REPORT ON OpenDreamKit DELIVERABLE D4.10

Second version of the PARI Python/Cython bindings

VINCENT DELECROIX, JEROEN DEMEYER, VINCENT KLEIN

Due on	31/08/2018 (M36)
Delivered on	31/08/2018
Lead	CNRS (CNRS)
Progress on and finalization of this deliverable has been tracked publicly at: `https://github.com/OpenDreamKit/OpenDreamKit/issues/84`	

CONTENTS

1. INTRODUCTION

The PARI library is a state-of-the-art library for number theory, developed at Université de Bordeaux. It is an important component of the SAGE computational system: SAGE uses it for example to implement number fields and elliptic curves. PARI itself is just a C library but it comes with a command-line interface called GP. Together, these form the software package PARI/GP. Thanks to OpenDreamKit (see D4.4), there is also a JUPYTER interface implementing the same language. Unfortunately, GP is a specialised language which is not so easy to integrate with other software.

This deliverable is a Python interface for PARI/GP. Since Python is a widely-used programming language, this makes it easy to integrate PARI/GP with other (scientific) software. As such, it is an important component of a VRE for researchers who require number-theoretical computations.

Since SAGE is implemented in Python, it is also necessary for SAGE to have a Python interface to PARI/GP.

In the first reporting period of the OpenDreamKit project, we created PYTHON bindings for PARI/GP in a package called *cypari2*. We reported on that in D4.1 and this deliverable is a follow-up.

2. Examples

Let us start by giving two concrete examples of using cypari2. These examples were run in the IPython command line interface to Python. The first is a basic example of computing the number of points on the elliptic curve $Y^2 = X^3 + 2X + 3$ over the finite field $\mathbb{F}_{11}$ (the answer is 12):

```
In [1]: from cypari2 import Pari; pari = Pari()

In [2]: E = pari.ellinit([2, 3])

In [3]: E.ellcard(11)
Out[3]: 12
```

We can easily use the parallel features of PARI/GP (see **T5.1**). In this example, we factor all numbers of the form $2^n - 1$ for n in $[1, 200]$:

```
In [1]: from cypari2 import Pari; pari = Pari()

In [2]: pari.default("nbthreads", 2)

In [3]: L = [2**n - 1 for n in range(1, 201)]

In [4]: %time res = pari.parapply("factor", L)
CPU times: user 6.76 s, sys: 170 ms, total: 6.93 s
Wall time: 3.6 s
```

The timings show that this computation took 6.93 seconds of CPU time but it finished in 3.6 seconds. This is because 2 threads were used in parallel (as configured in the `nbthreads` default).

3. New features

Since the first deliverable D4.1, we added various new features to the cypari2 package, which we list here.

3.1. Support for GP lists

The PARI/GP data type `t_LIST` is now supported. This is a dynamic array type, very similar to a Python `list`. It is not used internally in the PARI library, but it is a useful programming tool for the end user.

Here, we show how such a list can be used. We create a list from the vector `[1,2,3]` and then insert the element 5 at the third position:

```
In [1]: from cypari2 import Pari; pari = Pari()

In [2]: L = pari.List([1,2,3])

In [3]: L
Out[3]: List([1, 2, 3])

In [4]: L.listinsert(5, 3)
Out[4]: 5

In [5]: L
Out[5]: List([1, 2, 5, 3])
```

3.2. **Plotting support**

As part of the work for D4.7, we implemented SVG high-resolution plotting in PARI. This was first developed for the PARI/GP Jupyter kernel, but the same feature now also works using cypari2 in the PYTHON or SAGE JUPYTER kernels. Independently, our SVG implementation has been reused upstream as a basis for the `plotexport` command newly available in `pari-2.11.0`.

