Holonomic gradient method — a symbolic-numeric method to evaluate integrals with parameters

Nobuki Takayama[1][0000−0002−0061−514X], Takaharu Yaguchi[2][0000−0001−9025−6015], and Yi Zhang[3][0000−0001−5844−5107]

[1] Department of Mathematics, Kobe University, Kobe 657-8501, Japan, takayama@math.kobe-u.ac.jp
[2] Institute of Mathematics for Industry, Kyushu University, Fukuoka 819-0395, Japan
yaguchi@imi.kyushu-u.ac.jp
[3] Department of Applied Mathematics, Xi'an Jiaotong-Liverpool University, Suzhou 215123, China
Yi.Zhang03@xjtlu.edu.cn

Abstract. ODEs obtained via computational algebra, such as through the creative telescoping method or D-module integration, are often large in size, and most ODE solvers do not work well especially when significant figures of generalized boundary values are about 3 digits. We propose an effective method for performing this kind of numerical analysis and implement it on several computer algebra systems with an assistance of AI.

Keywords: holonomic gradient method · ODE solvers · AI assisted programming

1 Holonomic Gradient Method

The holonomic gradient method[4] (HGM) is a method for evaluating approximate values of parameterized multiple integrals through the following 3 steps:

1. Derive a holonomic system of linear partial differential equations (holonomic system) satisfied by the multiple integral with respect to its parameters, using a computational algebra algorithm.
2. Numerically compute approximate values of the multiple integral at a few parameter values.
3. Convert the holonomic system obtained via computational algebra into an ordinary differential equation (ODE), and numerically compute approximate values of the multiple integral for all parameter values using numerical solvers for ODEs.

Example 1. Consider an unnormalized probability distribution $\exp(tx - x^3)$ on $x \in [0, \infty)$ with parameter t. The normalizing constant of this distribution is the definite integral with the parameter t

$$Z(t) = \int_0^{+\infty} \exp(tx - x^3)dx.$$

It follows from $(3\partial_t^2 - t)\bullet\exp(tx - x^3) = -\partial_x \bullet \exp(tx - x^3)$ that $Z(t)$ satisfies the ODE $(3\partial_t^2 - t)\bullet Z(t) = 1$ where $\partial_t = \frac{d}{dt}$ (see, e.g., [18, Chap 6] as an introductory exposition). Determining the value of the normalization constant $Z(t)$ is necessary for solving various statistical problems.

[4] For a survey, see [30]

ODE $L \bullet f = 0$ obtained via computational algebra, such as through the creative telescoping method or D-module integration (see, e.g., references in [30]), are often large in size, and standard methods like the Runge-Kutta method frequently fail to work well for Step 3. This problem is also a concern in the field of computational physics, with research such as AMFlow [20] being conducted. The following sections discuss numerical methods suitable for Step 3 and propose an approach in Section 3.

2 Numerical Solvers for Differential Equations for Holonomic Gradient Method in Statistics

Various algorithms and implementations have been researched and provided for solving linear ordinary differential equations. Our arXiv preprint [31] compares several methods, primarily from the perspective of their use in the third step of HGM. We list only the methods we compared and their references in this paper because of the page limitation. For more details, please refer to the preprint.

In the HGM, the function $Z(t)$ to evaluate has an integral representation and satisfies a linear ODE $L \bullet f = b, \quad L = \sum_{k=0}^{r} c_k(t)\partial_t^k,\ \partial_t = \frac{d}{dt}$ where $b(t)$ is an inhomogeneous term and $f(t)$ is an unknown function.

The differential equation might have a solution u which dominates Z at $+\infty$. In other words, there might be a solution u such that $|u/Z| \to \infty$ when $t \to +\infty$. In such cases, it is not easy to obtain numerical values of $Z(t)$ globally by solving the ODE even when the almost accurate initial values of $Z(t), Z'(t), \ldots$ are known. In HGM, it is often necessary to find subdominant numerical solutions over as wide a range as possible.

