pisot Documentation

Release

Robert Dougherty-Bliss

Sep 25, 2017

Table of Contents

1 Maple and Python Comparison **1**
- 1.1 Detailed Comparison 2

2 Modules **5**
- 2.1 cfinite module 5
- 2.2 pisot module 6

3 Examples **9**

4 Classes **11**

5 Indices and tables **13**

Python Module Index **15**

CHAPTER 1

Maple and Python Comparison

Doron Zeilberger's original Pisot.txt was implemented in the computer algebra system Maple. Maple has a very robust symbolic computation system, but its programming language is clunky. I have chosen to implement Pisot.txt in the programming language Python to enhance its readability and show that Maple (and its ilk) are not the only languages that can perform symbolic computation.

Python was designed to be a readable and general purpose language. Applications specific to mathematicians are not very general purpose, so symbolic computation is not a native feature of Python. Fortunately the library *SymPy* implements symbolic computation in Python and along with it most features found in Maple. By performing all of the "behind the scenes" calculation with SymPy, we obtain both the readability of Python and the symbolic power of Maple. As a short example of the differences between Maple and Python, we will compare two snippets from Zeilberger's Maple procedure `Pis` and the corresponding Python version. The procedure `Pis` computes the absolute value of the second largest root of the characteristic equation for a C-finite recurrence. As a part of this procedure, Zeilberger discards `1` if it is a root. If `lu` contains the roots, then the following Maple snippet accomplishes this task:

```
if member(1,lu) then
lu:=convert({op(lu)} minus {1},list):
if lu=[] then
 RETURN(FAIL):
fi:
fi:
```

Perhaps this is an obvious design pattern to an experienced Maple programmer, but to my eyes, there is a cognitive barrier between what we want to do (remove the root 1) and what is being done (convert a list to a set, perform set subtraction, and then converting the result back to a list).

If `roots` is a dictionary of roots, then in Python the same task can be accomplished as:

```python
if 1 in roots:
    del roots[1]

if not roots:
    return None
```

The Python version communicates the intent of the code more clearly and without as many technical details.

Detailed Comparison

As an example of the differences between the two, let us compare the full implementations of Zeilberger's Maple procedure `Pis`. His procedure is as follows:

```
#Pis(C): Inputs a C-finite sequence and outputs the absolute value of the second-
↪largest root
#It is a Pisot number if it is less than 1.
#Fis([[1,1],[1,1]]);
#Pis([[10,219,4796,105030],[22,-3,18,-11]]);
Pis:=proc(C) local x,lu,i,aluf,mu:
lu:=[solve(x^nops(C[2])-add(C[2][i]*x^(nops(C[2])-i),i=1..nops(C[2])))]:

if nops(lu)<>nops(C[1]) then
 RETURN(FAIL):
fi:

if member(1,lu) then
lu:=convert({op(lu)} minus {1},list):
if lu=[] then
 RETURN(FAIL):
fi:
fi:

aluf:=1:

for i from 2 to nops(lu) do
 if abs(evalf(lu[i]))>abs(evalf(lu[aluf])) then
  aluf:=i:
 fi:
od:

mu:=evalf([op(1..aluf-1,lu),op(aluf+1..nops(lu),lu)]):

max(seq(abs(mu[i]),i=1..nops(mu))):


end:
```

Now, taking our implementation of *`cfinite.CFinite`* for granted, the Python implementation is as follows:

```
def pisot_root(c_seq):
    """
    Compute the absolute value of the second-largest root of the characteristic
    equation for the C-finite sequence.

    :c_seq: :class:`.CFinite` instance.

    :returns: Floating point evaluation of the absolute value of the root, or
              None.

    """
    roots = c_seq.characteristic_roots()
    n_roots = len(roots.keys())

    if n_roots != c_seq.degree:
        return None
    if 1 in roots:
        del roots[1]

    if not roots:
        return None

    root_norms = [abs(root) for root in roots.keys()]
    root_norms = [sympy.re(sympy.N(norm)) for norm in root_norms]

    max_index = root_norms.index(max(root_norms))
    del root_norms[max_index]

    return max(root_norms)
```

The procedures are about the same length, accounting for blank lines and comments.

The first feature is the documentation. Both versions document their inputs and outputs, but the Python version is written in a standard way that allows for automatic documentation generation. In fact, the documentation at *`pisot.pisot_root()`* is automatically generated, as is every other piece of documentation on this site.

