Correct by Construction Gram-Schmidt Process

João Victor Lopez Pereira
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
joaovictorlopezpereira@gmail.com

Daniel Kiyoshi Hashimoto
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
dkhashimoto@ic.ufrj.br

Hugo Musso Gualandi
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
hugomg@ic.ufrj.br

João Paixão
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
jpaixao@ic.ufrj.br

Abstract

The programming languages community has developed proof techniques for reasoning about correctness and equivalence of stream programs in a concise and precise way whilst not worrying about indexes. These ideas can be applied to areas, such as linear algebra, where indexes are the gold-standard. We showcase this through the lens of the Gram-Schmidt process for producing an orthonormal basis, which is an online algorithm that can be succinctly implemented as a purely functional stream program. Our first contribution is a specification of orthogonalization as a Galois Connection. Our second contribution is a correct-by-construction derivation of the classical Gram-Schmidt algorithm directly from its specification. Our third contribution is an induction-free proof that the more numerically stable modified Gram-Schmidt algorithm is equivalent to its classical counterpart. All of our proofs are index-free, in the sense that we do not use indexes to refer to vectors in a basis nor to elements in a vector.

Keywords

Streams, Linear Algebra, Equational Reasoning, Gram-Schmidt, Modified Gram-Schmidt, Galois Connection

1 Introduction

The orthogonalization of a given basis is one of the most fundamental ideas in linear algebra. Its true usefulness lies in preserving the span of a subspace while transforming a set of vectors into an orthonormal basis, making many calculations significantly easier. The most well-known algorithm for this purpose is the Gram-Schmidt process, which constructs an orthonormal basis from a linearly independent set of vectors.

Existing presentations of the Gram-Schmidt process usually separate the algorithm and its correctness argument: the algorithm is presented first, and its correctness is later established through induction proofs filled with indexes. This style of reasoning is far from derivational and tends to obscure the essence of the algorithm.

In this work, we characterize orthogonalization as a Galois connection between subspaces and orthonormal bases (Section 2.1). This perspective exposes the algebraic structure underlying the process and provides a high-level specification from which implementations can be systematically derived. We show, in particular, that the classical Gram-Schmidt algorithm can be obtained directly from its specification through equational reasoning, guaranteeing that the algorithm is correct by construction (Section 2.2). Although most presentations assume that the input vectors are linearly independent, our specification also accepts linearly dependent ones.

The classical Gram-Schmidt procedure is notable for its simplicity and strong geometric intuition, but may suffer from numerical instability when applied in finite-precision arithmetic, particularly when the input vectors are nearly linearly dependent. Another algorithm that implements the Gram-Schmidt process in a slightly different way is the modified Gram-Schmidt algorithm, which improves numerical stability by orthogonalizing each vector against the previously computed orthonormal vectors one step at a time, thereby reducing the accumulation of rounding errors in finite-precision computations.

Although both the classical and the modified algorithms produce the same orthonormal basis in exact arithmetic, proofs of their equivalence are traditionally carried out by induction and involve intricate index manipulations. Such proofs tend to be lengthy and obscure the structural relationship between the two algorithms. Rather than relying on a conventional inductive and index-based linear algebra proof, we employ Hinze's stream calculus [4] to establish that the two algorithms are equivalent in the more general case in which both the input and output are streams (Section 3).

Our contributions are:

(1) A specification of orthogonalization as a Galois Connection which supports linearly dependent inputs (Section 2.1);
(2) A correct-by-construction derivation of the classical Gram-Schmidt algorithm, from this specification (Section 2.2);
(3) An induction-free and index-free proof that the modified Gram-Schmidt algorithm is equivalent to the classical one, for linearly independent inputs (Section 3).

2 Orthogonalization as a Galois Connection

The Gram-Schmidt process is usually presented as a procedure that transforms a linearly independent set of vectors into an orthonormal basis spanning the same subspace. However, every algorithm implicitly implements a specification, and understanding this specification is often the key to understanding the algorithm itself.

In the case of Gram-Schmidt, the usual pedagogical order is somewhat reversed: students are first taught the algorithm and only later prove that the resulting orthonormal basis spans the same subspace as the original vectors. In other words, the implementation is typically presented before the underlying specification. Our goal is to reverse this order. We begin with a specification of orthogonalization and derive the algorithm from it.

2.1 Specification for Orthogonalization

Our first step is to specify an orthonormal basis. It is a list of unit vectors, represented as columns of a matrix, that are perpendicular to each other. As our proofs will proceed recursively on the columns of these matrices, it is convenient work with an inductive definition:

Definition 1. $\mathbb{O}^{m\times n}$ is the set of **orthogonal matrices** in $\mathbb{R}^{m\times n}$. Q is an orthogonal matrix if and only if one of the following holds:

(1) Q is the empty matrix $[]^{m\times 0}$;
(2) $Q = \begin{bmatrix} Q_1 & q_2 \end{bmatrix}$, where $Q_1 \in \mathbb{O}^{m\times(n-1)}$, $q_2 \in \mathbb{R}^m$, $\|q_2\| = 1$, and $Q_1^\intercal q_2 = \bar{0}$.

Second, we want to specify that the input vectors can be generated as linear combinations of the vectors in our orthonormal basis. Moreover, this should be done in a particular order. When the input vectors are linearly independent, the i-th input vector should be a linear combination of the first i vectors of the basis, so that each input vector results in the addition of one vector to the basis. In matricial notation, this would be expressed by a QR decomposition: the input matrix A gets decomposed into the product QR of an orthogonal matrix Q and an upper triangular matrix R.

To support linearly dependent inputs, we ensure that if one of the input vectors is a linear combination of the preceding ones, then no vectors should be added to the basis. To do so, we generalize upper triangular matrices to wide staircase matrices.

Definition 2. $\mathbb{W}^{b\times c}$ is the set of **wide staircase matrices** in $\mathbb{R}^{b\times c}$. R is a wide staircase matrix if and only if one of the following holds:

(1) R is the empty matrix $[]^{0\times 0}$;
(2) $R = \begin{bmatrix} R_1 & r_2 \end{bmatrix}$, where $R_1 \in \mathbb{W}^{b\times(c-1)}$ and $r_2 \in \mathbb{R}^b$;
(3) $R = \begin{bmatrix} R_{11} & r_{12} \\ \bar{0}^\intercal & r_{22} \end{bmatrix}$, where $R_{11} \in \mathbb{W}^{(b-1)\times(c-1)}$, $r_{12} \in \mathbb{R}^{b-1}$, $r_{22} \in \mathbb{R}$, and $r_{22} \neq 0$.

