Politechnika Wrocławska

Design and implementation issues of a computer algebra system in an interpreted, dynamically typed programming language

Mateusz Paprocki <mattpap@gmail.com>
Wrocław University of Technology

June 7, 2010

Introduction

Design and implementation issues
of a computer algebra system
in an interpreted, dynamically typed
programming language

- supervisor: dr inż. Krzysztof Juszczyszyn
- language: English

Plan of presentation

- Remainder of the previous talk
- Multi–level structure
- Implemented algorithms
- Ground types
- Using Cython
- Future plans

Thesis gloals

What I would like to achieve

Design and implement computer algebra module for SymPy:

- conform SymPy's goals
- introduce multi–level structure
- multiple ground types
- utilize pure mode Cython
- ...

Thesis gloals

What I would like to achieve

Design and implement computer algebra module for SymPy:

- conform SymPy's goals
- introduce multi–level structure
- multiple ground types
- utilize pure mode Cython
- ...

Thesis gloals

What I would like to achieve

Design and implement computer algebra module for SymPy:

- conform SymPy's goals
- introduce multi–level structure
- multiple ground types
- utilize pure mode Cython
- ...

Thesis gloals

What I would like to achieve

Design and implement computer algebra module for SymPy:

- conform SymPy's goals
- introduce multi–level structure
- multiple ground types
- utilize pure mode Cython
- . . .

Thesis gloals

What I would like to achieve

Design and implement computer algebra module for SymPy:

- conform SymPy's goals
- introduce multi–level structure
- multiple ground types
- utilize pure mode Cython
- ...

Multi–level structure

```
>>> f3 = x**10 - 1
>>> %timeit factor_list(f3)
100 loops, best of 3: 5.57 ms per loop

>>> f2 = Poly(x**10 - 1, x, domain='ZZ')
>>> %timeit f2.factor_list()
100 loops, best of 3: 2.15 ms per loop

>>> f1 = DMP([mpz(1), mpz(0), mpz(0), mpz(0), mpz(0),
... mpz(0), mpz(0), mpz(0), mpz(0), mpz(0), mpz(-1)], ZZ)
>>> %timeit f1.factor_list()
100 loops, best of 3: 1.90 ms per loop

>>> f0 = [mpz(1), mpz(0), mpz(0), mpz(0), mpz(0),
... mpz(0), mpz(0), mpz(0), mpz(0), mpz(0), mpz(-1)]
>>> %timeit dup_factor_list(f0, ZZ)
100 loops, best of 3: 1.88 ms per loop
```

Implemented algorithms

Only most remarkable cases here

- square–free decomposition
  - Yun
- factorization into irreducibles
  - finite fields
    - Berlekamp, Shoup, Zassenhaus
  - other domains
    - Zassenhaus, Wang
- functional decomposition
  - Landau–Zippel
- Gröbner bases
  - Buchberger
- root isolation
  - continued fractions, Collins–Krandick

