Towards Formalized Matrix Analysis and Algorithms

Carl Kwan

Department of Computer Science
The University of Texas at Austin
carlkwan@cs.utexas.edu

Abstract

We discuss ongoing efforts to formalize a theory of matrices and related ideas from analysis. In particular, we outline some of the challenges and our solutions to these challenges that the community may find interesting. Some of these issues are a result of performing our formalization in a first-order logic. The motivation for this work is the verification of numerical linear algebra algorithms and programs, which are relevant to many areas of science, including artificial intelligence.

Introduction

Mathematical software, such as numerical linear algebra libraries, are used not only by programmers, but also scientists, engineers, analysts, etc. for modelling, data processing, and inference. The pervasiveness of linear algebra in the real world indicates the existence of critical systems that depend on numerical linear algebra libraries, which are susceptible to bugs and the risks associated with floating-point computations. There is a need for formal methods in the mathematical software industry.

In this extended abstract, we discuss our approach to the formalization of linear algebra concepts, such as vector / matrix analysis and algorithms, possible applications, and our solutions to the challenges of performing such work in a first-order logic. We use ACL2(r), a version of the ACL2 first-order logic automated theorem prover with support for complex and real numbers via non-standard analysis (Gamboa and Kaufmann 2001). The advantage of using ACL2(r) is that it's capacity and rewriting heuristics make it particularly capable at reasoning about large and complex models.

First, we discuss how to reason about notions of continuity in ACL2(r). Then we consider the issue of executing formalized matrix operations efficiently. Finally, we look at the use of SMT solvers to aid ACL2(r) in reasoning about ad hoc or algebraic structures.

Reasoning about Continuity

There are several facets of continuity to consider in a first-order logic formalization of matrix analysis. These considerations range from the fundamental definition of continuity itself to more subtle soundness issues associated with recognizing "small" (in the norm sense) vectors or matrices.

The classical defnition of continuity for a function $f : M_1 \to M_2$ at $x \in M_1$,

$$\forall \varepsilon > 0,\ \exists \delta > 0,\ \forall y \in M_1,$$
$$\|x - y\|_{M_1} < \delta \implies \|f(x) - f(y)\|_{M_2} < \varepsilon,$$

involves a nest of quantifiers. While ACL2(r) can support reasoning with quantifiers (via Skolem functions), this reasoning can be slow and the non-standard analysis approach is more amenable to ACL2(r) rewriting.

ACL2(r) objects are either standard or non-standard. Standard objects are those found in the vanilla version of ACL2. Non-standard objects are those that involve infinitesimals, i.e. numbers that are smaller than $1/n$ for any natural n. Thus the ACL2(r) definition of a continuous function $f : M_1 \to M_2$ at a standard x is

$$\text{ismall}\left(\|x - y\|_{M_1}\right) \implies \text{ismall}\left(\|f(x) - f(y)\|_{M_2}\right),$$

where ismall is a recognizer for infinitesimals. When $M_1 = M_2 = \mathbb{R}$, the issue of recognizing infinitesmimal inputs and outputs is not hard. For multivariate vector-valued functions, the issue is more subtle.

Typically, to show that a function is continuous involves knowledge that entries of a vector are infinitesimal. Vectors and matrices are usually represented by lists of reals and lists of real vectors, respectively. A straight-forward manner of recognizing that a vector is infinitesimal under, say, the Euclidean norm would be to recurse down the vector, recognize each entry to be infinitesimal, and show that the square root of the sum of the entries is infinitesimal. However, this is impossible in ACL2(r) because defining a non-standard recursive function can introduce unsoundness. To avoid recursion, observe that for any entry x_i of $x \in \mathbb{R}^n$,

$$|x_i|^2 \leq \max_{j=1}^{n} |x_j|^2 \leq \sum_{j=1}^{n} |x_j|^2 = \|x\|_2^2$$

and $|x_i| \geq 0$. Now if $\|x\|_2$ is infinitesimal, so is $|x_i|$; this knowledge was obtained without recursion.

An astute reader may notice the approach taken above is similar to the proof for the equivalence of vector / matrix norms. Indeed, we've formalized these theorems in such a manner. The definition of a matrix

norm is slightly trickier than vector ones because matrix norms involve supremums. In ACL2(r), this can be solved by defining a Skolem function, which are functions with an outer-most quantifier and, in ACL2(r), admit a function known as a witness for which the quantified expression holds true. For example, the witness of the Skolem function,

$$\exists x,\ x \in \mathbb{R}^n \wedge \left(\forall y,\ y \in \mathbb{R}^n \implies \frac{\|Ay\|_\infty}{\|y\|_\infty} \leq \frac{\|Ax\|_\infty}{\|x\|_\infty}\right),$$

on input A can be thought of as the "argmax" of $\|Ax\|_\infty / \|x\|_\infty$. Then the infinity norm $\|\cdot\|_\infty$ on a matrix A is simply $\|Aw(A)\|_\infty / \|w(A)\|_\infty$ where $w(A)$ is the witness of the Skolem function on A.