Third, a vector basis can be transformed into an equivalent one by flipping the signs of zero or more of its vectors. This operation can be encoded as a sign-diagonal matrix.

Definition 3. $\mathbb{D}^{n\times n}$ is the set of **sign-diagonal matrices** in $\mathbb{R}^{n\times n}$. D is a sign-diagonal matrix if and only if one of the following holds:

(1) D is the empty matrix $[]^{0\times 0}$;
(2) $D = \begin{bmatrix} D_1 & \bar{0} \\ \bar{0}^\intercal & d_2 \end{bmatrix}$, where $D_1 \in \mathbb{D}^{(d-1)\times(d-1)}$ and $d_2 = \pm 1$.

The following properties appear in the sequel and can be verified by induction.

Property 4. Q is orthogonal if and only if $Q^\intercal Q = I$.

Property 5. If D is a sign-diagonal matrix, then $D = D^\intercal$.

Property 6. If D is a sign-diagonal matrix, then $DD = I$.

We are now ready to specify the two matrix decompositions that will appear in our main specification. The first decomposition states that some list of vectors B can generate some list C, which we write $B \leq C$. We specify this as a QR-like decomposition using a wide staircase matrix. For now we do not require that any of these matrices be orthogonal.

Definition 7. Let $B \in \mathbb{R}^{m\times b}$ and $C \in \mathbb{R}^{m\times c}$ be matrices.

$$B \leq C \iff \exists R \in \mathbb{W}^{b\times c}(BR = C).$$

It will be useful to break this definition into cases that mirror the inductive structure of the wide staircase matrix R that witnesses the decomposition. There are three possibilities for $BR = C$:

(1) $[]^{m\times 0} \cdot []^{0\times 0} = []^{m\times 0}$;
(2) $B \cdot \begin{bmatrix} R_1 & r_2 \end{bmatrix} = \begin{bmatrix} C_1 & c_2 \end{bmatrix}$;
(3) $\begin{bmatrix} B_1 & b_2 \end{bmatrix} \cdot \begin{bmatrix} R_{11} & r_1 \\ \bar{0}^\intercal & r_2 \end{bmatrix} = \begin{bmatrix} C_1 & c_2 \end{bmatrix}$, with $r_2 \neq 0$.

If we write these multiplications in full and replace $BR_1 = C_1$ by $B \leq C_1$, and so on, we obtain a lemma that describes $\leq$ inductively.

Lemma 8. Let $B \in \mathbb{R}^{m\times b}$ and $C \in \mathbb{R}^{m\times c}$ be matrices with $b \leq c$. We have $B \leq C$ if and only if one of the following cases holds:

(1) B and C are the empty matrix $[]^{m\times 0}$;
(2) $C = \begin{bmatrix} C_1 & c_2 \end{bmatrix}$, where $C_1 \in \mathbb{R}^{m\times(c-1)}$, $c_2 \in \mathbb{R}^m$, such that $B \leq C_1$ and $\exists r_2 \in \mathbb{R}^b(B\,r_2 = c_2)$;
(3) $B = \begin{bmatrix} B_1 & b_2 \end{bmatrix}$ and $C = \begin{bmatrix} C_1 & c_2 \end{bmatrix}$, where $B_1 \in \mathbb{R}^{m\times(b-1)}$, $b_2 \in \mathbb{R}^m$, $C_1 \in \mathbb{R}^{m\times(c-1)}$, $c_2 \in \mathbb{R}^m$, such that $B_1 \leq C_1$ and $\exists r_1 \in \mathbb{R}^{b-1} \exists r_2 \in \mathbb{R}(B_1 r_1 + b_2 r_2 = c_2 \wedge r_2 \neq 0)$.

The second case happens when c_2 is a linear combination of the vectors that generate C_1. The third case happens when we introduce a vector b_2 to the base and the condition $r_2 \neq 0$ ensures that this new vector is used in the linear combination that generates c_2.

Our second and last kind of matrix decomposition specifies that two orthogonal matrices are equivalent modulo sign-flipping, which we write $U \sqsubseteq V$. We only need to define this for orthogonal matrices, although the definition would also work for non-orthogonal ones.

Definition 9. Let $U \in \mathbb{O}^{m\times n}$ and $V \in \mathbb{O}^{m\times n}$ be matrices.

$$U \sqsubseteq V \iff \exists D \in \mathbb{D}^{n\times n}(UD = V).$$

It will again be useful to rewrite the decomposition inductively. There are two possible cases for $UD = V$:

(1) $[]^{m\times 0} \cdot []^{0\times 0} = []^{m\times 0}$;
(2) $\begin{bmatrix} U_1 & u_2 \end{bmatrix} \cdot \begin{bmatrix} D_1 & \bar{0} \\ \bar{0}^\intercal & d_2 \end{bmatrix} = \begin{bmatrix} V_1 & v_2 \end{bmatrix}$, with $d_2 = \pm 1$.

Lemma 10. Let $U \in \mathbb{O}^{m\times n}$ and $V \in \mathbb{O}^{m\times n}$ be matrices. We have $U \sqsubseteq V$ if and only if one of the following cases hold:

(1) U and V are the empty matrix $[]^{0\times 0}$;
(2) $U = \begin{bmatrix} U_1 & u_2 \end{bmatrix}$ and $V = \begin{bmatrix} V_1 & v_2 \end{bmatrix}$, where $U_1 \in \mathbb{O}^{m\times(n-1)}$, $u_2 \in \mathbb{R}^m$, $V_1 \in \mathbb{O}^{m\times(n-1)}$, and $v_2 \in \mathbb{R}^m$, such that $U_1 \sqsubseteq V_1$, and $\pm u_2 = v_2$.

