Symbolic-Numeric Integration of Univariate Expressions based on Sparse Regression

Shahriar Iravanian
Emory University, Atlanta, GA, USA
shahriar.iravanian@emoryhealthcare.org

Carl Julius Martensen
Otto-von-Guericke University, Magdeburg, Germany

Alessandro Cheli
University of Pisa, Pisa, Italy

Shashi Gowda
Massachusetts Institute of Technology, Boston, MA, USA

Anand Jain
Julia Computing, Boston, MA, USA

Yingbo Ma
Julia Computing, Boston, MA, USA

Chris Rackauckas
Massachusetts Institute of Technology, Boston, MA, USA
accounts@chrisrackauckas.com

Abstract

The majority of computer algebra systems (CAS) support symbolic integration using a combination of heuristic algebraic and rule-based (integration table) methods. In this paper, we present a hybrid (symbolic-numeric) method to calculate the indefinite integrals of univariate expressions. Our method is broadly similar to the Risch-Norman algorithm. The primary motivation for this work is to add symbolic integration functionality to a modern CAS (the symbolic manipulation packages of SciML, the Scientific Machine Learning ecosystem of the Julia programming language), which is designed for numerical and machine learning applications. The symbolic part of our method is based on the combination of candidate terms generation (ansatz generation using a methodology borrowed from the Homotopy operators theory) combined with rule-based expression transformations provided by the underlying CAS. The numeric part uses sparse regression, a component of the Sparse Identification of Nonlinear Dynamics (SINDy) technique, to find the coefficients of the candidate terms. We show that this system can solve a large variety of common integration problems using only a few dozen basic integration rules.

1 Introduction

Symbolic integration is a core competency of most Computer Algebra Systems (CAS) and has numerous applications. This paper presents a symbolic-numeric (hybrid) integration method, similar to the heuristic methods, that applies numerical computations to simplify the intermediate steps.

JuliaSymbolics is a subset organization of **SciML** (an open source organization maintaining hundreds of scientific computing packages for the Julia programming language). Historically, **SciML** began as a collection of ordinary differential equation (ODE) solvers. Symbolic computation was added as a domain-specific language to ease the definition of ODE systems and to aid with the automatic calculation of Jacobian and Hessian of such systems. With the expansion of the **SciML** ecosystem, the purely symbolic routines were decoupled from the ODE solvers. Considering its history and origin, **JuliaSymbolics** is geared toward symbolic differentiation and numerical integration but lacks direct symbolic integration

capabilities. Furthermore, **SciML** has grown to cover recent advances in scientific machine learning. As part of our method, we utilize one of these new packages (**DataDrivenDiffEq.jl**) that implements data-driven differential equation structural estimation and identification.

2 Symbolic-Numeric Integration

The integration method presented here is a variant of the *method of indeterminate coefficients* and is closely related to the *Risch-Norman (parallel Risch or poor man's integrator)* integration method [4]. The main idea is to write the expected solution as a linear combination of many candidate terms (ansatz) with unknown complex coefficients, then equate the derivative of the solution with the integrand, and, finally, find the unknown coefficients by solving a system of linear equations.

The process described in the previous paragraph may seem purely symbolic; however, numerical computation becomes necessary to avoid relying on **JuliaSymbolics** to convert expressions into unique *canonical* forms. Identities like $\sin^2 x + \cos^2 x = 1$ may be correctly applied in this case, but in general, according to Richardson's theorem, the problem of finding canonical forms of transcendental expressions is undecidable. Another reason for using numerical computation is that the list of candidates may not be (and usually is not) linearly independent. Finding a linearly independent subset of a set of expressions is facilitated using numerical methods.

We split the integration algorithm into two parts. The symbolic part is concerned with generating candidate terms. The numeric part finds the coefficients of the terms. Our method, similar to the parallel Risch algorithm and in contrast to the recursive (classic) Risch method, generates many extra candidates that do not contribute to the solution and are expected to have zero coefficients.

Let the input function to be integrated, $f\colon \mathbb{C} \to \mathbb{C}$, be a univariate expression of the independent variable x. Additionally, we assume that f is well defined in a closed subset of the complex plane with only isolated poles.

The candidate terms generation algorithm produces a list of generator expressions, $G_0, G_1, G_2, \cdots, G_L$, where L is a user-defined parameter that determines the number of generators to try. Each G_l is converted to a set of candidates, $T_l = \{\theta_k\}$, by, first, expanding it symbolically and then removing constant coefficient from its terms. We can ignore the constant leading coefficients because the final coefficients are calculated by the numerical part of the algorithm. The algorithm applies the numerical part to T_0 first. If it finds an acceptable answer, it is returned; otherwise, T_1 to T_L are sequentially tested. If no answer is acceptable after trying T_L, the algorithm fails.

