Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 1 (2024), pp. 183-188

http://www.ripublication.com

Enhancing the Performance of Computer Algebra Systems

Kostas Zotos

South-west University "Neofit Rilski" Faculty of Science and Mathematics
Department of Informatics
66 Ivan Michailov st. 2700 Blagoevgrad, Bulgaria
zwtos@yahoo.com

Abstract

In recent years, Computer Algebra Systems (CASs) have evolved into a crucial computational tool due to their ability to address various mathematical issues. Applications for CAS are abundant in domains where manual calculations are laborious, challenging, and prone to errors. Various programs that attempted to show that in the scientific domain, it is possible to move beyond the purely numerical area and use them to carry out a symbolic calculation first appeared in the late 1950s and early 1960s. Today's freeware symbolic tools are strong enough to serve as a workable, trustworthy substitute for pricey commercial symbolic packages. Desktop CASs are the preferred option for high-performance computational programs from businesses and professionals, even though the online versions operate with good speed and performance. In this paper, we are going to examine all these factors and see some techniques that significantly improve CAS's performance.

Keywords: *Computer Algebra Systems; CAS; CAS performance; MATLAB; Maple; Mathematica*

INTRODUCTION

Computer Algebra's two main objectives are: Firstly, to give the algorithms to perform operations on algebraic structures such as fields, vector spaces, rings, ideals, and modules. Secondly, employ the algorithms and how they are applied to resolve theoretical and practical mathematical issues.

Symbolic Algebra System (SAS), also known as Computer Algebra System (CAS), is a package that consists of a language for implementing algebraic objects, an

environment to use the language in, and mathematical algorithms and special functions for performing symbolic manipulations [1]. There are many different types of Computer Algebra Systems available today. Maple, Mathematics, and MATLAB are the most widely used and well-liked commercial systems.

In many ways, symbolic and hybrid symbolic-numeric approaches can be far better than strictly numerical ones. The benefits of the symbolic approach are:

(i) Offering an analytical solution that characterizes every potential solution for a particular control problem,
(ii) Enhancing parametric design and analysis.

It is a natural desire to use a computer to symbolically carry out a mathematical computation whenever a laborious and drawn-out series of manipulations is needed. When performing complex calculations, mathematicians, engineers, and computer scientists frequently turn to CASs. Their preference for one over the other stems from the system's ability to solve the problem classes that these users find interesting. There are examples where CAS does not give us the correct result in a mathematical operation. This happens even in the most famous algebraic systems [2]. Therefore, we must bear in mind that the performance of these systems has some limits and is not always accurate.

In this study, we will focus on the most famous MATLAB, Mathematica, and Maple. The documentation sheets of these CASs are the source of data that we used to compare them and examine their characteristics. In the rest of the paper, we are going to see ways to improve CAS performance.

TIPS FOR ENHANCING THE CAS PERFORMANCE

In this section, we will see some tips to handle CASs in a way that maximizes performance. Most of this material is general and applies to all CASs, but some tips apply only to specific CASs. Thus, if you want to perform better, pay attention to these guidelines:

If the type of data changes, create new variables. Steer clear of code that creates variables. Steer clear of global variables. Local functions are preferable to nested functions. Rather than using scripts, use functions. It’s a good practice to avoid loop-based code.

According to Toint (2021), bound-constrained mixed-integer derivative-free optimization algorithms have software implementations. Some of these implementations are also capable of handling constraints. They can handle both continuous and Integer variables. The most well-known are DAKOTA solvers (open-source and written in C++), DFL solvers, and Brute Force Optimizer (BFO), an open-source MATLAB implementation for nonlinear bound-constrained derivative-free optimization and equilibrium calculations with continuous and discrete variables [3].

Benchmarking, or comparing, mathematical algorithms is a challenging process that requires careful consideration of numerous subtle factors to produce an objective and

fair assessment. The first step in creating a suitable experimental design is to define the questions that need to be addressed. This entails choosing appropriate performance metrics and a test set following the study's goals. Transparency, justice, and thoroughness are required in the analysis and processing of the data [4].

