Sympy - Solvers: Extending `solveset`

Jogi Miglani

April 9, 2019

Abstract

Sympy currently is in the process of replacing `solve` with `solveset`. Many places are left to replace `solve` with `solveset`. I plan to do the following over the summer.

- Extending `transolve`: Completing lambert and handling modular equations.
- Handle nested trigonometric equations.
- Enhancing the set infrastructure.
- Integerating helper solvers with `solveset`.

Contents

1 About Me

The Student

Name Jogi Miglani

E-mail jmig5776@gmail.com

Alternate E-mail jogimiglani.cd.mat17@itbhu.ac.in

Github https://github.com/jmig5776

The Institute

University Indian Institute of Technology (BHU), Varanasi

Major Mathematics and Computing

Year Sophomore

Degree Integrated Dual 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, C++, Java, Javascript. For version control, I have been using git. I work on Ubuntu 18.04.1 LTS with *vim* as my primary editor. I have background knowledge in Differential Calculus, Linear Algebra and Abstract Algebra along with Probability and Statistics. I have good experience in python as well as git version control.

Solving equations is very important in mathematics. Solving transcedental equations using Sympy is a favourite feature for me.

```
>>> from sympy.solvers.solvers import _tsolve as tsolve
>>> from sympy.abc import x
>>> tsolve(3**(2*x + 5) - 4, x)
  [-5/2 + log(2)/log(3), (-5*log(3)/2 + log(2) + I*pi)/log(3)]
```

My Motivation

I have always been fascinated with solving algebraic equations fast and with accuracy. My interests matched with Sympy, So I decided to explore this library more.

Solvers module of sympy is very powerful and no doubt it fascinates me with its techniques and algorithms. So I went through the solvers module to get deep taste of its features.

Contributions

I started using Sympy in November and made my first contribution in beginning of December. I have been constantly contributing and learning new skills from the community.

Merged PR's

- 15607 - Fixed `sympy.plot` sampling issue with logarithmic axis
- 15712 - Fixed matrix power failure error, it raised `ValueError` error when called with variable not known to be negative.
- 15735 - Implemented support for equations of type *f(x)**g(x) = c* to include all the solutions in the result. Earlier following types of equation does not returned all solutions :

  ```
  >>> eq = Eq((x**2 - 7*x + 11)**(x**2 - 13*x + 42), 1)
  >>> solve(eq)
  [2, 5, 6, 7]
  ```

  It did not considered the case *(-1)**(2*k)* k being an positive integer. But it does now :

  ```
  >>> eq = Eq((x**2 - 7*x + 11)**(x**2 - 13*x + 42), 1)
  >>> solve(eq)
  [2, 3, 4, 5, 6, 7]
  ```

  And many more cases are also included in real and complex range.
- 15742 - Changed CSS in docs to prefer DejaVu Sans Mono.
- 15786 - Fixed `Piecewise` to choose appropriate backend, it raised `TypeError` earlier.

- 15809 - Fixed Min/Max when no args are given to return `oo`/`-oo`.
- 15855 - Fixed `summation` to prevent infinite loop.
- 15861 - Fixed docstring of `logcombine` method.
- 15957 - Fixed `checkodesol` method and made more robust when checking solutions involving Integrals.
- 15961 - Fixed simplification of an evaluated *tan(3*pi/8)*.
- 16429 - Fixed `logcombine` - combine single unknown log with others.

Unmerged PR's

- 15692 - Fixed `Mul.flatten` method such that it returns the correct return type in case of matrices addition/multiplication.
- 15708 - Fixed `powsimp` to simplify further.
- 15826 - Fixed *integrate(1/(2**(2*x/3)+1), (x,0,oo))*.
- 15935 - Fixed `evalf` when expression involving symbols.
- 16324 - [WIP] replacing `solve` with `solveset` in `_lambert` in `bivariate`
- 16427 - [WIP] handling linear modular equations in solveset.

Issues/Bugs Caught

- 15940 - `trigsimp` or `simplify` fails to simplify this inverse trigonometric identity:

```
>>> from sympy import Symbol, acos, sqrt, atan
>>> D = Symbol('D', positive=True)
>>> e = acos(D) + 2*atan(D/(sqrt(-D + 1)*sqrt(D + 1))
- 1/sqrt(-D**2 + 1))
>>> trigsimp(e)
acos(D) + 2*atan(D/(sqrt(-D + 1)*sqrt(D + 1))
- 1/sqrt(-D**2 + 1))
>>> simplify(e)
acos(D) + 2*atan(D/(sqrt(-D + 1)*sqrt(D + 1))
- 1/sqrt(-D**2 + 1))
```

2 The Project

In this section I will explain my project details and its vision. I expect this structure to be considerably enhanced under the guidance of my mentor.