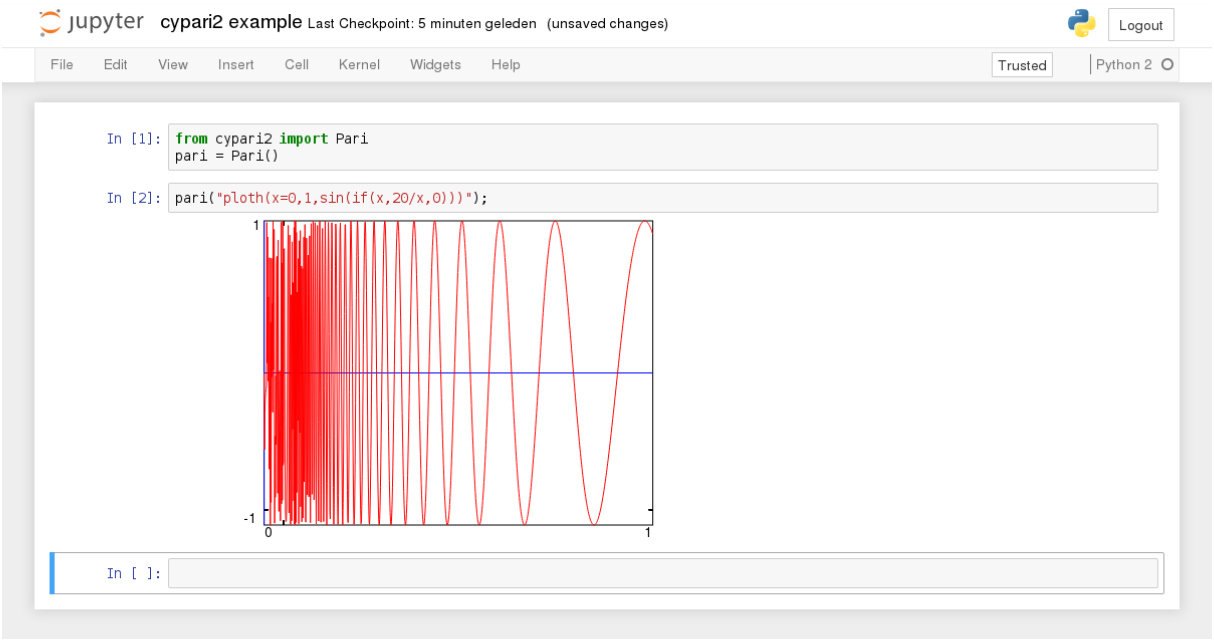

FIGURE 1. Plotting using cypari2 in the Python Jupyter kernel

3.3. **Avoiding copies from the PARI stack**

The first version of cypari2 always copied the results from a PARI/GP function to the heap (a part of memory where dynamic memory allocations happen). This meant that every function call had the overhead of allocating and copying memory (unrelated to the overhead of managing a Python object).

For efficiency, PARI internally uses a stack to store intermediate results. One of the goals stated for **T4.12** was to keep objects on the PARI stack. This has been implemented in a new version of cypari2 (which is not released yet). One disadvantage of a stack is that it is impossible to free an object in the middle of the used stack space: only the most recently allocated object can be freed. Therefore, cypari2 has a strategy to copy all objects from the stack to the heap once too much stack space is used. By delaying the copying, only the objects which are still referenced need to be copied. This is usually a significant saving over copying all objects since temporary objects typically do not need to be copied.

3.4. **Improved auto-generation**

As already reported in D4.1, large parts of the cypari2 source code are auto-generated. Indeed, PARI ships a file `pari.desc` containing, for each function, all machine-readable data which is needed to generate an interface to that function. This was originally meant to generate a C interface, but it applies also to a Cython interface. In pari-2.11, there are 1040 such auto-generated functions.

We have made two improvements to this auto-generation process. First of all, the Cython declarations to directly call the C functions corresponding to GP functions are now auto-generated. This is meant to be used by external Cython code (the original auto-generated code was only for