With AI, CASs can comprehend Mathematics effectively. Recent studies have demonstrated that by employing sample problems, machine learning techniques like support vector machines can enhance the performance of CASs [5]. Furthermore, new understandings of symbolic computation gained from explainable AI techniques can lead to innovative applications of CAS.

Use sparse arrays when appropriate (they can reduce computation time and require less memory). A sparse array is one in which the default value is null, or zero, and most of the elements have the same value so, in a sparse array, we only need to store the non-zero/null elements.

It is relatively simple to parallelize even complex algebraic algorithms using a CAS, as demonstrated by implementing various parallel programming paradigms. Multivariate nonlinear equation systems solving algorithms are implemented on different parallel architectures. For instance, Siegel (1993) successfully parallelized a simple solution to the intricate and significant problem of real root isolation, achieving a five-fold increase in speed over the original sequential Maple source [6].

According to Mantsika Matooane's (2002) thesis, memory space is an essential resource for parallel CAS. It is commonly known that memory overload is the primary cause of Computer Algebra System failures [7]. For example, some of the best algorithms currently in use for several Computer Algebra problems suffer from intermediate expression swelling, where the final result is of a reasonable size, but the intermediate calculation faces severe memory limitations.

The time complexity of the mathematical algorithms can be reduced with hashing, memoization, or memoisation (an optimization technique that avoids redundant computations), sorting, dynamic programming, and greedy algorithms.

Numerous smaller CASs have been created by lone investigators or small groups of researchers at academic institutions. Several certain mathematical computations are far more effective than large systems (for example, at large polynomials and matrices). Such systems are Fermat, MuPAD, Singular, CoCoA, GeoGebra, WolframAlpha, SageMath, Maxima, Scilab, Octave, R, FreeMat, Demetra+, and Pari-Gp [8]. Also, some simplifiers of small CASs have better execution times rather than famous Algebra systems (for example Nemo). Short mathematical expressions take very little time to parse, but their simplification—finding polynomial GCDs, for example—requires large computational resources because of the high powers of subexpressions. Not every simplifier completes the same mathematical expression in a fair amount of time. Symbolica (lib) and Maple simplifiers have the best times with the Mathematica simplifier being approximately 15 times slower [9].

Scott Frame and John W. Coffey (2014) compared functional and imperative programming techniques for mathematical software the results show that although the

functional approach offers highly expressive features that could make software development easier, functional programming languages are probably unsuitable for most mathematically intensive applications due to their performance issues [10]. Python and Julia are two very flexible programming languages that let you do both symbolic and numerical calculations and are much used for this purpose by students [11]. Both have strengths and weaknesses and are strong. Julia's speed is one of its main advantages; large datasets and complex calculations can become time-consuming in parallel computing and therefore the speed matters a lot. Code written in Python is more productive but slower, although its runtime is lighter. Additionally, Python's library selection makes it a better choice for data analysis and machine learning. On the other hand, when performing computationally demanding tasks like statistical computing Julia might be a better option for a developer [12],[13].

RESULTS

Large-scale numerical computations may result in excessive demands on the memory and processing capacity of computers. Several configurations can be made to enhance desktop CAS performance, including determining whether background processes use system resources, raising RAM, avoiding loop-based code, positioning independent operations outside of loops, and avoiding overloading functions (particularly built-in functions). The time complexity of the mathematical algorithms can be reduced with hashing, memoization, or memoisation, sorting, dynamic programming, and greedy algorithms.

Today's freeware symbolic tools are strong enough to serve as a workable, trustworthy substitute for pricey commercial symbolic packages. Python and Julia are two very flexible programming languages that let you do both symbolic and numerical calculations and are much used for this purpose by mathematicians.

In this paper, we have seen some techniques that significantly improve the performance of CASs. The measurements demonstrate a remarkable performance improvement when some of the earlier techniques are used [14]. Using effective built-in data structures (such as packed arrays or sparse arrays, which have far wider uses than one might assume based on their stated primary function) gives us better execution time, and the use of Mathematica built-in functions can sometimes make the code run four times faster [15],[16].