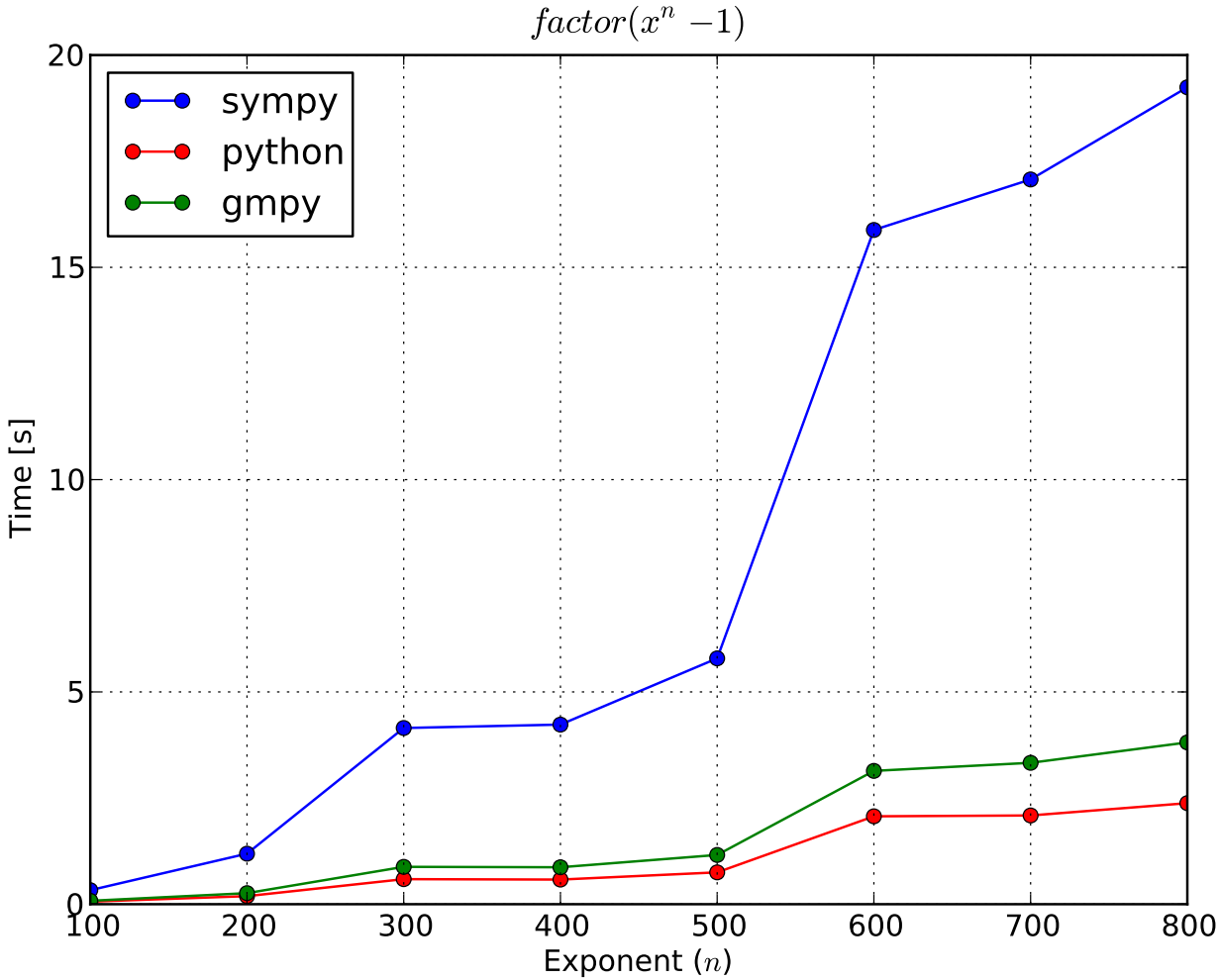
Ground types

An introduction

```
In [1]: from sympy.polys.algebratools import ZZ_python

In [2]: ZZ
Out[2]: ZZ
In [3]: ZZ.dtype
Out[3]: <built-in function mpz>

In [4]: zz = ZZ_python()

In [5]: zz
Out[5]: ZZ
In [6]: zz.dtype
Out[6]: <type 'int'>

In [7]: zz.rep = 'ZZpy'

In [8]: Poly(x**10 - 1, domain=zz)
Out[8]: Poly(x**10 - 1, x, domain='ZZpy')
```

