Sympy - Improving the `series` package and `limits`

Sartaj Singh

April 28, 2015

Abstract

Sympy currently lacks a proper structure for handling and manipulating series. All the series related functionality is defined as methods in `Expr` class. I plan to give the series package a concrete structure for future development and improvement. I plan to do the following over the summer.

- Sequence classes for defining sequences of coefficients.
- Classes to represent series in general.
- Implement 'Formal Power Series', using the above implemented structure.
- Computing limits of sequences.

Contents

1 About Me

The Student

Name Sartaj Singh

e-mail singhsartaj94@gmail.com

Github https://github.com/leosartaj

Blog https://leosartaj.blogspot.com

The Institute

University Indian Institute of Technology (BHU), Varanasi

Major Mathematics and Computing

Year Sophomore

Degree Integrated Masters Degree

Programming Experience and Mathematical Background

I have been programming from last two years, and from the past one year in python. Apart from python I have experience in C, javascript, PHP. For version control, I have been using git. I work on Arch Linux(rolling release rocks!) with *vim* as my primary editor. I like vim because its fast and productive. My favorite python feature is list comprehensions, simply because they are *beautiful*. I have completed courses on basic mathematical methods, series expansions.

Recurrence relations are a powerful tool in mathematics. Ability of sympy to solve them fascinates me.

```
>>> y = Function('y')
>>> rsolve(y(n) - 2*y(n-1), y(n))
>>> C0*2**n
```

My Motivation

I have always been fascinated with the power *Mathematica* [1] brings to the process of mathematical study. I soon discovered sympy and decided to give it a shot.

Series play an integral role in field of mathematics. Series have various applications, like solving differential equations. How a seemingly complex function can be simplified using series is what fascinates me the most.

```
>>> series(cos(4*acos(x))
1-8x**2+8x**4 # simple polynomial
```

My Projects

PyChat [2] - A simple asynchronous chat client, based on *Twisted*[3] framework.

Sub [4] - Simple command line tool for downloading subtitles.

autosign [5] - Easily add signatures to files

PyCross [6] - Single/multiplayer TicTacToe game. Single player is based on *minimax*[7] algorithm and *Monte Carlo*[8] Simulation.

All of my projects (including others) can be found on my github page.

Contributions

I started using sympy in January and made my first contributing in February. I have been constantly contributing and learning from the community ever since.

Merged PR's

- 8985, 9039 - Removed `\lvert` (not supported by `matplotlib`) in favor of `\left |` from latex(Abs)
- 9105 - Fixed `exp(x).taylor_term` method, it raised `TypeError` error when called (uncached).
- 9139 - `factorial(n) >= 1`, now returns `True` for a *nonnegative* integer `n`.
- 9142 - Implemented function `stieltjes` for computing stieltjes constants.
- 9156 - `Fibonacci(n).limit(n, oo)` and `Lucas(n).limit(n, oo)` returns `oo` rather than `Fibonacci(oo)` and `Lucas(oo)`. Result is now consistent with factorial and others.

Unmerged PR's

- 9036 - **[WIP]** Fixed `LambertW` to give series expansion at x=0
- 9038 - **[WIP]** zero=True assumption was ignored in Limit. First approach was not good. Working on a new approach to tackle the bug.
- 9050 - **[WIP]** Implemented Fourier sine/cosine series expansion.
- 8729 - Earlier `nan.is_rational()` returned `True`, fixed it to return `False`.
- 9075 - Earlier following type of limits computed incorrectly,

```
>>> ex = (6**(n+1)+n+1)/(6**n+n)
>>> ex.limit(n, oo)
1
```

Fixed it to return `6` (expected result)

```
>>> ex.limit(n, oo)
6
```

Issues/Bugs Caught

- 9104 - `exp(x).taylor_term(n, x)` returned `TypeError`. **Caught** and **Fixed** the bug.
- 9077 - Fourier sine/cosine implementation. Sympy should have an implementation of the same. Issued a pull request to address the same.

