SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Formalizing Linear Algebra in UniMath – Gaussian Elimination, Matrix Foundations & Applications

av

Daniel Skantz

2022 - No K16

Formalizing Linear Algebra in UniMath – Gaussian Elimination, Matrix Foundations & Applications

Daniel Skantz

Självständigt arbete i matematik 15 högskolepoäng, grundnivå

Handledare: Peter LeFanu Lumsdaine

2022

Formalizing Linear Algebra in UniMath – Gaussian Elimination, Matrix Foundations & Applications

Daniel Skantz, Stockholm University
BSc. Thesis

Supervised by Peter LeFanu Lumsdaine

2022

Abstract

We formalize fundamental linear algebra in UniMath, a minimal foundation for mathematics programmed in Coq. We develop the necessary background theory, including showing properties of sums over vectors, inverses, transposes and elementary row operations over matrices in order to construct and prove correctness of procedures for Gaussian elimination, including applications for solving systems of linear equations and constructing inverses of matrices. The formalization is done partly over semirings, with properties requiring additive identity, commutativity of multiplication or decidable equality being formalized over fields.

1 Introduction

In this work, we formalize fundamental results in linear algebra over the UniMath project [VAG+]. UniMath is an open-source project in constructive mathematics originally thought of by Vladimir Voevodsky [Voe10] and originally implemented together with co-authors Benedikt Ahrens and Daniel Grayson as a dialect of Coq with an alternative and minimal foundations based on homotopy type theory. We contribute a linear algebra module with material building up to Gaussian elimination and the construction of inverses of matrices. These theories form essential material for development and formalization of other important mathematical results, making a formalization a valuable contribution to UniMath. The most important result from this work is proving the following theorem in a constructive context.

Theorem 7 (Gaussian elimination)**.** *For every matrix A, there exists an invertible matrix B, such that the matrix product BA is in row echelon form.*

```
Definition gaussian_elimination_stmt
    {F : fld} {m n : nat} {A : Matrix _ m n}
    := Σ (B : Matrix _ _ _), (matrix_inverse B)
      × (is_row_echelon (B ** A)).

Theorem gaussian_elimination
    {m n : nat} {A : Matrix F m n}
    : @ gaussian_elimination_stmt _ F _ _ A.
```

The various applications of Gaussian elimination follow as corollaries of having proven this as a master theorem. The notation used in the constructive theorem statement given above, which is presented in the same fashion as all other theorems in this paper, as a direct quotation from the source code, will be introduced in Section 2. To construct B as given above, we first need to contribute theories such as identities on sums of vectors, properties of matrix operations such as associativity of multiplication, invertibility and transposes, and elementary row operations. Having built up such a constructive foundation, we formalize Gaussian elimination. In addition, we use the formalization to prove some fundamental applications, such as the construction of matrix inverses and solving systems of linear equations.

While the current formalization of linear algebra in UniMath is limited, the subject has previously been formalized for different theorem provers. One closely related example is the state-of-the-art in mainline Coq [CCDMS16], where the authors formalize linear algebra over elementary divisor rings, working up until providing proven algorithms for computing the Smith normal form. Providing this type of foundation for UniMath is valuable as it would provide a base for (co)-homology calculation through invariant factor decomposition. Part of our aim with this thesis is to provide a foundation for such work that would be of special interest to UniMath authors. Meanwhile, Gaussian elimination and its corollaries in solving systems of linear equations has a wide range of well known applications within mathematics; thus our contribution could be used to develop other significant theories within UniMath. Furthermore, another part of our contribution is providing a model for how to formalize linear algebra, having provided the first major work on the topic working over UniMath.

In presenting the material for this thesis, we start by covering formalization of matrix fundamentals in Section 3. Having thus worked up to showing the associativity of matrix multiplication, in Section 4, we define elementary row operations. In Section 7, we provide various summation identifies that will prove useful in Section 5 on elimination. Having defined and proven procedures for putting a matrix in row echelon form in Section 5, we use this procedure to construct matrix inverses and solve systems of linear equations in Section 6.

