SymPy — czyli matematyka w Pythonie

Mateusz Paprocki <mattpap@gmail.com>

Wrocław University of Technology
University of Nevada, Reno

8 października 2010

Plan prezentacji

- Matematyka w Pythonie
- Wprowadzenie do SymPy
- Architektura systemu
- Przykłady zastosowań
- Sesja interaktywna
- Plany na przyszłość

Matematyka w Pythonie

- Python:
  - `__add__`, `__sub__`, `__mul__`, `__div__`, ...
- `math`:
  - biblioteka numeryczna
  - jedynie funkcje rzeczywiste
  - logarytmy i funkcja wykładnicza
  - funkcje trygonometryczne i hiperboliczne
- NumPy & SciPy
  - biblioteki numeryczne
  - działania na macierzach
  - funkcje rzeczywiste i zespolone
  - optymalizacja, interpolacja, ...
- Swiginac, Pynac, Sage, ..., SymPy

Matematyka w Pythonie

- Python:
  - `__add__`, `__sub__`, `__mul__`, `__div__`, ...
- `math`:
  - biblioteka numeryczna
  - jedynie funkcje rzeczywiste
  - logarytmy i funkcja wykładnicza
  - funkcje trygonometryczne i hiperboliczne
- NumPy & SciPy
  - biblioteki numeryczne
  - działania na macierzach
  - funkcje rzeczywiste i zespolone
  - optymalizacja, interpolacja, ...
- Swiginac, Pynac, Sage, ..., SymPy

Matematyka w Pythonie

- Python:
  - `__add__`, `__sub__`, `__mul__`, `__div__`, ...
- `math`:
  - biblioteka numeryczna
  - jedynie funkcje rzeczywiste
  - logarytmy i funkcja wykładnicza
  - funkcje trygonometryczne i hiperboliczne
- NumPy & SciPy
  - biblioteki numeryczne
  - działania na macierzach
  - funkcje rzeczywiste i zespolone
  - optymalizacja, interpolacja, ...
- Swiginac, Pynac, Sage, ..., SymPy

Matematyka w Pythonie

- Python:
  - `__add__`, `__sub__`, `__mul__`, `__div__`, ...
- `math`:
  - biblioteka numeryczna
  - jedynie funkcje rzeczywiste
  - logarytmy i funkcja wykładnicza
  - funkcje trygonometryczne i hiperboliczne
- NumPy & SciPy
  - biblioteki numeryczne
  - działania na macierzach
  - funkcje rzeczywiste i zespolone
  - optymalizacja, interpolacja, ...
- Swiginac, Pynac, Sage, ..., SymPy

Dlaczego SymPy?

Istnieje wiele systemów matematycznych:

- Systemy komercyjne:
  - Mathematica, Maple, Magma, ...
- Systemy Open Source:
  - AXIOM, GiNaC, Maxima, PARI, Sage, Singular, Yacas, ...

Problemy:

- wszystkie wymyślają swój własny język programowania
  - musimy się takiego języka nauczyć (często bywają uciążliwe)
  - podział na jądro systemu oraz bibliotekę matematyczną
  - wyjątki: GiNaC and Sage
- wszystkie wymagają kompilacji
  - niekorzystne w użyciu interaktywnym

Dlaczego SymPy?

Istnieje wiele systemów matematycznych:

- Systemy komercyjne:
  - Mathematica, Maple, Magma, ...
- Systemy Open Source:
  - AXIOM, GiNaC, Maxima, PARI, Sage, Singular, Yacas, ...

Problemy:

- wszystkie wymyślają swój własny język programowania
  - musimy się takiego języka nauczyć (często bywają uciążliwe)
  - podział na jądro systemu oraz bibliotekę matematyczną
  - wyjątki: GiNaC and Sage
- wszystkie wymagają kompilacji
  - niekorzystne w użyciu interaktywnym

Dlaczego SymPy?

Istnieje wiele systemów matematycznych:

- Systemy komercyjne:
  - Mathematica, Maple, Magma, ...
- Systemy Open Source:
  - AXIOM, GiNaC, Maxima, PARI, Sage, Singular, Yacas, ...

Problemy:

- wszystkie wymyślają swój własny język programowania
  - musimy się takiego języka nauczyć (często bywają uciążliwe)
  - podział na jądro systemu oraz bibliotekę matematyczną
  - wyjątki: GiNaC and Sage
- wszystkie wymagają kompilacji
  - niekorzystne w użyciu interaktywnym

Co to jest SymPy?

SymPy jest to biblioteka pisana w Pythonie do wykonywania:

- obliczeń symbolicznych
  - np. wyznaczanie pochodnych, całek, sum, granic, szeregów
- obliczeń algebraicznych
  - np. wyznaczanie izomorfizmów ciał algebraicznych
- obliczeń numerycznych
  - np. rozwiązywanie równań nieliniowych

Co to jest SymPy?