The relations $\leq$ and $\sqsubseteq$ are reflexive, because the identity matrix is a member of $\mathbb{W}$ and $\mathbb{D}$. They are also transitive, because $\mathbb{W}$ and $\mathbb{D}$ are closed under matrix multiplication. Thus, these two relations are preorders. For all A, B, and C:

Property 11 ($\leq$ is reflexive)**.** $A \leq A$.

Property 12 ($\leq$ is transitive)**.** $A \leq B$ and $B \leq C$ implies $A \leq C$.

Property 13 ($\sqsubseteq$ is reflexive)**.** $A \sqsubseteq A$.

Property 14 ($\sqsubseteq$ is transitive)**.** $A \sqsubseteq B$ and $B \sqsubseteq C$ implies $A \sqsubseteq C$.

We are finally ready to specify the orthogonalization algorithm. The orthonormal basis should

(1) be able to generate the input vectors; and

(2) be equivalent modulo sign-flipping to any other basis that generates the input vectors.

Specification 1 (Direct)**.** An **orthogonalization function** is a map $\mathrm{O}\colon \mathbb{R}^{m\times n} \to \mathbb{O}^{m\times k}$ for some $k \le n$ such that, for all $A \in \mathbb{R}^{m\times n}$, both of the following hold:

(1) $\mathrm{O}(A) \le A$;
(2) $\forall Q\big(Q \le A \implies Q \sqsubseteq \mathrm{O}(A)\big)$.

Observe that this does not specify a canonical base; the algorithm is allowed to choose any base that is equivalent according to $\sqsubseteq$. Furthermore, our definition of $\le$ does not allow the basis Q to have superfluous vectors, only vectors strictly necessary to generate A can be included.

For the upcoming derivation of the algorithm, it will be useful to condense the specification into a single rule.

Specification 2 (Galois)**.** An **orthogonalization function** is a map $\mathrm{O}\colon \mathbb{R}^{m\times n} \to \mathbb{O}^{m\times k}$ for some $k \le n$ such that, for all Q and A,

$$Q \le A \iff Q \sqsubseteq \mathrm{O}(A).$$

Theorem 15. Specification 1 is equivalent to Specification 2.

Proof. The forwards direction of Specification 2 is clearly equivalent to Specification (1.2). It remains to show that the backwards direction of Specification 2 implies Specification (1.1)

$$\begin{aligned}
&\forall Q\big(Q \sqsubseteq \mathrm{O}(A) \implies Q \le A\big)\\
\Downarrow\quad &\{\ \text{Let } Q = \mathrm{O}(A)\ \}\\
&\mathrm{O}(A) \sqsubseteq \mathrm{O}(A) \implies \mathrm{O}(A) \le A\\
\Downarrow\quad &\{\ \sqsubseteq \text{ is reflexive (Prop 13)}\ \}\\
&\mathrm{O}(A) \le A
\end{aligned}$$

and that Specification (1.1) implies the backwards direction of Specification 2.

$$\begin{aligned}
&Q \sqsubseteq O(A)\\
\Downarrow\quad &\{\ \text{All sign-diagonal matrices are wide staircase}\ \}\\
&Q \le O(A)\\
\Downarrow\quad &\{\ \text{Hypothesis } \mathrm{O}(A) \le A \text{ and transitivity (Prop 12)}\ \}\\
&Q \le A
\end{aligned}$$ □

Specification 2 is an instance of a Galois connection, a common pattern that is common in program derivation [7].

Definition 16. Given two pre-ordered sets $(P, \sqsubseteq)$ and $(Q, \le)$, the functions $f\colon P \to Q$ and $g\colon Q \to P$ form a **Galois connection** whenever, for all $x \in P$ and $y \in Q$,

$$f(x) \le y \iff x \sqsubseteq g(y).$$

In our specification, the upper adjoint g is the orthogonalization function O; and the lower adjoint f is a function of type $\mathbb{O}^{m\times n} \to \mathbb{R}^{m\times n}$ which "forgets" that a matrix is orthogonal.

Although the direct specification captures the intended meaning of orthogonalization, it is not particularly suitable for program derivation. Its minimality condition is expressed through a global quantification over all admissible orthogonal factorizations and therefore provides little guidance on how a solution should be constructed. In contrast, the Galois specification characterizes orthogonalization through a local equivalence between the relations $\le$ and $\sqsubseteq$. This formulation exposes the algebraic structure of the problem, combines validity and optimality into a single law, and can be unfolded recursively during calculations. For these reasons, the Galois specification serves as a more convenient starting point for the derivation of the Gram–Schmidt algorithm.

2.2 Program Derivation

Having established the Galois characterization of orthogonalization, we now derive the implementation. Rather than design an algorithm and subsequently prove its correctness, we proceed in the opposite direction: the implementation will emerge from the specification itself. But first, we introduce some useful notation and concepts.

Definition 17. Let $U \in \mathbb{O}^{m\times n}$, and $b \in \mathbb{R}^m$. We say that P_U is the **orthogonal projection** of b into the subspace spanned by U, and $\mathrm{P}_{U^\perp}$ is the **residue** of this projection.

$$\mathrm{P}_U b = UU^\mathsf{T} b$$
$$\mathrm{P}_{U^\perp} b = b - (UU^\mathsf{T} b)$$

For any basis U, the residue $\mathrm{P}_{U^\perp} b$ is the only vector orthogonal to U that connects b to the subspace spanned by U.

Lemma 18. Let $U \in \mathbb{O}^{m\times n}$, and $e, b \in \mathbb{R}^m$. If $U^\mathsf{T} e = \bar{0}$ then

$$\exists x \in \mathbb{R}^n (Ux + e = b) \iff e = \mathrm{P}_{U^\perp} b.$$

Proof.

$$\begin{aligned}
&\exists x(Ux + e = b)\\
\Updownarrow\quad &\{\ \text{Multiply by } UU^\mathsf{T}\ \}\\
&\exists x(Ux + e = b \wedge UU^\mathsf{T}(Ux + e) = UU^\mathsf{T} b)\\
\Updownarrow\quad &\{\ U^\mathsf{T}U = I \text{ (Prop 4) and } U^\mathsf{T} e = \bar{0}\ \}\\
&\exists x(Ux + e = b \wedge Ux = UU^\mathsf{T} b)\\
\Updownarrow\quad &\{\ \text{Let } x = U^\mathsf{T} b\ \}\\
&UU^\mathsf{T} b + e = b\\
\Updownarrow\quad &\{\ \text{Definition of } \mathrm{P}_{U^\perp} \text{ (Def 17)}\ \}\\
&e = \mathrm{P}_{U^\perp} b
\end{aligned}$$ □

Equivalent orthonormal bases have the same orthogonal projections. This ability to switch from one basis to the other will come in handy in our derivation of the Gram-Schmidt algorithm.