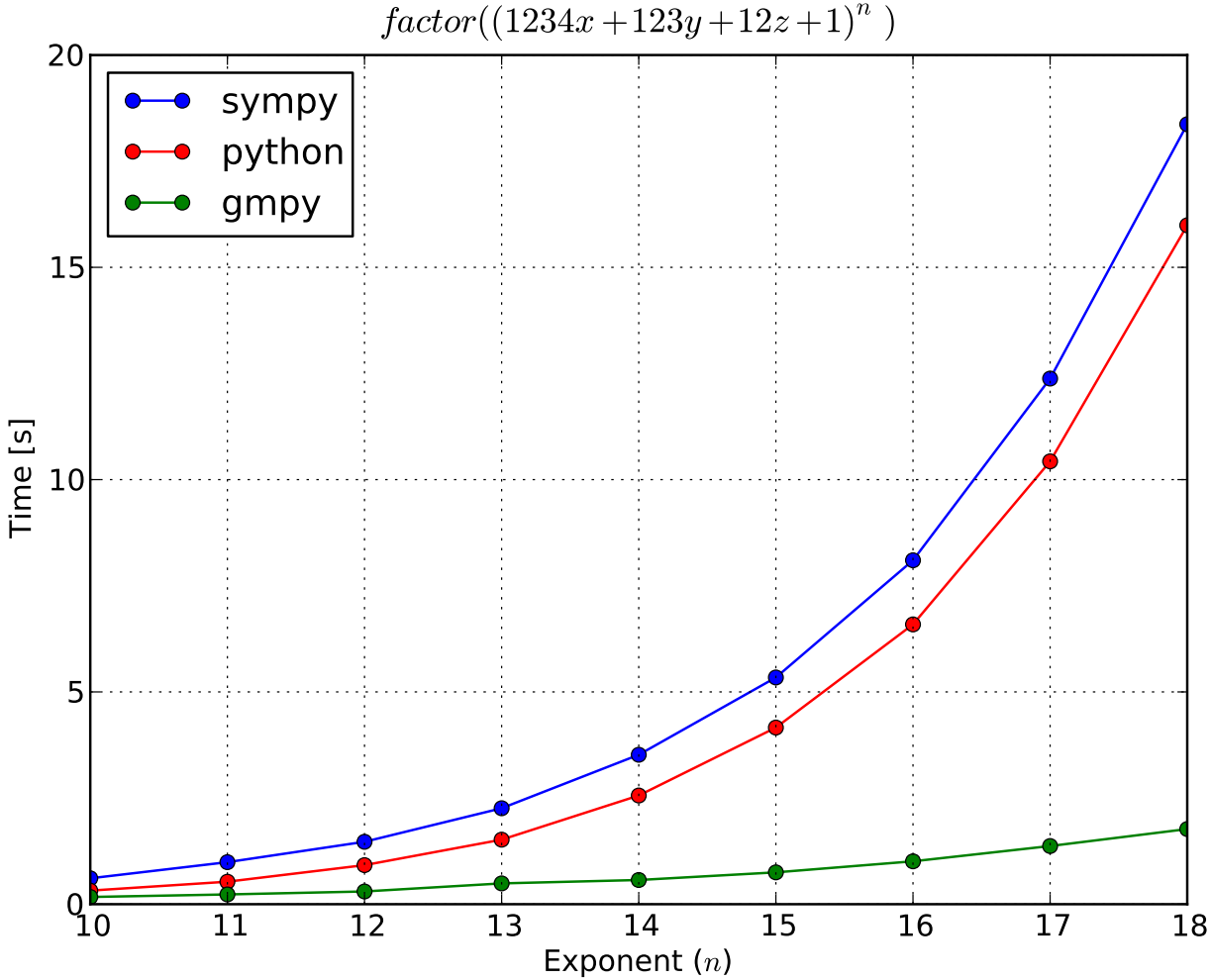
Ground types

Benchmark: small coefficients



Ground types

Benchmark: large coefficients



Polynomial representations

An introduction

- dense representation
  - list of list

