expsagetex – a fully expandable interface for SageTeX[*]

Florent Rougon[†]

January 11, 2020

Abstract

This package is based on `sagetex.sty`, the style file for SageTeX. One deficiency of current `sagetex.sty` (version 3.3 from 2019/01/09) is that it doesn't provide any user-accessible way to retrieve a result computed by Sage in places where only expansion happens. This packages uses the low-level SageTeX machinery to implement a set of commands that can "record" results computed by Sage and a fully expandable command that can yield any such result wherever you need it, even in expansion-only contexts.

Contents

[*]This file describes expsagetex v0.1, last revised 2020-01-11.

[†]E-mail: mailto:f.rougon@free.fr

Besides the Quick start section below, there is introductory material to expsagetex on TeX.stackexchange.com.

1 Quick start

Getting a result from Sage using expsagetex involves three steps in the LaTeX code:

1. Recording a Python expression to be written to the `.sagetex.sage` file. This step is done with one of the `\est*Record` functions.
2. Using the result where you need it. This is done with `\estGet` and works in many places, in particular in expansion-only contexts (inside `\edef`, `\message`, `\write`, etc.).
3. Declaring that you use the result, so that a warning can be displayed if the result isn't available yet (`\estGet` can't print the warning itself because it must work in expansion-only contexts). This is done with `\estRefUsed`.

Step 3 can be done before or after step 2, but must come after step 1. When you run `sage` on the `.sagetex.sage` file written by step 1, the Python expression is evaluated, converted to a Python string and written to the `.sagetex.sout` file. This file is in turn read by `sagetex.sty` at startup, which makes the result available for `\estGet`.

Thus, a very simple example can be $45^6 \equiv 1 \pmod 7$. This was obtained with:

```
\estRecordFormatted{\littleFermatExple}{(45**6) \percent 7}%
\estRefUsed{\littleFermatExple}%
$45^6 \equiv \estGet{\littleFermatExple} \pmod 7$
```

For compiling, use the same method as with sagetex (which expsagetex relies on): you need one LaTeX run, one `sage` run on the `.sagetex.sage` file followed by at least one more LaTeX run (if more are needed, warnings are printed by LaTeX as usual).

2 Examples

2.1 Integers

2.1.1 Local assignments

When using `\estRecordFormatted{`⟨*macro*⟩`}{`⟨*expr*⟩`}` with a Python expression ⟨*expr*⟩ that evaluates to an integer, `\estGet[`⟨*fallback*⟩`]{`⟨*macro*⟩`}` should give (i.e., expand to) the expected result. This is because calling `str()` on a Python integer doesn't use any exponent notation, thus `latex()` should not add any formatting. Let's execute the following Python statement using the `sageblock` environment:

```
    x = 2**333
```

Now, record the value of x and associate it with macro `\mymacro`. This makes `\mymacro` a kind of reference to the saved value, but to retrieve the tokens associated to this reference, you have to use `\estGet` instead of `\ref` (this can be done in an expansion-only context like inside `\edef`, which is impossible with `\ref`). See section 5 below for more details on this. Recover the saved value and format it with the `\num` macro of siunitx:
17 498 005 798 264 095 394 980 017 816 940 970 922 825 355 447 145 699 491 406 164 851 279 623 993 595 007 385 788
This was achieved with the following calls:

```
\estRecordFormatted{\mymacro}{x}
\num{\estGet[-1]{\mymacro}}
\estRefUsed{\mymacro}
```

Another way to do the same without having Sage call `latex()` at all is to use `\estRecordStr` to record a string representation of the integer we are interested in:

```
\estRecordStr{\mymacro}{str(x)}
\num{\estGet[-1]{\mymacro}}
\estRefUsed{\mymacro}
```

Notes:

- `\estGet` wraps its return value within `\unexpanded` (`\exp_not:n`), which implies that it won't expand further when used in `\edef` or an expl3 `x`-type argument.
- In order to use the result of evaluating a Python expression in the LaTeX document, it is not necessary to store it in a Python variable. Example:

$$2^{10} = 1024$$

Another example with variables:

```
    x = 23 % 8          # That is, x = 7.
    y = 2*x
```

Save the value of $1000000 \times (y - 10)$ (as computed by Sage) and associate it with macro `\numberBasedOnY`.

$$1000000 \times (y - 10) = 4\,000\,000$$

2.1.2 Global assignments

While `\estRecordFormatted` assigns locally to the macro given by its first argument, all `\est*Record` functions have a global variant that performs a global assignment instead. The global variant of `\estRecordFormatted` is `\estGRecordFormatted`, that of `\estRecordStr` is `\estGRecordStr`, that of `\estARecordFormatted` is `\estGARecordFormatted`, etc. Example:

```
x = 3+4
```

Save the value of x and globally assign macro `\Iamseven` to allow us to retrieve the saved value. Get it back from outside the group where this was done, and format it with `\num`: the result is 7.