Lemma 19 (Basis Swap)**.** If two orthogonal bases $U, V \in \mathbb{O}^{m\times n}$ are equivalent ($U \sqsubseteq V$), then $\mathrm{P}_U = \mathrm{P}_V$ and $\mathrm{P}_{U^\perp} = \mathrm{P}_{V^\perp}$.

Proof. By hypothesis, there exists some sign-diagonal D such that $UD = V$. From properties 5 and 6 we know that $DD^\mathsf{T} = I$ and therefore $\mathrm{P}_V = VV^\mathsf{T} = UDD^\mathsf{T}U^\mathsf{T} = UU^\mathsf{T} = \mathrm{P}_U$. It follows, from the definition of residue, that $\mathrm{P}_{V^\perp} = \mathrm{P}_{U^\perp}$. □

The residue is orthogonal to the basis and is a good candidate to extend the basis.

Lemma 20. Let $V \in \mathbb{O}^{m\times n}$ and $b \in \mathbb{R}^m$. We have $V^\mathsf{T}(\mathrm{P}_{V^\perp} b) = \bar{0}$.

Proof. Since V is orthogonal, $V^\mathsf{T}V = I$ and thus

$$V^\mathsf{T}(\mathrm{P}_{V^\perp} b) = V^\mathsf{T}(b - VV^\mathsf{T} b) = V^\mathsf{T} b - V^\mathsf{T}VV^\mathsf{T} b = V^\mathsf{T} b - V^\mathsf{T} b = \bar{0}.$$ □

The next lemma characterizes the scalar relating a vector to a colinear unit vector. We will use it to identify the coefficient multiplying the new orthonormal vector when we extend the basis.

Lemma 21. Let $u \in \mathbb{R}^m$ be a unit vector, let $b \in \mathbb{R}^m$ be any a vector. If $ux = b$ for some scalar x then $x = \pm \|b\|$.

Proof. We have $\|b\| = \|ux\| = |x| = \pm x$. □

At last, we are ready to derive the Gram-Schmidt algorithm.

Theorem 22 (Gram-Schmidt Derivation by Galois Specification)**.** There exists an orthogonalization function $\mathrm{O} : \mathbb{R}^{m\times n} \to \mathbb{O}^{m\times k}$ which follows the Specification 2, which says that for all Q and A,

$$Q \le A \iff Q \sqsubseteq \mathrm{O}(A).$$

Proof. Superficially, our derivation is a proof by induction that the function O satisfies the specification. However, at the start we do not yet know what O should be. It's only at the end that we discover which implementation completes the proof.

The induction is on the columns of A. In the base case $A = []^{m\times 0}$ and in the inductive case A is composed of $A_1 \in \mathbb{R}^{m\times n}$ and $a_2 \in \mathbb{R}^m$. We need to prove, for all Q,

$$Q \le [] \iff Q \sqsubseteq O([]) \qquad \text{(Base case)}$$
$$Q \le \begin{bmatrix} A_1 & a_2 \end{bmatrix} \iff Q \sqsubseteq \mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right) \qquad \text{(Inductive case)}$$

Base case. We begin our calculations on the $\le$ side and work towards the $\sqsubseteq$ side. Our first step is to invoke Lemma 8, which uncovers that Q must be the empty matrix.

$$\begin{aligned} & Q \le []^{m\times 0} \\ \Updownarrow \quad & \{ \text{ Cases for } \le \text{ (Lemma 8) } \} \\ & Q = []^{m\times 0} \\ \Updownarrow \quad & \{ \text{ Cases for } \sqsubseteq \text{ (Lemma 10) } \} \\ & Q \sqsubseteq []^{m\times 0} \end{aligned}$$

The last step suggests that to complete the proof we should define

$$\mathrm{O}([]^{m\times 0}) = []^{m\times 0}.$$

Inductive case. Lemma 8 tells us that there are two possible cases when $A = \begin{bmatrix} A_1 & a_2 \end{bmatrix}$. In one case a_2 can be generated by the same orthogonal vectors that generate A_1 and in the other case it cannot. We will work through these two cases in turn and each will produce a separate clause in the implementation of the orthogonalization function.

Linearly dependent subcase. In this case, Lemma 8 tells us that $Q \le A_1$ and $\exists r(Qr = a_2)$. The calculation proceeds from that point.

$$\begin{aligned} & Q \le A_1 \wedge \exists r(Qr = a_2) \\ \Updownarrow \quad & \{ \text{ Induction hypothesis } \} \\ & Q \sqsubseteq \mathrm{O}(A_1) \wedge \exists r(Qr = a_2) \\ \Updownarrow \quad & \{ \text{ Lemma 18 } \} \\ & Q \sqsubseteq \mathrm{O}(A_1) \wedge \mathrm{P}_{Q^\perp} a_2 = \bar{0} \\ \Updownarrow \quad & \{ \text{ Basis swap (Lemma 19) } \} \\ & Q \sqsubseteq \mathrm{O}(A_1) \wedge \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 = \bar{0} \end{aligned}$$