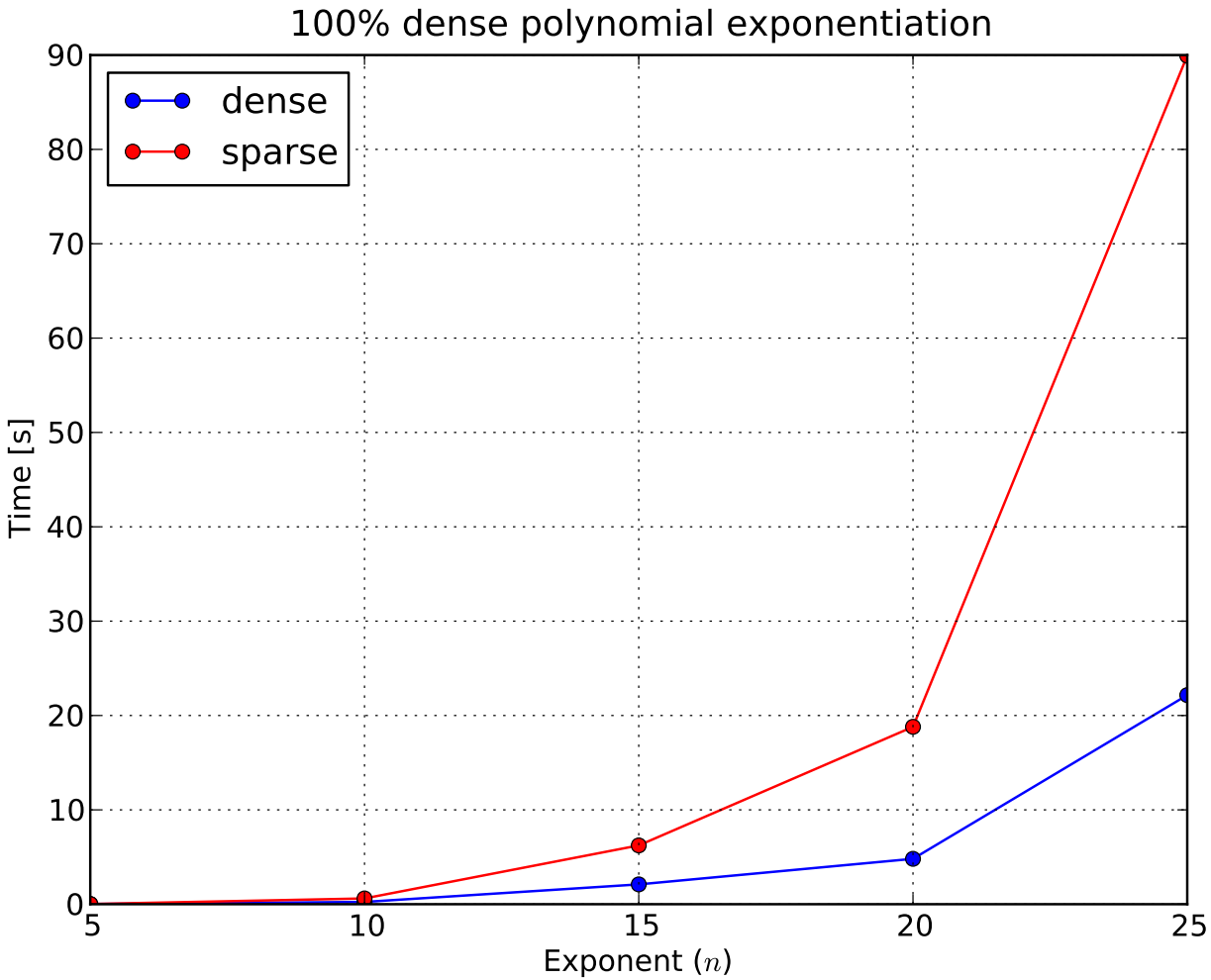
$$[[\ldots], [\ldots], \ldots, [\ldots]]$$

- sparse representation
  - dictionary

$$[(\mathrm{M_n}, \mathrm{c_n}), (\mathrm{M_{n-1}}, \mathrm{c_{n-1}}) \ldots, (\mathrm{M_0}, \mathrm{c_0})]$$

