Risch algorithm for symbolic integration

Chetna Gupta

May 1, 2013

Abstract

The Risch algorithm is a complete algorithm to integrate any elementary function. Given an elementary function, it will either produce an antiderivative, or prove that none exists.The project is to continue where Aaron Meurer left off in his 2010 GSoC project, implementing the algorithm from Manuel Bronstein's book, Symbolic Integration I: Transcendental Functions. A successful implementation of this would not only provide concrete algorithms for transcendental functions but would also form the basis for very similar Karr Algorithm, decision procedure for symbolic summation, yet to be implemented in sympy

1 Personal Details

- Name: Chetna Gupta
- Email: cheta.gup@gmail.com
- IRC: chetna on freenode
- Skype: chetna.gupta3
- Github: cheatiiit

2 Backgound

I am a second year Computer Science student at International Institute of Information and Technology, Hyderabad, India. I am planning to pursue

my honors in Visual Image processing. I am a Microsft Windows 8 Official App developer and have been one amongst the winning teams of Microsoft Acad Accelerator . Being a member of the Open Source Development Group at my college and after participating in various hackathons I realized that open source is based on scratching a developer's personal itch. Complex data signal analysis, integrating filters with signals to recover data etc, as would be required for my honors motivated me to look for python based CAS and indeed I started looking into sympy's codebase . Maths has always been not only a field of knowledge but also encouragement for me. I am quite comfortable with Linear Algebra and Abstract Algebra.

3 Coding

I have done coding in several high-level languages (C, C++, C#, Python, Java); What I like about Python, as a mathematician, is the simplicity and readability of the language which allow for what I find to be very elegant programs. For over two years now I've been coding on a Linux workstation. I cannot say I'm considerably fluent in Python but I see my level somewhere around intermediate. I have written various scripts to analyze data in various fields of my interest, this includes use of python to interpret dynamic nature of signals mostly Wi-Fi- Access Points. building backend for Search engines , data-file parsers, python web-crawler, using python based web development softwares like web2py and also developing game using n-curses library as a course project.

I have been using svn and had heard of git before, but I started using it because of my involvement in the sympy project. I'm still learning the little tricks, but now that I've submitted some of the patches enhancing the code under the integral module which make me feel more confident about using it.

4 Contributions to Sympy

My contributions to Sympy are as follows:

4.1 Patch Requirements

- https://github.com/sympy/sympy/pull/2002: Implementing Method for recognizing derivatives , logarithmic derivatives for rational functions Section 5.12 from [1] which has been bottlenecking most of the other unimplemented algorithms
- https://github.com/sympy/sympy/pull/2049 Addition of hypertangent and exponential cases to Partial Differntial Equations
- https://github.com/sympy/sympy/pull/2034: Parametric Poly Risch Differential Equation (deg(b) $\leq$ 0)
- https://github.com/sympy/sympy/pull/1978 : Changed SPDE from recursive to iterative (Prevents RuntimeError when called recursively by functions like rischDE, paramrischDE)
- https://github.com/sympy/sympy/pull/1957: Changed integrate-primitive-polynomial from recursive to iterative (Prevents RuntimeError: maximum recursion depth exceeded for test-cases with large powers eg risch-integrate($(log(x)+1)^{1001}$)
- https://github.com/sympy/sympy/pull/1948 : Enhanced Hermite Algorithm from quadratic version to Mack's Linear version (Reduced time complexity from n^2 to $n-1$)
- https://github.com/sympy/sympy/pull/1917 : Implemented risch_integrate for trigonometric, inverse trigonometric and hyperbolic functions (Raised NonImplementedError earlier)
- https://github.com/sympy/sympy/pull/1906 : Optimized number of permutations of mappings in heurisch (Reduced order from

5 Timeline

As I have already started following the text for implementation and improvisation of risch algorithm, I plan to immediately start working on the same.

- Week 1-2
  Recognizing derivatives, log derivatives , log derivatives in k radical

along with test cases and also the unimplemented sub-algorithms of Risch Differential Equations.