Linearly independent subcase. This time Lemma 8 tells us that $Q = \begin{bmatrix} Q_1 & q_2 \end{bmatrix}$ with $Q_1 \in \mathbb{O}^{m\times n}$ and $q_2 \in \mathbb{R}^m$ such that $Q_1 \le A_1$ and that there exist $r_1 \in \mathbb{R}^n$, $r_2 \in \mathbb{R}$ such that $Q_1 r_1 + q_2 r_2 = a_2$ and $r_2 \neq 0$. Additionally, because Q is orthogonal, we know that $Q_1^\intercal q_2 = \bar{0}$, and $\|q_2\| = 1$. With this information at hand, we calculate:

$$\begin{aligned} & Q_1 \le A_1 \wedge \exists r_1 \exists r_2 (Q_1 r_1 + q_2 r_2 = a_2 \wedge r_2 \neq 0) \\ \Updownarrow \quad & \{ \text{ Inductive hypothesis } \} \\ & Q_1 \sqsubseteq \mathrm{O}(A_1) \wedge \exists r_1 \exists r_2 (Q_1 r_1 + q_2 r_2 = a_2 \wedge r_2 \neq 0) \\ \Updownarrow \quad & \{ \text{ Lemma 18 } \} \\ & Q_1 \sqsubseteq \mathrm{O}(A_1) \wedge \exists r_2 (q_2 r_2 = \mathrm{P}_{Q_1^\perp} a_2 \wedge r_2 \neq 0) \\ \Updownarrow \quad & \{ \text{ Basis swap (Lemma 19) } \} \\ & Q_1 \sqsubseteq \mathrm{O}(A_1) \wedge \exists r_2 (q_2 r_2 = \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \wedge r_2 \neq 0) \\ \Updownarrow \quad & \{ \text{ Going down: Lemma 21;} \\ & \quad \text{Going up: Let } r_2 = \pm \left\| \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \right\| \ \} \\ & Q_1 \sqsubseteq \mathrm{O}(A_1) \wedge \pm q_2 = \frac{\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2}{\left\| \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \right\|} \wedge \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0} \\ \Updownarrow \quad & \{ \text{ Lemma 10 using Lemma 20, which says that} \\ & \quad \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \text{ is orthogonal to } \mathrm{O}(A_1) \ \} \\ & \begin{bmatrix} Q_1 & q_2 \end{bmatrix} \sqsubseteq \begin{bmatrix} \mathrm{O}(A_1) & \frac{\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2}{\left\| \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \right\|} \end{bmatrix} \wedge \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0} \end{aligned}$$

Summing up. Conveniently, the side conditions $\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 = \bar{0}$ and $\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0}$ are complementary. This invites us to define $\mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right)$ by cases:

$$\mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right) = \begin{cases} \mathrm{O}(A_1) & \text{if } \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 = \bar{0} \\ \begin{bmatrix} \mathrm{O}(A_1) & \frac{\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2}{\left\| \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \right\|} \end{bmatrix} & \text{if } \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0}. \end{cases}$$

We can now finish the proof of the inductive case:

$$\begin{aligned} & Q \le \begin{bmatrix} A_1 & a_2 \end{bmatrix} \\ \Updownarrow \quad & \{ \text{ Is } a_2 \text{ is in the span of } Q? \\ & \quad \text{Break in cases with Lemma 10. } \} \\ & \left(Q \sqsubseteq \mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right) \wedge \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 = \bar{0}\right) \\ & \vee \left(Q \sqsubseteq \mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right) \wedge \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0}\right) \\ \Updownarrow \quad & \{ \text{ Is } a_2 \text{ in the span of } \mathrm{O}(A_1)? \\ & \quad \text{Break in cases for zero and nonzero } \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2. \ \} \\ & Q \sqsubseteq \mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right). \end{aligned}$$

This completes the proof and the construction of the algorithm. To recapitulate the main result, we can summarize the derived implementation as follows:

$$\mathrm{O}([]) = []$$

$$\mathrm{O}\left(\begin{bmatrix} A_1 & a_2 \end{bmatrix}\right) = \begin{cases} \mathrm{O}(A_1) & \text{if } \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 = \bar{0} \\ \begin{bmatrix} \mathrm{O}(A_1) & \frac{\mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2}{\left\| \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \right\|} \end{bmatrix} & \text{if } \mathrm{P}_{\mathrm{O}(A_1)^\perp} a_2 \neq \bar{0}. \end{cases}$$

□

The derivation we just presented is noteworthy because the Gram-Schmidt algorithm was not postulated in advance. Instead, it

emerged as the unique construction capable of satisfying the Galois specification recursively. The familiar projection-and-normalization steps therefore appear as consequences of the specification rather than as design choices. This illustrates the correct-by-construction nature of the derivation: correctness is inherited directly from the specification used throughout the calculation.

3 Equivalence of the Classical and Modified Gram-Schmidt Algorithms

The previous section showed how the classical Gram-Schmidt algorithm can be derived from its specification in a correct-by-construction fashion. We now turn to a different question. Rather than deriving an algorithm from first principles, we investigate the relationship between two existing implementations: the classical and the modified Gram-Schmidt algorithms. To this end, we first present both algorithms, then introduce the stream-calculus machinery required for the proof, and finally establish their equivalence through equational reasoning, avoiding induction and index manipulations.

In this section, we work with the traditional versions of the Gram-Schmidt algorithm, which are defined only for linearly independent inputs. Consequently, we no longer consider the generalized version of Gram-Schmidt derived in the previous section, which explicitly handles linearly dependent inputs.

3.1 The Two Algorithms

Before proving the equivalence between the programs, we must first present the programs themselves. We begin with a pseudo-code implementation of the classical Gram-Schmidt algorithm:

```
function Classical-Gram-Schmidt(A)
    for i in 1 : n
        v_i = a_i
        for j in 1 : i − 1
            v_i = v_i − q_j q_j^T a_i
        end
        q_i = v_i/‖v_i‖
    end
    return Q
end
```

There are several forms of the modified Gram-Schmidt algorithm. We will consider the following variant, which is close to the presentation in Golub and Loan [3].

```
function Modified-Gram-Schmidt(A)
    for i in 1 : n
        v_i = a_i
        for j in 1 : i − 1
            v_i = v_i − q_j q_j^T v_i
        end
        q_i = v_i/‖v_i‖
    end
    return Q
end
```

Our code follows some conventions. The value n should be understood as the number of columns of the input matrix A. The columns of the matrices A and Q are vectors a_i and q_i. Accordingly, the assignment $q_i = v_i/\|v_i\|$ should be interpreted as assigning the normalization of v_i to the i-th column of Q.

Notice that the only visible difference between the classical and the modified algorithm is a single character. In the inner loop, the classical algorithm uses a_i where the modified algorithm uses v_i.

