

## REPORT ON OpenDreamKit DELIVERABLE D3.5

### Integration between SAGEMATHCLOUD and SAGE's TRAC server

ERIK BRAY



Due on	31/08/2018 (M36)
Delivered on	31/08/2018
Lead	Université de Versailles Saint-Quentin (UVSQ)
Progress on and finalization of this deliverable has been tracked publicly at: <a href="https://github.com/OpenDreamKit/OpenDreamKit/issues/64">https://github.com/OpenDreamKit/OpenDreamKit/issues/64</a>	

#### 1. STATUS REPORT

Task **T3.7** of WP3 aims to improve the development workflow in mathematical software, to ease contributions by all actors, from experts developers on those projects or coming from other projects, to students, novice users, or non-programmers.

For this deliverable we focused on engaging newcomers to become contributors by reducing the barrier of entry to SAGE development. In particular, trivial contributions ought to be trivial: for example, a user or technical writer that notices a typo or a potential phrasing improvement in the documentation should be encouraged and empowered to fix it on the spot and submit it for review and integration into SAGE.

The Sage community has a history of blurring the line between developer and user, with even new users often encouraged to make code contributions. However, the core group of frequent development contributors remains relatively small, and has a well-established workflow and culture surrounding its existing development tools—in particular the source code hosting and issue tracking service TRAC. This works well for experienced Sage developers, but also creates certain barriers to entry for new contributors, especially those who are new to open source development.

To tackle those barriers, we aimed for a better integration between Sage's Trac-based workflow with platforms that are more modern and familiar to open source novices, such as GITHUB and GITLAB, without immediately breaking the Sage project's existing workflow.<sup>1</sup>

Beyond tackling the immediate goal of this deliverable, we believe that this approach also opens the doors to long-term improvements in Sage's development workflow, as the advantages of using these new tools become more apparent. Part of this work also serves as a prerequisite to improvements in Sage's continuous integration practices, which are being explored in D3.8: "Continuous integration platform for multi-platform build/test."

An aspect that Sage (and indeed any open source mathematics software) prides itself on over closed-source alternatives is the ability to peer inside the software to see how it works. It is not a black box and one does not have to take its results for granted—it is possible to look at the source code in order to understand how a particular algorithm works, or even modify and improve it. To this end we sought ways to more tightly couple Sage's API documentation to its source code and, through access to the source code, encourage contribution of improvements.

To these ends, this deliverable includes the following improvements to Sage's development platform and documentation:

<sup>1</sup>Upon writing the proposal we envisioned using SAGEMATHCLOUD (now COCALC) as a major development platform for Sage, hence the title of this deliverable. However, while COCALC remains quite viable for this purpose, it became evident that there was a need for solutions not explicitly tied to COCALC. See Section 2 for discussion.

- (1) Enable users with existing GitHub credentials to log into Sage’s Trac site.
- (2) Create a presence for Sage on the GitLab collaborative source code sharing service, including the ability to accept patches to Sage’s documentation and source code in the form of *merge requests* (a.k.a. *pull requests*).
- (3) Link GITLAB merge requests to tickets in Sage’s Trac issue tracker in a manner that fully integrates with Sage’s existing development workflow.
- (4) Create a direct path from Sage’s online API documentation to the source code, and from the source code directly an interface to edit the source code and submit a patch (as a merge request) all without the user having to leave their web browser.

## 2. REDESIGNED WORK PLAN TO BEST ACHIEVE THE DELIVERABLE AIM

This deliverable was originally entitled “Integration between SageMathCloud and Sage’s TRAC server”. The vision and rationale were as follows: compiling Sage and its dependencies requires a significant amount of computing power, and also a number of development software prerequisites that can be hard to set up on a developer’s personal computer, especially in the context of a one or two day workshop.

At the time of the proposal, COCALC (then still named SAGEMATHCLOUD) had recently begun providing a cloud-hosted virtual environment that was immediately suited to building, running, and developing Sage. This looked promising and we envisioned that users doing development on Sage in a COCALC environment ought to have a way to submit their development efforts directly to Sage’s TRAC (hereafter referred to as Sage-Trac).

We started the work on this deliverable by an in depth review of its aim – reducing the entry barrier to Sage-development – and best means to achieve it.