2 Background

Constructive mathematics facilitates use of computerized methods to verify that each step in producing a mathematical theorem has been made without an error. That is one of the founding principles of the UniMath project, providing a foundation on which modern mathematics can be formalized and developed building upon a computerized assurance of correctness for increasingly complex proofs, making for a highly rigorous body of mathematics. In addition, the equivalence of a computer program to a mathematical statement and its associated proof allow the mathematician and programmer to use powerful computational methods, for instance to verify complicated algorithms and extract a corresponding program, to develop certified compilers, and to employ automation for proof search. There is moreover rich mathematical content involved in computer friendly foundations of mathematics and mathematical formalization, involving frontier research in category theory, type theory and homotopy theory.

The UniMath kernel is a minimalistic design that uses a subset of constructs found in the Coq theorem prover, notably eschewing general inductive types. UniMath as a computer system is a realization of a theoretical system that is intensional Martin-Löf. This entails having the well-known constructs of dependent pair type $\sum$, dependent function type $\prod$, coproduct $\coprod$ and identity type. In addition to the well known type theory constructs, a key contribution of Unimath is a mathematical foundation built on univalence, giving notions of equivalence and equality with beneficial computational properties. For a treatment of univalent theory, the reader can refer to the HoTT book [Uni13], and for a gentle introduction to UniMath in particular, they can refer to Grayson's primer in [Gra18]. To follow the material presented in this thesis, some familiarity with general type theory and constructive mathematics will be sufficient. To make definitions and theorems presented in source code format throughout the thesis understandable to the reader, we give the following brief introduction to some useful constructs and notations.

The base types of UniMath follow the Curry-Howard correspondence between logic and programming, where the sigma type $\sum$ has a corresponding interpretation in existential quantification $\exists$. The product type $\prod$ corresponds to universal quantification $\forall$, and the coproduct $\coprod$ corresponds to disjunction; thus Theorem 7 might be read as saying that there exists ($\sum$) a matrix B, such that B is invertible (stronger, we provide an explicit inverse), and ($\times$) such that the product BA is in row echelon form. The product type $\times$ is thus a special case of the dependent sum, corresponding to conjunction in the logic setting. Appearing in e.g. Definition 1 are symbols from the UniMath lambda calculus, where a function such as $\lambda\ i\ j,\ i^2 + j$ is the same as $(i, j) \mapsto i^2 + j$ in traditional mathematics. All functions in Coq are curried. This means a function such as $f : (A \times B \times C) \mapsto D$ has a representation as a function of one argument returning a function in terms of the remaining arguments $f : A \mapsto (B \mapsto (C \mapsto D))$.

The linear algebra material formalized in this thesis mostly follows a standard exposition that could be found in an textbook on linear algebra (such as [Str06]; we formalize material that is in part covered by the first few chapters of this book). There are a few potential deviations of note. We seek to formalize material at a high level of generality, defining operations on vectors and matrices over semirings where possible. When additive identity or commutativity of multiplication, or decidable equality is needed, we tend to use fields, or on occasion, commutative rings. Useful notations to keep in mind while reading our paper is that R tends to refer to a fixed semiring, while F tends to refer to a fixed field. The successor of a natural number n is interchangeably referred to as $S\ n$ or $n+1$. We might overload the $\sum$ symbol to also mean a finite sum over a semiring when that would be unambiguous. Indexing in UniMath starts at zero, as in most programming languages. Another convention we use is that matrix multiplication has as shorthand notation ** when presented in UniMath code.

3 Matrices

In the UniMath project there are presently theories developed over vectors and matrices. A formalization of matrices was previously contributed to UniMath by Langston Barrett. In his matrix module, matrices are defined as vectors of vectors, where a vector in turn is a function $[\![n]\!] \to X$ for some natural number n, the standard finite set $[\![n]\!]$ (also referred to as `stn n`, the sigma type of natural numbers with a proof that they are less than n) and X being of arbitrary type. Theories on pointwise and entrywise binary operations are developed within this module on a general level. Later, specialization on semirings (rings without necessarily an additive inverse), referred to in UniMath as rigs, is made to define operations such as matrix multiplication. Fix a semiring R for the definitions and theorems in this section.

