github.com/sympy/sympy/issues/20063



sympy.TableForm produces invalid math-mode latex #20063

eric-wieser opened this issue on Sep 9, 2020 · 12 comments

eric-wieser commented on Sep 9, 2020 • edited

The contract of `LatexPrinter`'s `_latex` method is to return math-mode latex. However, for `TableForm` it returns in text mode, which leads to invalid latex for code like

```
from sympy import latex, TableForm

t = TableForm([[1, 2]])
latex([t])
```

which gives

```
\left[ \begin{tabular}{l l}
$1$ & $2$ \\
\end{tabular}\right]
```

Note that `\left` and `\right` are only legal inside math-mode

eric-wieser added Bug printing.latex labels on Sep 9, 2020

Assignees
No one assigned

Labels
Bug printing.latex

Projects
None yet

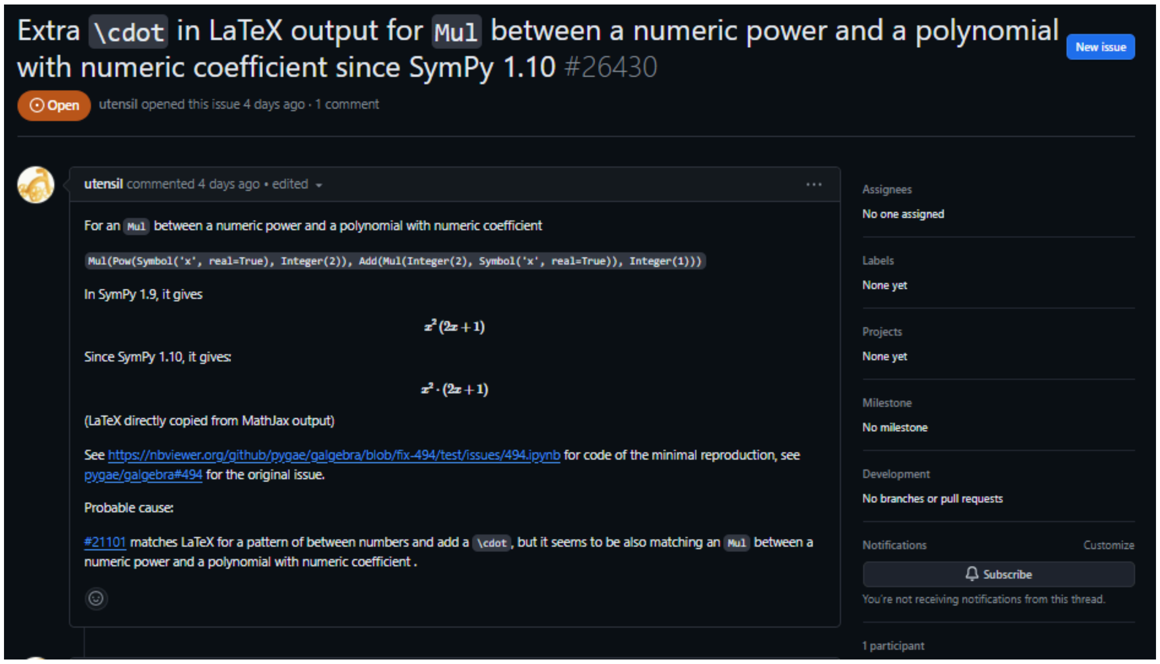
Milestone
No milestone

Development
No branches or pull requests

4 participants

Requirements: This task will involve further investigation of the comment that the post references. Although the bug is easy to comprehend, we will likely have to do a lot of testing in order to fully understand this problem and make the correct changes. This appears to be an issue with how the floor function is implemented, and although this is an algorithmic issue, there is not a textbook solution for this issue and will require us to utilize the quality assurance skills we have acquired this semester.

Task 3: https://github.com/sympy/sympy/issues/26430



Extra `\cdot` in LaTeX output for `Mul` between a numeric power and a polynomial with numeric coefficient since SymPy 1.10 #26430

utensil opened this issue 4 days ago · 1 comment

utensil commented 4 days ago • edited

For an `Mul` between a numeric power and a polynomial with numeric coefficient

`Mul(Pow(Symbol('x', real=True), Integer(2)), Add(Mul(Integer(2), Symbol('x', real=True)), Integer(1)))`

In SymPy 1.9, it gives

$$x^2 (2x+1)$$

Since SymPy 1.10, it gives:

$$x^2 \cdot (2x+1)$$

(LaTeX directly copied from MathJax output)

See https://nbviewer.org/github/pygae/galgebra/blob/fix-494/test/issues/494.ipynb for code of the minimal reproduction, see pygae/galgebra#494 for the original issue.