Of course, there are issues of existence and this Skolem function definition of a matrix norm is not very useful for computational purposes. For this particular case, we compute $\|A\|_\infty = \max_{i \in [1,n]} \|a_i\|_1$ where a_i is the i-th row of A and $\|\cdot\|_1$ is defined in the usual way in ACL2(r). The proof of their equivalence is standard, but this is a good example of balancing reasoning and computation.

Executing Formalized Matrix Operations

A common trade-off of using theorem provers is speed; with ACL2(r), this is minimized by the use of single-threaded objects (stobjs), which are data structures restricted such that only one instance ever exists. Updates to a stobj can then be made by destructively modifying only the memory locations of the stobj fields to be changed. This prevents the need for an entirely new object to be created, which is costly in both time and space. The only "bottleneck" is the speed at which ACL2(r)'s underlying language, Lisp, can destructively update memory locations.

Logically, a stobj is merely an ACL2(r) object, with ordinary ACL2(r) functions that allow the user to access and update its fields, similar to a list. Reasoning with and about stobjs is no harder (from a user perspective) than proving usual ACL2(r) theorems. However, the single-threaded properties are leveraged when execution speed is vital.

We use matrices defined by stobjs for this dual purpose. The clean applicative semantics for ACL2(r) stobjs mean that theorems about stobj-defined matrices will be as intelligible as regular ACL2(r) theorems. Furthermore, efficiency is not lost as stobjs enable fast execution speeds when evaluating operations and algorithms on explicit matrices. Thus, we can develop a fully formal numerical linear algebra library that is as fast as a widely-used programming language's ability to destructively update memory locations.

Reasoning about Algebraic Structures

In principle, the only necessary tool is ACL2(r). However, theorem provers can be challenging and time-consuming to use, especially if the theorems concern ad hoc constructions. In these cases, we leverage SMT solvers by way of Smtlink to assist in the automation of proof search. Smtlink is an ACL2-based mechanism that calls SMT solvers, which can then be used to efficiently perform intricate algebraic manipulations (Peng and Greenstreet 2018). We exploit SMT techniques for problems where the reasoning is largely "algebraic", such as those involving vector or matrix operations. For example, we use Smtlink to prove the Cauchy-Schwarz inequality over abstract vector spaces (Kwan, Peng, and Greenstreet 2020). Reasoning about objects of abstract type is a particularly appropriate use-case for SMT-based reasoning, so long as the SMT solver supports uninterpreted functions and sorts. Smtlink enables efficient reasoning for ad hoc structures without the need for developing sophisticated rewriting books.

Conclusion

The ubiquitous nature of linear algebra in science suggests a breadth of potential applications for a formalized numerical linear algebra library. We have used our ACL2(r) theory of vector analysis to mechanize theorems involving convex functions (Kwan and Greenstreet 2018), and matrices in ACL2(r) are currently being used to model rapid single-flux quantum circuits. Other applicable areas in which the community may also be interested include the verification of neural networks, which can be represented with matrices, or the modeling of zonotopes, which appear in the control of cyberphysical systems and can also be represented with matrices. Underlying many of these critical areas is the need to reason about vector-valued or matrix-valued functions over the reals, their convergence, continuity, or even differentiability, etc. By formalizing these notions – especially in a manner which retains computational efficiency – we enable the verification of any hardware / software models in which linear algebra is used, which is nearly everywhere.

Acknowledgments

Special thanks to Adrian She, Matt Kaufmann, and Warren A. Hunt, Jr.

References

Gamboa, R. A.; and Kaufmann, M. 2001. Nonstandard Analysis in ACL2. Journal of Automated Reasoning, 27(4): 323–351.

Kwan, C.; and Greenstreet, M. R. 2018. Convex Functions in ACL2(r). Electronic Proceedings in Theoretical Computer Science, 280: 128–142.

Kwan, C.; Peng, Y.; and Greenstreet, M. R. 2020. Cauchy-Schwarz in ACL2(r) Abstract Vector Spaces. Electronic Proceedings in Theoretical Computer Science, 327: 90–92.

Peng, Y.; and Greenstreet, M. R. 2018. Smtlink 2.0. Electronic Proceedings in Theoretical Computer Science, 280: 143–160.