We noted that the existing command-line development tools for Sage (e.g. the community-maintained `git trac` utility) already provided integration with TRAC independently of the development environment, and had gained wide adoption by the community. So it was unclear exactly what the advantage would have been to adding features to COCALC solely for the purpose of integrating with Sage-Trac, especially since those features would be very specific to Sage and would not integrate well into the direction COCALC has moved in, as a more general-purpose computing environment. It was also unclear how many people were actually using COCALC as their primary platform for Sage development, while it is certain that most of the core Sage developers are not doing so.

Additionally, as previously noted, modern cloud-based code sharing and issue tracking sites such as GITHUB and GITLAB have been gaining incredible momentum, and almost anyone working on modern open source projects—be they veteran developers or novices – are likely to have encountered them and be familiar with them. They are also backed by large companies and sponsors and are unlikely to go away soon.

Meanwhile TRAC, while an excellent system and familiar to many developers who worked with it in the early 2000’s, is showing its age. The TRAC software has few active developers anymore, and relative difficulty of setting up a self-hosted Trac-based project (as opposed to the near zero effort of adding a project to cloud-based services) means very few new projects are adopting it.<sup>2</sup>

Therefore it seemed a better focus of our efforts to open Sage’s development workflow up to the use of more modern development platforms (see the next section for discussion of how we are integrating GITLAB into the workflow). We expect this will have benefits in the future – more modern development tools integrate well with services like GITHUB or GITLAB out-of-the-box,

---

<sup>2</sup>The author of this report is a former developer of Trac, having contributed non-trivial feature enhancements and bug fixes, and authored numerous plug-ins to Trac.

whereas integrating development tools with TRAC requires extra work on the part of the Sage community that might be better spent elsewhere.

Despite not being tied to COCALC, all the work carried out here also benefits SAGE contributors that use COCALC, or any of the other OpenDreamKit-related VRE. We have had some discussion about how to further integrate issue tracking and source code contributions into VREs; while no consensus has emerged as to what form this would take (Would one submit issues against a VRE? Against the libraries and software it uses? Both?), it would most likely integrate best with modern code-sharing services.

### 3. DESCRIPTION OF THE ACHIEVEMENTS

#### 3.1. Login to Trac via GitHub usernames

A perennial problem in managing the Sage community's self-hosted TRAC site has been dealing with user registration and authentication. As with many bug-tracking sites, Sage-Trac requires users to register and log into the site in order to post bugs and feature requests.

Allowing anyone on the internet to register an account—not to mention allowing anonymous users to post to the site—regularly opened it up to a flood of spam. Even adding CAPTCHAs to the registration process was not fool-proof, as CAPTCHAs are being defeated more and more by either machine learning or cheap manual labor. Therefore, for several years the Sage community has resorted to a process of manual user registration: potential users of the site would send an e-mail to a dedicated address with a manually written application form, and eventually a site administrator—if the application appeared genuine—would approve the application and assign the user a username and temporary password, again over e-mail.

This process has worked, and does not generally leave anybody out. But it is still a slow, and labor-intensive process. One would not be surprised if it has discouraged potential bug-reports, with users deciding it was not worth their time, or forgetting about the issue while waiting for their account registration to be approved.

Therefore, we sought to make logging into Sage-Trac a more painless and instantaneous process.

GITHUB, with its over 40 million users as of writing, is the largest source code hosting site in the world. Many people, even who are not full-time software developers, have a GitHub account if they have worked at all peripherally with open source software (if nothing else, in order to submit bug reports to projects). By making it possible to log into Sage-Trac with one's existing GitHub account, this opens Sage development up instantly to any of those 40+ million users without additional effort. To date, this has not contributed any spam to Sage-Trac either, as we are able to leverage GitHub's own spam management.

We still also allow existing accounts to log in, and maintain the manual registration system so that users who do not already have – or prefer not to obtain – GitHub accounts may be registered. However, in the four months since launching this new feature on February 27, 2018, nearly one fourth (31 out of 130) of the users who submitted new tickets were logged into the site using GitHub. Of the other 130, only a handful were new manual registrations, while the rest were using accounts they already had on the system before February 27. We expect the number of contributions from GitHub users to grow, and will consider adding the ability to log in via other external authentication providers, such as GITLAB and Google.