SymPy jest to biblioteka pisana w Pythonie do wykonywania:

- obliczeń symbolicznych
  - np. wyznaczanie pochodnych, całek, sum, granic, szeregów
- obliczeń algebraicznych
  - np. wyznaczanie izomorfizmów ciał algebraicznych
- obliczeń numerycznych
  - np. rozwiązywanie równań nieliniowych

Co to jest SymPy?

SymPy jest to biblioteka pisana w Pythonie do wykonywania:

- obliczeń symbolicznych
  - np. wyznaczanie pochodnych, całek, sum, granic, szeregów
- obliczeń algebraicznych
  - np. wyznaczanie izomorfizmów ciał algebraicznych
- obliczeń numerycznych
  - np. rozwiązywanie równań nieliniowych

Co to jest SymPy?

SymPy jest to biblioteka pisana w Pythonie do wykonywania:

- obliczeń symbolicznych
  - np. wyznaczanie pochodnych, całek, sum, granic, szeregów
- obliczeń algebraicznych
  - np. wyznaczanie izomorfizmów ciał algebraicznych
- obliczeń numerycznych
  - np. rozwiązywanie równań nieliniowych

Przykładowa sesja z SymPy

- import `sympy` i definicja symboli

```
>>> from sympy import *
>>> var('x,y')
```

- całkowanie oraz upraszczanie wyrażeń

```
>>> f = (x - tan(x)) / tan(x)**2 + tan(x)

>>> integrate(f, x)
log(1 + tan(x)**2)/2 - x/tan(x) - x**2/2

>>> ratsimp(_.diff(x)) == f
True
```

- rozkład wielomianów na czynniki

```
>>> factor(x**2 + y**2, extension=I)
(x - I*y)*(x + I*y)
```

- wyznaczanie wartości funkcji specjalnych

```
>>> gamma(pi).evalf(n=30)
2.28803779534003241795958890906
```

Przykładowa sesja z SymPy

- import sympy i definicja symboli

```
>>> from sympy import *
>>> var('x,y')
```

- całkowanie oraz upraszczanie wyrażeń

```
>>> f = (x - tan(x)) / tan(x)**2 + tan(x)

>>> integrate(f, x)
log(1 + tan(x)**2)/2 - x/tan(x) - x**2/2

>>> ratsimp(_.diff(x)) == f
True
```

- rozkład wielomianów na czynniki

```
>>> factor(x**2 + y**2, extension=I)
(x - I*y)*(x + I*y)
```

- wyznaczanie wartości funkcji specjalnych

```
>>> gamma(pi).evalf(n=30)
2.28803779534003241795958890906
```

Przykładowa sesja z SymPy

- import `sympy` i definicja symboli

```
>>> from sympy import *
>>> var('x,y')
```

- całkowanie oraz upraszczanie wyrażeń

```
>>> f = (x - tan(x)) / tan(x)**2 + tan(x)

>>> integrate(f, x)
log(1 + tan(x)**2)/2 - x/tan(x) - x**2/2

>>> ratsimp(_.diff(x)) == f
True
```

- rozkład wielomianów na czynniki

```
>>> factor(x**2 + y**2, extension=I)
(x - I*y)*(x + I*y)
```

- wyznaczanie wartości funkcji specjalnych

```
>>> gamma(pi).evalf(n=30)
2.28803779534003241795958890906
```

Przykładowa sesja z SymPy

- import `sympy` i definicja symboli

```
>>> from sympy import *
>>> var('x,y')
```

- całkowanie oraz upraszczanie wyrażeń

```
>>> f = (x - tan(x)) / tan(x)**2 + tan(x)

>>> integrate(f, x)
log(1 + tan(x)**2)/2 - x/tan(x) - x**2/2

>>> ratsimp(_.diff(x)) == f
True
```

- rozkład wielomianów na czynniki

```
>>> factor(x**2 + y**2, extension=I)
(x - I*y)*(x + I*y)
```

- wyznaczanie wartości funkcji specjalnych

```
>>> gamma(pi).evalf(n=30)
2.28803779534003241795958890906
```

Założenia projektu

- biblioteka pisana w Pythonie
  - bez nowego środowiska, języka, ...
  - działa od razu na dowolnej platformie
  - moduły nie–Pythonowe mogą być opcjonalne
- prostota architektury
  - relatywnie mała baza kodu źródłowego
  - łatwość w rozbudowie na dowolnym poziomie
- szeroka funkcjonalność
  - obsługa najważniejszych działów matematyki
  - wspieranie zaawansowanych metod i algorytmów
- optymalizacja wydajności w Cythonie
  - opcjonalnie, jako dodatek do wersji interpretowanej
- liberalna licencja: BSD
  - duża swoboda w użytkowaniu SymPy