Definition 1 (Matrix Multiplication)**.** For A an $m \times n$ matrix, and B an $n \times p$ matrix, the matrix product AB is defined using the formula $AB_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$.

```
Definition matrix_mult {m n : nat} (mat1 : Matrix R m n)
                       {p : nat} (mat2 : Matrix R n p) : (Matrix R m p) :=
  λ i j, ∑ ((row mat1 i) ^ (col mat2 j)).
```

Above, the infix operator stands for the pointwise application of a binary operation, in our case that of multiplication. In turn, sums are defined upstream as iterated binary operations over semirings, with the base case being additive identity.

```
Local Notation ∑ := (iterop_fun rigunel1 op1).
```

The existing matrix library in UniMath leaves off with the following formalization of the property of matrix multiplication associativity, along with a note encouraging the contribution of a future proof of this statement.

```
Definition matrix_mult_assoc_statement : UU :=
    ∏ (m n : nat) (mat1 : Matrix R m n)
      (p : nat) (mat2 : Matrix R n p)
      (q : nat) (mat3 : Matrix R p q),
    ((mat1 ** mat2) ** mat3) = (mat1 ** (mat2 ** mat3)).
```

We found this a suitable starting point for formalizing theories over matrices and the property was later used to prove several useful properties pertaining to Gaussian elimination. In proving associativity corresponding to Definition 3, we first prove a number of properties of sums over semirings. Several of these properties can be found described in Section 7. We tend to use proof by induction to show identities over sums, and for showing equivalences between vectors and matrices, we tend to do this by showing equivalence on each entry. Putting the lemmata proven in Section 7 together, we are able to prove the associativity of matrix multiplication as follows. For illustrative purposes, we present the proof in UniMath code.

Lemma 2 (Matrix Multiplication Associativity)**.**

We now show that it is indeed true that for all matrices A, B, C of dimensions $m \times n$, $n \times p$, and $p \times q$ respectively, we have $(AB)C = A(BC)$.

```
Lemma matrix_mult_assoc :
    ∏ {m n : nat} (mat1 : Matrix R m n)
      {p : nat} (mat2 : Matrix R n p)
      {q : nat} (mat3 : Matrix R p q),
    ((mat1 ** mat2) ** mat3) = (mat1 ** (mat2 ** mat3)).
  Proof.
    intros; unfold matrix_mult.
    apply funextfun; intro i; apply funextfun; intro j.
    etrans.
    2: { symmetry.
         apply maponpaths, funextfun. intros k.
         apply sum_is_ldistr. }
    etrans.
      { apply maponpaths. apply funextfun. intros k.
        apply sum_is_rdistr. }
    rewrite interchange_sums.
    apply maponpaths, funextfun; intros k.
    apply maponpaths, funextfun; intros l.
    apply rigassoc2.
  Defined.
```

This is a typical proof in our UniMath library displaying use of various standard tactics. Intros will introduce the universally quantified matrices as hypotheses. The *funextfun* property is short for functional extensionality, a constructively non-trivial property that for every function f and g, if for all i, $f(i) = g(i)$, then $f = g$ (in

UniMath, it is a consequence of the univalence axiom). Thus applying this property twice will transform the proof goal into an entrywise obligation. The *apply maponpaths* tactic application allows us to show that $f(i) = f(j)$ from $i = j$. Moreover, *etrans* allows us to rewrite on either side of some equalities, where we apply the properties of distributivity of multiplication over sums; the latter, and some other constructively non-trivial properties, are proven in Section 7.

Having formalized up to this point, we have enough background material to also cover fundamental material on matrix inverses, transposes and properties on their products; thus formalizing material covered e.g. in part in Section 1.6 of [Str06].

4 Elementary row operations

Having proven properties of matrices such as the associativity of matrix multiplication in the previous section, we can now define matrices corresponding to the three elementary row operations of interchanging, adding, or multiplying a row by a scalar as multiplication by an appropriately constructed invertible matrix. This in turn allows us to define multistep operations on a matrix A that retain invertibility; moreover, from the product $E_n ... E_2 E_1 A$ of multiplying elementary matrices by A, the input matrix can be obtained again by left multiplication with the matrix $E_1^{-1} E_2^{-1} ... E_n^{-1}$.