2.1 Brief Overview

One of the pretty powerful module in Sympy is Solvers. It has `solve` which is very powerful but also has some major issues as follows:

- Its output are of various types which is not considered to be consistent.
- The input API takes a lot of parameters which are not needed and they makes it hard for the user and the developers to work on solvers.
- There are cases like finding the maxima and minima of function using critical points where it is important to know if it has returned all the solutions. `solve` does not guarantee this.

To fix the above issues `solveset` module was created and is under development. We had to replace `solve` with `solveset` completely. My main concern of this project are:

- Completing `transolve`
- Handle nested trigonometric equations
- Enhancing the set infrastructure.
- Integrating helper solvers with `solveset`

2.2 The Plan

2.2.1 Completing transolve

As work done by @Yathartha22[1] in the summer of 2018 results that now transolve is fully designed and is able to handle logarithmic and exponential equations for solveset. I want to continue his work and want to complete the following :

- Equations solvable by LambertW function

  Lambert solver is almost complete by @Yathartha22[1] in 14972 but currently it is not able to return all solutions. I will complete that and some leftovers are to pass some tests of bivariate solvers and to replace solve with solveset in _lambert in bivariate. I had opened a 16324 to complete the Lambert solver.

- Handling Modular Equations

  In 13178 one could possibly define _invert_modular to come up with a general fix of the issue.
  Issue:

  ```
  >>> n = symbols('n', integer=True)
  >>> a = 742938285
  >>> z = 1898888478
  >>> m = 2**31 - 1
  >>> x = 20170816
  >>> solveset(x - Mod(a**n*z, m), n, S.Integers)
  ```

  Possible fix: Defining _invert_modular that will be inverting modular equations. One approach is here 14284 but it needs to be generalised and worked upon.
  I had tried to solve the problem with different approach in 16427 and had also opened a discussion about it in 16428.

- Nested Trigonometric Equation Solver

  solveset uses _solve_trig to solve such trigonometric equations and focuses on returning general solutions. But still _solve_trig is unable to solve a variety of problems. Following are the issues that it faces and I would like to fix this during my GSoC term.

  Examples to be worked upon:

  The problem is inability to solve nested trignometric expressions 10217

  ```
  >>> eq=cos((sin(x)+1)) - 1
  >>> solveset(eq,domain=S.Reals)
  ConditionSet(x, Eq(cos(sin(x) + 1) - 1, 0), (-oo, oo))
  ```

  Possible fix: such problems are the ones that can be converted to polynomial equations(by the principle of Decomposition and Rewriting) and thus solved. For these type of problems we can have an internal call to solve_decomposition which solves the problem

2.2.2 Defining ImageSet Union for Trigonometric Equation Solver

This issue in Sympy still prevails due to the lack of a proper algorithm for unifying two ImageSet objects such as `2*n*pi` and `2*n*pi + pi`. And many much of its use case is in trigonometric equation solving.

```
>>> solveset(sin(2*x) + sin(4*x) + sin(6*x) , x, S.Reals)
Union(ImageSet(Lambda(_n, 2*_n*pi), Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi), Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/2), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 3*pi/2), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 4*pi/3), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 2*pi/3), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/3), Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/3), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 5*pi/4), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 3*pi/4), Integers),
ImageSet(Lambda(_n, 2*_n*pi + 7*pi/4), Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi/4), Integers))

# this should be
Union(ImageSet(Lambda(_n, _n*pi/4), Integers),
ImageSet(Lambda(_n, _n*pi + pi/6), Integers),
ImageSet(Lambda(_n, _n*pi - pi/6), Integers))

>>> solveset(sin(x), x)
Union(ImageSet(Lambda(_n, 2*_n*pi), Integers),
ImageSet(Lambda(_n, 2*_n*pi + pi), Integers))

# This should be
ImageSet(Lambda(_n, _n*pi), Integers)
```

The above problem can be solved by defining Image set Union. Various approaches are tried by contributors, some are :
@Shekharrajak[2] tried to solve this problem in these PRs 10733 10898 10713. The approach is good but it needs to be implemented in a general way which will solve the problem.