the Python interface) wanting to interface efficiently with PARI. The Cython interface has the advantage of avoiding the overhead of calling Python methods and creating Python objects. It also allows finer control over PARI internals. This is an example showing the difference between using the Python interface of cypari2 and direct calling of PARI functions:

```
In [1]: %load_ext cython

In [2]: %%cython
   ...: from cysignals.signals cimport sig_on
   ...: from cypari2.gen cimport Gen, GEN
   ...: from cypari2.paridecl cimport gcos, DEFAULTPREC
   ...: from cypari2.stack cimport new_gen
   ...:
   ...: def cosfixpoint_py(Gen a, Py_ssize_t iterations):
   ...:     cdef Py_ssize_t i
   ...:     for i in range(iterations):
   ...:         a = a.cos()
   ...:     return a
   ...:
   ...: def cosfixpoint_cy(Gen a, Py_ssize_t iterations):
   ...:     cdef Py_ssize_t i
   ...:     cdef GEN g = a.g
   ...:     sig_on()
   ...:     for i in range(iterations):
   ...:         g = gcos(g, DEFAULTPREC)
   ...:     return new_gen(g)

In [3]: from cypari2 import Pari; pari = Pari()

In [4]: %time cosfixpoint_py(pari(0.0), 100000)
CPU times: user 177 ms, sys: 0 ns, total: 177 ms
Wall time: 177 ms
Out[4]: 0.739085133215161

In [5]: %time cosfixpoint_cy(pari(0.0), 100000)
CPU times: user 144 ms, sys: 0 ns, total: 144 ms
Wall time: 143 ms
Out[5]: 0.739085133215161
```

Second, cypari2 now automatically generates a `DeprecationWarning` for obsolete PARI/GP functions. The following example shows such a deprecation warning, including the date when the function was deprecated:

```
In [1]: from cypari2 import Pari; pari = Pari()

In [2]: pari.polred("x^2 + 4")
/usr/local/src/sage-config/local/bin/ipython:1:
DeprecationWarning: the PARI/GP function polred
is obsolete (2013-03-27)
  #!/usr/local/src/sage-config/local/bin/python2
Out[2]: [x, x^2 + 1]
```

3.5. **Documentation**

PARI/GP has very extensive documentation for each function, called *help*. This help is now translated to *docstrings* (in-line documentation for Python code) for the Python interface. This requires a translation from LaTeX (the language used for the PARI/GP help) to reStructuredText (which is understood by SPHINX, the package building the documentation for cypari2).

Since the documentation appears in Cython-generated functions, we needed a few improvements in SPHINX to make it work with Cython functions. This is detailed in the report of D4.13. Following standard practice in the Python community, the built documentation is now also available on the documentation hosting service ReadTheDocs; see `https://cypari2.readthedocs.io/`.

4. CYPARI2 AS A STAND-ALONE PYTHON PACKAGE

The primary motivation of the cypari2 package was to be used as part of the SAGE distribution. However, we also want it to be usable also by itself, outside of SAGE. Moving the PARI/GP interface from SAGE to a separate package cypari2 is also a testcase for more generally moving code away from SAGE. This test was positive: interfacing PARI/GP from SAGE using cypari2 works equally well now as when it was part of SAGE. Moreover, it simplifies the SAGE build system: the auto-generation was a complication for building the SAGE library. This complication still exists in cypari2, but there it is less of a problem since cyari2 is a lot smaller and probably not built as often as SAGE is.

At the May 2018 OpenDreamKit workshop in Cernay, Jeroen Demeyer used cypari2 as a use-case in a presentation about writing Cython bindings for a C library. In the mean time, other similar interfaces are also being moved away from SAGE. One such example is pplpy, an interface to PPL (Parma Polyhedra Library).

We made several improvements to make cypari2 behave better as stand-alone package: the code is now fully compatible with PYTHON 3 (more precisely, version 3.4 or later) and with multiple versions of PARI/GP (from 2.9.0 to the most recent release, 2.11.0). To ensure that it remains compatible, multiple PYTHON and PARI/GP versions are tested on the continuous integration platform Travis CI (see D3.8 for more information about continuous integration in general).

One of the original goals of this deliverable was replacing the older cypari package. The main stumbling block for this is Windows compatibility of cysignals, a package for interrupt, signal and error handling of Cython code, written by OpenDreamKit as part of **T4.12**. This package is a dependency of cypari2 and it currently does not work natively on Windows, only on Unix-like systems (in particular on Linux and Mac OS X but also Cygwin on Windows). Because of that, cypari2 does not work on Windows either. Significant progress has been made to port cysignals to Windows. We plan to finish this task in the final year of OpenDreamKit, mainly with resources from Ghent University and Université de Bordeaux. Given that PARI/GP itself does work on Windows, it should then be relatively easy to get cypari2 working on Windows. Once that is done, it will be a viable replacement for cypari.