2 The Project

In this section I will detail my vision for how the API will look and how the code will be structured. I expect this structure to be considerably enhanced under the guidance of my mentor.

2.1 Sequences

For the purpose of defining coefficients, I plan to implement Sequence based Classes inspired from sequences [9] branch by Alexander U. Gudchenko. A `Sequence` will be finite or infinite *lazily* evaluated list. Coefficients will only be evaluated when required. Coefficients evaluated once will be saved in an internal `dictionary` (sparse representation).

2.1.1 Sequences based on formula

To define coefficients based on a particular formula, a special `SeqFormula` class will be implemented.

```
>>> SeqFormula(n*(n+1), (n, 0, oo))
[0, 2, 6, 12, 20, ...]
```

2.1.2 Periodical Sequences

Similar to sequences based on formulas, sequences can also be defined periodically, such as

```
>>> SeqPer((1, 3, 8), (n, 0, oo))
[1, 3, 8, 1, 3, ...]
```

2.1.3 Sequences from a function

Sequences can also be generated using functions.

```
>>> SeqFunc(lambda n: n*(n+1), (n, 0, oo))
[0, 2, 6, 12, 20, ...]
```

2.1.4 Containment

All the above sequences can be generated using `Sequence` class.

```
>>> Sequence(function=lambda n: n*(n+1), bounds=(n, 0, oo))
[0, 2, 6, 12, 20, ...]
>>> Sequence(formula=n*(n+1), bounds=(n, 0, oo))
[0, 2, 6, 12, 20, ...]
>>> Sequence(periodical=(1, 3, 8), bounds=(n, 0, oo))
[1, 3, 8, 1, 3, ...]
```

2.1.5 Methods

The `coeff` method can be used for getting a coefficient for a particular n.

```
>>> per = Sequence(periodical=(1, 3, 8), bounds=(n, 0, oo))
>>> per.coeff(7)
1
>>> fun = Sequence(function=lambda n: n*(n+1), bounds=(n, 0, oo))
>>> fun.coeff(7)
56
>>> form = Sequence(formula=n*(n+1), bounds=(n, 0, oo))
>>> form.coeff(n-1) # supports symbolic coefficients
n*(n-1)
```

2.1.6 Operations

Addition, Subtraction, Multiplication, Division will be defined element-wise.

```
>>> a = Sequence(formula=n**2, bounds=(n, 0, oo))
>>> b = Sequence(formula=n, bounds=(n, 0, oo))
>>> a + b
[0, 2, 6, 12, 20, ...]
>>> a - b
```

```
[0, 0, 2, 6, 12, ...]
>>> a * b
[0, 8, 27, 64, 125, ...]
```

Cauchy-Product [10] can also be implemented, where

$$cn = \sum_{k=0}^{n} a_k b_{n-k}$$

2.2 SeriesData class

This class will represent series. All other series will use `SeriesData` to represent a series. Internally `SeriesData` class will use `Sequence` objects to define a series. Series expansion of `exp(x)` can be modeled using *sequences*.

```
>>> an = Sequence(formula=1/factorial(n)), bounds=(n, 0, oo))
>>> xn = Sequence(formula=x**n), bounds=(n, 0, oo))
>>> a * b
[1, x, x**2/2, x**3/6, x**4/24, ...]
```

In general `SeriesData` just zips the two sequences together over *multiplication*. For constructing a series from known coefficients.

```
>>> an = Sequence(formula=1/factorial(n), bounds=(n, 0, oo)) # coefficients
>>> xn = Sequence(formula=x**n, bounds=(n, 0, oo)) # powers of x
>>> SeriesData(an, xn) # expansion of exp(x)
1 + x + x**2/2 + x**3/6 + x**4/24 + O(x**5)
```

This class can further have methods for generating *term* for a particular n, giving *truncated* or *infinite* series. Further, more classes can be formed which represent series in `x`, `sin(x)` and `cos(x)` (Fourier series), etc

2.3 SeriesX class

This class will represent series in x and can be used to model various series based on powers of x.