Założenia projektu

- biblioteka pisana w Pythonie
  - bez nowego środowiska, języka, ...
  - działa od razu na dowolnej platformie
  - moduły nie–Pythonowe mogą być opcjonalne
- prostota architektury
  - relatywnie mała baza kodu źródłowego
  - łatwość w rozbudowie na dowolnym poziomie
- szeroka funkcjonalność
  - obsługa najważniejszych działów matematyki
  - wspieranie zaawansowanych metod i algorytmów
- optymalizacja wydajności w Cythonie
  - opcjonalnie, jako dodatek do wersji interpretowanej
- liberalna licencja: BSD
  - duża swoboda w użytkowaniu SymPy

Założenia projektu

- biblioteka pisana w Pythonie
  - bez nowego środowiska, języka, ...
  - działa od razu na dowolnej platformie
  - moduły nie–Pythonowe mogą być opcjonalne
- prostota architektury
  - relatywnie mała baza kodu źródłowego
  - łatwość w rozbudowie na dowolnym poziomie
- szeroka funkcjonalność
  - obsługa najważniejszych działów matematyki
  - wspieranie zaawansowanych metod i algorytmów
- optymalizacja wydajności w Cythonie
  - opcjonalnie, jako dodatek do wersji interpretowanej
- liberalna licencja: BSD
  - duża swoboda w użytkowaniu SymPy

Założenia projektu

- biblioteka pisana w Pythonie
  - bez nowego środowiska, języka, ...
  - działa od razu na dowolnej platformie
  - moduły nie–Pythonowe mogą być opcjonalne
- prostota architektury
  - relatywnie mała baza kodu źródłowego
  - łatwość w rozbudowie na dowolnym poziomie
- szeroka funkcjonalność
  - obsługa najważniejszych działów matematyki
  - wspieranie zaawansowanych metod i algorytmów
- optymalizacja wydajności w Cythonie
  - opcjonalnie, jako dodatek do wersji interpretowanej
- liberalna licencja: BSD
  - duża swoboda w użytkowaniu SymPy

Założenia projektu

- biblioteka pisana w Pythonie
  - bez nowego środowiska, języka, ...
  - działa od razu na dowolnej platformie
  - moduły nie–Pythonowe mogą być opcjonalne
- prostota architektury
  - relatywnie mała baza kodu źródłowego
  - łatwość w rozbudowie na dowolnym poziomie
- szeroka funkcjonalność
  - obsługa najważniejszych działów matematyki
  - wspieranie zaawansowanych metod i algorytmów
- optymalizacja wydajności w Cythonie
  - opcjonalnie, jako dodatek do wersji interpretowanej
- liberalna licencja: BSD
  - duża swoboda w użytkowaniu SymPy

SymPy w liczbach

- 2006–teraz
- 100 autorów
- 150 tysięcy linii kodu
  - 500 klas
  - 8000 funkcji
- 17 tysięcy testów
  - czas wykonania: 8 minut (Atom 1.6)
- 11 prezentacji na konferencjach i warsztatach
- 16 projektów w Google Summer of Code
  - oraz Google Highly Open Participation Contest
- jedna praca dyplomowa

SymPy w liczbach

- 2006–teraz
- 100 autorów
- 150 tysięcy linii kodu
  - 500 klas
  - 8000 funkcji
- 17 tysięcy testów
  - czas wykonania: 8 minut (Atom 1.6)
- 11 prezentacji na konferencjach i warsztatach
- 16 projektów w Google Summer of Code
  - oraz Google Highly Open Participation Contest
- jedna praca dyplomowa

SymPy w liczbach

- 2006–teraz
- 100 autorów
- 150 tysięcy linii kodu
  - 500 klas
  - 8000 funkcji
- 17 tysięcy testów
  - czas wykonania: 8 minut (Atom 1.6)
- 11 prezentacji na konferencjach i warsztatach
- 16 projektów w Google Summer of Code
  - oraz Google Highly Open Participation Contest
- jedna praca dyplomowa

Informacje kontaktowe

- Strona główna projektu:
  - `www.sympy.org`
- Strony dodatkowe:
  - `docs.sympy.org`
  - `wiki.sympy.org`
  - `live.sympy.org`
- Lista mailingowa:
  - `sympy@googlegroups.com`
- Kanał IRC:
  - `#sympy` na FreeNode
- Repozytorium `git`:

```
git clone git://github.com/sympy/sympy.git
```

Informacje kontaktowe

- Strona główna projektu:
  - `www.sympy.org`
- Strony dodatkowe:
  - `docs.sympy.org`
  - `wiki.sympy.org`
  - `live.sympy.org`
- Lista mailingowa:
  - `sympy@googlegroups.com`
- Kanał IRC:
  - `#sympy` na FreeNode
- Repozytorium `git`:

```
git clone git://github.com/sympy/sympy.git
```

Informacje kontaktowe

- Strona główna projektu:
  - `www.sympy.org`
- Strony dodatkowe:
  - `docs.sympy.org`
  - `wiki.sympy.org`
  - `live.sympy.org`
- Lista mailingowa:
  - `sympy@googlegroups.com`
- Kanał IRC:
  - `#sympy` na FreeNode
- Repozytorium `git`:

```
git clone git://github.com/sympy/sympy.git
```