2.2.3 Integrating helper solvers with solveset

Here are three solvers which I will be going to integrate in solveset.

- `linsolve`: Solves a system of linear equations

- `nonlinsolve`: Solves a system of non-linear equations and improving it to handle transcedental/trigonometric equations at a time.
- `solve_decomposition`: Solves a varied class of equations using the concept of Rewriting and Decomposition(It will be already implemented while covering trigonometric equation solver.)

2.3 Execution

Whatever issues are there, they are not solved unless properly executed and planned. Therefore to get a better result out of what I have proposed I divide my work into three phases.

2.3.1 Phase 1

- **Completing Lambert Solver** :
  During the first phase, I will be first completing the lambert solver. Currently the Lambert solver implemented by @Yathartha22[1] in 14972 is not returning all solutions.

  Implementation
  Basically the problem in `_solve_lambert` is that it tries to convert all type of lambert expressions to this form *F(X, a..f) = a*log(b*X + c) + d*X + f = 0* in which while converting them to log some solutions are missed.
  So while converting exponential equations to required logarithmic form, We should take *abs* of what inside the log we will be taking. For eg :

  ```
  if mainpow and symbol in mainpow.exp.free_symbols:
       lhs = collect(lhs, mainpow)
       if lhs.is_Mul and rhs != 0:
          soln = _lambert(expand_log(log(abs(lhs)) -
                        log(abs(rhs))), symbol)
  # see the abs used.
  ```

  Moreover I had implemented a sample code for completing lambert in 14972.

- **Handling modular equations** :
  Currently `solveset` is unable to handle modular equations i.e returns `ConditionSet`. I have tried to think of an approach to implement this

solver.
Implementation
General equation that will be handled here is:

```
A - Mod(B, C)
```

where A, B can be function of x Firstly I will try to extract the terms containing x on one side of equation using `invert`. And then by further solving we will get the required answer.
Here is a code snippet that I implement in 16427 :

```
# In solveset
elif f.has(Mod) and not f.has(TrigonometricFunction):
    modterm = list(f.atoms(Mod))[0]
    t = Dummy('t', integer=True)
    f = f.xreplace({modterm: t})
    g_ys = solver(f , t, S.Integers)
    if isinstance(g_ys, FiniteSet):
        for rhs in g_ys:
            if not rhs.has(symbol) and modterm.has(symbol):
                result += _solve_modular(modterm, rhs, symbol, domain)

# Function _solve_modular
def _solve_modular(modterm, rhs, symbol, domain):
    "Helper function to solve modular equations"
    #Documentation to be added and more implementation also to be added
    #It gives idea of implementation
    a, m = modterm.args
    n = Dummy('n', real=True)

    if a is symbol:
        return ImageSet(Lambda(n, m*n + rhs%m), S.Integers)
    if a.is_Add:
        g, h = a.as_independent(symbol)
        u = Dummy('u', integer=True)
        sol_set = _solveset(u - rhs + Mod(g, m) , u, domain)
        result = EmptySet()
        if isinstance(sol_set, FiniteSet):
            for s in sol_set:
                result += _solveset(Mod(h, m) - s, symbol, domain)
        elif isinstance(sol_set, ImageSet):
            soln = _solveset(Mod(h, m) - sol_set.lamda.expr, symbol,
            S.Integers)
        else:
            result = ConditionSet(symbol, Eq(modterm - rhs, 0), domain)
        return result
    if a.is_Mul:
        g, h = a.as_independent(symbol)
        return _solveset(Mod(h, m) - rhs*invert(g, m), symbol, domain)
    if a.is_Pow:
        if a.exp is symbol:
            return _solveset(a.exp - discrete_log(m, rhs, a.base),
            symbol, domain)
    return ConditionSet(symbol, Eq(modterm - rhs, 0), domain)
```

This above code is implemented in 16427.
This above code solves the issue of 13178 :

```
>>> n = symbols('n', integer=True)
>>> a = 742938285
>>> z = 1898888478
>>> m = 2**31 - 1
>>> x = 20170816
>>> solveset(x - Mod(a**n*z, m), n, S.Integers)
{100}
```

