Towards Verified Linear Algebra Programs Through Equivalence*

Yihan Yang
yangyihan121@gmail.com
Harvey Mudd College

Mohit Tekriwal
tekriwal1@llnl.gov
Lawrence Livermore National Laboratory

John Sarracino
sarracino2@llnl.gov
Lawrence Livermore National Laboratory

Matthew Sottile
sottile2@llnl.gov
Lawrence Livermore National Laboratory

Ignacio Laguna
lagunaperalt1@llnl.gov
Lawrence Livermore National Laboratory

1 Introduction

Correctness in scientific computing (SC) is gaining increasing attention as evident from a recently published report by the US Department of Energy (DOE) workshop on correctness [6]. The SC software stack is moving towards a new programming paradigm dominated by approximate computing principles, which involves the use of custom floating-point precision and data compression formats, and is adapting to new heterogeneous hardware architecture to aid performance of scientific simulation. As existing algorithms are being redesigned to support these new programming paradigms, implementation and approximation errors creep into almost all stages of SC software design, whose end-to-end correctness is classified into five key abstraction layers [6]. One correctness abstraction layer is the *implementation* of numerical algorithms, which we target in this work.

There are two main challenges in verifying correctness of numerical algorithms: (a) Lack of a comprehensive mechanized theory of common building blocks like matrix factorization, orthogonalization, interpolation, etc. In the past, there have only been a few works in this area, which includes formalization of Cholesky decomposition [9], Jordan canonical forms [4], and convergence of stationary iterative algorithms [10, 11]; and (b) Lack of equivalence proofs between different variants of an algorithm. Numerical algorithms for linear algebra, widely used in solving partial differential equations, generally have *variants*, which are developed to have favorable numerical properties, such as better numerical stability or a faster rate of convergence. For instance, in the case of a fundamental orthogonalization algorithm – the Gram-Schmidt process [3], two common variants are the Classical Gram-Schmidt (CGS) and the Modified Gram-Schmidt (MGS) processes. A key mathematical invariant for these variants is that each orthogonal vector is computed by subtracting the original vector with a projection of this original vector onto other input vectors. However, these variants vary in the order in which these projections are subtracted in finite precision, which leads to MGS being more stable numerically than CGS. There are several other variants of both CGS and MGS like their block counterparts and their implementation with reorthogonalization. As the SC moves towards approximate computing principles and heterogeneous hardware, leading to reimplementation of classical numerical algorithms in form of new variants, the research question we address in this work is *"how do we verify that these variants are mathematically correct with respect to their classical counterparts?"*. This is important because the classical counterparts have been studied extensively in the numerical literature and act as a ground truth against which the new variants can be evaluated. For the cases where the variation in implementation is due to precision, mathematical correctness boils down to exact equivalence in reals.

Contributions. In this work, we present preliminary work, in Coq, on proving an exact equivalence between different variants of a classical numerical algorithm in reals. We demonstrate a proof structure for proving exact real equivalence between MGS and CGS. In the process of developing this proof structure, we also mechanize several properties of orthogonalization algorithms, which address the challenge (a), discussed earlier. To mechanize GS in Coq, we developed a detailed pen-and-paper correctness proof of CGS, an extension of the proof to MGS, and a mechanization of the pen-and-paper proofs. We are in the process of mechanizing both of these proofs and detail our formal library, mechanization progress, and future applications in the sequel.

2 Mechanizing and Verifying Gram-Schmidt

The Gram-Schmidt process [3] (GS) is a fundamental method which computes an orthogonal basis from a given set of linearly independent vectors, and is defined as follows. Given a set of n linearly independent vectors $x_1, x_2, \ldots, x_n$, both CGS and MGS compute an orthogonal set of vectors $v_1, v_2, \ldots, v_n$:

$$v_j := x_j - \sum_{k=1}^{j-1} (q_k^T x_j) q_k;\ q_k = v_k / ||v_k||_2;\ j = 1 : n \quad \text{[CGS]}$$

$$v_k^{j+1} := v_k^j - (q_j^T v_k^j) q_j;\ q_j = v_j / ||v_j||_2;\ k = j+1 : n;\ j = 1 : n; \ \text{[MGS]}$$

$$v_k^1 = x_k; \quad v_k := v_k^k$$

The difference between CGS and MGS is that CGS orthogonalizes each output vector against previous results in a single large computation. By contrast, MGS takes the set of output vectors and gradually orthogonalizes all of the outputs simultaneously. When computed over Real numbers, these processes have identical results. When computed over floating-point numbers, the MGS computation has significantly better numerical stability due to less accumulation of rounding error. This is because the orthogonalization in CGS explicitly depends on prior inputs and their accumulated error, which generates more error, compounding on each iteration.