The essential disparity between the algorithms lies in how their inner loops accumulate the orthogonal projections. In the classical version the inner assignment $v_i = v_i - q_j q_j^\top a_i$ projects the input vector a_i and each projection is independent from the previous ones. Therefore, the corrections accumulate additively:

$$v_i = a_i - (q_1 q_1^\top a_i) \cdots - (q_{i-1} q_{i-1}^\top a_i).$$

In contrast, the modified assignment $v_i = v_i - q_j q_j^\top v_i$ updates v_i based on the previous value of v_i. In the modified algorithm, the corrections accumulate multiplicatively:

$$v_i = (I - q_{i-1} q_{i-1}^\top) \cdots (I - q_1 q_1^\top)\, a_i.$$

This difference becomes more noticeable if we rewrite the algorithms using capital Σ and capital Π notation. Now they no longer differ by a single character.

```
function Classical-Gram-Schmidt(A)
    for i in 1 : n
        v_i = ( I − Σ_{j=1}^{i−1} q_j q_j^T ) a_i
        q_i = v_i/‖v_i‖
    end
    return Q
end

function Modified-Gram-Schmidt(A)
    for i in 1 : n
        v_i = ( Π_{j=i−1}^{1} ( I − q_j q_j^T ) ) a_i
        q_i = v_i/‖v_i‖
    end
    return Q
end
```

The goal is also more apparent now. To show that these programs are equivalent, it would suffice to prove, for all orthogonal Q, that

$$I - \left(\sum_{i=1}^{n} q_i q_i^\top \right) \;=\; \prod_{i=1}^{n} \left(I - q_i q_i^\top \right).$$

The traditional way to prove such a result would be via induction over n, resulting in a proof that requires careful management of indices. We explore alternative methods which reduce the need for induction and allow us to work without indices.

3.2 Stream Calculus

One way to avoid indices is to reach for a stream-based language where the sequences of iterations are first-class objects of discourse.

We adopt Hinze's stream calculus [4], which provides various operations to generate and transform streams as well as proof techniques to show when two streams are identical.

Before we apply the stream calculus to our problem, we should review how it works. A stream is an infinite list of values. The simplest examples are constant streams, which repeat the same value indefinitely. The simplest operation is the constructor (:), read *cons*, which prepends an element to a stream. For example, the constant stream of zeros is the solution to the recursive equation

```
z = 0 : z
```

Arithmetic operations act element-wise on streams. For example, the stream of natural numbers can be defined as the solution to

```
n = 0 : 1 + n
```

Numeric literals on the left of a (:) stand for actual numbers, but elsewhere they stand for infinite constant streams. Hence, the stream of natural numbers begins with zero and continues with the stream that we obtain if we increment each natural number.

This style of programming is similar to the one that is encountered in lazy programming languages. For example, in Haskell we could write the above example using `repeat` for the constant lists and `zipWith` for the element-wise addition.

```
nats = 0 : zipWith (+) (repeat 1) nats
```

The running theme in this section is that we define recurrences using recursive stream equations whose operations act on streams as a whole. We never individually index the i-th element of a stream.

We now turn to the question of whether a stream equation has a well-defined solution. An attentive reader may have noticed that at several points we mentioned *the* solution of an equation, instead of merely *a* solution. As it turns out, all the equations we presented so far do have unique solutions, but this fact must be established. Whether an unique solution exists is in general undecidable, but there is a syntactic condition that ensures uniqueness.

Definition 23. A stream equation is **admissible** if it has the form $x = h : t$ where h is a fixed constant and t is a stream expression that does not use the `tail` operator.

It can be proven that a system of admissible stream equations has an unique solution [4]. The basic idea is that the admissibility condition ensures that each element of the stream depends only on elements that appear before it. This rules out paradoxical streams where one element would be defined in terms of itself. For instance, the following Haskell equations are not admissible and lead to infinite loops if they are evaluated.

```
loop = loop
bad = 0 : tail bad
```

Knowing that an equation has an unique solution is powerful. It allows us to define a stream as the solution to an equation, and it also offers a way to prove that two streams are equal: whenever two streams are solutions of same admissible equation, then those streams must be identical.

We complete our tour of the stream calculus by introducing accumulation operators that compute running sums and products. They will model the for-loops used in the Gram-Schmidt algorithm.

Definition 24 (Operator Σ)**.** Let a be a stream and let 0 denote the additive identity of the type of a.

$$\Sigma\, a = (0 : a + \Sigma\, a).$$

Definition 25 (Operator Π)**.** Let a be a stream and let 1 denote the multiplicative identity of the type of a.

$$\Pi\, a = (1 : a \cdot \Pi\, a).$$

These operators can be understood as analogous to `scanl` in Haskell. Each element of the stream contains the partial sum or the partial product accumulated up to that point.

These definitions not only provide a compact specification of the operators, but also characterize them as fixed points of stream equations.

Lemma 26 (Universal Property of Σ)**.** Let a be a stream.

$$b = \Sigma\, a \iff b = (0 : a + b).$$

Lemma 27 (Universal Property of Π)**.** Let a be a stream.

$$b = \Pi\, a \iff b = (1 : a \cdot b).$$

Both the directions of the equivalences above play distinct roles. The forward direction (from left to right) asserts that $\Sigma\, a$ and $\Pi\, a$ satisfy their corresponding recursive equations, ensuring consistency with their intended construction. The backwards direction (from right to left) guarantees uniqueness: any stream satisfying the same equation must coincide with the one defined by the respective operator. We know that the solutions are unique because both of the equations are admissible.

3.3 Gram-Schmidt as Stream Equations

Having defined the accumulation operators, our pseudo-code can be written in terms of stream equations as follows:

```
Classical-Gram-Schmidt a = q
    where
        v = (I − Σ p)  a
        q = v/‖v‖
        p = qqᵀ

Modified-Gram-Schmidt a = q
    where
        v = (Π(I − p))  a
        q = v/‖v‖
        p = qqᵀ
```

In these programs, a is a stream of linearly independent input vectors, v is the a stream of orthogonal vectors that span a, q is the output stream of orthogonal unit vectors, and p is the stream whose i-th element is the orthogonal projection matrix $q_i q_i^{\mathsf{T}}$. All operations, including normalization and transposition, work element-wise on streams.