#### 3.2. GITLAB-TRAC integration

While making it easier for new contributors to SAGE to log into its TRAC site eliminates one barrier to contribution, another barrier is learning its somewhat unique development and source code contribution and review workflow. This workflow is well-documented and custom tooling is provided to make it easier (e.g. the `git trac` command for posting Git branches directly to Trac from the command line); nevertheless it is still not an entirely familiar workflow for anyone

new to Sage—experienced developers and novices alike. Experience gained through our many training workshops suggests that it usually takes a one hour demo and a couple walk-me-through first contributions (often spread over several such workshops) to get used to the workflow and become autonomous. And few people – mostly already advanced programmers – learn the workflow by themselves outside of training workshops, even for trivial contributions.

Novices in particular – especially those who became involved in open source development within the last five years – will tend to be more familiar with what is known as the “GitHub Workflow”<sup>3</sup> as was popularized by GITHUB.<sup>4</sup> Thus, the availability of a complementary GitHub-like workflow would lower the barrier to contributions. As we will see in the next section, this also makes it easier to go directly from browsing Sage’s documentation to making small source code and documentation contributions.

In order to accomplish this, we might have used a mirror of Sage’s Git repository on GITHUB (indeed, such a mirror already exists). However, for this purpose we chose instead to focus on GITLAB—a competitor to GITHUB that provides a similar product, but with an *open core* business model, meaning that the majority of the software that runs the service `GitLab.com` is provided under an open source license, while the company maintains optional “enterprise” features under a separate proprietary license.

A few reasons we chose GITLAB for this purpose over GITHUB<sup>5</sup> include:

- (1) Many members of Sage’s community have strongly-held political objections to relying on closed-source commercial software for development, and would be reluctant to adopt GitHub into their workflow (the aforementioned GitHub login for Trac being acceptable due only to it being optional). We hope that GitLab, itself being *mostly* open source, will be a more politically viable alternative. Sage itself being an open source project, we are happy to work hand-in-hand with other open source projects.
- (2) Although we now host a mirror of the Sage project on the cloud-based `GitLab.com` service, should the need arise for any reason we could easily migrate this mirror to a any other instance of the GitLab service, e.g. self hosted using GitLab’s *software*.

Indeed, the Sage community has long prided itself on being almost entirely independent of commercial products or services for hosting its development tools; while using `GitLab.com` is convenient (less effort spent on server maintenance that could be spent improving Sage itself), it is encouraging to know that the possibility exists.

- (3) `GitLab.com` allows users to log in with their GitHub usernames, Google accounts, and other third-party authentication providers (in addition to registration directly with GitLab). Thus, although GitHub has a wider existing user base, all users with GitHub accounts can log into GitLab without significant effort.
- (4) GitLab has some features that are not present on GitHub. In particular, GitLab has a powerful built-in continuous integration and deployment platform that we have found well-suited for Sage. Our efforts in using GitLab’s continuous integration/deployment tools are reported on in D3.8: “Continuous integration platform for multi-platform build/test.”. Some of this work directly impacts Task **T3.7** by reducing the entry barrier to the *review* part of the development workflow.

<sup>3</sup><https://guides.github.com/introduction/flow/>

<sup>4</sup>Sage’s existing process is not that different from the GitHub workflow, but small differences in semantics and user experience make this less immediately obvious.

<sup>5</sup>The choice of GitLab for hosting a Sage project may appear to stand in contrast with the previous section of this deliverable, which added authentication to Sage-Trac via *GitHub*. However, as noted above, GitLab also allows authentication with GitHub accounts, taking advantage of GitHub’s existing wide user base. Therefore, we focused primarily on GitHub-based authentication. In the future we would also like to add the ability for users to authenticate with Sage-Trac via GitLab, if there is demand for it.

- (5) Although GitLab the company is currently based out of San Francisco (albeit with only one employee at its San Francisco office) the project was started in Europe (Ukraine) and a large portion of its all-remote employees are based in Europe.

For these reasons, among others, we have set up a new mirror<sup>6</sup> of Sage’s Git repository on GitLab, with the ability to accept contributions in the form of *merge requests*, which are the same as what GitHub calls “pull requests”.

The challenge in opening up Sage to accepting contributions via GitLab was doing so in a way that did not significantly disrupt Sage’s existing Trac-based development workflow and to make gradual introduction of a new kind of workflow more socially acceptable; the community indeed tends to have strong opinions on the subject.