Probable cause:

#21101 matches LaTeX for a pattern of between numbers and add a `\cdot`, but it seems to be also matching an `Mul` between a numeric power and a polynomial with numeric coefficient .

Assignees
No one assigned

Labels
None yet

Projects
None yet

Milestone
No milestone

Development
No branches or pull requests

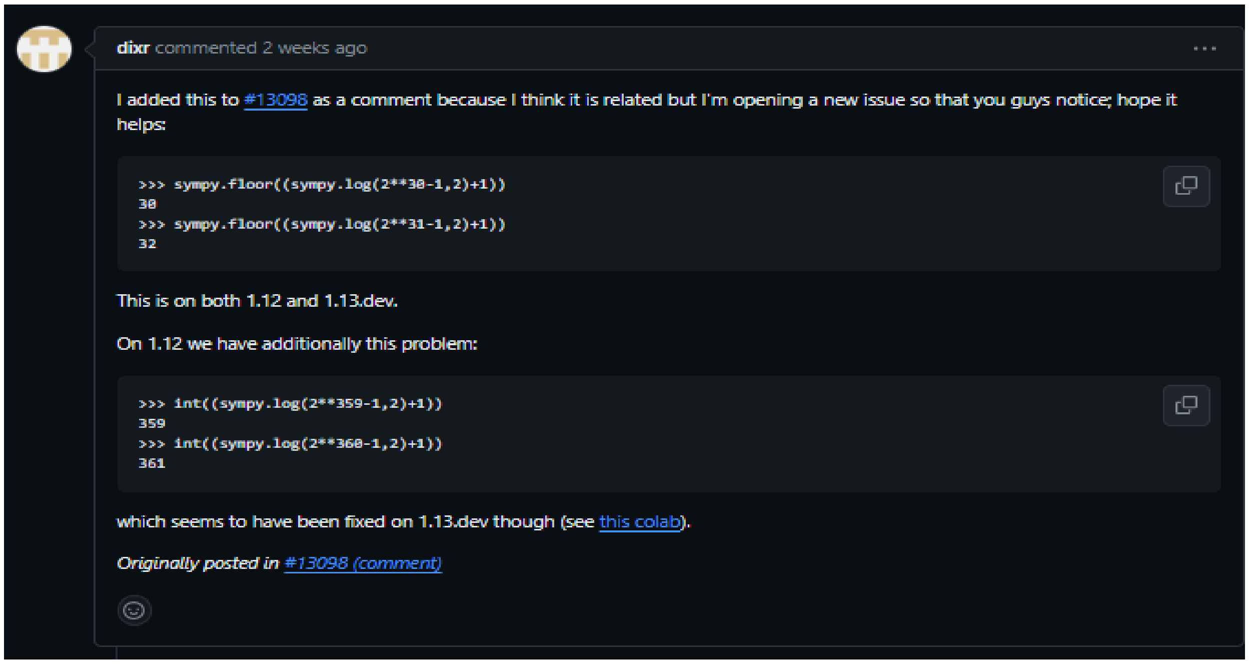
Notifications

You're not receiving notifications from this thread.

1 participant

Requirements: This task will require us to understand the intended behavior of the Mul operator and the implementation of the galgebra python library. This issue is not a bug for real numbers, but it is a bug for particular objects in the galgebra library. We hope to fix this issue by identifying the objects which the bug affects and modifying the behavior of the Mul function in order to reflect the correct behavior.

Task 4: https://github.com/sympy/sympy/issues/26368



dixr commented 2 weeks ago

I added this to #13098 as a comment because I think it is related but I'm opening a new issue so that you guys notice; hope it helps:

```
>>> sympy.floor((sympy.log(2**30-1,2)+1))
30
>>> sympy.floor((sympy.log(2**31-1,2)+1))
32
```

This is on both 1.12 and 1.13.dev.

On 1.12 we have additionally this problem:

```
>>> int((sympy.log(2**359-1,2)+1))
359
>>> int((sympy.log(2**360-1,2)+1))
361
```

which seems to have been fixed on 1.13.dev though (see this colab).

Originally posted in #13098 (comment)

Requirements: This task would require understanding of the floor function implementation in SymPy in order to create a minimal test case and isolate the issue. Additionally, we would need to look into the issue referenced in the above image, since it may have additional information on what we would be required to do.