Here are methods we tried in [31]. (A) The Runge-Kutta method (see, e.g., [15, Chapter 2]). (B) The implicit Runge-Kutta method (see, e.g., [15, Chapters 2 and 4], [25], [33]). (C) Multi-step methods (see, e.g., [15, Chapters 3 and 5]). (D) The spectral method in the approximation theory (see, e.g., [2], [5], [7], [24], [32]). (E) Defusing method (filter method, restricting the initial value vector to a sub linear space) and the discrete QR method, the continuous orthonormalization, the Riccati transformation [1], [8], [3], [9], [10], [11]. (F) Holonomic sparse interpolations/extrapolations by ODE: solving a generalized boundary value problem by solving an optimization problem (see, e.g., [3], [4], [6] and the Section 3).

The robustness for input errors is an important point in our comparison, because initial values or generalized boundary values [5] might be evaluated by Monte-Carlo integrators in HGM. The last two methods of the defusing method and holonomic sparse interpolation/extrapolation method are expected to be used mainly under this situation of the HGM.

3 Holonomic Sparse Interpolation/Extrapolation Methods

We found the Holonomic Sparse Interpolation/Extrapolation Method B in [31] (Holonomic SIE method B or HIE method B, abbreviated notation) particularly useful for analyzing large ODEs appearing in HGM using low-precision generalized boundary values obtained from Monte Carlo simulations. This method is a type of the least squares spectral method (LSSM) (see, e.g., [3],[4], [6]). The difference from conventional LSSM is that it utilizes rigorous rational number calculations or big floats as preprocessing steps for analyzing large ODEs and uses "soft constraints" in an

[5] which are also called data points or values at collocation points [3]

optimization step to avoid to overfit to noisy data. We name Algorithm 1 holonomic SIE method B to emphasize these differences.

We solve the ODE $L \bullet f = b$ of rank r when (approximate) values of $f(t) = Z(t)$ at $t = p_1, p_2, \ldots, p_{r'}$ are known. We call the points $(p_i, q_i), q_i = f(p_i)$ *data points* or *generalized boundary values*. In other words, we want to interpolate or extrapolate values of f from these r' values. The number of data points r' may be more than the rank r.

We minimize

$$\int_{t_s}^{t_e} |L \bullet f(t) - b(t)|^2 d\mu(t) \tag{1}$$

where $d\mu(t)$ is a measure in the t space. We approximate this integral by a numerical integration scheme. The scheme for a function g can be expressed as

$$I_N(g) = \sum_{j=0}^{N} T_j g(t_j) \tag{2}$$

where $t_0 = t_2 < t_1 < \cdots < t_{N-1} < t_N = t_e$ and weight $T_j \in \mathbf{R}_{\geq 0}$. We fix a numerical integration method. For example, the trapezoidal method can be expressed as $h = \frac{t_e - t_s}{N}$, $t_i = t_s + hi$, $T_j = h$ for $1 \leq j < N$ and $T_0 = T_N = h/2$. We approximately expand the solution f by given basis functions $\{e_k(t)\}$, $k = 0, 1, \ldots, M$ as

$$f(t) = \sum_{k=0}^{M} f_k e_k(t), \quad f_k \in \mathbf{R}. \tag{3}$$

Put this expression into $L \bullet f = b$. We minimize the following loss function, which is approximately equal to (1),

$$\begin{aligned} \ell(\{f_k\}) &= \sum_{j=0}^{N} |(Lf)(t_j) - b(t_j)|^2 T_j \\ &= \sum_{j=0}^{N} \left| \sqrt{T_j} \sum_{k=0}^{M} f_k (Le_k)(t_j) - \sqrt{T_j} b(t_j) \right|^2 \end{aligned} \tag{4}$$

under the constraints[6] at data points

$$\sum_{k=0}^{M} f_k \cdot e_k(p_i) = q_i, \quad i = 1, 2, \ldots, r'. \tag{5}$$

This is a least mean square problem under constraints (see, e.g., [23]). Since the loss function (4) is defined by the numerical integration formula (2), we have the following estimate.