- Week 3
  Implementation of Cancellation Algorithm for the Liouvillian Case

- Week 4
  Consideration of Non-Linear Cases in Parametric Risch Differntial Equation

- Week 5
  Solution for Hyper-tangent Cases by using Cancellation Algorithm

- Week 6
  Would start looking into code for implementation of coupled differential equation and would solve CDS for the primitive cases, more sophisticated testing and benchmark

- Week 7-8
  (Mid-term Evaluation) Solution of CDE for hypertangent and exponential cases with test cases

- Week 9
  Implementation of non-linear cases for Coupled Differential System

- Week 10
  Would review all the pull requests and would add more complex test cases for each

- Week 11
  Structure theorem for cases other than linear & exponential and solving parametric logarithmic derivative problem

- Week 12
  Extending Limited integration Algorithm, Adding complex test Cases for all the cases

- Week 13
  Look back at the entire code and make improvements & stylistic changes if necessary; more sophisticated testing, benchmarks (if not done already); add details to documentation

- Week 14
  Final Evaluation

6 Implementation Details

6.1 Implementation of methods to recognize derivatives , logarithmic derivatives, logarithmic derivatives k(t) radical

A code-block for recognizing derivatives, log derivatives and log derivatives k radical, "integrals" module would enable us to implement most of the algorithms for the Risch differential Equation which have been raising a NonImplementedErrors.

Examples from the code

- Special-denom in integrals/rde.py : This method though works for primitive, exponential and base cases but it fails to solve hypertangent cases raising a NotImplementedError("Tangent case not implemented yet for RDE special-denom().") because of unavailability of algorithm to check if there exist a log derivative of a function (say) $B = dv/v$ for some v in K^*.

6.1.1 Theory Planned

- Recognizing Derivatives: In order to recognize if a given rational function (say) F is a derivative of a rational function without actually computing the integral of F, I plan to use a couple of criterion as follows:
  - Laurent Series Method:
    For a give function F let A,D be the numerator and denomintor of F respectively where A,D belong to K(x) with D monic and nonzero, gcd(A, D) = 1, If $D = D_1, D_2 \dots D_n$ is squarefree factorization of D. Then, using only rational operations over K, for each D_i (i varies from 1 to n) I compute H_{ij} belonging to K[x] for $1 \leq j \leq i \leq n$ such that the partial fraction decomposition of A/D is
    $$\text{A/D} = \text{P} + \sum_{i=1}^{n} \sum_{a, D_i(a)=0} (H_{ii}(a)/(x-a)^i + \dots + H_{i1}(a)/(x-a))$$

After computing Hi1 for each Di,we write Di as Di = GiEi where Gj = gcd(Hi1, Di) and gcd(Ei,Hi1) = 1. Since the residues of F at the roots of G_j are all 0, and the residue of F at a root "a" of E_i is $H_{i1}(a) \neq 0$, f is the derivative of a rational function if and only if $E_i = 1$ for each i, which is equivalent to $D_i | H_{i1}$ for each i .

- Marik's Criteria:
  The second approach could be to compute the squarefree factorization $D = D_1, D_2 \dots D_m$ of the denominator of F, and write F $\sum_{i=1}^{n} A_i/D_i^i$ If F is the derivative of a rational function, then $D_1 | A_1$, since the residuesof F at the roots of Di would be nonzero otherwise. If $D_1 | A_1$, then f is the derivative of a rational function if and only if each Ai/Di^i is the derivative of a rational function for i ¿ 1, and we can use Marik's criterion, which states that A/D^m is the derivative of a rational function for $m > 1 iff$ D | Wronskian$(dD/dx, dD^2/dx, ...dD^m - 1/dx, A)$

I have already started working on this by including methods like: laurent-series, recognize-derivatives (refer to
PR https://github.com/sympy/sympy/pull/2002 for more details)

Although these two approaches would solve most of the cases but still it is not an alternative to the normal method of using Hermite Reduction and calculating integral(returning True if an integral is found, False otherwise) therefore if none of them prove that the function is a derivative of a rational function then we shift to the last available choice of hermite reduction. I am mostly done with the laurent series approach but still need to add complex test cases and code the 'Marik Criteria' and 'Hermite Reduction Method' so that we can get a valid result for more general functions. As a result alot still needs to be done in this section. As there is no psudeo code present for solving this in [1] or any other on-line references, therefore I would have to write more unit-testing test cases to check and verify each line of the code. This would require me to exhaust all possibilities where the above may fail. I intend to put laborious calculations to get this through as not much examples are available to solve this.