The next part computes the roots of the characteristic polynomial. Maple:

```
lu:=[solve(x^nops(C[2])-add(C[2][i]*x^(nops(C[2])-i),i=1..nops(C[2])))]:
```

Python (taking advantage of *`cfinite.CFinite.characteristic_roots()`*):

```
roots = c_seq.characteristic_roots()
```

Using Python's classes, it is clear that the characteristic roots are a property of the C-finite sequence, and that these roots are what we are computing.

Next, we look to see if there are any repeated roots. This is true if the number of distinct roots is less than the degree of the sequence. Maple:

```
if nops(lu)<>nops(C[1]) then
 RETURN(FAIL):
fi:
```

Python:

```
n_roots = len(roots.keys())

if n_roots != c_seq.degree:
    return None
```

The Maple version counts the number of coefficients in the C-finite sequence and calls this the degree. The structure of the Maple C-finite sequences is `[[coeffs], [initials]]`, so it counts the number of elements in the first element of a nested list. The Python version, relying on the class *`cfinite.CFinite`*, simply asks for the sequence's degree. Again, Python's classes allow us to embed information into an object, rather than relying on its actual representation.

CHAPTER 2

Modules

cfinite module

This module is a translation of (parts of) Doron Zeilberger's CFinite.txt from Maple to Python. It contains classes and functions to do two things:

1. Work with linear recurrence relations with constant coefficients, called C-finite sequences or recurrences.
2. Guess possible C-finite recurrences a given list of terms might satisfy.

Code to do the second item already exists in sympy, but I thought it would be fun to write it again.

class `cfinite.`**`CFinite`**`(`*initial*, *coeffs*`)`

Bases: `sympy.series.sequences.SeqBase`

This class provides procedures for working with linear recurrence relations with constant coefficients, called C-finite sequences.

We inhereit from sympy's `SeqBase` class, though our use is not currently (2017-09-20) optimized for recursion. (This goes for pisot.py as well.) SeqBase uses sympy's @cacheit decorator, which we should try to take advantage of for recurrences. (For example, after computing CFinite.coeff(100), CFinite.coeff(99) is not cached.)

`characteristic_poly``(`*var=x*`)`

Create the characteristic polynomial of the recurrence relation in *var*.

Var Symbol to use as the polynomial's variable.

Returns Sympy expression.

`characteristic_roots``()`

Compute the roots of the characteristic equation.

Returns List of roots returned by `sympy`.

`default_assumptions` **= {'commutative': True}**

`gen``()`

Yield a generator of terms.

To compute terms, we will need to keep track of the previous *degree* terms. We need to access all of them, but also constantly delete the first term and add a new one. For simplicity, we currently use a list for this, despite the O(n) deletion. It is possible to use cyclic lists for O(1) runtime in both, should we wish later.

`get_terms`(*k*)
Compute the first k terms as a list.

`interval`
Interval on which sequence is defined ((0, ∞)).

`is_commutative` = **True**

`start`
Start of sequence (0).

`stop`
End of sequence (∞).

`cfinite.`**`guess_cfinite_degree`**(*terms*, *degree*)
Try to guess a C-finite recurrence of the given degree that the terms might satisfy.

If the terms do satisfy a linear recurrence relation, then they satisfy a certain linear system, where the coefficients of the recurrence are the unknowns, and the terms form the coefficient matrix. To try and guess what the coefficients should be, we form the system and try to solve it. If it works out, then we have a guess. If it doesn't work out, then no possible C-finite recurrence can occur of the given degree.

Terms Terms of the sequence.

Degree Degree of the recurrence to check.

Returns A CFinite instance, or None.

`cfinite.`**`guess_cfinite_recurrence`**(*terms*, *max_degree=None*)
Guess a C-finite recurrence that the terms might satisfy, up to a maximum degree.

Sympy has something that does this more efficiently, but I wanted to write my own that gives prettier error messages (see guess_cfinite_degree()).

Terms Terms of the sequence.

Max_degree Maximum degree to check.

Returns A CFinite instance if a guess is found, otherwise None.

pisot module

This module is a translation of (part of) Doron Zeilberger's Pisot.txt from Maple to Python.