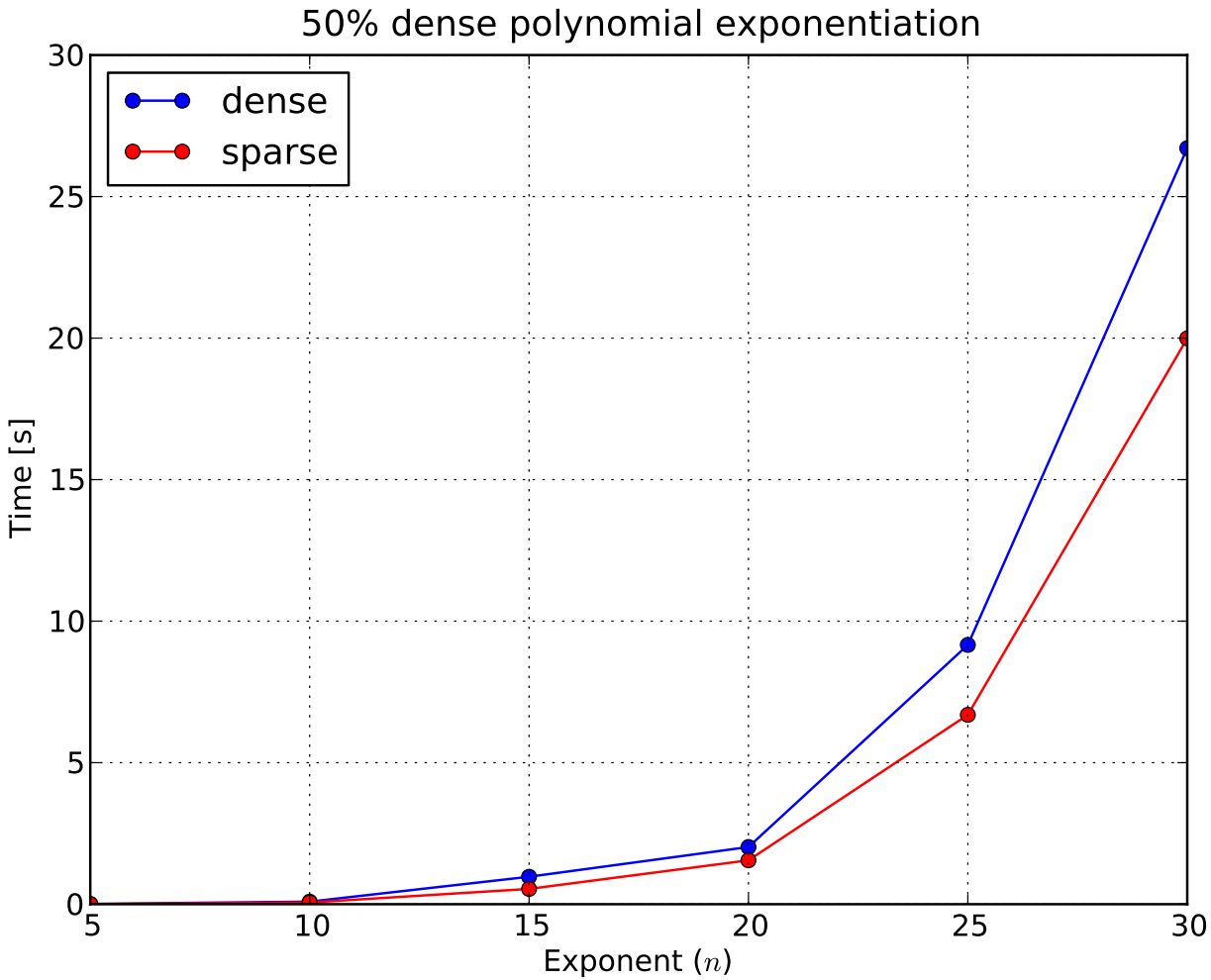
Polynomial representations

Benchmark: 100% dense exponentiation



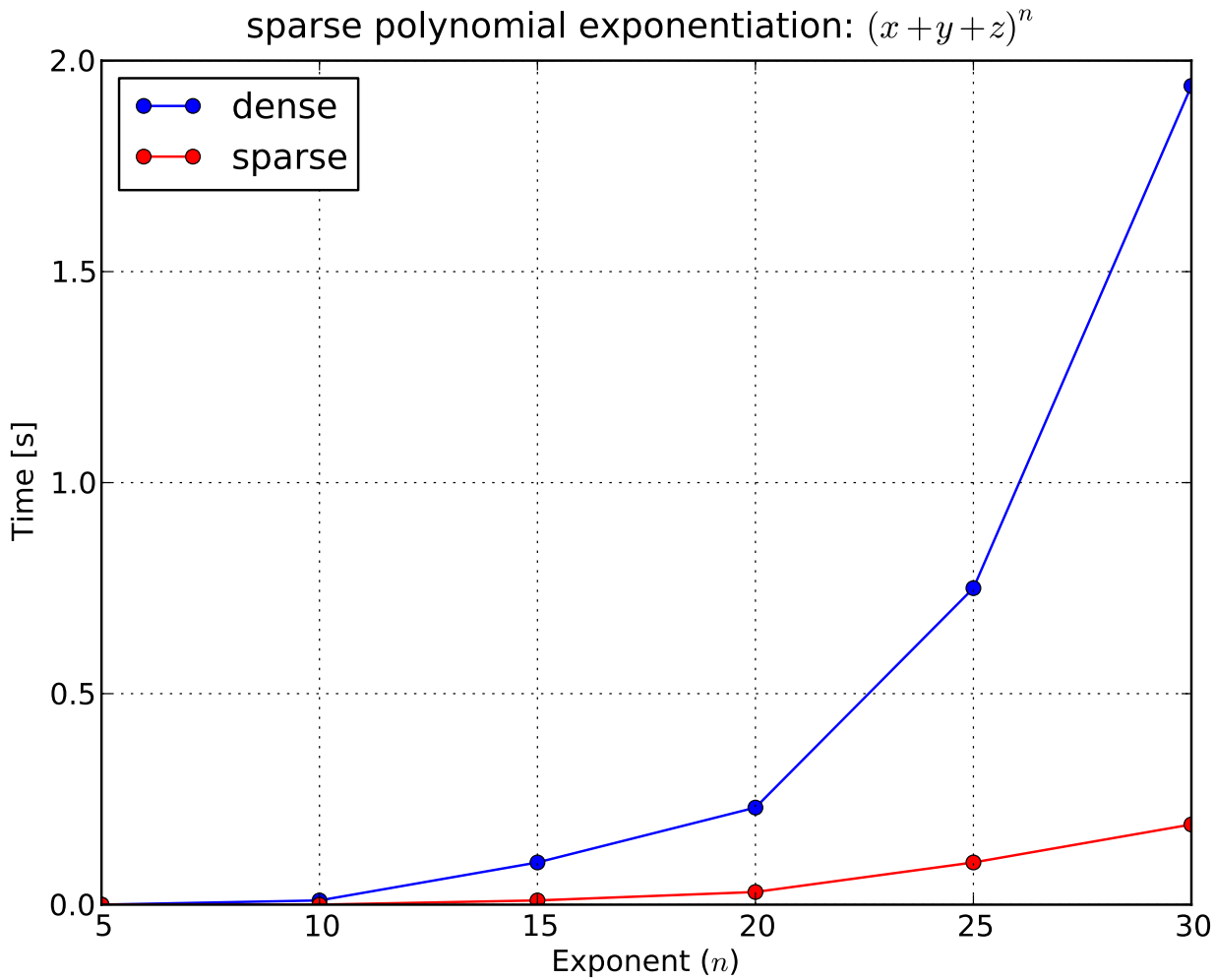