Organizacja pracy

- korzystamy z systemu SCM `git`
- jedno główne repozytorium (GitHub)
  - tylko jedna gałąź — `master`
- każdy twórca ma własny "fork" na GitHubie
  - zazwyczaj wiele gałęzi
- żeby zacząć pracę z jedną z gałęzi:

```
git remote add github git://github.com/mattpap/sympy-polys.git
git fetch github
git branch --track polys11 github/polys11
```

- korzystamy z mechanizmu "pull request" do łączenia gałęzi
- testy zawsze muszą wykonywać się poprawnie
  - używamy buildbotów do testowania różnych konfiguracji

Organizacja pracy

- korzystamy z systemu SCM `git`
- jedno główne repozytorium (GitHub)
  - tylko jedna gałąź — `master`
- każdy twórca ma własny "fork" na GitHubie
  - zazwyczaj wiele gałęzi
- żeby zacząć pracę z jedną z gałęzi:

```
git remote add github git://github.com/mattpap/sympy-polys.git
git fetch github
git branch --track polys11 github/polys11
```

- korzystamy z mechanizmu "pull request" do łączenia gałęzi
- testy zawsze muszą wykonywać się poprawnie
  - używamy buildbotów do testowania różnych konfiguracji

Organizacja pracy

- korzystamy z systemu SCM `git`
- jedno główne repozytorium (GitHub)
  - tylko jedna gałąź — `master`
- każdy twórca ma własny "fork" na GitHubie
  - zazwyczaj wiele gałęzi
- żeby zacząć pracę z jedną z gałęzi:

```
git remote add github git://github.com/mattpap/sympy-polys.git
git fetch github
git branch --track polys11 github/polys11
```

- korzystamy z mechanizmu "pull request" do łączenia gałęzi
- testy zawsze muszą wykonywać się poprawnie
  - używamy buildbotów do testowania różnych konfiguracji

Organizacja pracy

- korzystamy z systemu SCM `git`
- jedno główne repozytorium (GitHub)
  - tylko jedna gałąź — `master`
- każdy twórca ma własny "fork" na GitHubie
  - zazwyczaj wiele gałęzi
- żeby zacząć pracę z jedną z gałęzi:

```
git remote add github git://github.com/mattpap/sympy-polys.git
git fetch github
git branch --track polys11 github/polys11
```

- korzystamy z mechanizmu "pull request" do łączenia gałęzi
- testy zawsze muszą wykonywać się poprawnie
  - używamy buildbotów do testowania różnych konfiguracji

Organizacja pracy

- korzystamy z systemu SCM `git`
- jedno główne repozytorium (GitHub)
  - tylko jedna gałąź — `master`
- każdy twórca ma własny "fork" na GitHubie
  - zazwyczaj wiele gałęzi
- żeby zacząć pracę z jedną z gałęzi:

```
git remote add github git://github.com/mattpap/sympy-polys.git
git fetch github
git branch --track polys11 github/polys11
```

- korzystamy z mechanizmu "pull request" do łączenia gałęzi
- testy zawsze muszą wykonywać się poprawnie
  - używamy buildbotów do testowania różnych konfiguracji

Moja rola w projekcie

- początek współpracy w marcu 2007 roku
  - kilka prostych poprawek i rozszerzeń
- następnie Google Summer of Code 2007
  - algorytmy rozwiązywania równań rekurencyjnych
  - algorytmy sumowania nieoznaczonego i oznaczonego
- no i tak już zostało:
  - algorytmy całkowania symbolicznego
  - struktury algebraiczne, wielomiany
  - upraszczanie wyrażeń algebraicznych, ...
- poza tym:
  - Google Summer of Code 2009, 2010 mentor (PSU, PSF)
  - EuroSciPy 2009, 2010; py4science (UC Berkeley)
  - praca dyplomowa

Moja rola w projekcie

- początek współpracy w marcu 2007 roku
  - kilka prostych poprawek i rozszerzeń
- następnie Google Summer of Code 2007
  - algorytmy rozwiązywania równań rekurencyjnych
  - algorytmy sumowania nieoznaczonego i oznaczonego
- no i tak już zostało:
  - algorytmy całkowania symbolicznego
  - struktury algebraiczne, wielomiany
  - upraszczanie wyrażeń algebraicznych, ...
- poza tym:
  - Google Summer of Code 2009, 2010 mentor (PSU, PSF)
  - EuroSciPy 2009, 2010; py4science (UC Berkeley)
  - praca dyplomowa

Moja rola w projekcie

- początek współpracy w marcu 2007 roku
  - kilka prostych poprawek i rozszerzeń
- następnie Google Summer of Code 2007
  - algorytmy rozwiązywania równań rekurencyjnych
  - algorytmy sumowania nieoznaczonego i oznaczonego
- no i tak już zostało:
  - algorytmy całkowania symbolicznego
  - struktury algebraiczne, wielomiany
  - upraszczanie wyrażeń algebraicznych, ...
- poza tym:
  - Google Summer of Code 2009, 2010 mentor (PSU, PSF)
  - EuroSciPy 2009, 2010; py4science (UC Berkeley)
  - praca dyplomowa