These stream-based formulations describe the the Gram-Schmidt procedure in a declarative and index-free manner, in contrast to the traditional imperative implementations which rely on assignments, for loops, and plenty of vector indexing.

3.4 The Equivalence Proof

We will now prove that the stream-based formulations of the classical and modified Gram-Schmidt algorithms produce the same result. To do so, we show that the part that differs between the algorithms actually computes the same stream.

Theorem 28. Let a, v, q, and p be streams and let I be the stream of identity matrices, such that

$$v = (I - \Sigma\, p)a, \quad q = v/\|v\|\,, \quad \text{and} \quad p = qq^{\mathsf{T}}.$$

Then

$$I - \Sigma\, p = \Pi(I - p).$$

Proof. We use Lemma 27, the universal property of Π. To prove $I - \Sigma\, p = \Pi(I - p)$, it suffices to show $I - \Sigma\, p = I : (I - p) \cdot (I - \Sigma\, p)$.

$$\begin{aligned}
& I - \Sigma\, p \\
= \quad & \{ \text{ Definitions of } I \text{ and } \Sigma \ \} \\
& (I : I) - (0 : p + \Sigma\, p) \\
= \quad & \{ \text{ Definition of } (-) \ \} \\
& (I - 0) : (I - (p + \Sigma\, p)) \\
= \quad & \{ \text{ Simplification } \} \\
& I : (I - p - \Sigma\, p) \\
= \quad & \{ \ p\, \Sigma\, p = 0 \text{ by upcoming Theorem 29 } \} \\
& I : (I - p - \Sigma\, p + p\, \Sigma\, p) \\
= \quad & \{ \text{ Distributive } \} \\
& I : (I - p) \cdot (I - \Sigma\, p)
\end{aligned}$$ □

We find this proof easier to read if we read the first three steps from the top to bottom and the last step from the bottom to the top. Doing so, we discover that, to complete the proof, all that remains is to show that $p\, \Sigma\, p = 0$, which we will prove in the next theorem.

The statement $p\, \Sigma\, p = 0$ encapsulates the hypothesis that the vectors q are orthogonal. Indeed, since $p = qq^{\mathsf{T}}$, the condition $p\, \Sigma\, p = 0$ expresses that each projection matrix qq^{T} nullifies the contributions accumulated from all previous projections, which is equivalent to stating that each vector q_i is orthogonal to the span of vectors q_j for $j < i$.

Theorem 29. Let a, v, q and p be streams and let I be the stream of identity matrices, such that

$$v = (I - \Sigma\, p)a, \quad q = v/\|v\| \quad \text{and} \quad p = qq^{\mathsf{T}}.$$

Then

$$p\, \Sigma\, p = 0.$$

Proof.

$$\begin{aligned}
& p\, \Sigma\, p \\
= \quad & \{ \text{ Definitions of } p, q, \text{ and } v \ \} \\
& q \frac{((I - \Sigma\, p)\, a)^{\mathsf{T}}}{\|v\|}\, \Sigma\, p \\
= \quad & \{ \text{ Transpose of the product } \} \\
& q \frac{a^{\mathsf{T}}}{\|v\|}\, (I - \Sigma\, p)^{\mathsf{T}}\, \Sigma\, p \\
= \quad & \{ \text{ Let } \alpha = q \frac{a^{\mathsf{T}}}{\|v\|} \ \} \\
& \alpha\, (I - \Sigma\, p)^{\mathsf{T}}\, \Sigma\, p \\
= \quad & \{ \ (\Sigma\, p)^{\mathsf{T}} = \Sigma\, p \ \} \\
& \alpha \left(\Sigma\, p - (\Sigma\, p)^2\right)
\end{aligned}$$

Thus, to prove $p\, \Sigma\, p = 0$ it suffices to show that $\Sigma\, p - (\Sigma\, p)^2 = 0$. To do that we take a slightly more circuitous route. Instead of directly showing that $\Sigma\, p - (\Sigma\, p)^2$ is a solution to the equation that defines the zero stream, we show that both it and the zero stream are solutions of a third equation — an equation which we will only discover after we finish our calculations.

$$\begin{aligned}
& \Sigma\, p - (\Sigma\, p)^2 \\
= \quad & \{ \text{ Definition 24 } \} \\
& (0 : p + \Sigma\, p) - (0 : p + \Sigma\, p)^2 \\
= \quad & \{ \text{ Distributive } \} \\
& (0 : p + \Sigma\, p) - (0 : p^2 + p\, \Sigma\, p + (\Sigma\, p)p + (\Sigma\, p)^2) \\
= \quad & \{ \text{ Definition of } (-) \ \} \\
& 0 : p + \Sigma\, p - p^2 - p\, \Sigma\, p - (\Sigma\, p)p - (\Sigma\, p)^2 \\
= \quad & \{ \ p = p^2, p = p^{\mathsf{T}}, \Sigma\, p = (\Sigma\, p)^{\mathsf{T}} \text{ and transpose of the product } \} \\
& 0 : p + \Sigma\, p - p - p\, \Sigma\, p - (p\, \Sigma\, p)^{\mathsf{T}} - (\Sigma\, p)^2 \\
= \quad & \{ \text{ Simplification and commutativity } \} \\
& 0 : \Sigma\, p - (\Sigma\, p)^2 - p\, \Sigma\, p - (p\, \Sigma\, p)^{\mathsf{T}} \\
= \quad & \{ \text{ Previous calculation: } p\, \Sigma\, p = \alpha \left(\Sigma\, p - (\Sigma\, p)^2\right) \ \} \\
& 0 : \Sigma\, p - (\Sigma\, p)^2 - \alpha \left(\Sigma\, p - (\Sigma\, p)^2\right) - \left(\alpha \left(\Sigma\, p - (\Sigma\, p)^2\right)\right)^{\mathsf{T}}
\end{aligned}$$

From the above, we can observe that $\Sigma\, p - (\Sigma\, p)^2$ is a solution to the admissible stream equation

$$x = 0 : x - \alpha x - (\alpha x)^{\mathsf{T}},$$

which has an unique solution. Since the zero stream is also a solution, we conclude that $\Sigma\, p - (\Sigma\, p)^2 = 0$ and, therefore, $p\, \Sigma\, p = 0$. □

The equality $\Sigma\, p = (\Sigma\, p)^2$ has an interesting interpretation in linear algebra: $\Sigma\, p$, being the stream of projection matrices, is necessarily idempotent, since each of its elements is in fact idempotent. The intuition behind this fact is that once a vector is projected onto a subspace, projecting it again onto the same subspace must produce no further change.