Polynomial representations

Benchmark: 50% dense exponentiation



Polynomial representations

Benchmark: sparse exponentiation



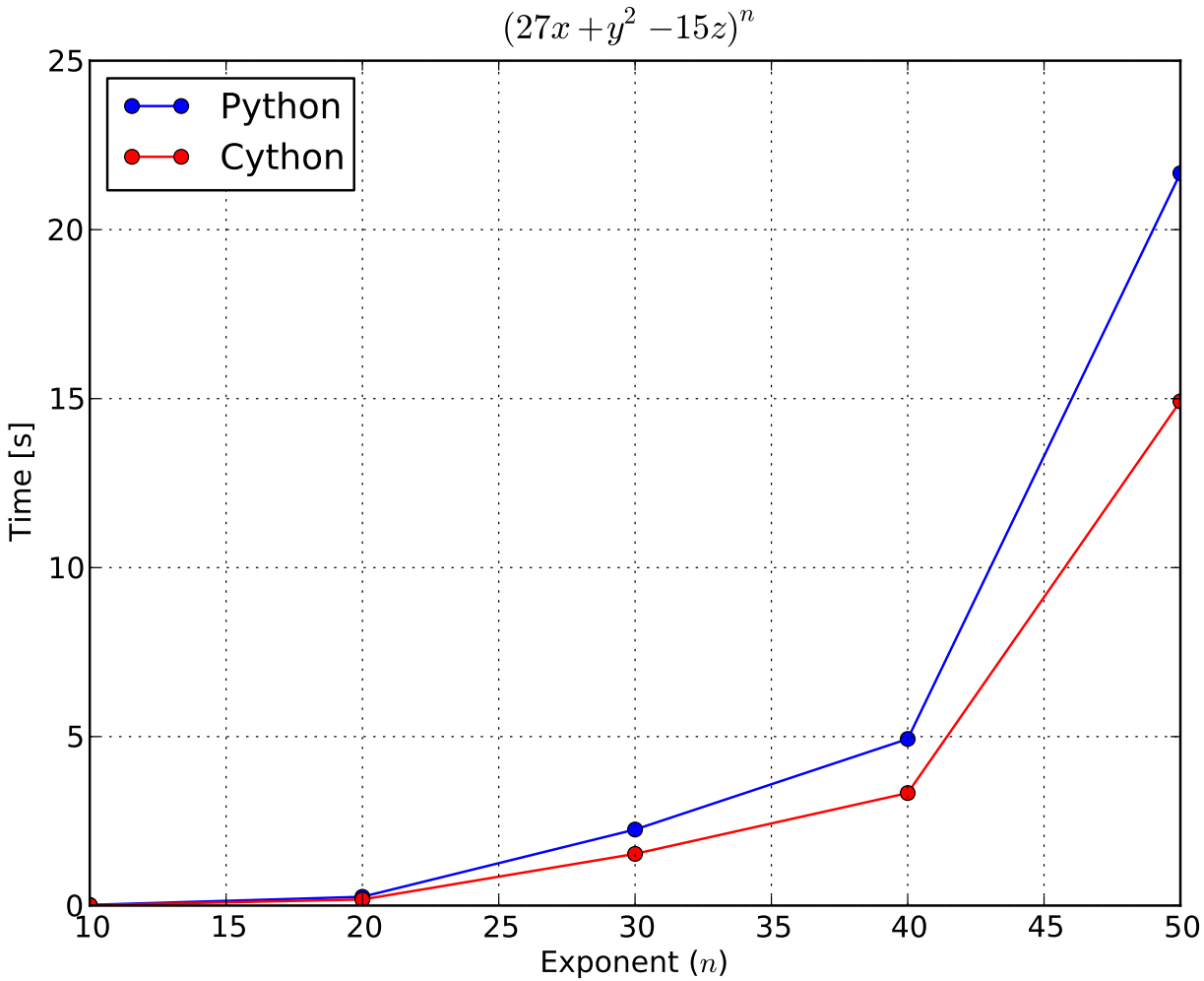
Using pure mode Cython

An introduction

- install Cython (`www.cython.org`)
- take advantage of `cythonized` decorator
- compile, i.e. wait 20 s (on 1.7 GHz CPU)

