Report on the paper
"SymPy: Symbolic computing in Python" (#11410)
by A. Meurer et al.

The paper introduces SymPy, a well-known pure Python library for symbolic computation. It covers many aspects of the library: architecture, basic usage, overview of modules, a more detailed look into some of them, and physics application.

The writing style is mostly clear and easy to understand. Some example code is provided to further explain the use of various module, classes, and functions.

Unfortunately, the paper is very unconnected and disorganized, with some unnecessary repetition and some things left undefined. Further, parts of the supplement should be moved to the main paper and vice versa. All of this makes the paper look more like a collection of unconnected or, at best, very loosely connected parts, instead of a meaningful whole.

Another big problem of the paper is the lack of aim. Some parts of SymPy are covered at informative level (short descriptions of the elements related to some subject), some at the beginners level (basic usage examples), while some go deep in internal SymPy implementation of certain features. This structure leaves the impression that different, yet mutually intermingled sections aim at different audiences.

Moreover, these vastly differently approached elements come in no specific order, and with no obvious reason why each of them is picked to be covered at all and, specifically, at the chosen level of complexity in the approach.

I suggest a major rewrite of the paper, to improve the structure and group differently approached subjects. It would probably be best to:

1. Create a new section (following the introduction of SymPy) on projects that use SymPy, and put in it the materials currently available in supplement's sections 8, 9, 11. The comparison with Mathematica (supplement's section 10) should be moved either to the introduction as a section, or right after the introduction as its own section, but it should not be in the middle of the description of SymPy-powered projects.

2. This should be followed by the list of SymPy packages and modules (currently section 3) and descriptions of selected modules (currently sections 3.1, 3.3, 3.5, supplement's sections 5, 7, and 9).

3. Now, basic usage can be given as its own section (currently done in sections 2.1 and 2.3), followed by introductions on usage of various modules (currently in sections 3.2, 3.4, 3.6, 4, 5, and supplement's sections 2, 3, 4, 6).

4. In-depth architecture (most of the so far unmentioned sections) can be given either as its own section, or made into a supplement of its own (as it is naturally far more technical and less interesting to general audience).

5. The current conclusion works fine as the finishing section of the paper.

The suggested reorganization still leaves the paper unbalanced with respect to which SymPy features are covered and in how much detail. I suggest that the authors expand the sections that are currently only descriptive. Further, some motivation should be given for the choice of the features that are explained (as opposed to just being listed in what is currently section 3).

More specific comments and correction suggestions for the paper follow.

1. Explaining what Python is (lines 70–72) should go before talking about SymPy as a "CAS written in Python". Further, the paper assumes a moderate familiarity with Python (for example, Python's console, OOP, and exceptions), and this should be specified.
   There should be a short note on the used Python console (`>>>` is the prompt, with the results of computation following immediately in the lines after it).
   The citation [25] from line 65 should be moved next to "Python" in line 70.

2. Line 73 has outdated information. Sage was renamed to SageMath and it no longer aims only at pure mathematics but also at algebra, numerical analysis, etc. The reference [40] should be replaced by a more up-to-date one.

3. The plural "CASs" is usually written as "CASes" or "CAS's", with the latter being somewhat problematic due to it looking like it implies possession.

4. Line 88 mentions "printers", but it doesn't state what they are, which is confusing for those readers that are yet to learn the concept in section 3.4.

5. Also in line 88, Jupyter's citation [30] is actually about IPython and should be replaced by a more up-to-date version.

6. The word "software" in line 91 is ambiguous; "library" or "package" would make a better choice.

7. Lines 91–96: "we discuss/look at/etc" is the preferred form, instead of "section discusses/looks at/etc".

8. The paragraph in lines 103–105 should be moved to the introduction, and the footnote from line 104 should be added to that paragraph as a full-blown sentence, expanded by all the relevant technical information (Python version, OS,...). Given that the end of life for Python 2 is 2020., a comment on whether all the presented examples work in both Python 2 and 3 should be included as well.
   Further, emphasise that wildcard imports, "`import *`", should almost never be used in programs (see PEP 8, the item "wildcard imports"). The same goes for the `import` mentioned in lines 484–487.

9. Line 119 should be removed, as it is basically a copy of the previous line.

10. In line 121 the word "stored" should be replaced by "used as keys".

11. What do the authors mean by "thereby permitting features such as caching" in line 122? Caching can be done for mutable types as well, just not through hashing.