The elementary row operation matrices are constructed in a canonical way, which we describe in the following; on the left-hand side, the matrix corresponding to a row operation (when multiplied on the right by the input matrix), and on the right side its inverse.

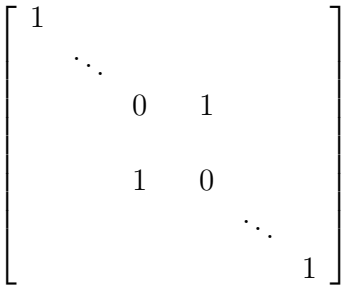
$E_1(r, m)$: Multiplication of a row r by a non-zero scalar m.

$$\begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & m & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

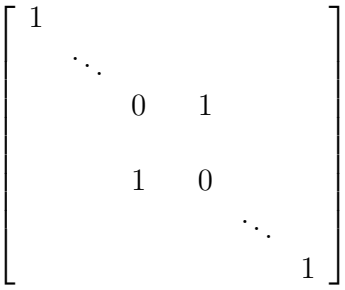
Inverse of scalar row multiplication; inverse element of m at same entry.

$$\begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & m^{-1} & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}$$

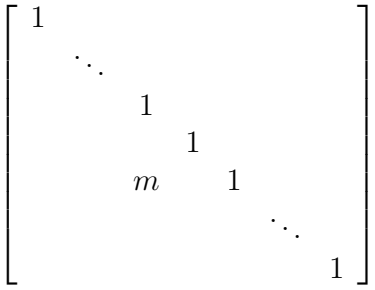
$E_2(r_1, r_2)$: Switching two rows.



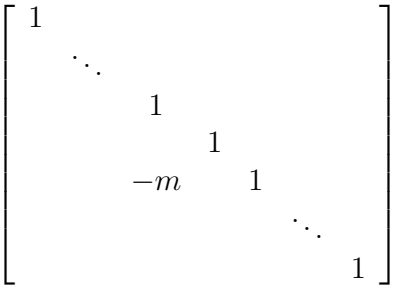
Inverse of row switch (same as left).



$E_3(r_1, r_2, m)$: Adding a scalar multiple m of one row to another.



Inverse of row addition; inverse element of m at same entry.



The three matrix constructions, the signatures of which we leave out for brevity, are each respectively accompanied by one lemma demonstrating their invertibility.

Lemma 3 (Invertibility of scalar multiplication matrix)**.**

$E_1(r, s)$ is invertible if $s \neq 0$, and $E_1(r, s)^{-1} = E_1(r, s^{-1})$.

```
Lemma scalar_mult_matrix_is_inv
    { n : nat }
    ( i : [| n |]%stn ) ( s : F ) ( ne : s != 0%ring )
  : @matrix_inverse F n (mult_row_matrix s i).
```

Lemma 4 (Invertibility of switch row matrix)**.**
$E_2(r_1, r_2)$ is invertible and $E_2(r_1, r_2)^{-1} = E_2(r_2, r_1)$.

```
Lemma switch_row_matrix_is_inv
    { n : nat } ( r1 r2 : [| n |]%stn )
    : @matrix_inverse F n (switch_row_matrix r1 r2).
```

Lemma 5 (Invertibility of add row matrix)**.**

$E_3(r_1, r_2, s)$ is invertible if $r_1 \neq r_2$, and $E_3(r_1, r_2, s)^{-1} = E_3(r_1, r_2, -s)$.

```
Lemma add_row_matrix_is_inv { n : nat } ( r1 r2 : [| n |]%stn )
  (r1_neq_r2 : r1 ≠ r2) ( s : F )
   : @matrix_inverse F n (add_row_matrix r1 r2 s).
```

Proofs. Proving that $A * A^{-1} = I$ for each of the three respective elementary matrices A is done by case analysis on rows and columns, showing that $\sum_{k=1}^{n} A_{i,k} * A^{-1}_{k,j} = 1$ for $i = j$ and $\sum_{k=1}^{n} A_{i,k} * A^{-1}_{k,j} = 0$ for $i \neq j$ respectively.