CONCLUSIONS

Mathematicians should be aware of the most recent developments in the Sciences, and scientists should have access to the most advanced mathematical software. The connections between Mathematics and the Sciences are especially crucial. A fundamental element of Science, Mathematics is a part of its structure, its universal language, and its invaluable repository of knowledge. In turn, Science stimulates and inspires Mathematics by raising fresh issues and generating novel ideas.

Computer Algebra Systems are being extensively used. However, there is still room for improvement. The CAS development is continuously receiving new tools and technologies. Over the coming years, Artificial Intelligence (AI) will become more and more significant. We hope that shortly the new versions of CASs will be even faster and more efficient. Future developments in CAS are anticipated to be influenced by emerging technologies such as artificial neural networks. There's only one thing that is certain in the uncertain future of CAS: the trend toward web-based applications will not stop.

REFERENCES

[1] **S. Rachev, M. Racheva, A. Andreev and D. Ganchev** (2021). *Mathematical software tools applicable to remote learning and scientific research in case of isolation*. INTERNATIONAL SCIENTIFIC JOURNAL "MATHEMATICAL MODELING" YEAR V, ISSUE 1, P.P. 8-12 (2021).

[2] **Antonio J. Duran, M. Perez and Juan L. Varona** (2014). The misfortunes of a trio of mathematicians using Computer Algebra Systems. Can we trust in them? Notices Amer. Math. Soc. 61 (2014), 1249-1252.

[3] **Toint, P.L., Porcelli**, M.: BFO—brute-force optimizer (current as of 15 March 2021). https://sites.google.com/site/bfocode/home

[4] **V. Beiranvand, W. Hare, and Y. Lucet** (2017). Best Practices for Comparing Optimization Algorithms. Optim Eng (2017) 18:815–848, DOI 10.1007/s11081-017-9366

[5] **L. A. Toro-Carvajal, F. N. Jiménez-García, H. H. Ortíz-Álvarez, J. de Jesús Agudelo-Calle** (2012). *Los sistemas cognitivos artifi ciales en la enseñanza de la Matemática*. Educ.Educ. Vol. 15. No. 2 | Mayo-agosto de 2012 | pp. 167-183.

[6] **Kurt Siegl** (1993). *Parallelizing Algorithms for Symbolic Computation using Maple*. 4th ACM 0.89791-589.5/93/0005/0179

[7] **Mantsika Matooane** (2002). *Parallel systems in symbolic and algebraic computation*. UCAM-CL-TR-537 ISSN 1476-2986

[8] **Robert H. Lewis** (1999). *Comparison of polynomial-oriented computer algebra systems*. ACM SIGSAM Bulletin 34(1):24-24.

[9] **K. Mokrov, A. Smirnov and M. Zeng** (2023), "Rational Function Simplification for Integration-by-Parts Reduction and Beyond," doi:10.26089/NumMet.v24r425 [arXiv: 2304. 13418 [hep-ph]].

[10] **Scott Frame and John W. Coffey (**2014). *A Comparison of Functional and Imperative Programming Techniques for Mathematical Software Development.* Systemics, Cybernetics and Informatics Volume 12 - Number 2 - Year 2014.

[11] **V. G. Martínez, L. H. Encinas, A. M. Muñoz and A. Q. Dios** (2021). *Using Free Mathematical Software in Engineering Classes*. Axioms 2021, 10, 253. https://doi.org/10.3390/axioms10040253

[12] https://hyperskill.org/blog/post/julia-vs-python-a-comparison-of-two-popular-programming-languages

[13] https://datascientest.com/en/python-vs-julia-which-is-the-best-language-for-

data-science
[14] https://www.mathworks.com/products/matlab/performance.html
[15] https://sudonull.com/post/96601-10-Tips-for-Writing-Quick-Code-in-Mathematica
[16] **S. Boragan Aruoba,** Jesus Fernandez-Villaverde (2015). A Comparison of Programming Languages in Economics. Journal of Economic Dynamics and Control Volume 58, September 2015, Pages 265-273.