Pisot.txt is a Maple program that implements procedures to assist in the study of Pisot sequences. In particular, it can guess (and prove!) whether or not a given Pisot sequence satisfies a linear recurrence relation with constant coefficients (up to all degrees checked). To do this, it relies on the Maple program Cfinite.txt, also written by Zeilberger.

Due to its mathematical complexity, Pisot.txt can be difficult to read. This is exacerbated by the fact that Maple, though useful in its domain, is not an elegant programming language. I hope that this Python implementation will (eventually) provide a more accessible demonstration of the applications of symbollic computing.

class `pisot.`**`Pisot`**(*x*, *y*, *r*)
Bases: `sympy.series.sequences.SeqBase`

This class defines basic methods for dealing with Pisot sequences.

The Pisot sequence $E_r(x, y)$ is defined by

$$a_0 = x \tag{2.1}$$

$$a_1 = y \tag{2.2}$$

$$a_n = \left\lfloor \frac{a_{n-1}^2}{a_{n-2}} + r \right\rfloor \tag{2.3}$$

where $0 < x < y$ are integers, and r is some constant.

`default_assumptions` = {‘commutative’: True}

`find_cfinite_recurrence`(*n_terms*)

Try to guess a C-finite recurrence of the given degree that the first n_terms terms might satisfy, using sympy’s find_linear_recurrence() function.

N_terms Number of terms to check.

Returns A *`CFinite`* instance or None.

`gen`()

Yield a generator of terms.

`get_terms`(*k*)

Compute the first k terms as a list.

`interval`

Interval on which sequence is defined ((0, ∞)).

`is_commutative` = True

`start`

Start of sequence (0).

`stop`

End of sequence (∞).

`pisot.`**`pisot_root`**(*c_seq*)

Compute the absolute value of the second-largest root of the characteristic equation for the C-finite sequence, excluding any possible “1”s. It is assumed that the single root case is handled before calling this.

Zeilberger does additional things in this method. If there are repeated roots, we return None.

Parameters **`c_seq`** – *`CFinite`* instance.

Returns Floating point evaluation of the absolute value of the root, or None.

`pisot.`**`pisot_to_cfinite`**(*pisot*, *guess_length*, *check_length*, *verbose=False*)

Check if the given Pisot sequence satisfies a linear recurrence relation with finite coefficients.

We “correct” (I have not yet tried the single root case) the behavior of Pisot.txt as follows:

Let p be the single coefficient. Then, the conjectured form is $a_n = pa_{n-1}$, or $a_n = p^n x$. Rewritten, the conjecture is that $a_n = b_n$, where $b_n = p^n x$ for some real p. This is true iff

$$p^n x = floor(p^n x + r),$$

which holds iff 0 <= r < 1.

That is, if it looks like the sequence is a trivial geometric sequence, then it is as long as 0 <= r < 1.

More formally: If $y/x = p$, $p^n x$ is an integer for all nonnegative integers n, and 0 <= r < 1, then $E_r(x, y)$ is given by $a_n = p^n x$.

As an important special case, if x divides y and 0 <= r < 1, then this is true. We only handle this case.

Pisot *Pisot* sequence.

Guess_length Number of terms to use when guessing the recurrence. This should be somewhat small. If the sequence fails to satisfy a linear recurrence at a large number, then this method will spend a long time trying to look for one.

Check_length Number of terms of the sequence to check the conjectured linear recurrence for. This should be a large number.

Returns A *CFinite* instance, or None.

Pisot sequences are a family of recursive sequences defined by

$$a_0 = x \tag{2.4}$$

$$a_1 = y \tag{2.5}$$

$$a_n = \left\lfloor \frac{a_{n-1}^2}{a_{n-2}} + r \right\rfloor \tag{2.6}$$

where $0 < x < y$ are integers and r is some constant. These modules define procedures to work with Pisot sequences and determine whether or not they satisfy linear recurrence relations with constant coefficients. This work is a translation of Doron Zeilberger's Maple package Pisot.txt, based on this article by Zeilberger and Neil Sloane. It is my hope that this Python implementation provides a more accessible example of the uses of symbolic computation in languages more general than Maple.

To use the package, ensure that SymPy is installed, then clone the GitHub project (via `git clone https://github.com/rwbogl/pisot.git`, or downloading a ZIP file from the page, etc.). On the terminal, navigate to the directory that `pisot.py` and `cfinite.py` are in, then open a Python terminal. After this, import the `pisot` module with `import pisot` or some variant. Then every function defined in `pisot.py` will be available as `pisot.func_name`.