12. There is no need to repeat "(CAS)" in line 124, as it was already given in line 64.

13. In the same line, the word "represents" should be replaced by "stores".

14. In line 184, "symbols are" should be replaced by "`t` is" (the general rule is already given in line 179).

15. The code in line 210 should be made into its own line (like a displaymath formula), for typesetting reasons and better readability.

16. The footnote 4 should be moved from line 221 to line 222, right after "`Basic`".

17. The part "which defines some basic methods for symbolic expression trees" should be removed from line 222, as it was already given in line 130.

18. In line 225, the sentence "Not all SymPy classes are subclasses of `Expr`." sounds confusing as a reader new to SymPy wouldn't expect, for example, symbols to inherit "Expr". It would be better to expand this, for example "Most of the SymPy classes (including `Symbol`) are subclasses of `Expr`, but there are exceptions to this rule".

19. The title "Features" in line 276 is ambiguous, as a "feature" has no precise meaning in Python (or even software libraries in general). It should be replaced by "Packages and modules" or a similar more precise wording. The same goes for 'feature" in most other places in the paper (for example, the caption of Table 1. and line 495).
    It would be very useful to also include actual names of the packages/modules in Table 1., as well as in any section covering those packages/modules.

20. The sets support listing is unnaturally split in two by the "This includes..." sentence which would fit better in parentheses.

21. Line 355, add a sentence explaining that in SymPy `str == repr`, because in Python `repr` is used to get an unambiguous valid Python code representation, while the return value of `str` is meant to be human-readable.

22. Lines 359 and 379: what is 2D text representation? It seems that "2D" shouldn't be here.

23. Line 427: every dictionary is a "dictionary of keys". This should be a dictionary with coordinate tuples as keys associated with the appropriate values.

24. Section 4 would benefit from an introduction, and lines 440– should become a new subsection 4.1. (named "`Float`" or "Real numbers support" or similar).

25. I suggest a better example for lines 459–460: a computation of $(e^{100}+1)-e^{100}$:

```
>>> (exp(100)+1).evalf() - exp(100).evalf()
0
>>> ((exp(100)+1) - exp(100)).evalf()
1.00000000000000
>>> (exp(100)+1) - exp(100)
1
```

or two different ways to compute the 100th Fibonacci number:

```
>>> phi = (1+sqrt(5))/2
>>> psi = (1-sqrt(5))/2
>>> ((phi**100-psi**100)/sqrt(5)).evalf()-fibonacci(100)
65536.0000000000
>>> ((phi**100-psi**100)/sqrt(5) - fibonacci(100)).evalf()
0.e-104
```

Obviously, like your own example, these are problematic because a part of the computation is relying on Python's builtin floats. Please include a comment on whether symbolic computation (i.e., applying `evalf()` on the whole expression) always avoids these errors or not, possibly with an example when it doesn't resolve this problem.

26. The footnote from line 477 should be moved as a sentence in its own right to the introduction, with other technical specifications.
27. In line 495, the word "solving" seems more appropriate than "solutions".
28. In line 504 "is" should be used instead of "are" (because "array" is singular).
29. In line 518, "produces" should become plural.
30. The title of section 5, "Domain specific submodules", seems inappropriate because the section only covers Physics package (not "submodules"). It should either be expanded with a short introduction listing other domain specific packages, or it should be renamed to "Physics package".
31. The word "symbolics" in line 528 should be removed (as almost everything in the paper deals with symbolic computation).
32. In line 530, `sympy.physics.vector` is a module, not a package.
33. It is unclear what "both of these objects" refer to in line 532. My guess is vectors and dyadic objects, but this should be reworded to make it more clear.
34. In lines 543 and 545, "rad" should be removed. Radians are assumed when no other measure (like degrees) is given.
35. In lines 567–568, "performing symbolic quantum mechanics" makes no sense. This should probably be "computations", "solving problems related to", etc.

36. The sentence "SymPy expressions are immutable trees of Python objects." doesn't belong in the conclusion. This can be moved to the appropriate place when discussing SymPy's architecture.

37. All "submodules" should be replaced by "modules" (examples: lines 657, 662).

38. The sentences in lines 657–661 should swap places, because "areas of mathematics" are discrete mathematics, concrete mathematics, etc., while simplifying expressions, performing common calculus operations, pretty printing expressions, etc. belongs to common operations ("other areas" is also fine, albeit slightly wrong).

39. In line 662 "classical mechanics and quantum mechanics" are listed as the only example of the support for specific domains, as in section 5, which leaves the impression that physics the only one. Either more domains should be listed, or it should be reworded to recognize the fact that there are no others.