Taylor Series [11]

```
>>> an = Sequence(formula=1/factorial(n), bounds=(n, 0, oo))
>>> SeriesX(an, x, a)
1 + e*x + e**2*x**2/2 + e**3*x**3/6 + e**4*x**4/24 + O(x**5)
```

2.3.1 Methods and Operations

- Addition, subtraction will be term-wise

```
>>> an = Sequence(formula=n**2, bounds=(n, 0, oo))
>>> bn = Sequence(formula=n, bounds=(n, 0, oo))
>>> s1 = SeriesX(an, x)
>>> s2 = SeriesX(bn, x)
>>> s1 + s2
2*x + 6*x**2 + 12*x**3 + 20*x**4 + O(x**5)
>>> s1 - s2
3*x**2 + 6*x**3 + 12*x**4 + O(x**5)
```

- Differentiation and integration will also be term-wise

```
>>> s1.diff(x)
1 + 8*x + 27*x**2 + 64*x**3 + O(x**4)
>>> integrate(s1, x)
x**2/2 + 4*x**3/3 + 9*x**4/4 + 16*x**5/5 + O(x**6)
```

- Division, raised to power, inverse, composition
  All the methods and operations for truncated series will be modeled on the algorithms implemented in `ring_series`[12].

- Infinite representation
  Series can also be represented in infinite form (if available)

```
>>> s1.infinite
Sum(n**2*x**n, (n, 0, oo))
```

2.4 Formal Power Series

Formal Power Series will allow returning infinite series expansions in the form of

$$\sum_{n=0}^{\infty} a_n x^n$$

Currently Mathematica and others have this routine. Sympy should have it too. Last year some work was done on implementation of Formal Power Series[13][14]. Some part of the implementation is done and works correctly. I plan to integrate it with this system and complete the implementation.

```
>>> fps = FormalPowerSeries(exp(x), x, 0)
1 + x + x**2/2 + x**3/6 + x**4/24 + O(x**5)
>>> fps.infinite
Sum(x**n/factorial(n), (n, 0, oo))
```

2.4.1 Algorithm details

Algorithm used for computing *Formal Power Series* is described in detail in papers[15][16].

Find a Simple DE

For the algorithm to work, a *simple* DE of form

$$f^n(x) + \sum_{k=0}^{n-1} A_k f^k(x)$$

is to be obtained. DE can be obtained by following steps:

1. Set n := 1
2. Form the DE and solve for all A_n
3. if all A_n are rational stop, else increase n and repeat steps.
4. If no solution is found for a suitable n(papers suggest 4), stop

```
>>> simpleDE(exp(x), x)
-f(x) + Derivative(f(x), x)
>>> simpleDE(erf(x), x)
Derivative(f(x), x, x) + 2*x*Derivative(f(x), x)
```

Forming Corresponding RE

1. This can be achieved by the substitution

$$x^j * f^k(x) \to pochhammer(n+1-j,k) * a(n+k-j)$$

2. In the case, where all the coefficients of DE are constants, substitute

$$a(n) \to b(n) * n!$$

This substitution is equivalent to

$$f^k(x) \to b(n+k)$$

Example:

```
>>> DE = simpleDE(exp(x), x)
>>> a = Function("a")
>>> DEtoRE(DE, a(n))
(n + 1) * a(n + 1) - a(n)
```

Solving the RE

The RE obtained, can be solved using `rsolve` routine, already implemented in sympy using initial conditions.

To solve for any general point x_0 other than 0, following routine can be used

$x \to y + x_0$

`FormalPowerSeries(f(y+x0), y, 0)`

$y \to x - x_0$

2.5 Limits of Sequences

Implementing this algorithm will allow computing limits of series/summations, which are *not* currently computed by sympy. This will improve the range of admissible functions of which limit can be computed. Algorithm for computing limits of sequences is explained in the paper[17].