Back-up Tasks: In the unfortunate case that one of our tasks is resolved before we are able to make a contribution, we have each chosen a back-up task.

Coy's Back-up Task: This task deals with legend errors when plotting parametric functions.

Krithika's Back-up Task: This task deals with enhancing the graph of the floor function.

6. **Requirements. A description of each task's requirements, both functional and quality. Document these requirements at whatever level of detail you consider appropriate: use cases might be helpful, for example, but are not required. You might consider how your proposed task fits into the overall goals of the project. We do not want any sort of full formal software requirements specification. Instead, we want lightweight documentation of your task's requirements and evidence that you understand how they fit into the larger project. (At most half a page per task.)**

(See underneath each image for requirements)

7. **Initial time plan. Choose any format as long as it is clear (e.g., Gantt diagram, plain text). This should include at least: individual tasks and milestones, with deliverables (e.g., to the project maintainers); estimated effort for each (sub)task; dependencies between tasks; and a best-effort assignment of (sub)tasks to team members. We encourage you to include supporting evidence for your estimates. We will grade you on the presence of your planning but not its accuracy; it is completely acceptable if plans change. Be sure to schedule time for QA activities. See the final team report for more on QA. (At most one page.)**
   Individual Task Distribution
   Coy: 1 & 3
   Krithika: 2 & 4

   Milestones:
   - Talking to a current contributor
   - Run an example program
   - Reproduce error
   - Find minimal error producing input
   - Understanding the files associated with the bug
   - Baby makes their first change
   - Testing changes (Unit or Regression)
   - Submitting a PR

   Deliverables:
   - Comprehensive test suite
   - PR
   - Final Report

   Estimated Effort Per Task:

1. Considerable research due to the complex nature of the functions/algorithms involved. The Bessel Function, or functions are a class of well-studied differential equations. I, however, have not studied them in great detail and will most likely need to read about what it is and the methods used to solve Bessel functions. I estimate about a week's worth of work.
2. Research to look into how Sympy renders LaTeX and then work on how to return math mode instead of text mode for TableForm. I have some experience with working with LaTeX so I think this will help me better understand how Sympy renders LaTeX in order to be able to solve the issue. About a week for writing the code and a few days to test this out by reproducing the error and seeing if we get the expected output.
3. I estimate myself spending a substantial amount of effort reading documentation and understanding the dependencies related to this bug. I would estimate this taking another week to complete. This seems like another very niche and math related bug. I will need to read about the library, galgebra, which this bug is dependent on, and likely do more

research about what types of mathematical objects this library deals with to understand what the correct output should be in all cases.

4. Looking into how Sympy's log expressions are evaluated with floors in order to figure out a solution for the precision errors. This could take 1 week for the research and 1 week for looking at what solution to code.

   Dependencies between tasks: There are no dependencies between the tasks that we have chosen.

8. **Short risk assessment. Identify and briefly describe key risks in each task and discuss how you plan to mitigate those risks. (1 paragraph, or 1 paragraph per task.)**
   These tasks, while simple, may end up taking much more time than we anticipate. In order to stay on top of time management for our tasks, we plan to stick to our schedule described below. As first time contributors/developers, we are aware that our code may be inefficient, poorly written relative to the rest of the project, or simply incorrect. We plan to mitigate these issues by communicating with active members of the project, trying our best to mimic the current style of coding in the repository, and applying unit tests and regression tests to the code which we modify and/or add. In order to have a good idea of the scope of our tasks, we intend to talk to current contributors and see what they have to say about these tasks. By following these guidelines, we hope that our development process has minimal risks and goes according to plan.

9. **Initial process plan. Describe the process you plan to follow. This should mention quality assurance and how you plan to communicate and collaborate as well as divide and integrate work. (At most half a page.)**

   Process:

   1. Collect Information to research by reading code and documentation for each of our tasks.
   2. Reaching out to active community members. SymPy Has a google group of almost four thousand members. Reaching out to some contributors and developers may give us more guidance for completing our tasks more efficiently.
   3. Meet once to twice a week (schedules allowing) to check in and discuss our progress.
   4. Daily work on each of our respective tasks.