2.1 Gram-Schmidt Mechanization in Coq

We used the Mathematical Components library [7] in Coq to develop implementations for MGS, CGS, and the equivalence statement between the two. While there exists similar formalization of GS algorithm in Isabelle/HOL [2] and LEAN [1], these formalisms either target only CGS (in Isabelle/HOL) or use a non-computable definition of CGS (in LEAN). Thus, the novelty of our work lies in the fact that we also attempt to prove the numerically stable variant of GS, i.e., MGS, with the goal of proving interesting numerical properties like stability and convergence. Moreover, beyond just the formalization, we aim to tackle a broader problem of proving

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

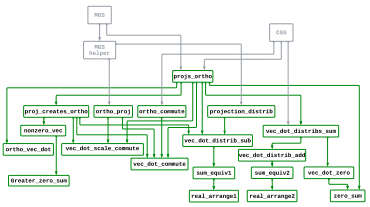

Figure 1. GS operation library. All lemmas are mechanized and proved.

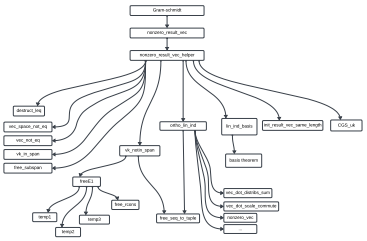

Figure 2. CGS correctness proof. All lemmas are proved except for CGS_uk.

equivalence (both in reals and in floats) between different variants of linear algebra algorithms. This work is therefore a building block to achieve this broader objective.

A key challenge for this task is the lack of a comprehensive formal theory on common linear algebra operations. We extended MathComp to common linear algebra concepts, illustrated in the dependency graphs Fig. 1 and Fig. 2. Each node of the graph is a lemma, and two lemmas A and B are connected by an arrow if A depends on B. Our library is aimed to be extensible to floating-point proofs, which would equip each linear algebra operation with an associated error bound, which needs to be propagated to obtain the final bound on the loss of orthogonality.

2.2 Key Results

The main Gram-Schmidt theorem [8] (classical version) that we formalize in Coq is stated as follows.

Lemma 2.1 (nonzero_result_vec). *Let $v_0, v_1, ..., v_{m-1} \in \mathbb{R}^n$ be the input linearly independent vectors of GS. Let $u_0, .., u_{m-1}$ be the result of GS where $u_k = v_k - \sum_{i=0}^{k-1} proj_{u_i} v_k$. Then the number of result vectors is same as the number of initial vectors, and for all $k \leq m$, $\{u_0, ..u_{k-1}\}$ are all nonzero and pairwise orthogonal. Furthermore, $span(v_0, .., v_{k-1}) = span(u_0, ..., u_{k-1})$.*

We mechanize the Gram-Schmidt theorem into an equivalent Coq lemma over sequences and vectors, using our linear algebra library and MathComp:

```
Lemma nonzero_result_vec_helper {n:nat} (init_vec : seq (r_vector
     n)):
    free init_vec →
    forall k, (k <= (size init_vec)) →
    let result_vec := classical_GS n init_vec in
        (forall i, (i < k) → result_vec`_i != 0) /\
        (forall i j, (i < k) → (j < k) → i != j →
            ortho n result_vec`_i result_vec`_j) /\
        (span (take k result_vec) = span (take k init_vec)).
```

The dependencies for the proof of Lemma nonzero_result_vec are illustrated in Fig. 2. We do not discuss them here for the sake of brevity. The second important lemma that we prove in Coq is the Orthogonal decomposition theorem [8].

Lemma 2.2 (projs_ortho). *Let $a, v_0, v_1, ..., v_{m-1}$ be vectors in $\mathbb{R}^n$. For any natural number k where $k \leq m$, if $v_0, .., v_{k-1}$ are nonzero pairwise orthogonal, then $a - \sum_{i=0}^{k-1} proj_{v_i} a$ is orthogonal to $v_0, ..., v_{k-1}$.*

This is a property of the vectors independent from CGS and MGS. More importantly, both the proof of CGS and MGS uses this lemma.