Moreover it returns solutions to equation like this as:

```
>>> x = Symbol('x')
>>> n = Dummy('n', real=True)
>>> solveset(3-Mod(x,5), x)
ImageSet(Lambda(n, 5*n + 3), S.Integers)
>>> solveset(3-Mod(x + 2, 5), x)
ImageSet(Lambda(n, 5*n + 1), S.Integers)
>>> solveset(3-Mod(5*x + 2, 7), x)
ImageSet(Lambda(n, 7*n + 3), S.Integers)
>>> solveset(3-Mod(5*x,7), x)
ImageSet(Lambda(n, 7*n + 2), S.Integers)
```

One more approach by @ishanaj[3] has been implemented in 14284 defining a proper `_invert_modular` function. After discussing with mentors a full proof method will be implemented.

- **Nested Trigonometric Equation Solver** :
  As above it is described that for solving trigonometric equation `_solve_trig` is used. In 12340, discussion by @Yathartha22[1] had mostly cleared the path to implement it.
  - Equations which can be reduced to polynomial ones can be solved by rewriting and decomposition. And for the rewriting and decomposition we can have an internal call to `solve_decomposition` which will convert the given equation in reduced polynomial or required form and further we can solve the problem.
  - Most of the solutions returned by `solveset` for trigonometric equations does not get unified properly. So to fix this ImageSet Union will be needed to defined and this will be done in Phase 2.

2.3.2 Phase 2

- First thing is to complete all the leftovers of **Phase 1** so that no problem should arise while implementation of the things of **Phase 2**.
- **Defining ImageSet Union for Trigonometric Equation Solver**

  Discussion in 12340, by @Yathartha22[1] describes that a proper ImageSet Union had to be defined.
  - First of all the sets with linear expressions will be separated out to call a particular function to unify those sets.

    Here is the code snippet to do that:-

```
if isinstance(other, ImageSet):
#This is just a rough implementation.
# Will be improved after discussing with mentor
    if len(self.lamda.variables) > 1 or
    len(other.lamda.variables) > 1:
        # Only single lambda supported
        return None
    self_sym, other_sym = self.lamda.variables, other.lamda.variables
    self_expr, other_expr = self.lamda.expr, other.lamda.expr
    linear = Poly(self_expr, self_sym).is_linear and \
        Poly(other_expr, other_sym).is_linear
    if not linear:
        # Does not fulfill the needs of linearity
```

```
            return None
        base = other.base_set
        if base != self.base_set:
            # Base should be same
            return None
        result = unify_img(base, self_expr, other_expr,
                            self_sym, other_sym)
        return result
```

– Then proper algorithm will be applied to reduce the expression after discussing the proper plan with the mentor.
  Here I had tried to think of an algorithm which can be implemented to solve the problem:

  1. Let us suppose the two expressions be:
  *a*n + b* and *c*n + d*
  where the expressions will be reduced satisfying b < a and d < c.

  2. Now p = *lcm(a, c)* and two rings of *mod p* will be created for two expressions like below:
  For *a*n + b*: Add *b%p* to every element of modular ring of a in mod p.
  For *c*n + d*: Add *d%p* to every element of modular ring of c in mod p.

  3. Now 2 rings of p will be obtained from step 2. Now here we should merge the two rings.

  4. Now in the final ring if difference of any two elements comes out to be not co-prime with p then it will be further reduced accordingly.

  For example :
  2πn + 3π/4 and 2πn + 7π/4
  As expressions are already reduced according to step no. 1 so reduced expressions are:
  2πn + 3π/4 and 2πn + 7π/4
  Now p = *lcm(2π, 2π)*, so two rings of *mod p* i.e *mod 2π* are:
  2πn + 3π/4 —> $\{3\pi/4\}$
  2πn + 7π/4 —> $\{7\pi/4\}$
  So now merging the two rings we get $\{3\pi/4, 7\pi/4\}$ of *mod 2π*.

Now as the difference of the elements is π which is not co-prime with 2π. So 2π*difference/p equals to $2\pi * \pi / 2\pi = \pi$ (it will be taken as coefficient of n) and the initial or small phase angle is $3\pi/4$.
So the answer is $\pi n + 3\pi/4$.