CHAPTER 3

Examples

The highlight of the package is *pisot_to_cfinite()*. Given a *Pisot* instance, it tries to determine whether or not it satisfies a linear recurrence relation with constant coefficients.

Example:

```
In [1]: import pisot

In [2]: from sympy import Rational

In [3]: p = pisot.Pisot(5, 17, Rational(1, 2))

In [4]: guess_terms = 10

In [5]: check_terms = 1000

In [6]: c = pisot.pisot_to_cfinite(p, guess_terms, check_terms)

In [7]: c
Out[7]: CFinite([5, 17], [4, -2])

In [8]: c.get_terms(10)
Out[8]: [5, 17, 58, 198, 676, 2308, 7880, 26904, 91856, 313616]

In [9]: p.get_terms(10)
Out[9]: [5, 17, 58, 198, 676, 2308, 7880, 26904, 91856, 313616]
```

Or, using the `verbose` flag:

```
In [10]: pisot.pisot_to_cfinite(p, guess_terms, check_terms, verbose=True)
The Pisot sequence E_{1/2}(5, 17), whose first few terms are
    [5, 17, 58, 198, 676, 2308, 7880, 26904, 91856, 313616],
appears to satisfy the C-finite recurrence CFinite([5, 17], [4, -2]) whose first few␣
↪terms are
    [5, 17, 58, 198, 676, 2308, 7880, 26904, 91856, 313616],
The absolute value of the second-largest root of the C-finite sequence is 0.
↪585786437626905 <= 1.
Therefore our conjecture holds.
Out[10]: CFinite([5, 17], [4, -2])

In [11]: p = pisot.Pisot(8, 16, Rational(1, 2))

In [12]: pisot.pisot_to_cfinite(p, guess_terms, check_terms, verbose=True)
The Pisot sequence E_{1/2}(8, 16), whose first few terms are
    [8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096],
appears to satisfy the C-finite recurrence CFinite([8], [2]) whose first few terms are
    [8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096],

This C-finite sequence looks like a geometric sequence, so this is easier to check.
The conjecture holds if x divides y; the guessed ratio equals y / x; and r is in [0,␣
↪1).

We already know that r satisfies this.
The conjectured geometric sequence has ratio 2.

The ratio is an integer and equals y / x, so our conjecture holds.
Out[12]: CFinite([8], [2])
```

CHAPTER 4

Classes

We define the classes `pisot.Pisot` and `cfinite.CFinite`. These inherit from `SeqBase`, so they are fully-fledged sequences that can be used in SymPy. The classes encapsulate some important operations and properties of C-finite and Pisot sequences, namely:

- Computing lists of terms is done with `Pisot.get_terms()` and `CFinite.get_terms()`.
- For CFinite sequences, the characteristic polynomial is computed with `CFinite.characteristic_poly()`, and its roots with `CFinite.characteristic_roots()`.
- Guessing a linear recurrence with constant coefficients can be done with `Pisot.find_cfinite_recurrence()`.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

`cfinite`, 5

p

`pisot`, 6

Index

C

CFinite (class in cfinite), 5
cfinite (module), 5
characteristic_poly() (cfinite.CFinite method), 5
characteristic_roots() (cfinite.CFinite method), 5

D

default_assumptions (cfinite.CFinite attribute), 5
default_assumptions (pisot.Pisot attribute), 7

F

find_cfinite_recurrence() (pisot.Pisot method), 7

G

gen() (cfinite.CFinite method), 5
gen() (pisot.Pisot method), 7
get_terms() (cfinite.CFinite method), 6
get_terms() (pisot.Pisot method), 7
guess_cfinite_degree() (in module cfinite), 6
guess_cfinite_recurrence() (in module cfinite), 6

I

interval (cfinite.CFinite attribute), 6
interval (pisot.Pisot attribute), 7
is_commutative (cfinite.CFinite attribute), 6
is_commutative (pisot.Pisot attribute), 7

P

Pisot (class in pisot), 6
pisot (module), 6
pisot_root() (in module pisot), 7
pisot_to_cfinite() (in module pisot), 7

S

start (cfinite.CFinite attribute), 6
start (pisot.Pisot attribute), 7
stop (cfinite.CFinite attribute), 6
stop (pisot.Pisot attribute), 7