2.1 Symbolic Computations (Candidate Generation)

One key observation is that the form of the anti-derivative of some functions is similar to their derivative forms. These functions can be defined in terms of the polynomials of the exponential function and its inverse, i.e., $f \in \mathbb{C}[e^x, e^{-x}]$, $(e^x)' = e^x$, and $(e^{-x})' = -e^{-x}$; therefore $\mathbb{C}[e^x, e^{-x}]$ is closed under differentiation and integration.

In general, most integrands are not in $\mathbb{C}[e^x, e^{-x}]$; however, repetitive differentiation is still the backbone of the general algorithm. The essence of candidate generation is integration by parts followed by repetitive differentiation. The continuous homotopy operators method provides a systematic way to automate integration by parts [3]. Here, we borrow some of the machinery of the homotopy operators methodology to enhance the generation of candidate expressions. We start by rewriting the integrand, $f(x)$, into a form suitable for integration,

$$\boxed{f(x) = \prod_{i=1}^{N} u_i(x)^{n_i}\,,} \tag{1}$$

where $u_i(x) = g_i(v_i(x))$ such that $g_i(v)$ is a function that can be integrated easily using a rule-based system, $v_i(x)$ defines the arguments of $g(v)$, and $n_i \in \mathbb{Z}^+$. For example, if $f(x) = (x+1)^2 \sin(x^2-1)$, then $f = u_1^2 u_2$ for $u_1 = v_1$, $v_1 = x+1$, $u_2 = \sin(v_2)$ and $v_2 = x^2 - 1$.

We generate G_0 by integrating each u_i (with respect to v_i) in turn. We integrate the first factor of f, i.e. $u_1^{n_1}$, by multiplying f by v_1'/v_1' to split it into $u_1 v_1'$ and $f/(u_1 v_1')$. Ignoring the constant coefficients, the second part can be written as $(v_1')^{-1}\partial f/\partial u_1$. Considering that $u = g(v)$, we have $\int uv'\,dx = \int g(v)v'\,dx = \int g(v)\,dv$ (remember that g is chosen to be easy to integrate). Therefore,

$$\boxed{G_0 = \sum_{i=1}^{N} \left(1 + \int g_i(v)\,dv\right)_{v \leftarrow v_i} \left(1 + v_i^{-1}\frac{\partial f}{\partial u_i}\right).} \tag{2}$$

In $\left(1 + \int g_i(v)\,dv\right)$, 1 represents the constant of integration. Next, we generate $G_1, G_2, \cdots$ by repetitive differentiation. In addition, we introduce the powers of x into the results by integration-by-parts,

$$\int f\,dx = \int (x)' f\,dx = xf - \int xf'\,dx\,. \tag{3}$$

Putting all together,

$$\boxed{G_{l+1} = (1+x)\,(\mathbf{1} + \mathcal{D}_x)\,G_l\,,} \tag{4}$$

where $\mathcal{D}_x$ is the total derivative operator. Converting integrands to a form compatible with Eq. 1 and finding $\int g_i(v)\,dv$ in Eq. 2 are facilitated by the term-rewriting and rule definition functionality of **JuliaSymbolics** [2]. Nearly fifty rules are sufficient to cover the integration of elementary functions (exponential, logarithmic, trigonometric, hyperbolic, and their inverses).

2.2 Numerical Computations

After the symbolic part provides a set of n candidate terms $T_l = \{\theta_k\}$, we generate n random numbers x_i (test points) in $\mathbb{D}_d$, an open disk of radius d centered at the origin (d is a parameter provided by the user). Using x_is, the algorithm creates an n-by-n matrix $\mathbf{A} = (a_{ij})$ and an n-element vector $\mathbf{b} = (b_i)$ filled, respectively, with the values of the derivatives of the candidate terms and the input function at the test points, i.e., $a_{ij} \leftarrow \frac{d\theta_j}{dx}(x_i)$ and $b_i \leftarrow f(x_i)$.

As discussed above, a potential complication at this stage is that the columns of $\mathbf{A}$, corresponding to different candidate terms, may be linearly dependent. We remedy this problem by utilizing the pivoted QR algorithm. In short, $\mathbf{A}$ is decomposed into $\mathbf{A} = \mathbf{QRP}^T$, where $\mathbf{Q}$ is an orthogonal rotation matrix, $\mathbf{R}$ is an upper triangular matrix, and $\mathbf{P}$ is the permutation matrix. We locate the small absolute values on the diagonal of $\mathbf{R}$ to find $P \subset \{1, 2, \ldots, n\}$, such that the columns of $\mathbf{A}$ not in P form a maximally linearly independent set. Then, for each $i \in P$, we remove the ith row and column from $\mathbf{A}$, the ith row from $\mathbf{b}$, and the ith element from T_l.