2.2 Floating point numbers

`\estRecordFormatted` works with floating point numbers, but it uses LaTeX markup if the string representation of the result (in Python) uses exponent notation. When this is not desirable—i.e., when you just want a numerical result without any formatting—use `\estRecordFloat` or one of its variants. Example:

```
x = 3.1415927
```

x rounded to two decimal places is 3.14. This was rounded by Python. Thanks to the expandable nature of `\estGet`, one can also do this: x rounded to four decimal places is 3.1416. This was rounded by siunitx.

Of course, one can also compute a float with Sage without storing it into a Python variable. Here is an approximation of π using an expression based on Machin's formula and the power series expansion of arctan:

$$\begin{aligned}\pi &= 4\sum_{n=0}^{+\infty}\frac{(-1)^n}{(2n+1)}\left(4\left(\frac{1}{5}\right)^{2n+1}-\left(\frac{1}{239}\right)^{2n+1}\right)\\ &\approx 3.141593\end{aligned}$$

2.3 Strings

Functions working on Python strings are the most general here. For instance, `\estRecordFormatted` first evaluates `latex(...)` in Sage (Python), where `...` is the Python expression you entered; this results in a string with LaTeX math-mode formatting commands (e.g., if evaluating `str(...)` in Python string yields something containing brackets or exponents); then this string is written to the `.sout` file inside an argument of `\newlabel`, which sagetex reads at startup. Similarly, `\estRecordFloat` causes Python to convert a float to a string using the user-specified format, then this string is written to the `.sout` file inside an argument of `\newlabel`, just as with `\estRecordFormatted`. This applies to global and array variants of the expsagetex functions as well.

So, `\estRecordStr` and its variants can be very useful in case your Python code returns a data type that isn't handled in a satisfactory way by functions belonging to the families of `\estRecordFormatted` or `\estRecordFloat`. All you have to do is to make your Python code format your data as a string with a syntax that is convenient for the LaTeX document; you record this string with `\estRecordStr` or one of its variants, use `\estGet` and `\estRefUsed` as usual and voilà, your new data type is nicely handled by expsagetex. We'll see an example of this approach in section 2.6, where Python code generates a large amount of (x_i, y_i) coordinates from two lists of floats $[x_1, \ldots, x_n]$ and $[y_1, \ldots, y_n]$. The Python code formats the list of (x_i, y_i) coordinates in a form that is suitable for the Ti*k*Z `plot coordinates` operation, which allows one to plot the data right away.

Here is a very simple example with `\estRecordStr` used to record a Sage (Python) string:

```
    some_string = "abc  def"
```

Xabc defY

The two spaces between `abc` and `def` are present in the `.sout` file, but get coalesced into a single space token when `\newlabel` tokenizes its second argument. If this is not desired, the easiest way is probably to use another character instead of space (e.g., `~`).

2.4 Using array data

`expsagetex` offers variants of `\estRecordFormatted`, `\estGRecordFormatted`, `\estRecordFloat`, `\estGRecordFloat`, `\estRecordStr` and `\estGRecordStr` that allow one to easily use an index inside the macro name given for the recording operation. These variants all have the letter `A` (standing for "array") right before `Record`.

Where the non-array versions of the `\est*Record*` commands take one argument specifying the destination macro name, the array versions accept two arguments ⟨*base*⟩ and ⟨*index*⟩ and determine the destination macro name as the result of expanding `\csname` ⟨*base*⟩@⟨*index*⟩`\endcsname`. As a consequence, the ⟨*base*⟩ and ⟨*index*⟩ arguments are recursively expanded, which can be used to dynamically compute a numeric index using for instance `\numexpr`, or whatever you want.

Here is an example that uses Python to evaluate the elements of a Vandermonde matrix:

$$\mathcal{V}\Big(2, \frac{\sqrt{2}}{2}, \frac{2}{7}, \frac{4}{3}, \frac{1}{5}\Big) = \begin{bmatrix} 1 & 2 & 4 & 8 & 16 \\ 1 & \frac{1}{2}\sqrt{2} & \frac{1}{2} & \frac{1}{4}\sqrt{2} & \frac{1}{4} \\ 1 & \frac{2}{7} & \frac{4}{49} & \frac{8}{343} & \frac{16}{2401} \\ 1 & \frac{4}{3} & \frac{16}{9} & \frac{64}{27} & \frac{256}{81} \\ 1 & \frac{1}{5} & \frac{1}{25} & \frac{1}{125} & \frac{1}{625} \end{bmatrix}$$

This technique can be used to plot curves with Ti*k*Z based on coordinates computed by Python, but this is rather wasteful. We'll see a better method in section 2.6.