- Recognizing Log Derivative Inorder to determine whether there exists u in K(x)* such that du/dx = uf ie F is the logarithmic derivative of a

rational function, if and only if F can be written as F = A/D where D is squarefree, deg(A) ¡ deg(D), gcd(A, D) = 1,and all the roots of the Rothstein-Trager resultant are integers.
I am mostly done with this section.Have implemented:recognize-log-derivative(refer to PR https://github.com/sympy/sympy/pull/2002 for more details)

I have also included test cases to recheck and verify each and every line of the code. Therefore there is nothing much left to do for this part. But as some of the Risch Differential Equation algorithms have sub-parts which particularly depend on the recognizability of a function to be a log derivative, I plan to complete all such possible sub-parts using this method. Hence exhausting and completing the sub-algorithms that haven't been implemented yet from Chapter 6 of [1]

6.2 Implementation and Improvisation of Parametric Risch Differential Equations

To solve Parametric Differential Equations currently the only cases implemented in integral module for equations of the form Dq + bq = Sum(i,1,m) ciqi are as follows:

- D = d/dt or deg(b) > max(0, $\delta(t) - 1$)
- deg(b) < $\delta(t) - 1$ or $\delta(t) \geq 2$

We still need non-cancellation algorithm for cases of the form $\delta(t) \geq 2$ and deg(b) = $\delta(t) - 1$ as well as cancellation algorithms for

- Liouvillian Cases → $D \neq d/dt$ and $deg(t) <= 1$
- Nonlinear Cases → $\delta(t) \geq 2$ and $deg(b) = \delta(t) - 1$ and lc(b) = n*lambda(t)
- Hypertangent Cases → $Dt/(t^2 + 1)$ = eta in k such that $\delta(t) = 2$

6.3 Implementation of Coupled Differential System

As there is no method to address a solution for coupled differential system, I plan to write various cancellation algorithms which would help in extending Coupled Differential Systems to a coupled system of real and imaginary part of risch differential equations. This would enable us to find solutions of the form (say) (y_1, y_2) where y_1, y_2, y satisfy

$Dy + (f_1 + f_2 \times \sqrt{a}) = (g_1 + g_2 \times \sqrt{a})$

- $y = y_1 + \sqrt{a} \times y_2$
- $y \in K(\sqrt{a}))$
- $y_1, y_2 \in K$
- $\sqrt{a} \notin K$
- for a given Differential Extension K

  (a = -1 being a specific case for the above conditions)

6.3.1 Theory Planned

Since coupled differential systems are generated by integration algorithms only when $\sqrt{-1}$ [in general $\sqrt{a}$] $\notin$ K, therefore inorder to find a solution for y $\in K(\sqrt{-1})$ of

$Dy + (f_1 + f_2 \times \sqrt{a}) = (g_1 + g_2 \times \sqrt{a})$

I would consider real (say y_1) and imaginary (say y_2) parts of y. Hence $y = y_1 + y_2 \times \sqrt{-1}$ putting this in above equation we get

$D(y_1 + \sqrt{-1} \times y_2) + (f_1 + f_2 \times \sqrt{-1}) \times (y_1 + \sqrt{-1} \times y_2) = g_1 + g_2\sqrt{-1}$

$Dy_1 + \sqrt{-1} \times Dy_2 + f_1y_1 - f_2y_2 + \sqrt{-1}(f_1y_2 + f_2y_1) = g_1 + g_2(\sqrt{-1})$