Moja rola w projekcie

- początek współpracy w marcu 2007 roku
  - kilka prostych poprawek i rozszerzeń
- następnie Google Summer of Code 2007
  - algorytmy rozwiązywania równań rekurencyjnych
  - algorytmy sumowania nieoznaczonego i oznaczonego
- no i tak już zostało:
  - algorytmy całkowania symbolicznego
  - struktury algebraiczne, wielomiany
  - upraszczanie wyrażeń algebraicznych, ...
- poza tym:
  - Google Summer of Code 2009, 2010 mentor (PSU, PSF)
  - EuroSciPy 2009, 2010; py4science (UC Berkeley)
  - praca dyplomowa

Moduły SymPy

- assumptions
- concrete
- core
- functions
- galgebra
- geometry
- integrals
- interactive
- logic
- matrices
- mpmath
- ntheory
- parsing
- physics
- plotting
- polys
- printing
- series
- simplify
- solvers
- statistics
- tensor
- thirdparty
- utilities

Moduły SymPy

- assumptions
- concrete
- core
- functions
- galgebra
- geometry
- integrals
- interactive
- logic
- matrices
- mpmath
- ntheory
- parsing
- physics
- plotting
- polys
- printing
- series
- simplify
- solvers
- statistics
- tensor
- thirdparty
- utilities

Moduły SymPy

- assumptions
- concrete
- core
- functions
- galgebra
- geometry
- integrals
- interactive
- logic
- matrices
- mpmath
- ntheory
- parsing
- physics
- plotting
- polys
- printing
- series
- simplify
- solvers
- statistics
- tensor
- thirdparty
- utilities

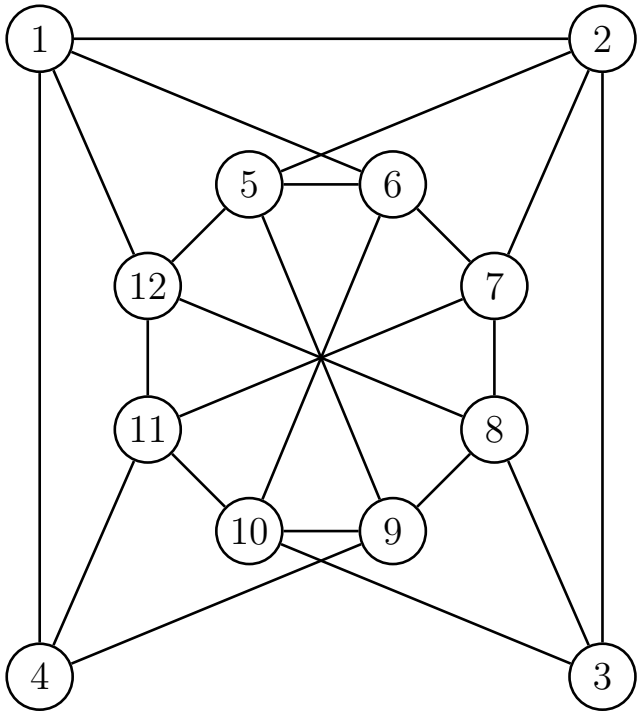
Zastosowania SymPy

- rozwiązywanie problemów matematycznych
  - np. nauczanie matematyki
  - Dlaczego?
    - łatwy do nauczenia język programowania
    - użytkownik ma dostęp do wszystkich algorytmów
  - Przykład: k–kolorowanie grafów
- osadzanie SymPy w innych programach
  - np. generacja kodu na podstawie wyrażeń matematycznych
  - Dlaczego?
    - mała biblioteka bez żadnych zależności
    - nie wymaga kompilacji, instalacji, konfiguracji
  - Przykład: generowanie kodu C