Lemma 1. *The norm* $\int_{t_s}^{t_e} |Lf(t) - b(t)|^2 d\mu(t)$ *is bounded by* $\ell(\{f_k\}) + e_N(|Lf - b|^2)$ *where* $e_N(|Lf - b|^2)$ *is the error of the numerical integration method.*

Solvers of the least mean square problem output the value of the loss function, then we can estimate the L^2 norm of $Lf - b$ by this Lemma where f is assumed to be (3).

[6] If data are given in derivative values of f, these constraints become relations among q_i's.

We can also consider a least mean square problem with no constraints by the loss function

$$\tilde{\ell}(\{f_k\}) = \alpha\ell(\{f_k\}) + \beta\sum_{i=1}^{r}\left(\sum_{k=0}^{M} f_k \cdot e_k(p_i) - q_i\right)^2 + \gamma\sum_{i=0}^{N} f_i^2. \tag{6}$$

Here α, β, γ are hyperparameters for the optimization. These hyperparameters should be adjusted according to the magnitude of the error. For example, if the error in the calculated integral value is large, the hyperparameter β should be reduced to mitigate the effect of the error. The hyperparameter $\gamma > 0$ is used to avoid an overfitting. While standard Least-Squares Spectral Methods (LSSM) typically treat boundary conditions as strict constraints (setting $\beta \gg \alpha$), our generalized boundary values derived from Monte Carlo simulations inherently contain noise. Therefore, our formulation essentially adopts a machine-learning-inspired optimization framework similar to Physics-Informed Neural Networks (PINNs). By introducing hyperparameters and intentionally setting β to an extremely small value (soft constraints) alongside Tikhonov regularization γ, we prevent high-degree polynomial basis from overfitting to the noisy data, successfully suppressing parasitic solutions.

Let us summarize the procedure of the holonomic SIE method B.

Algorithm 1 *Input: data points (p_i, q_i) (generalized boundary values), basis $\{e_k\}$, numerical integration scheme (2), integration domain $[t_s, t_e]$, hyperparameters $\alpha \geq \beta, \gamma$.*
Output: Approximation of f of the form (3).
Step 1: Evaluate numerically $(L \bullet e_k)(t_j)$'s, $b(t_j)$'s, $\sqrt{T_j}$, $e_k(p_i)$'s by rational approximation of $t_j, \sqrt{T_j}, (p_i, q_i)$ and by exact rational arithmetics or big floats (Computer algebra part).
Step 2: Solve the least square problem (6) by transforming it into the form $\|Af - B\|^2$ where A is a matrix, B is a vector, and $f = (f_0, \ldots, f_M)^T$ (the accuracy of A and B can be reduced to the accuracy required for normal numerical analysis).

Example 2. (Continuation of Example 1) We solve the ODE $\partial_t(3\partial_t^2 - t) \bullet Z(t) = 0$ with the input data: $(p_1, q_1) = (-20, 124907/2500000)$, $(p_2, q_2) = (-4, 117521/500000)$, basis=$\{$Chebychev polynomials upto degree 29$\}$, Chebychev quadratur integration scheme, $[t_s, t_e] = [-20, 6]$, $\alpha = 1, \beta = 1, \gamma = 0$. The output graph is the left one in Figure 1. Note that in this example, increasing the approximation order (using Chebychev polynomials upto degree 79) will result in a parasitic solution that is highly sensitive to errors in the initial values with 2 data points. See the right one in Figure 1. Note that 3 data points suppress this parasitic solution.