Our solution was to implement a plug-in for Sage-Trac that uses GitLab’s API to automatically open a *ticket* on Sage-Trac for each merge request opened on GitLab. That way, Sage developers watching Sage-Trac for new contributions are notified even without having to go to GitLab. The proposed code changes in the merge request – which are hosted on GitLab – are also automatically synchronized to Sage’s main Git repository, so that all of Sage’s existing maintenance tools continue to work as normal even for contributions that came originally through GitLab. When a ticket is *closed* on Sage-Trac (i.e. the changes were accepted, or rejected) the corresponding merge request on GitLab is also automatically closed. In this way, veteran Sage developers may, if they wish, accept contributions through GitLab without ever actually having to go to GitLab.

That said, we hope that most of the Sage development community *will* consider using GitLab. In particular, we believe that the more modern code review tools provided by GitLab will prove too compelling to ignore. There is a small risk associated with this: many code change tickets generate discussion – sometimes a significant amount – in the form of code review comments or sometimes even debate. If an issue is tracked on both Sage-Trac and on GitLab, there is a risk of that discussion becoming fragmented between the two sites and difficult to follow. This can be mitigated either through policy (e.g. keep discussion of a single issue on one site or the other, but not both), or in the form of a technical solution (e.g. enhance the existing synchronization plug-in to synchronize discussions between the two sites; this is non-trivial, however).

Either way, there is prior art for this process of adopting a modern code sharing site while maintaining a legacy issue tracking system. Most notably, the CPython project (the reference implementation of the Python language and its standard library – what is typically referred to as just “Python”) switched to a similar model beginning in February 2017: Issues are still tracked primarily on the project’s legacy bug tracking system – which carries in it a wealth of historical knowledge and lingering open issues – while bug fixes and new features can be submitted as pull requests on GitHub which are automatically linked to their corresponding issues on the legacy system. Since then, over 6500<sup>7</sup> merge requests have been accepted through GitHub. In an informal query<sup>8</sup> to the CPython core developers one year after switching to the new hybrid workflow, its primary architect, Brett Cannon, asked how the new workflow was working and the response was overwhelmingly positive. Frequently cited as a benefit were the several bots that were created by the community to help manage pull request submissions. Many of these bots would be useful for Sage as well, and although they were created to run on GitHub, the GitLab API is similar-enough that these bots could be ported to GitLab with minimal effort. There was also a general interest (including by Python’s creator Guido van Rossum) to move forward with migrating all issues to GitHub, and setting the legacy bug tracker to read-only; the details of this migration, however, remain a subject of debate.<sup>9</sup> We believe a similar outcome for Sage’s partial

<sup>6</sup><https://gitlab.com/sagemath/sage>

<sup>7</sup><https://github.com/python/cpython/pulls?utf8=%E2%9C%93&q=is%3Apr+is%3Aclosed+is%3Amerged>

<sup>8</sup><https://mail.python.org/pipermail/python-dev/2018-February/152200.html>

<sup>9</sup><https://lwn.net/Articles/754779/>



migration to GitLab would be desirable, but it remains to be seen how well it will be adopted by the Sage community.

### 3.3. From Sage API documentation to source code contributions

As mentioned in the introduction, a selling point of Sage over closed-source mathematics software is easy access to the source code; the ability to actually inspect its algorithms – and maybe even improve them – rather than simply plug in input and trust that the output is correct.

Sage has always made it easy to view the source code for its classes and functions, not just by downloading and viewing the sources directly, but also at runtime using its detailed inspection capabilities. For example, to see the source code for Sage’s `Integer` class it is possible, at the Sage input prompt, to write `Integer??` (this is in fact a standard feature of the IPYTHON interface upon which Sage’s user interface is built).

However, the command prompt is not always the easiest place to explore Sage’s capabilities – sometimes reading the online documentation gives a bigger picture. Reciprocally, while the documentation includes English language prose and usage examples that explain what different classes and functions do, sometimes browsing the source code helps get the full picture.

To this end, we have added<sup>10</sup> two new hyperlinks that appear in the headings of the API documentation for all classes, methods, functions, and other documented objects in Sage.

```
class sage.categories.euclidean_domains.EuclideanDomains(s=None) \[edit on gitlab\] \[source\]
Bases: sage.categories.category_singleton.Category_singleton
```

The category of constructive euclidean domains, i.e., one can divide producing a quotient and a remainder where the remainder is either zero or its `ElementMethods.euclidean_degree()` is smaller than the divisor.