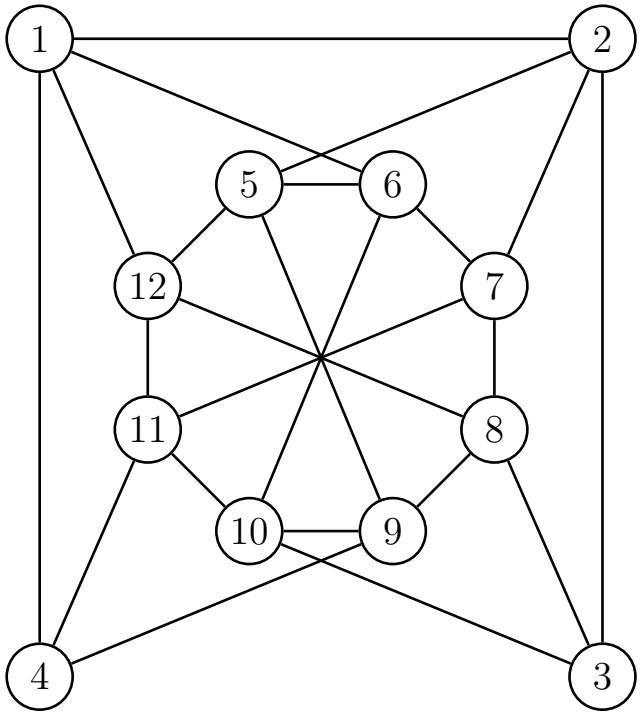
Zastosowania SymPy

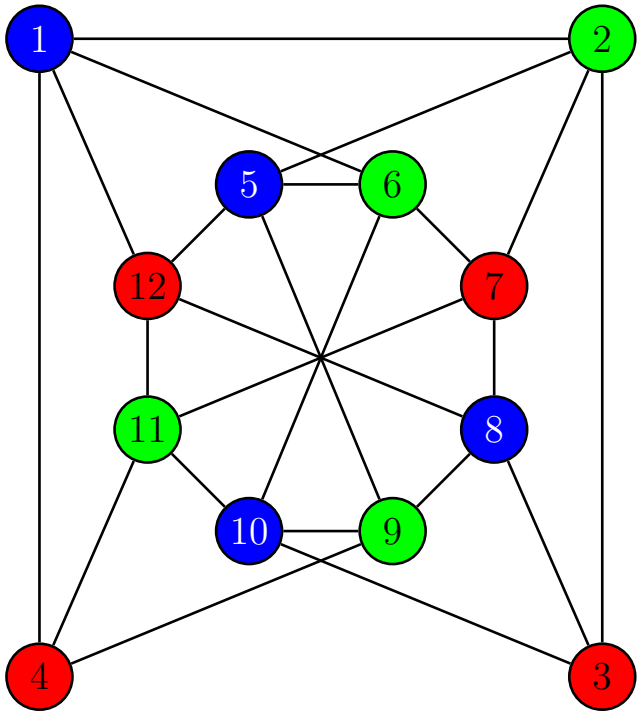
- rozwiązywanie problemów matematycznych
  - np. nauczanie matematyki
  - Dlaczego?
    - łatwy do nauczenia język programowania
    - użytkownik ma dostęp do wszystkich algorytmów
  - Przykład: k–kolorowanie grafów
- osadzanie SymPy w innych programach
  - np. generacja kodu na podstawie wyrażeń matematycznych
  - Dlaczego?
    - mała biblioteka bez żadnych zależności
    - nie wymaga kompilacji, instalacji, konfiguracji
  - Przykład: generowanie kodu C

k–kolorowanie grafów (1)



k–kolorowanie grafów (1)





k–kolorowanie grafów (2)

Dany jest $\mathcal{G}(V, E)$. Wprowadzamy dwa układy równań wielomianowych:

- I_k — dozwolone jest przypisanie jednego z k kolorów do wierzchołka x_i

$$I_k = \{x_i^k - 1 : i \in V\}$$

- $I_{\mathcal{G}}$ — przyległe wierzchołki muszą mieć przypisane różne kolory

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \ldots + x_i x_j^{k-2} + x_j^{k-1} : (i,j) \in E\}$$

Następnie rozwiązujemy $I_k \cup I_{\mathcal{G}}$ metodą baz Gröbnera.

k–kolorowanie grafów (2)

Dany jest $\mathcal{G}(V, E)$. Wprowadzamy dwa układy równań wielomianowych:

- I_k — dozwolone jest przypisanie jednego z k kolorów do wierzchołka x_i

$$I_k = \{x_i^k - 1 : i \in V\}$$

- $I_{\mathcal{G}}$ — przyległe wierzchołki muszą mieć przypisane różne kolory

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \ldots + x_i x_j^{k-2} + x_j^{k-1} : (i,j) \in E\}$$

Następnie rozwiązujemy $I_k \cup I_{\mathcal{G}}$ metodą baz Gröbnera.

k–kolorowanie grafów (2)

Dany jest $\mathcal{G}(V, E)$. Wprowadzamy dwa układy równań wielomianowych:

- I_k — dozwolone jest przypisanie jednego z k kolorów do wierzchołka x_i

$$I_k = \{x_i^k - 1 : i \in V\}$$

- $I_{\mathcal{G}}$ — przyległe wierzchołki muszą mieć przypisane różne kolory

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \ldots + x_i x_j^{k-2} + x_j^{k-1} : (i,j) \in E\}$$

Następnie rozwiązujemy $I_k \cup I_{\mathcal{G}}$ metodą baz Gröbnera.

k–kolorowanie grafów (2)

Dany jest $\mathcal{G}(V, E)$. Wprowadzamy dwa układy równań wielomianowych:

- I_k — dozwolone jest przypisanie jednego z k kolorów do wierzchołka x_i

$$I_k = \{x_i^k - 1 : i \in V\}$$

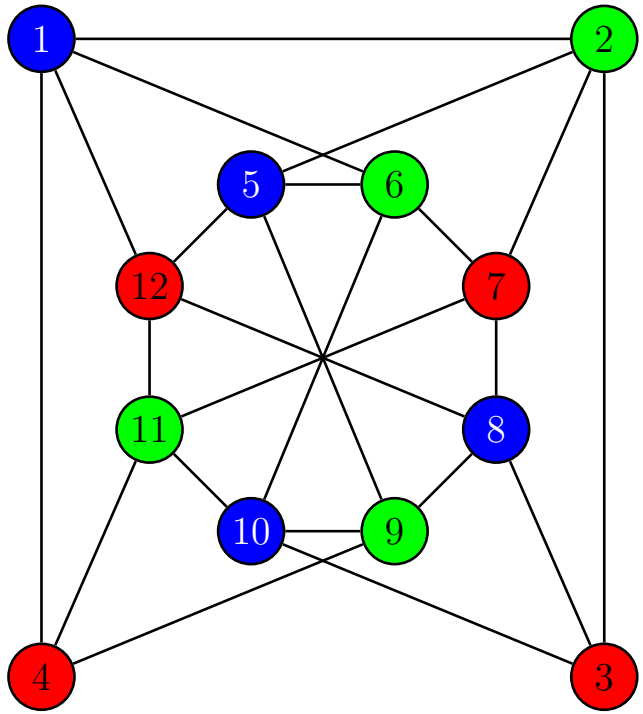
- $I_{\mathcal{G}}$ — przyległe wierzchołki muszą mieć przypisane różne kolory

$$I_{\mathcal{G}} = \{x_i^{k-1} + x_i^{k-2}x_j + \ldots + x_i x_j^{k-2} + x_j^{k-1} : (i,j) \in E\}$$

Następnie rozwiązujemy $I_k \cup I_{\mathcal{G}}$ metodą baz Gröbnera.

k–kolorowanie grafów (3)

$$\{x_1 + x_{11} + x_{12},$$
$$x_2 - x_{11},$$
$$x_3 - x_{12},$$
$$x_4 - x_{12},$$
$$x_5 + x_{11} + x_{12},$$
$$x_6 - x_{11},$$
$$x_7 - x_{12},$$
$$x_8 + x_{11} + x_{12},$$
$$x_9 - x_{11},$$
$$x_{10} + x_{11} + x_{12},$$
$${x_{11}}^2 + x_{11}x_{12} + x_{12}^2,$$
$${x_{12}}^3 - 1\}$$

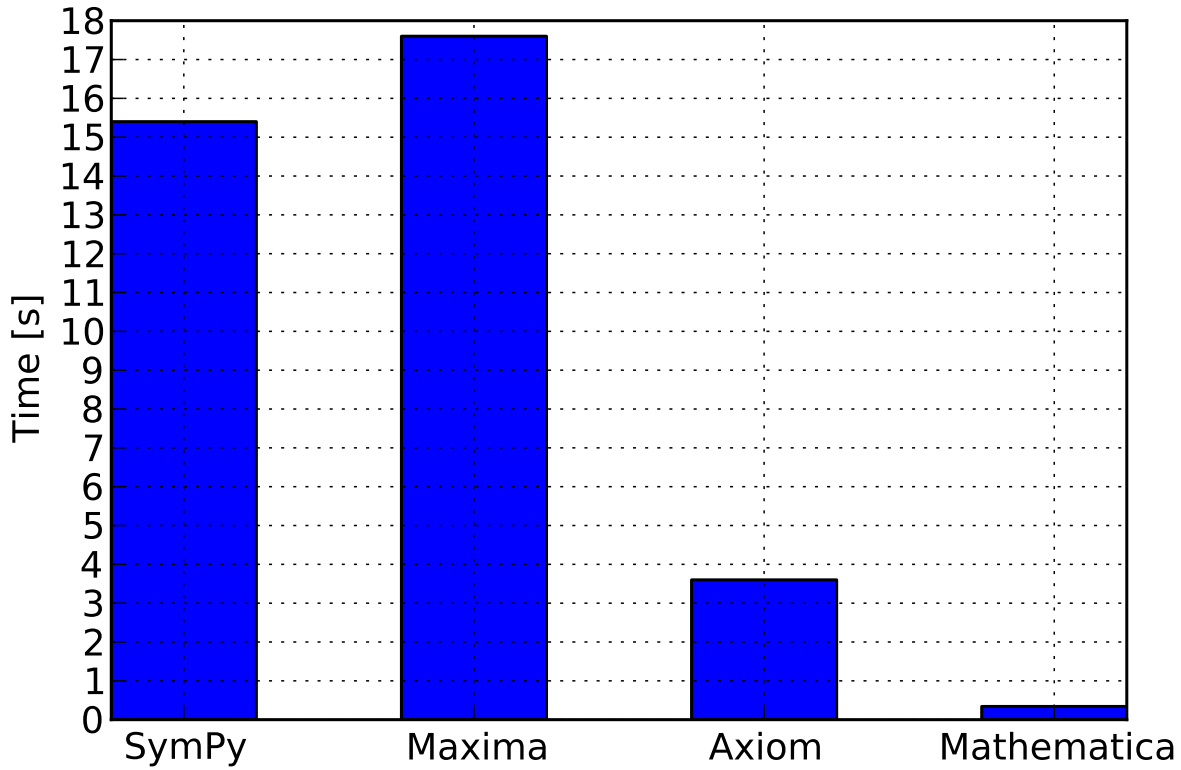


k–kolorowanie grafów (4)