2.5.1 Applicability Criteria

This algorithm can be applied if the following conditions are fulfilled

- Applies on expressions of the form p_n/q_n where $p_n, q_n \to \infty$ when $n \to \infty$
- q_n should be asymptotically *strictly* monotonous
- terms are built from rational functions, indefinite sums and indefinite products over an indeterminate n, called $\Pi\Sigma - expressions$

2.5.2 Algorithm Details

Difference operator, Δ is defined as

$$\Delta a_n = a_{n+1} - a_n$$

Also, if

$$\lim_{n \to oo} p_n/q_n = 0$$

then $q_n \succ p_n$ or q_n is the dominant term

- Check for the applicability criteria as described in the above section.
- Use the difference operator on the numerator and denominator.
- Find the *dominant* term of the numerator and the denominator. Drop all other terms.
- Keep on doing recursively, until limit converges to a value.

2.5.3 Implementation

- **Difference Operator** For this a new function can be implemented for finding the delta of an expression

  ```
  >>> difference(n*2**n)
  (n+2)*2**n
  >>> difference(Sum(n*2**n, (n, 0, m)))
  (m+1)*2**(m+1)
  ```

- **Dominant Term** A simple function that computes the limit at infinity will do the trick

  ```
  >>> dominant(n, n**2)
  n**2 # dominant term
  ```

  This process is similar to finding the maximum number from a list of numbers, and can be done in a similar fashion.

- **Integrating into Limit** This algorithm can then be used inside `Limit` for finding the applicable limits.

3 Road-map

I will not be available for two or three separate days in June (travelling). Apart from this I have no prior commitments. I have been following Sympy's mailing list discussions for a while now and have looked at some of the codebase. I will be able to devote 40-45 hrs a week.

3.1 Community Bonding Period

- Discuss the project with my mentor in further detail.
- Get more acquainted with Sympy's codebase
- Read the below listed references in further detail.

3.2 Coding Period

- Implement Sequence classes (First week of June)
- Operations on sequences (Second week of June)
- SeriesData and SeriesX classes (Third week of June)
- Operations on SeriesX (First week of July)
- Implement Formal Power Series (Third week of July)
- Limits of Sequences (Second week of August)
- One buffer week for any unexpected delays (or more enhancements)

3.3 Post GSoC

Sympy provides me with a good platform to hone my programming skills and put my mathematical skills to good use. Series are one of the many great ideas to have evolved in mathematics. They are among my favorite mathematical topics. There are other areas in the field of mathematics where SymPy could do a much better job, and I would love to contribute and enhance them.

References

[1] Wolfram Research, `Mathematica`
http://www.wolfram.com/mathematica

[2] Sartaj Singh, `PyChat`
https://github.com/leosartaj/PyChat

[3] `Twisted`
https://twistedmatrix.com

[4] Sartaj Singh, `sub`
https://github.com/leosartaj/sub

[5] Sartaj Singh, `autosign`
https://github.com/leosartaj/autosign

[6] Sartaj Singh, `PyCross`
https://github.com/leosartaj/PyCross

[7] *Minimax*
http://en.wikipedia.org/wiki/Minimax

[8] *Monte-Carlo*
http://en.wikipedia.org/wiki/Monte_Carlo_method

[9] Alexander U. Gudchenko, `sequences`
https://github.com/goodok/sympy/tree/sequences

[10] *convolution*
http://en.wikipedia.org/wiki/Cauchy_product

[11] *Taylor/Maclaurin Series*
http://en.wikipedia.org/wiki/Taylor_series

[12] Mario Pernici, `ring_series`
https://github.com/sympy/sympy/blob/master/sympy/polys/ring_series.py

[13] Avichal Dayal, `#7846`
https://github.com/sympy/sympy/pull/7846

[14] Avichal Dayal, `#7618`
https://github.com/sympy/sympy/pull/7618

[15] Dominik Gruntz and Wolfram Koepf, *Formal Power Series*

[16] Wolfram Koepf, *Power Series in Computer Algebra*

[17] Manuel Kauers, *Computing Limits of Sequences*