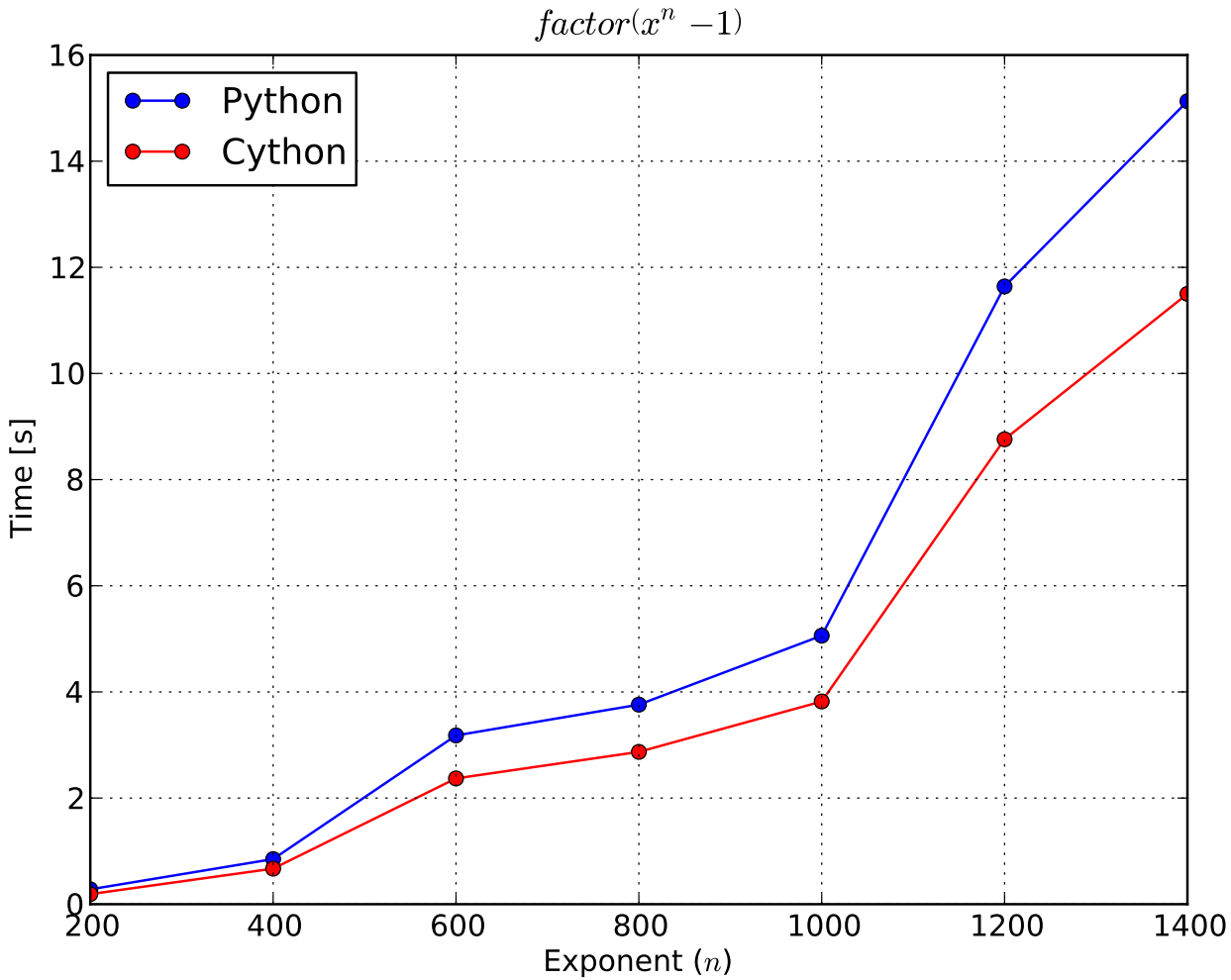
Using pure mode Cython

Benchmark: polynomial exponentiation



Using pure mode Cython

Benchmark: polynomial factorization



Future plans

- implement better algorithms
  - factorization, Gröbner bases, ...
- write more extensive documentation
- use the module for pratical things
  - GSoC project: Algorithms for Symbolic Integration

Thank you for your attention!

Questions, remarks, discussion ...