#### EXAMPLES:

```
sage: EuclideanDomains()
Category of euclidean domains
sage: EuclideanDomains().super_categories()
[Category of principal ideal domains]
```

```
class ElementMethods \[edit on gitlab\] \[source\]
```

```
euclidean_degree() \[edit on gitlab\] \[source\]
```

Return the degree of this element as an element of an Euclidean domain, i.e., for elements  $a$ ,  $b$  the euclidean degree  $f$  satisfies the usual properties:

1. if  $b$  is not zero, then there are elements  $q$  and  $r$  such that  $a = bq + r$  with  $r = 0$  or  $f(r) < f(b)$
2. if  $a, b$  are not zero, then  $f(a) \leq f(ab)$

**Note:** The name `euclidean_degree` was chosen because the euclidean function has different names in different contexts, e.g., absolute value for integers, degree for polynomials.

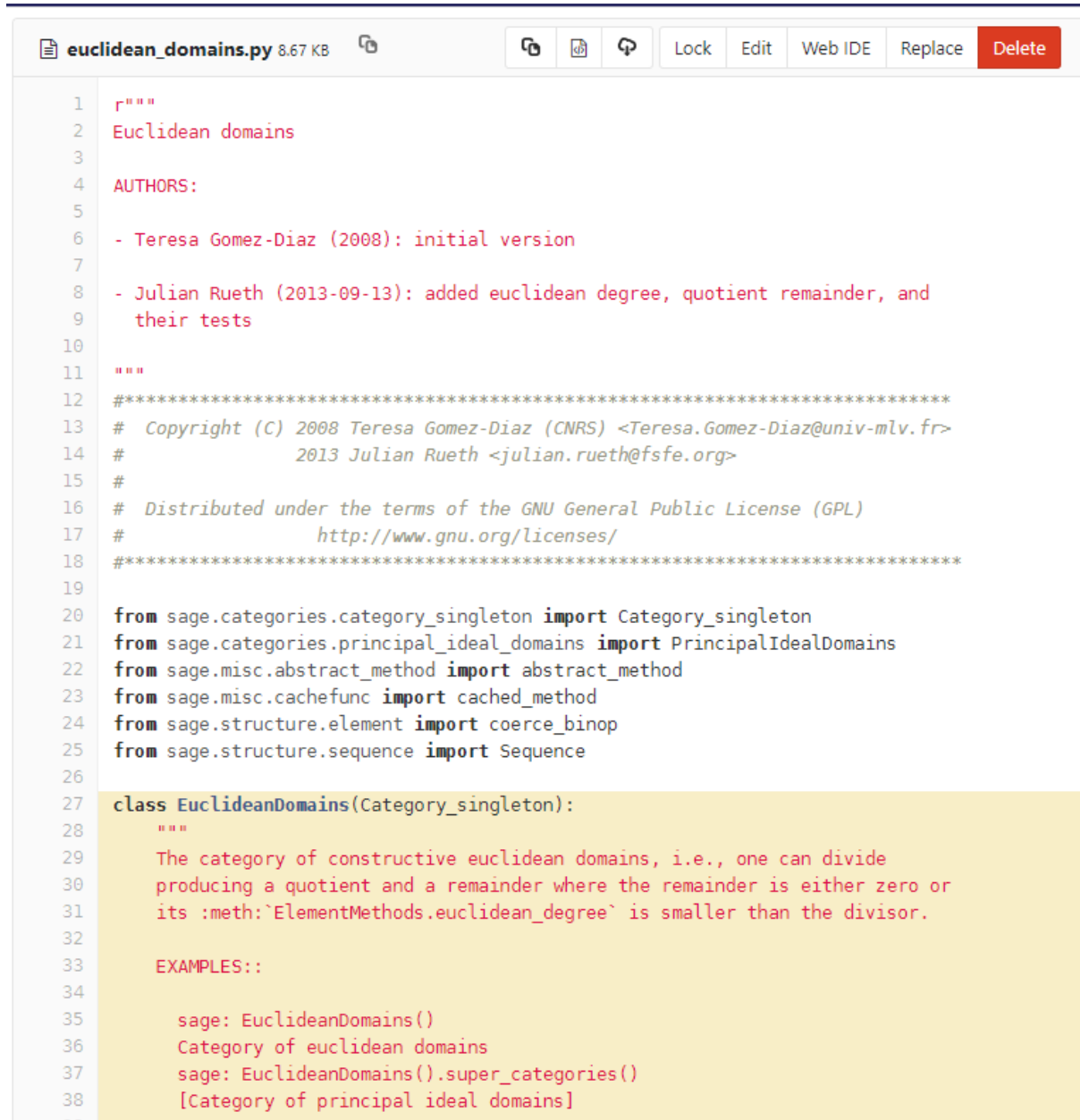
FIGURE 1. Example Sage API documentation with new source links.