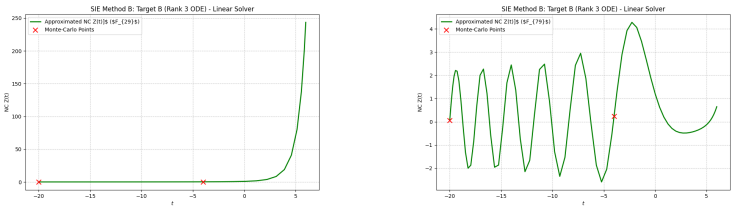

Fig. 1. Left: Graph of $Z(t)$. Right: Parasitic solution by too higher-order approximation and a few data data points with very small errors

Example 3. We consider the integral studied in [29]

$$\begin{aligned}\mathrm{E}[\chi(M_t)] &= F(s_1, s_2, m_{11}, m_{21}, m_{22}; t)\\ &= \frac{1}{2}\int_t^{\infty} d\sigma \int_{-\infty}^{\infty} db \int_0^{2\pi} d\theta \int_0^{2\pi} d\phi(\sigma^2 - b^2)\frac{s_1 s_2}{(2\pi)^2}\exp\Big\{-\frac{1}{2}R\Big\}, \end{aligned} \tag{7}$$

where

$$\begin{aligned}R = s_1\left(b\sin\theta\sin\phi + \sigma\cos\theta\cos\phi - m_{11}\right)^2 + s_2\left(\sigma\sin\theta\cos\phi - b\cos\theta\sin\phi - m_{21}\right)^2\\ +s_1\left(\sigma\cos\theta\sin\phi - b\sin\theta\cos\phi\right)^2 + s_2\left(b\cos\theta\cos\phi + \sigma\sin\theta\sin\phi - m_{22}\right)^2.\end{aligned}$$

and $m_{11} = 1, m_{21} = 2, m_{22} = 3, s_1 = 10^3, s_2 = 10^2$. By virtue of the Euler characteristic method, the expectation $E[\chi(M_t)]$ of the Euler characteristic of a random manifold M_t approximates the probability that the maximal eigenvalue of 2×2 random matrices is less than t. See [29, Sec 3] for details.

The function $E[\chi(M_t)]$ is a solution of the rank 11 ODE $L \bullet f = 0$ discussed in [29, Example 5]. The operator L is of the form $\left((-4.72 \times 10^{-52}t^{29} + \cdots)\partial_t^{10} + \cdots + (-7.78 \times 10^{-22}t^{35} + \cdots)\right)\partial_t$ by multiplying a constant $10^{-31}/8.66$ [7] and ∂_t from the right to the operator given in `https://yzhang1616.github.io/ec1/ec1.html` (25 kbytes), which is derived by the creative telescoping method. This system will be solved by holonomic SIE method B in the next section.

Note that the holonomic SIE method B gives the Chebyshev function method [2], [32] to solve ODEs when we use Gauss-Chebychev equadrature integration scheme and Chebychev polynomials as basis functions [31]. The SIE method is more robust to errors in generalized boundary values by solving an optimization problem that includes hyperparameters. We also note that this method can also be applied to systems of first-order ODEs that restrict the Pfaffian system associated with a holonomic system.

4 Multi-language Implementation of a holonomic SIE method B Solver for the HGM and Comparison of Design Philosophies in Computer Algebra Systems

We implemented "holonomic SIE Method B" (Algorithm 1) on several computer algebra systems with the assistance of Gemini 3.1 Pro [12]. Gemini 3.1 Pro is a system strong in mathematics and coding. We deployed this method across five mathematical software systems: starting with a hybrid Risa/Asir [26] and Python (SciPy) approach, followed by SageMath [27], Mathematica [22], Julia/OSCAR [19], and Maple [21]. All implementations are obtainable from [14]. See them for detailed setting of input data and hyperparameters.

The new programming paradigm, where humans strictly define the mathematical essence (What/Why) of an algorithm and rely on AI assistance, can significantly reduce development time. During the porting of the identical mathematical algorithm to different systems, the unique design philosophies of each system such as their handling of catastrophic cancellation and significance arithmetic were brought to light. We compare and discuss these technical barriers and our approaches to overcoming them.

[7] This constant is chosen so that the maximal absolute value of the coefficients of f_k's in (4) is 1.

Ordinary differential equations (ODEs, $Lf = 0, \ldots$) derived in HGM often become high-order singular perturbation equations that not only have massive polynomial coefficients but also possess numerous singular points. However, because the solutions in HGM represent statistical quantities, they are often smooth functions. SIE Method B was devised with the aim of functioning effectively even in such scenarios.