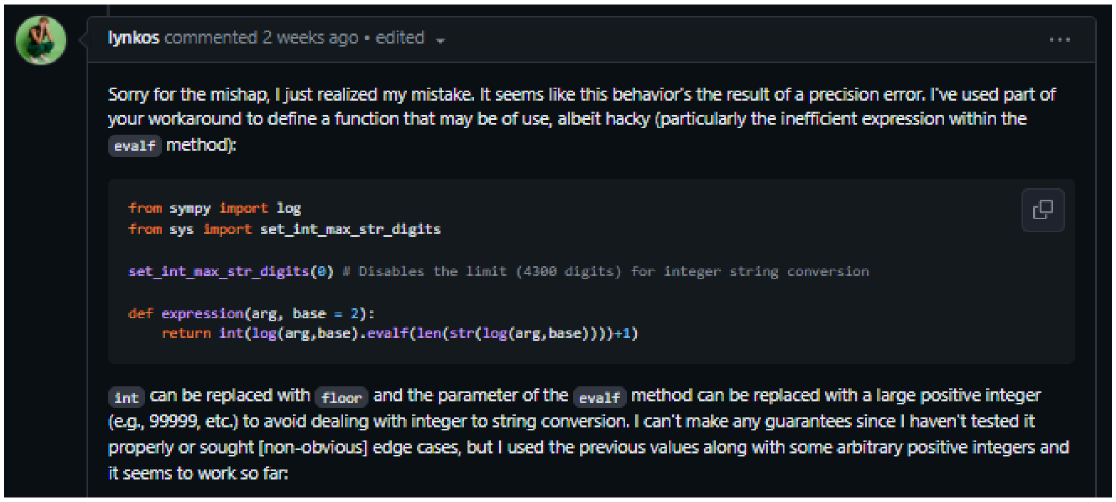
   Schedule:

   We plan to meet on Mondays and Fridays to discuss our research and progress. Additionally, we plan to make some progress each day, whether that be research, reading documentation, reaching out to developers, checking in with one another, etc. We plan to be done with our projects in about two weeks total time. Our intent is to make our meetings similar to a code review, where we check each other's ideas and code. We hope this allows us to maintain high code quality throughout the duration of the project.

10. **Task scope justification. Evidence that the tasks are of a sufficient and reasonable size and complexity (for you or your team) for this assignment. You may want to review the task scope bullet point under Task Selection and the Plan Updates bullet point under Project Report for ideas about what to include here. (1 paragraph per task)**

    The SymPy project has been active since its start in 2005 with a large community. As per the SymPy wikipedia page, the core of SymPy contains around 260,000 lines of code and includes a comprehensive set of self-tests totalling over 100,000 lines of code in 350 files (as of version 0.7.5).

    Our tasks' opening dates range from extremely recent to four years old. There are indications that others have attempted to solve some of the same problems we are but were not successful or fully implemented. For example, this comment corresponding to task four indicates that people have been working on the same tasks we are but finding basic solutions or workarounds and not actually fixing the code defects:


lynkos commented 2 weeks ago • edited

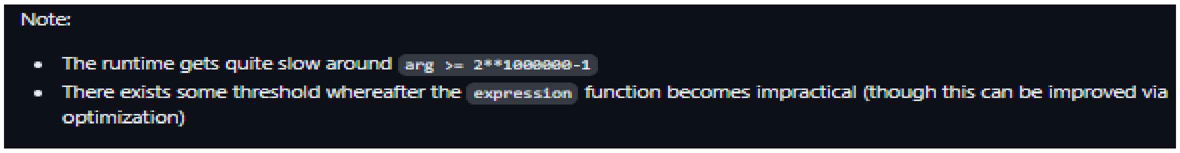
Sorry for the mishap, I just realized my mistake. It seems like this behavior's the result of a precision error. I've used part of your workaround to define a function that may be of use, albeit hacky (particularly the inefficient expression within the `evalf` method):

```
from sympy import log
from sys import set_int_max_str_digits

set_int_max_str_digits(0) # Disables the limit (4300 digits) for integer string conversion

def expression(arg, base = 2):
    return int(log(arg,base).evalf(len(str(log(arg,base))))+1)
```

`int` can be replaced with `floor` and the parameter of the `evalf` method can be replaced with a large positive integer (e.g., 99999, etc.) to avoid dealing with integer to string conversion. I can't make any guarantees since I haven't tested it properly or sought [non-obvious] edge cases, but I used the previous values along with some arbitrary positive integers and it seems to work so far:


Note:

- The runtime gets quite slow around `arg >= 2**1000000-1`
- There exists some threshold whereafter the `expression` function becomes impractical (though this can be improved via optimization)

This is a good indication that our tasks will be sufficiently difficult and require a substantial amount of time and effort in order to come up with a potential solution, yet they aren't so unreasonable that no one has any insight as to how to solve the issue. Similar comments can be found under some of our other tasks, indicating that our task selection is reasonably large and manageable for two people.