The `[source]` link displays the source code (in this case for the `EuclideanDomains` class) in the context of the source file it’s defined in, without leaving the Sage documentation site (<http://doc.sagemath.org>). The second link, `[edit on gitlab]`, is what partially

<sup>10</sup><https://trac.sagemath.org/ticket/25914>

motivates having a mirror of the Sage repository hosted on GitLab, with the ability to accept merge requests, as described in the previous section. This feature is inspired by a similar feature used by the Astropy project in its API documentation<sup>11</sup>.

This link opens the source file containing the object in question (e.g. `EuclideanDomains`) in the source code browser for the Sage project on GitLab (see Figure 2). Users who are logged into, or who subsequently log into their GitLab accounts, then have the option to edit that source file directly in the browser using the web-based code editor provided by GitLab. Because nobody has permission to edit the Sage source code directly, when they save their changes this results in a Git commit to the user's personal copy of the repository, and a *merge request* is opened in the main Sage repository. With the GitLab-Trac integration this also results in a ticket on Sage's Trac for the requested changes to the code and/or documentation, triggering the review process.



```

1  r"""
2  Euclidean domains
3
4  AUTHORS:
5
6  - Teresa Gomez-Diaz (2008): initial version
7
8  - Julian Rueth (2013-09-13): added euclidean degree, quotient remainder, and
9    their tests
10
11  """
12  #*****
13  # Copyright (C) 2008 Teresa Gomez-Diaz (CNRS) <Teresa.Gomez-Diaz@univ-mlv.fr>
14  #      2013 Julian Rueth <julian.rueth@fsfe.org>
15  #
16  # Distributed under the terms of the GNU General Public License (GPL)
17  #      http://www.gnu.org/licenses/
18  #*****
19
20  from sage.categories.category_singleton import Category_singleton
21  from sage.categories.principal_ideal_domains import PrincipalIdealDomains
22  from sage.misc.abstract_method import abstract_method
23  from sage.misc.cachefunc import cached_method
24  from sage.structure.element import coerce_binop
25  from sage.structure.sequence import Sequence
26
27  class EuclideanDomains(Category_singleton):
28      """
29      The category of constructive euclidean domains, i.e., one can divide
30      producing a quotient and a remainder where the remainder is either zero or
31      its :meth:`ElementMethods.euclidean_degree` is smaller than the divisor.
32
33      EXAMPLES::
34
35      sage: EuclideanDomains()
36      Category of euclidean domains
37      sage: EuclideanDomains().super_categories()
38      [Category of principal ideal domains]
39  """

```

FIGURE 2. Editing Sage source code on GitLab via documentation link.

<sup>11</sup>See for example <http://docs.astropy.org/en/stable/api/astropy.units.Quantity.html#astropy.units.Quantity>

In addition to API documentation, much of which is automatically generated from the source code, Sage also has a large amount of prose documentation in English and other languages, such as high-level usage documentation for Sage and tutorials. This material is all also tracked in Sage's source code repository, and is something we want to make easy to crowd-edit. Therefore we also add an `[edit on gitlab]` link on all the prose documentation, which works the same as for the API documentation: users are allowed to edit the source files for the prose documentation and submit their edits as merge requests. This would be encouraged for anything from typos and minor clarifications, to major documentation updates. By making it a process of a few clicks from reading the documentation to proposing edits, we believe this will lead to more rapid quality improvements in Sage's documentation.