2.3 Equivalency Proof Sketch

We want to show that for the set of linearly independent initial vectors $v_1, v_2, \ldots, v_n$, the resulting set of orthogonal vectors from CGS, $u_1, u_2, \ldots, u_n$ are exactly equivalent to $v_1^1, v_2^2, \ldots, v_n^n$ in reals. From the definition of CGS and MGS respectively, we have:

$$u_i = v_i - \sum_{k=1}^{i-1} proj_{u_k} v_i; \quad v_i^i = v_i - \sum_{k=1}^{i-1} proj_{v_k^k} v_i^k$$

The equivalence proof then proceeds by induction on i with the base case: $u_1 = v_1^1$. To prove this equivalence, we need to prove the properties of CGS and MGS independently. We have mechanized the proof of CGS, stated in the Lemma 2.1. We are still in the process of mechanizing the theorem for MGS. This theorem has mostly the same structure as the CGS proof and so can reuse the CGS helper lemmas. An additional lemma, which is proved on pen-and-paper but unmechanized, is the following: *For all $0 \leq k < m$, if $v_i^i \perp v_j^j$ for $i, j = 0, ...k-1$, then $v_k^h = v_k - \sum_{i=0}^{h-1} proj_{v_i^i} v_k^i$. for $0 \leq h \leq k$.* Notice that when $h = k$, this is very close to what we want to prove in the main theorem. With additional steps on the proof of orthogonality using projs_ortho, we can prove the correctness of MGS. Once we have mechanized the proof of MGS, we can compose the correctness statements of CGS and MGS to prove the exact equivalence.

3 Future Work

One of the immediate goals on closing the proof is to coordinate with the Mathematical Components team to add our proofs to the library so that it can be used for future developments related to linear algebra proofs. Aside from narrowing the Trusted Computing Base (TCB) and closing our proofs, we see three broad avenues for future work: **(a) Reasoning about floating-point programs**: This extension would require first adapting the equality constraints of our lemmas to symbolic bounds, and then constructing new proofs using the adapted lemmas. **(b) Reasoning about low-level High Performance Computing (HPC) programs**: Another direction is to extend our equivalence results to more fine-grained models of program behavior. This can be done by extending a separation logic (such as Iris, VST, or CFML) with support for floating-point primitives and operations. **(c) Modular equivalence framework for floating-point programs**: One final direction is to extend recent mechanized relational equivalence frameworks such as LGTM [5] with support for floats and proof rules for common linear-algebra relational patterns, as well as higher-order parameterization to enable modular reasoning. This would significantly automate the currently very tedious and error-prone method of manually constructing FP error bounds.

References

[1] [n. d.]. analysis.inner_product_space.gram_schmidt_ortho - mathlib3 docs — leanprover-community.github.io. https://leanprover-community.github.io/mathlib_docs/analysis/inner_product_space/gram_schmidt_ortho.html.

[2] Jesús Aransay and Jose Divasón. 2017. A formalisation in HOL of the fundamental theorem of linear algebra and its application to the solution of the least squares problem. *Journal of Automated Reasoning* 58, 4 (2017), 509–535.

[3] Åke Björck. 1994. Numerics of gram-schmidt orthogonalization. *Linear Algebra and Its Applications* 197 (1994), 297–316.

[4] Guillaume Cano. 2014. *Interaction entre algèbre linéaire et analyse en formalisation des mathématiques*. Theses. Université Nice Sophia Antipolis. https://theses.hal.science/tel-00986283

[5] Vladimir Gladshtein, Qiyuan Zhao, Willow Ahrens, Saman Amarasinghe, and Ilya Sergey. 2024. Mechanised Hypersafety Proofs about Structured Data. *Proc. ACM Program. Lang.* 8, PLDI, Article 173 (June 2024), 24 pages. https://doi.org/10.1145/3656403

[6] Maya Gokhale, Ganesh Gopalakrishnan, Jackson Mayo, Santosh Nagarakatte, Cindy Rubio-González, and Stephen F Siegel. 2023. Report of the DOE/NSF Workshop on Correctness in Scientific Computing, June 2023, Orlando, FL. *arXiv preprint arXiv:2312.15640* (2023).

[7] Assia Mahboubi and Enrico Tassi. 2021. Mathematical components. (2021).

[8] David Poole. 2015. Linear algebra: A modern introduction. (2015).

[9] Pierre Roux. 2016. Formal Proofs of Rounding Error Bounds: With Application to an Automatic Positive Definiteness Check. *Journal of Automated Reasoning* 57 (2016), 135–156.

[10] Mohit Tekriwal, Andrew W Appel, Ariel E Kellison, David Bindel, and Jean-Baptiste Jeannin. 2023. Verified correctness, accuracy, and convergence of a stationary iterative linear solver: Jacobi method. In *International Conference on Intelligent Computer Mathematics*. Springer, 206–221.

[11] Mohit Tekriwal, Joshua Miller, and Jean-Baptiste Jeannin. 2024. Formalization of Asymptotic Convergence for Stationary Iterative Methods. In *NASA Formal Methods Symposium*. Springer, 37–56.