In this implementation, we use Chebyshev polynomials as the basis and approximate the integration using Chebyshev-Gaussian quadrature to approximately express $\int_{t_s}^{t_e} |L\sum_{k=0}^{M} f_k e_k(t)|^2 d\mu(t)$ as a product of matrices where $\{e_k\}$ is a basis to express the solution f. Numerical values, which we call generalized boundary values, of $f(t)$ at a few interpolation/extrapolation points t obtained from Monte Carlo simulations and the product of matrices are then used to solve a constrained least squares problem. A major advantage of this method is its flexibility, allowing the basis and quadrature approximation methods to be modified according to the specific problem.

4.1 Implementation with Risa/Asir and Python

Since Risa/Asir is a computer algebra system we have developed from scratch and we know the details of the system characteristics, we first implemented the algorithm with the assistance of gem "Risa/Asir programming assistant v7"[8] [13] of Gemini.

In this implementation, the algorithm is presented to the gem, and the roles of Risa/Asir and Python are instructed as follows:

1. Risa/Asir approximates the generalized boundary conditions (p_i, q_i), Chebyshev points t_j, and integral weights w_j with rational numbers, and performs all calculations using polynomials with rational coefficients.
2. The basis $\{e_k\}$ consists of the Chebychev polynomials up to the degree 29.
3. Data w_j, t_j, $(Le_k)(t_j)$, p_i, q_i for constructing the matrix for the least squares problem are calculated as rational numbers, and the data passed to Python is converted from these rational numbers to 64-bit floating-point numbers using `number_eval`.
4. Python should use the GPU if available for calculations.

When the implementation using gem was executed, the system Risa/Asir worked without problems, but an issue arose where the least squares problem in Python could not be solved correctly. Upon examining the matrix elements, it was found to be a matrix with entries of approximately 10^{135}. Normalizing the columns did not change the answer to the least squares problem, so column normalization allowed the problem to be solved correctly. It took 2.42 seconds[9] on Risa/Asir to generate data for Python. It took 1.43 seconds to execute the Python code without Cupy.

4.2 Hyperparameters and Soft Constraints

In this initial implementation (Risa/Asir + Python), we observed that the high-degree polynomial basis overfitted the generalized boundary conditions derived from noisy simulation data, causing severe Runge's phenomenon. To prevent this, we set the constraint weight β in the loss function to an extremely small value ($\beta = 10^{-13}$). By introducing these "soft constraints," the simulation data acts merely as an anchor to choose a relevant solution, thereby achieving a smooth interpolation/extrapolation that strictly obeys the laws of the ODE (Figure 2, left).

[8] a custom persona configured for Risa/Asir in Gemimi

[9] on Intel(R) Core(TM) i7-10700K CPU @ 3.80GHz (3.80 GHz), 16GB memory

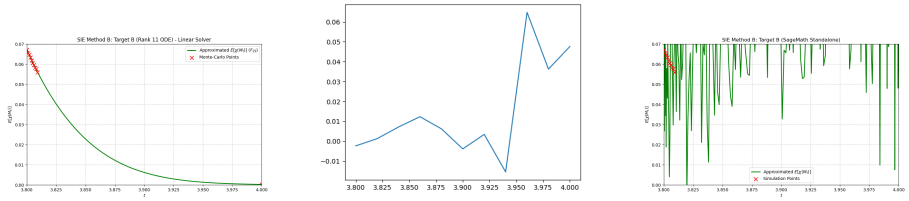

Fig. 2. Left: solution. Middle: relative errors with simulation values with adding data at $x = 4$. Right: catastrophic cancellation.

The least squares problem is solved by `scipy.optimize.least_squares` [28] or GPU-accelerated `cupy.linalg.lstsq` depending on the matrix condition. The SVD method for the least squares problem is known to operate stably even for ill-conditioned matrices.