The stream-based formulation revealed that the equivalence between the classical and the modified Gram-Schmidt algorithms is not fundamentally a statement about matrix indexes and nested loops, but rather a consequence of the algebraic properties of the projection matrices involved. By expressing both algorithms as stream equations, the proof reduces to showing that two recursively defined streams satisfy the same characterization.

This perspective avoids the complexity usually associated with index manipulations and induction arguments. Instead, the proof proceeds equationally. In this sense, streams provide not only a more compact notation, but also a semantic view of the algorithm in which equational, induction-free and index-free proofs can be done.

4 Related Work

The classical and modified Gram-Schmidt algorithms are covered extensively in Linear Algebra textbooks. The presentation in Golub and Loan [3] is specially similar to the one we use. Trefethen and Bau [11] present an argument of equivalence between the two algorithms and compare their numerical stability.

Several classical results on program equivalences and Galois connections also inform our approach [1, 2, 6–8]. These works establish how relationships between ordered structures can guide correct and efficient reasoning about programs.

The foundations for Section 3 lie in Hinze's stream calculus [4] and the underlying approach to this style of reasoning originates with the behavioural differential stream equations of Rutten [9, 10].

Hutton and Jaskelioff [5] offer some insight about admissible stream equations from the point of view of metric topology. They highlight that, under a suitable metric, streams form a complete metric space and admissible stream equations are contractions. As a result, the fact that admissible equations have an unique solution can be seen as a corollary of Banach's fixed point theorem.

5 Conclusion and Future Work

In this paper, we have shown how proof techniques originating in the programming languages community can be applied to the study of linear algebra algorithms. By moving from implementations to specifications and from index manipulations to equational reasoning, we obtain derivations and correctness arguments that remain close to the programs themselves.

Our first contribution was a characterization of orthogonalization as a Galois connection. This specification exposes the algebraic structure underlying the orthogonalization process and provides a high-level specification independent of any particular implementation. Rather than viewing Gram-Schmidt as an algorithm to be proven correct in a later proof, orthogonalization is presented as the solution to an abstract optimization problem induced by an adjunction between two ordering structures.

Our second contribution is a derivation of the classical Gram-Schmidt algorithm directly from the aforementioned specification. The familiar projection-and-normalization steps emerge naturally from the derivation, rather than being postulated in advance. In this sense, correctness is built into the construction itself: the resulting algorithm satisfies the specification by design. Another advantage of our approach is that our specification does not require the input vectors to be linearly independent; the algorithm we derived naturally discards the redundant directions.

Our third contribution was a formulation of the classical and the modified Gram-Schmidt algorithms via recursive stream equations. This allowed us to establish the equivalence between the two algorithms through index-free and induction-free equational reasoning. Rather than reason about individual matrix entries, we work directly with recursive definitions and their algebraic properties. The equivalence between the classical and the modified algorithm emerges as a consequence of the properties of projection operators and of the universal properties of the stream constructions involved. The resulting proofs are compact, calculational, and closely aligned with the structure of the programs they describe.

A natural direction for future work is to unify the two parts of the development more thoroughly. Although the Galois-connection specification accommodates linearly dependent inputs, the stream-based treatment still does not.

Artifact Availability

This paper does not provide any research artifacts.

Acknowledgements

We would like to show our gratitude to our friend and colleague Gustavo de Mendonça Freire for helping us with some calculations and valuable suggestions.

We also thank our friend and colleague William Victor Quintela Paixão for his careful proofreading, valuable suggestions, and help in improving the final version of this paper.

Funded by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Project Code 403601/2023-1. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

[1] Roland Backhouse. 2003. *Program Construction: Calculating Implementations from Specifications.* Wiley, Chichester, England.

[2] Edsger Wybe Dijkstra and Willem H. J. Feijen. 1988. *A Method of Programming.* Addison-Wesley, Reading, MA.

[3] Gene H. Golub and Charles F. Van Loan. 2013. *Matrix Computations.* JHU press, Baltimore, MD. A presentation of the Modified Gram-Schmidt algorithm and a discussion of its numerical accuracy can be found in Chapter 5.

[4] Ralf Hinze. 2010. Concrete Stream Calculus: An Extended Study. *Journal of Functional Programming* 20, 5-6 (2010), 463–535. doi:10.1017/S0956796810000213

[5] Graham Hutton and Mauro Jaskelioff. 2013. *Representing Contractive Functions on Streams (Extended Version).* Technical Report. School of Computer Science, University of Nottingham. https://people.cs.nott.ac.uk/pszgmh/contractive-extended.pdf

[6] Shin-Cheng Mu and José Nuno Oliveira. 2012. Programming from Galois Connections. *The Journal of Logic and Algebraic Programming* 81, 6 (2012), 680–704. doi:10.1016/j.jlap.2012.05.003 12th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS 2011).

[7] José Nuno Oliveira. 2023. Why Adjunctions Matter—A Functional Programmer Perspective. In *Recent Trends in Algebraic Development Techniques*, Alexandre Madeira and Manuel A. Martins (Eds.). Springer Nature Switzerland, Cham, 25–59.

[8] Paulo Ricardo Antunes Pereira. 2023. *Back to Programming from Galois Connections.* Master's thesis. Universidade do Minho.

[9] Jan J. M. M. Rutten. 2003. Behavioural Differential Equations: A Coinductive Calculus of Streams, Automata, and Power Series. *Theoretical Computer Science* 308, 1–3 (2003), 1–53. doi:10.1016/S0304-3975(02)00895-2

[10] Jan J. M. M. Rutten. 2005. A Coinductive Calculus of Streams. *Mathematical Structures in Computer Science* 15, 1 (2005), 93–147. doi:10.1017/S0960129504004507

[11] Lloyd N. Trefethen and David Bau. 1997. *Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA. The classical and modified Gram-Schmidt algorithms are discussed in lectures 7 and 8. Their numerical stability is discussed in lecture 19.