Rozwiązanie problemu 3–kolorowania w SymPy:

```
In [1]: V = range(1, 12+1)
In [2]: E = [(1,2),(2,3),(1,4),(1,6),(1,12),(2,5),(2,7),
(3,8),(3,10),(4,11),(4,9),(5,6),(6,7),(7,8),(8,9),(9,10),
(10,11),(11,12),(5,12),(5,9),(6,10),(7,11),(8,12)]

In [3]: X = [ Symbol('x' + str(i)) for i in V ]
In [4]: E = [ (X[i-1], X[j-1]) for i, j in E ]

In [5]: I3 = [ x**3 - 1 for x in X ]
In [6]: Ig = [ x**2 + x*y + y**2 for x, y in E ]

In [7]: G = groebner(I3 + Ig, X, order='lex')

In [8]: G != [1]
Out[8]: True
```

k–kolorowanie grafów (5)



Rysunek: Średni czas obliczania 3–kolorowania dla grafu $\mathcal{G}(V, E)$.

Generowanie kodu C (1)

Sformułowanie problemu:

- transformacja wyrażeń do postaci Hornera
- generacja odpowiadającego im kodu w języku C
  - np. dla celów szybkiego wyznaczenia wartości dla wielu punktów
- przyjmijmy, że nie chcemy/nie możemy skorzystać z funkcji `pow()`

Jak możemy rozwiązać tak sformułowany problem?

- używamy funkcji `horner()` do transformacji wyrażeń
- definiujemy nową drukarkę do wygenerowania kodu

Generowanie kodu C (1)

Sformułowanie problemu:

- transformacja wyrażeń do postaci Hornera
- generacja odpowiadającego im kodu w języku C
  - np. dla celów szybkiego wyznaczenia wartości dla wielu punktów
- przyjmijmy, że nie chcemy/nie możemy skorzystać z funkcji `pow()`

Jak możemy rozwiązać tak sformułowany problem?

- używamy funkcji `horner()` do transformacji wyrażeń
- definiujemy nową drukarkę do wygenerowania kodu

Generowanie kodu C (2)

Definiujemy nową drukarkę dla sformułowanego problemu:

```
from sympy.printing import StrPrinter

class CPrinter(StrPrinter):
    """Print Lambda as C function and unroll Pow. """

    counter = 0

    def _print_Lambda(self, expr):
        self.counter += 1

        return """long _f%i(long %s) {\n  return (%s);\n}""" % \
            (self.counter, expr.args[0], self.doprint(expr.args[1]))

    def _print_Pow(self, expr):
        if expr.exp.is_Integer:
            return '*'.join([str(expr.base)]*int(expr.exp))
        else:
            return StrPrinter._print_Pow(self, expr)
```

Generowanie kodu C (3)

Chcemy wygenerować kod w C dla wielomianu:

$$x^6 + 2x^3 + 3x^2 + 4x + 5$$

Użyjemy do tego celu `CPrinter`:

```
In [1]: from sympy import horner, Lambda
In [2]: from sympy.abc import x

In [3]: f = horner(x**6 + 2*x**3 + 3*x**2 + 4*x + 5)

In [4]: print CPrinter().doprint(Lambda(x, f))
Out[4]:
double _f1(double _x) {
  return (5 + (4 + (3 + (2 + _x*_x*_x)*_x)*_x)*_x);
}
```

Generowanie kodu C (3)

Chcemy wygenerować kod w C dla wielomianu:

$$x^6 + 2x^3 + 3x^2 + 4x + 5$$

Użyjemy do tego celu `CPrinter`:

```
In [1]: from sympy import horner, Lambda
In [2]: from sympy.abc import x

In [3]: f = horner(x**6 + 2*x**3 + 3*x**2 + 4*x + 5)

In [4]: print CPrinter().doprint(Lambda(x, f))
Out[4]:
double _f1(double _x) {
  return (5 + (4 + (3 + (2 + _x*_x*_x)*_x)*_x)*_x);
}
```

Sesja interaktywna

- podstawy
- tworzenie symboli
- kolorowanie grafów

Plany na przyszłość

O co musimy zadbać:

- lepsze pokrycie kodu testami
- szczegółowe benchmarki
- funkcjonalność
- poprawność

Dziękuję za uwagę!

Pytania, uwagi, dyskusja ...