4.3 Comparison of Implementation and Design Philosophy in Each Mathematical Software System

Now that the implementations in Risa/Asir and Python were successful, we instructed Gemini to implement them in SageMath, Mathematica, Julia/OSCAR, and Maple. Gemini suggested that these systems implement efficient least-squares problem solvers, and therefore, we should create closed implementations using these systems. Following this suggestion, Gemini implemented the solutions, and interestingly, the unique features of each system's functions for symbolic and numerical fusion computation became apparent. This is reported below.

SageMath: Overcoming Catastrophic Cancellation During the porting to SageMath, floating-point numbers were introduced in the generation process of the Chebyshev basis, leading to severe catastrophic cancellation when expanding polynomials to the 29th degree. Consequently, while the cost function approached almost zero `4.4008e-45`, the resulting graph became pure noise (right graph in Figure 2). This issue was resolved by strictly defining the evaluation points and intervals as rational numbers (`QQ`), explicitly casting them to a 500-bit precision real field (`RealField(500)`) during polynomial evaluation, and utilizing Horner's method for fast, high-precision substitution.

Mathematica: The Wall of Significance Arithmetic In Mathematica, due to careless programming, we encounter "contamination" where the inclusion of a single floating-point number (e.g., `1.0`) in matrix elements downgraded the entire computation to machine precision. Furthermore, when performing Singular Value Decomposition (SVD) using arbitrary-precision arithmetic on an ill-conditioned matrix, the system automatically detected the loss of significant digits, truncating the precision of the result from 150 digits to 110 digits. Consequently, the `Plot` function halted rendering, stating it could not meet the requested drawing precision (`Plot::precw` error). This was bypassed by using the `Rationalize` function, forcefully converting the obtained decimals into exact fractions with infinite precision.

Julia / OSCAR: Strict Type Systems and the Benefits of JIT Compilation In OSCAR, a computer algebra system built on Julia, implicit type conversion between the field of rational numbers (`QQFieldElem`) and floating-point numbers (`BigFloat`) is strictly prohibited to maintain mathematical rigor (`InexactError`). Therefore, we implemented a bridging process that individually extracts the `numerator` and `denominator` of a rational number and explicitly reconstructs it as a `BigFloat` fraction. Once the barrier of the type system was overcome, Julia's JIT compilation and powerful LAPACK backend achieved the fastest execution time 1.75 seconds among all tested systems.

Maple: Traps in Symbolic Computation and the Tolerance of the Normal Equations In Maple, values like $\sqrt{10^{-13}}$ were maintained as exact symbolic expressions rather than floating-point approximations, leading to rejection when inserted into a floating-point matrix type (`datatype=float`). This was resolved through explicit numerical evaluation using the `evalf` function. Additionally, due to handling ill-conditioned matrices under an extremely high precision of 150 digits, the off-diagonal elements in the SVD iterative method failed to converge. To address this, we switched to directly solving the normal equations for least squares regressions, which is a method typically avoided due to error amplification. The absolute precision of 150 digits safely absorbs the severe error amplification usually associated with the normal equations.

4.4 Conclusion

The new programming paradigm—where humans strictly define the mathematical essence (What/Why) of an algorithm, and rely on AI assistance to absorb and translate into the dialects and design philosophies (How) of each CAS—functioned exceptionally well. We demonstrated that even when evaluating the same mathematical formula, the "philosophy" of where to draw the line between exactness and numerical approximation differs entirely among systems. This serves as a highly significant case study for comparative research in mathematical software. Ultimately, the authors also believe that this collaborative process with AI will become a standard practice in developing mathematical software systems.

Acknowledgments. The first author is supported by Kakenhi ???. The second author is supported by JST CREST JPMJCR24Q5 (JGN). The last author is supported by the NSFC grant No. 12371520.