I am not sure where this method fails but it solves most of the cases. But some more approaches implemented by previous GSoCer's I want to share.@Shekharrajak[2] and @Kshitij10496[9] had tried to solve this problem by implementing some heuristics(such as genform). And @Shekharrajak[2] has also tried to solve the problem of ImageSet Union in 12011 and 11188. During my work period I will try to improve his implemented method to pass the problem currently it is facing. I will make sure to discuss with mentor and properly complete his implementation in an efficient manner and get his pull request merged.
@oscargus[8] has also tried to solve this in 16033. After proper discussion with mentors regarding which method to be implemented, ImageSet Union will be defined.

- **Integrating helper solvers with solveset**

  During this period I will also be integrating `linsolve` and `nonlinsolve` in the `solveset` to increase more coverage of `solveset`.

2.3.3 Phase 3

- In this Phase, the pending tasks of Phase 1 and Phase 2 will be completed. Like `ImageSet Union`, proper test cases will be added.

- Issues or minor bugs arise in Phase 1 and Phase 2 will be fixed in this Phase.

- I will start implementing the foundation for transolve in complex domain.

- In this Phase I will try to replace `solve` calls with `solveset` and will be exploring solveset more making it more efficient.

This Phase is for more exploration of `solveset` and I will try to close 10006.

3 Timeline

This project will make `solveset` as strong as `solve` and I assure that I will devote all my time in this project and try to finish what I proposed. I will be having my semester exams till mid of May, but even during this time I will try to give 2-3 hours daily and thereafter I will devote all of my time(about 40-50 hrs a week or even more).I might not be available for 1-2 days in July(travelling). College will start in August and not enough work happens during the first month so I can work with full devotion.

3.1 Community Bonding Period

- Discuss the project with my mentor in further detail.
- Get more acquainted with Sympy's `solvers` codebase
- Also I will have discussions and meetings with the mentors and we will come up with the efficient way

3.2 Coding Period

- **Week 1,2,3 (May 30 - June 19)**

  Completing `lambert`
  Adding test cases and Implementing modular equation handler

- **Week 4,5 (June 20 - July 3)**

  Implementing nested Trigonometric equation solver
  Fixing bugs and issues that may have raised during previous implementation

- **Week 6 (July 4 - July 10)**

  Improvement of ImageSet starts and adding test cases
  Analyzing how ImageSet works and discussing with mentors and arriving at a plan.

- **Week 7,8(July 11 - July 24)**

  Work on ImageSet Union, make it efficient and try to merge it.
  Minor bug fixes

- **Week 9,10,11(July 25 - Aug 14)**

  Integrating `linsolve` with solveset and writing test cases.
  Integrating `nonlinsolve` with solveset
  Improving `nonlinsolve` to handle transcedental/trigonometric equations at a time.

- **Week 12,13(Aug 15 - Aug 29)**

  Fixing minor bugs/issues.
  Finishing above work if it is remaining.
  Replace all internal calls from `solve` to `solveset`
  Analyzing what all things that can be to improve trigonometric equation solver to make it more better.
  Implementing the foundation for transolve in complex domain.

3.3 Post GSoC

If there are things that are left unimplemented I will try to complete post GSoC and even I will also continue my contributions to sympy and help the new contributors.I will also try to implement and complete `transolve` for complex domain. Sympy provides me with a good platform to hone my programming skills. I would love to contribute more to Sympy and its solvers module to make it more powerful and robust.

References

[1] Yathartha Joshi
https://github.com/Yathartha22

[2] Shekhar Prasad Rajak
https://github.com/Shekharrajak

[3] PR for handling modular equation by Ishan Joshi
https://github.com/sympy/sympy/pull/14284

[4] Modular Arithmetic Wikipedia Source
https://en.wikipedia.org/wiki/Modular_arithmetic

[5] Yathartha Joshi Gsoc Application
https://github.com/sympy/sympy/wiki/

GSoC-2017-Yathartha-Anirudh-Joshi-Application:-Solvers:
-Extending-Solveset-II

[6] Shekhar Prasad Rajak Gsoc Application
https://github.com/sympy/sympy/wiki/
GSoC-2016-Shekhar-Prasad-Rajak-Application%3A-Solvers%
3A-Completing-Solveset

[7] Shekhar Prasad Rajak PR on ImageSet Union
https://github.com/sympy/sympy/pull/12011

[8] Oscar's PR on ImageSet Union
https://github.com/sympy/sympy/pull/16033

[9] Kshitij Saraogi
https://github.com/Kshitij10496