Separating Real and imaginary parts we get

- $Dy_1 + f_1y_1 - f_2y_2 = g_1$
- $Dy_2 + f_1y_2 + f_2y_1 = g_2$

Since $(1, \sqrt{-1})$ is a vector space basis for $K(\sqrt{-1})$ over K, therefore if f_1 ,f_2, g_1 and g_2 are in K(t) then $y_1 + \sqrt{-1} \times y_2$ is a definite solution to first equation. In order to compute the above let $f = f_1 + f_2 \times \sqrt{a}$, $g = g_1 + g_2 \times \sqrt{a}$ So we get:

$Dy + fy = g$

Applying Rothsien SPDE Algorithm to the above equation, I intend to prove that either there is no solution for y in k(sqrt(a)(t) or there exist b, c, d, α, β in $k(\sqrt{a})[t]$ such that if a solution exists then y is of the form $y = (\alpha \times q + \beta)/d$ where q in $k(\sqrt{a})[t]$ is a solution of $Dq + bq = c$. Depending on the value of b we can check if the result from SPDE algorithm falls into non-cancellation cases ie:

- If degree(b) is large enough $b \neq 0$ and $D = d/dt$ or $deg(b) > max(0, \delta(t) - 1)$
- If $deg(q) > 0, deg(b) < \delta(t) - 1$, or either $\delta(t) \geq 2 or D = d/dt$,
- If $\delta(t) > 2, deg(b) = \delta(t) - 1, deg(g) > 0$ and $deg(g)! = -lc(b)/\lambda(t)$ where $\delta(t) = D$ degree of t , $\lambda(t) = D$ leading coefficient of t

If any of the above conditions are satisfied then we can apply the corresponding no-cancellation algorithms to the reduced equation Dq + bq = c since they do not generate any recursive problem over $k(\sqrt{a})$. Thus, in the non-cancellation cases, we can either prove that there is has no solution in $k(\sqrt{a})(t)$, or compute such a solution.

Apart from non-cancellaton cases CDS should also handle cancellation cases that is:

- $\delta(t) < 1, b \in k(\sqrt{a}) and D \neq d/dt$,
- $\delta(t) \geq 2, deg(b) = \delta(t) - 1$, and $deg(q) = -lc(b)/\lambda(t)$.

Note, some of these are straightforward to implement. For example, After going through [1] I found that it may not take much time in implementation of methods to recognize derivatives or logarithmic derivatives . While others such as Structure theorem and CDE will require more work to ensure that they catch all the types of possibilities that really fit their form. For this reason, it is possible that I may spend more time on one than I anticipated.

7 Logistics / Disclaimers

I have no plans other than traveling back home in the beginning of May , though this won't affect my productivity. I would be back in college by

Mid-May for next 3 months. Inorder to accomplish the opportunity I seek through this project, I would dedicate a whole-hearted 40 hours a week to work completely according to the timeline. I am mostly a self taught coder and learner but as most of the implementation left does not have a psudeo code or examples, though i would refer to various possible research papers, I would need some help to structure my work accordingly. I plan to an active sympy developer even after GSoC time-period and would continue the work I would undertake this summer

7.0.2 References

1 - M. Bronstein. Symbolic integration I: transcendental functions. Springer Verlag, 2005.

2 - Lattice Basis Reduction Part II: Algorithms Sanzheng Qiao Department of Computing and Software

3 - Basic Concepts of Differential Algebra Andreas W McMaster University, Canada qiao@mcmaster.ca www.cas.mcmaster.caqiao

4 - Wronskian - http://mathworld.wolfram.com/Wronskian.html

5 - Rothstein-Trager - http://www.apmaths.uwo.ca/ rcorless/AM563/NOTES/Nov-16-95/node12.html

6 - http://www.maths.surrey.ac.uk/explore/vithyaspages/coupled.html

7 - Barry M. Trager. On the Integration of Algebraic Functions. PhD thesis, MIT, 1984.

8 - James H. Davenport. On the Integration of Algebraic Functions. Lecture Notes in Computer Science 102. Springer, 1981.