References

1. F. S. Acton, Numerical Methods that Work, 1990, Mathematical Association of America.
2. `ApproxFun.jl`, Julia package for function approximation, `https://github.com/JuliaApproximation/ApproxFun.jl`, `https://juliaapproximation.github.io/ApproxFun.jl/latest/`.
3. U.M.Ascher , R.M.M.Mattheij, R.D.Russel, Numerical Solution of Boundary Value Problems for Ordinary Differential Equations, 1987, SIAM.
4. J. P. Boyd, Chebyshev and Fourier Spectral Methods, 2nd ed., Dover Publications, 2001.
5. F.Bréhard, N.Brisebarre, M.Joldes, Validated and numerically efficient Chebyshev spectral methods for linear ordinary differential equations, ACM Transactions on Mathematical Software, 44 (2018), 1-42. `https://doi.org/10.1145/3208103`
6. C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Zang, Spectral Methods: Fundamentals in Single Domains, Springer, 2006.

7. chebfun for Matlab, https://chebfun.org
8. S. D. Conte, The numerical solution of linear boundary value problems, SIAM Review 8, No. 3 (1966), 309–321.
9. A. Davey, An automatic orthonormalization method for solving stiff boundary-value problems, Journal of Computational Physics, 51 No. 2 (1983), 343–356.
10. L. Dieci, M. Osborne, R. Russell, A Riccati Transformation Method for Solving Linear BFPs I, SIAM Journal on Numerical Analysis 25 (1988), 1055–1073.
11. L.Dieci, R.D.Russell, E.S.Van Vleck, On the computation of Lyapunov exponents for continuous dynamical systems. SIAM Journal on Numerical Analysis, 31(1) (1994), 261-281.
12. Google Gemini, https://gemini.google.com
13. https://gemini.google.com/gem/1-OoG766iQlM7YbW9hAS1aYMebKOpGEKc?usp=sharing
14. https://github.com/nobuki-takayama/sie-method-b/
15. E.Hailer, S.P.Norsett, G.Wanner, Solving Ordinary Differential Equations I, II. Springer, 1993.
16. H.Hashiguchi, Y.Numata, N.Takayama, A.Takemura, Holonomic gradient method for the distribution function of the largest root of a Wishart matrix, Journal of Multivariate Analysis, 117, (2013) 296-312.
17. References for HGM, https://www.math.kobe-u.ac.jp/OpenXM/Math/hgm/ref-hgm.html
18. T. Hibi and et al, Gröbner Bases: Statistics and Software Systems, 2013, Springer
19. Julia/OSCAR, https://oscar-system.org
20. X.Liu, Y.Q.Ma, AMFlow: a Mathematica package for Feynman integrals computation via Auxiliary Mass Flow, Computer physics communications (2023) 108565 https://doi.org/https://doi.org/10.1016/j.cpc.2022.108565
21. Maple, https://maplesoft.com
22. Mathematica, https://wolfram.com/mathematica/
23. J.Nocedal, S.Wright, Numerical Optimization, 2006, Springer.
24. S. Olver, A. Townsend, A fast and well-conditioned spectral method, SIAM Review 55 (2013), 462-489.
25. pySDC project, a Python implementation of the spectral diferred correction, https://parallel-in-time.org/pySDC/
26. Risa/Asir, https://openxm.org
27. SageMath, https://www.sagemath.org
28. scipy.optimize.least_squares, https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html
29. N.Takayama, L.Jiu, S.Kuriki, Y.Zhang, Computations of the Expected Euler Characteristic for the Largest Eigenvalue of a Real Wishart Matrix, Journal of Multivariate Analysis 179 (2020), 104642.
30. N. Takayama, Hypergeometric systems, statistics, and algorithms, Combinatorial, Computational, and Applied Algebraic Geometry (Proceedings of Symposia in Pure Mathematics, Vol. 111), AMS, 2025, pp. 113–136.
31. N.Takayama, T.Yaguchi, Y.Zhang, Comparison of Numerical Solvers for Differential Equations for Holonomic Gradient Method in Statistics, https://arxiv.org/abs/2111.10947
32. L. N. Trefethen, Approximation Theory and Approximation Practice, 2020, SIAM.
33. M.Winkel, R.Speck, D.Ruprecht, A high-order Boris integrator, Journal of Computational Physics 295 (2015), 456–474.