2.5 Displaying lists

`\estRecordFormatted` is fine for a list of integers:

$$[2, 5, 8, 11, 14, 17, 20]$$

It may do what you want for floats too:

$$[2.0, 5.0, 8.0, 11.0, 14.0, 17.0, 20.0]$$

But beware: if you use `\estRecordFormatted`, floats are formatted by Sage's `latex()` function:

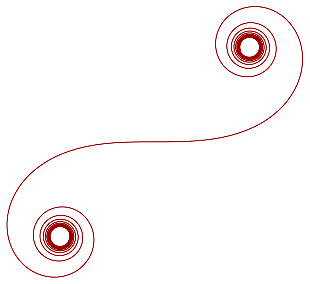
$$\left[1.00000000000000 \times 10^{-10}, 5.00000000000000, 3.00000000000000 \times 10^{-14}\right]$$

Here is a way to use a Python list of floats formatted with no exponent and using a chosen number of decimal places:

$$[0.00000000010000, 5.00000000000000, 0.00000000000003]$$

2.6 Plotting with Ti*k*Z based on Sage computations

Plotting functions based on coordinates computed by Sage can be done using array-style functions of `expsagetex` (see section 2.4), however such an approach is vastly inefficient if the plot uses a significant number of points. A better approach is to prepare one string on the Python side containing all needed coordinates in a format that the LaTeX plotting code can easily handle. This way, only one reference—as in `\label` and `\ref`—is used to pass data for the whole plot. As an example, here is a part of the Euler spiral where all coordinates are computed by Sage and transferred in one go to the LaTeX side (for the full Euler Spiral, you need to make t vary from $-\infty$ to $+\infty$).



$$\begin{cases} x(t) = \displaystyle\int_0^t \cos s^2 \,\mathrm{d}s \\ y(t) = \displaystyle\int_0^t \sin s^2 \,\mathrm{d}s \end{cases}, \quad t \in [-\tfrac{5\pi}{2}, \tfrac{5\pi}{2}]$$

3 Expandable commands work everywhere!

Since `\estGet` is expandable and we have `\usepackage{xfp}` in the preamble, we can do:

```
$\fpeval{3*\estGet[-1]{\Iamseven}}$
```

This expands to 21, since `\Iamseven` contains the last value of x computed by Sage, which is 7 (if Sage hasn't been run yet, it expands to -3 due to the fallback value `-1` specified in the optional argument of `\estGet`).

Given that TeX expands the right-hand side of integer, dimen and glue assignments until this yields the proper syntactic element, one can also use `\estGet` like this:

```
  \count2=4
  \multiply \count2 by \estGet[-1]{\Iamseven}
  \the\count2
  \estRefUsed{\Iamseven}% just to be clean
```

→ 28

Same thing with a `\dimen` register:

```
\dimen0=1pt
\dimen0=\estGet[-1]{\Iamseven}\dimen0
\the\dimen0
\estRefUsed{\Iamseven}% just to be clean
```

→ 7.0pt

And since LaTeX lengths are `\skipdef` tokens and `\setlength` is basically a glue assignment, one can have fun with `\estGet[-1]{\Iamseven}` expanding to the ⟨*factor*⟩ of a ⟨*normal dimen*⟩ (cf. TeXbook p. 270):

```
\dimen0=2pt
\dimen2=3pt
\newlength{\mylength}
\setlength{\mylength}{%
       \estGet[-1]{\Iamseven}\dimen0
  plus \estGet[-1]{\Iamseven}\dimen2}%
\the\mylength
\estRefUsed{\Iamseven}% just to be clean
```

→ 14.0pt plus 21.0pt

4 Troubleshooting

In case something goes wrong when using `sagetex` or `expsagetex`, you may get stuck and unable to perform a full LaTeX run. This can for instance happen if one of your Python expressions is invalid. This kind of error is similar to problems that may occur when something “bad” was written to the `.aux` file. The cure is similar too: locate the error (inspecting the `.sagetex.sage` and `.sagetex.sout` files can help; if the problem is an invalid Python expression, the `sage` run will tell you), regenerate the files containing invalid things after fixing their source, then rerun the LaTeX and `sage` commands as usual. Typically, you'll fix a Python expression in your `.tex` file, remove the `.sagetex.sout` file and rerun LaTeX, `sage` and again LaTeX. If you want to be *really* sure to restart from a clean state, remove the `.sagetex.sage`, `.sagetex.sout` and `.aux` files before redoing the LaTeX & `sage` dance.

5 Technical details

Some comments about what a call such as `\estRecordStr{\mymacro}{`*s*`}` as seen below really does (*s* must evaluate in Python to a string). Informally, one might be tempted to say that this saves the value of *s* in the specified macro, but that would be rather inaccurate. After the call to `\estRecordStr`, the specified macro contains an integer which is used to form a reference name—a label, if you wish—through which we can get the value of *s* using inner gears of the `\label` and `\ref` machinery. The `.sagetex.sout` file written when running Sage on the `.sagetex.sage` file contains `\newlabel` commands that define the associated text (tokens) for reference names `@sageinline0`, `@sageinline1`, etc. The trailing number is what is really stored by the above call to `\estRecordStr` as the replacement text of `\mymacro`. `sagetex.sty` reads this `.sagetex.sout` file at startup when it exists, which defines the labels from the point of view of the LaTeX kernel and allows us to retrieve the Sage output for each recorded expression (after the `\newlabel` commands have been executed, the Sage outputs are available, after some simple data extraction, in macros `\r@@sageinline0`, `\r@@sageinline1`, etc.).