40. Lines 670–678 contain explanations what some of the authors' institutions are. It is customary to use acknowledgements to thank people and institutions, while institutions' details should be provided in the authors' footnotes in the documents' head.

41. The citation [5] in line 691 is missing identification data, probably a URL.

42. The citation [21] in line 728 should have "2D" instead of "2d".

More specific comments and correction suggestions for the supplement follow.

43. The supplement should have an introduction, explaining what is being covered in it in general and in each section.

44. Since the Guntz algorithm is covered in depth, it would be good to include how an interested reader can see SymPy's steps of computation:

```
>>> import os
>>> os.environ['SYMPY_DEBUG'] = 'True'
>>> from sympy import *
>>> x = symbols('x')
>>> limit(sin(x)/x, x, 0)
DEBUG: parsing of expression [(0, 1, None, None)] with symbol _w
DEBUG: returned None
DEBUG: parsing of expression [(_w, 1, None, None)] with symbol _w
DEBUG: returned ([], [(_w, 1, None, None)], 1, False)
DEBUG: parsing of expression [(0, 1, None, None)] with symbol _w
DEBUG: returned None
DEBUG: parsing of expression [(_w, 1, None, None)] with symbol _w
DEBUG: returned ([], [(_w, 1, None, None)], 1, False)
DEBUG: parsing of expression [(_w, 1, None, None)] with symbol _w
DEBUG: returned ([], [(_w, 1, None, None)], 1, False)
DEBUG: parsing of expression [(0, 1, None, None)] with symbol _w
```

```
DEBUG: returned None
DEBUG: parsing of expression [(_w, 1, None, None)] with symbol _w
DEBUG: returned ([], [(_w, 1, None, None)], 1, False)
DEBUG: parsing of expression [(0, 1, None, None)] with symbol _w
DEBUG: returned None
DEBUG: parsing of expression [(_w, 1, None, None)] with symbol _w
DEBUG: returned ([], [(_w, 1, None, None)], 1, False)
DEBUG: parsing of expression [(1, 1, None, None)] with symbol _w
DEBUG: returned None
limitinf(x*sin(1/x), x) = 1
+-mrv_leadterm(_p*sin(1/_p), _p) = (1, 0)
| +-mrv(_p*sin(1/_p), _p) = ({_p: _Dummy_14}, {}, _Dummy_14*sin(1/_Dummy_14))
| | +-mrv(_p, _p) = ({_p: _Dummy_14}, {}, _Dummy_14)
| | +-mrv(sin(1/_p), _p) = ({_p: _Dummy_15}, {}, sin(1/_Dummy_15))
| |   +-mrv(1/_p, _p) = ({_p: _Dummy_15}, {}, 1/_Dummy_15)
| |     +-mrv(_p, _p) = ({_p: _Dummy_15}, {}, _Dummy_15)
| +-rewrite(_Dummy_14*sin(1/_Dummy_14), {exp(_p): _Dummy_14}, {}, _p, _w)
        = (sin(_w)/_w, -_p)
| | +-sign(_p, _p) = 1
| | +-limitinf(1, _p) = 1
| +-calculate_series(sin(_w)/_w, _w) = 1
|   +-limitinf(_w, _w) = oo
|   | +-mrv_leadterm(_w, _w) = (1, -1)
|   | | +-mrv(_w, _w) = ({_w: _Dummy_18}, {}, _Dummy_18)
|   | | +-rewrite(_Dummy_18, {exp(_w): _Dummy_18}, {}, _w, _w) = (1/_w, -_w)
|   | | | +-sign(_w, _w) = 1
|   | | | +-limitinf(1, _w) = 1
|   | | +-calculate_series(1/_w, _w) = 1/_w
|   | +-sign(-1, _w) = -1
|   | +-sign(1, _w) = 1
|   +-limitinf(_w, _w) = oo
|   +-limitinf(_w, _w) = oo
|   +-limitinf(_w, _w) = oo
+-sign(0, _p) = 0
+-limitinf(1, _p) = 1
```

Emphasize that the environment variable `SYMPY_DEBUG` must be set before importing SymPy for the first time.

45. In line 183, syntax `symbols(a:d)` is used without being previously defined or explained.

46. Figure 2 should not be a screenshot, but rather proper code with LaTeX-rendered results and explanations, as was done in the rest of the paper. A shortened link to SymPy Gamma for the example's expression can be included for the user to try this for themselves.

Because it will need a fair bit of rewriting while merging with the paper, I did not go into more detail with remarks related to the supplement.