Using full-rank $\mathbf{A}$ and $\mathbf{b}$, we find $\mathbf{q}$ such that $\mathbf{Aq} = \mathbf{b}$. If $\mathbf{A}$ is low dimensional, this can be done simply as $\mathbf{q} = \mathbf{A}^{-1}\mathbf{b}$ (by construction, $\mathbf{A}$ is a square matrix). However, this process has the drawback of tending to use all the terms, even those with numerically small coefficients, which obscures the results and differs from the expected answer to integration problems. We need a parsimonious (sparse) model such that $\mathbf{q}$ has the minimum number of non-zero elements while still solves $\mathbf{Aq} = \mathbf{b}$ with an acceptable accuracy. We can achieve this by recasting the problem as an optimization problem to solve

$$\min_q \|\mathbf{Aq} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{q}\|_i\,, \tag{5}$$

for $\mathbf{q}$, where $i \in \{1, 2\}$, and λ is a regularization parameter. In this paper, we use the sequential thresholded least-squares (STLSQ) algorithm, which is a component of the Sparse Identification of Nonlinear

Dynamics (SINDy) technique [1] and uses ℓ_2-norm. This method has been chosen due to robust behavior within the scope of many problems related to SINDy. The sparse regression code is provided by **DataDrivenDiffEq.jl** [5].

Finally, we put everything together and generate $y = \sum_j q_j \theta_j$, the anti-derivative of the input.

3 Results

SymbolicNumericIntegration.jl is available at

https://github.com/SciML/SymbolicNumericIntegration.jl.

It can correctly solve 670 out of 937 test integrals from the RUBI test suite based on classic calculus textbooks.

We showcase the strengths and weaknesses of the symbolic-numeric integration algorithm with following examples. The algorithm can successfully solve the following integral, $\int \frac{\log x}{x\sqrt{1+\log x}}\,dx = \frac{2}{3}x\log x\sqrt{1+\log x} - \frac{4}{3}\sqrt{1+\log x}$. The reason for the success is that the implicit substitution $u = 1 + \log x$ transforms the integral to easily solvable $\int (u-1)/\sqrt{u}\,du$. On the other hand, the algorithm fails to solve the following simple integral, $\int \frac{1}{1+2\cos x}\,dx$, where no simple substitution can solve the integral (it requires the tangent of half-angle trick).

Because our method is related to the Risch-Norman method, it is particularly effective in dealing with expressions composed of exps and logs. For example, it can solve the following test example, $\int \frac{\exp\left(\frac{1}{\log x}\right)\left((\log x)^2-1\right)}{(\log x)^2}\,dx = x\exp\left(\frac{1}{\log x}\right)$. The candidate generation algorithm does not automatically transform the transcendental functions into logarithmic and exponential functions; therefore, the final result is more readable and consistent with the user's expectations. For example, compare the output of the recursive Risch algorithm, $\int e^x\cos 2x = \left(\frac{1}{10} - \frac{1}{5}i\right)e^x e^{2ix} + \left(\frac{1}{10} + \frac{1}{5}i\right)e^x e^{-2ix}$ to our method, $\int e^x\cos 2x = \frac{1}{5}e^x\cos 2x + \frac{2}{5}e^x\sin 2x$.

The algorithm occasionally fails to solve an integral due to numerical accuracy issues and round-off errors. For example, $\int x^3\sin^3 x\,dx$ is solved correctly, but not $\int x^3\sin^4 x\,dx$, even though the symbolic part of the algorithm generates all the required candidate terms. The root cause of the problem is that many candidate terms are approximately linearly dependent while not marked as such by the QR algorithm. Therefore, the resulting matrix $\mathbf{A}$ becomes ill-conditioned.

References

[1] Brunton, S. L., Proctor, J. L., Kutz, J. N., and Bialek, W. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America 113*, 15 (apr 2016), 3932–3937.

[2] Cheli, A., and Rackauckas, C. Automated code optimization with e-graphs, 2021.

[3] Deconinck, B., and Nivala, M. Symbolic integration using homotopy methods. *Mathematics and Computers in Simulation 80*, 4 (dec 2009), 825–836.

[4] Geddes, K. O., and Stefanus, L. Y. On the risch-norman integration method and its implementation in maple. *Proceedings of the International Symposium on Symbolic and Algebraic Computation, ISSAC Part F130182* (7 1989), 212–217.

[5] Martensen, J., Rackauckas, C., et al. Datadrivendiffeq.jl, July 2021.