#### 4. FUTURE WORK

It is currently possible in Sage to view the documentation for an object directly from the Sage command prompt by entering the name of the object followed by a question mark (e.g. `Integer?`). However, this is currently a plain-text representation of the object's documentation, and lacks the rich text features such as fully-rendered equations and images that Sage's web-based HTML documentation has. Even in the web-based Jupyter Notebook environment, only the plain-text documentation is displayed, not the HTML documentation. The plain-text documentation displayed in the Sage console does not give any link to the richer HTML documentation; users have to know *a priori* that the HTML documentation exists, and to seek it out.

We realized upon completion of this deliverable that we should close the loop on that: The plain-text documentation displayed in the console should include a URL to the correct version of the online documentation for an object, so that it is easily discoverable by the user. This would allow a complete path of using an object or function in Sage → viewing the in-place documentation for that object → viewing the online HTML documentation for the object → (if desired) editing the documentation and/or source code of that object and proposing changes back to the Sage project via GITLAB. Furthermore, in the web-based Jupyter Notebook, there should be no need to display the plain-text documentation in the first place: When entering `Integer?` in the notebook it should take the user directly to the relevant HTML documentation.

In fact, the SAGE kernel for Jupyter already serves Sage's HTML documentation via a URL in the notebook, and which can be found through a "Help" menu. However, we do not yet have the ability to go directly from querying a specific object to displaying the HTML documentation for that object. The user instead has to load the documentation from the "Help" menu and use the documentation's search feature to find the documentation for a specific object. We believe no more than a few additional days of work should be necessary to close this disconnect.

As a related issue, once a merge request has been created for a documentation change, the author and reviewers would like to browse a version of the documentation that includes the changes. Work done in D3.8: "Continuous integration platform for multi-platform build/test." automates the process of producing a build of Sage, including its documentation, from each merge request that is opened on Sage's GitLab project, as well as for tickets created directly on Sage-Trac. The result of this build, if it succeeds, is a Docker image containing the built version of Sage which the author of the merge request (or anyone else) may then run and try out online through cloud-hosted notebook services like Binder<sup>12</sup> or a JupyterHub<sup>13</sup> instance.

With this work we have made it easier to make simple contributions: in particular documentation and trivial code changes. Non-trivial code changes or additions requires checking that the updated code builds successfully and passes the test suite. This remains an onerous process

---

<sup>12</sup><https://mybinder.org/>

<sup>13</sup><https://jupyterhub.readthedocs.io/>



```
Signature:      laplace(ex, t, s, algorithm='maxima')
Docstring:
Return the Laplace transform with respect to the variable t and
transform parameter s, if possible.

If this function cannot find a solution, a formal function is
returned. The function that is returned may be viewed as a function
of s.

DEFINITION:

The Laplace transform of a function f(t), defined for all real
numbers t >= 0, is the function F(s) defined by

    F(s) = int_{0}^{infty} e^{-st} f(t) dt.

INPUT:

* "ex" - a symbolic expression
* "t" - independent variable
* "s" - transform parameter
```

`sage.calculus.calculus.laplace(ex, t, s, algorithm='maxima')` [\[edit on gitlab\]](#) [\[source\]](#)

Return the Laplace transform with respect to the variable  $t$  and transform parameter  $s$ , if possible.

If this function cannot find a solution, a formal function is returned. The function that is returned may be viewed as a function of  $s$ .

DEFINITION:

The Laplace transform of a function  $f(t)$ , defined for all real numbers  $t \geq 0$ , is the function  $F(s)$  defined by

$$F(s) = \int_0^{\infty} e^{-st} f(t) dt.$$

INPUT:

- $ex$  - a symbolic expression
- $t$  - independent variable
- $s$  - transform parameter

FIGURE 3. Example of the plain-text documentation shown in the Sage console versus the web-based documentation for the same object; we will make the latter more accessible from the former, and directly accessible through the notebook with greater ease.

for Sage in particular, as one must set up the development environment and go through the lengthy build process. The work done in D3.8 will make it easier to (near) instantly launch a Sage development environment in a Docker container, as well as build new changes on top of it. Streamlining the process of making source code edits, pushing those edits to GitLab,

and obtaining and running working build of Sage based on those edits is an avenue for further investigation.

Disclaimer: this report, together with its annexes and the reports for the earlier deliverables, is self contained for auditing and reviewing purposes. Hyperlinks to external resources are meant as a convenience for casual readers wishing to follow our progress; such links have been checked for correctness at the time of submission of the deliverable, but there is no guarantee implied that they will remain valid.