To reduce the work in carrying out repeated case distinction in the formalization, we additionally make use of the following properties. Let $E_1(m, r)$ denote the scalar multiplication matrix, $E_2(r_1, r_2)$ the row switching matrix, and $E_3(s, r_1, r_2)$ the row addition matrix. The following easy identities were useful in the constructive proofs.

$$E_1(r, s_1)E_1(r, s_2) = E_1(r, s_2)E_1(r, s_1)$$
$$E_1(r, s_1)E_1(r, s_2) = E_1(r, s_1 s_2)$$
$$E_3(r_1, r_2, s_1)E_3(r_1, r_2, s_2) = E_3(r_1, r_2, s_1 + s_2)$$
$$E_3(r_1, r_2, s_1)E_3(r_1, r_2, s_2) = E_3(r_1, r_2, s_2)E_3(r_1, r_2, s_1)$$ □

In addition, we provide a lemma for the invertibility of product of two invertible matrices.

Lemma 6 (Invertibility of product of invertible matrices)**.**
If A and B are invertible matrices, the product AB is invertible too. An inverse of AB is $B^{-1}A^{-1}$.

```
Lemma inv_matrix_prod_is_inv {n : nat} (A : Matrix R n n)
    (A' : Matrix R n n) (pa : matrix_inverse A) (pb : matrix_inverse A' ) :
    (matrix_inverse (A ** A' )).
```

Proof. We can first show this for left and right-invertible matrices. If A, B are left invertible, the left inverse of AB is $B_L^{-1}A_L^{-1}$ by Lemma 2. Correspondingly, for right invertible A and B, $(AB)_R^{-1} = B_R^{-1}A_R^{-1}$. Consequentially, for two-side invertible matrices A and B, the inverse of AB is $B^{-1}A^{-1}$. □

Using Lemma 6 inductively, this allows us to show that any product $E_n ... E_2 E_1 B$, for an invertible B and elementary row operation corresponding matrices E_i, is also invertible. As described on page 55 in [Str06], this will later allow us to define Gaussian elimination as a sequence of multiplications by elementary matrices.

5 Elimination

Having previously established lemmata pertaining to elementary row operations, we can proceed to define procedures for carrying out Gaussian elimination. Roughly, our idea is to clear a matrix entry by means of an elementary row operation. For computational reasons, proving the equivalence between this operation and a direct manipulation of the target row proved beneficial. Having thus defined a procedure for clearing an entry of the matrix, we provide an inductive procedure for clearing a whole column segment. This in turn is done inductively over newly chosen pivots,

where the column-wise first non-zero element is chosen and switched to the current row of iteration before clearing the next column segment.

The main theorem that we prove in this module is that of Theorem 7 presented in the background of this thesis, which we restate here.

Theorem 7 (Gaussian elimination)**.** *For every matrix A, there exists an invertible matrix B, such that the matrix product BA is in row echelon form.*

```
Definition gaussian_elimination_stmt
    {F : fld} {m n : nat} {A : Matrix _ m n}
    := Σ (B : Matrix _ _ _), (matrix_inverse B)
      × (is_row_echelon (B ** A)).

Theorem gaussian_elimination
    {m n : nat} {A : Matrix F m n}
    : @gaussian_elimination_stmt _ F _ _ A.
```

In literature and practice, there are various definitions of row echelon form. We choose to work over the following form.

Definition 8. Row echelon form.
A matrix A is said to be in row echelon form if

1. Each leading entry is located strictly to the right of all leading entries of previous rows.
2. Every empty row is located below every non-empty row.

In UniMath, we phrase this as below.

```
Definition is_row_echelon {F : fld} {m n : nat} (mat : Matrix F m n) :=
    ∏ i_1 i_2 : ⟦ m ⟧%stn,
    (∏ j_1 j_2 : ⟦ n ⟧%stn,
      is_leading_entry (mat i_1) j_2
      → i_1 < i_2 → j_1 ≤ j_2
      → mat i_2 j_1 = 0%ring)
    × ((mat i_1 = const_vec 0%ring)
      → (i_1 < i_2)
      → (mat i_2 = const_vec 0%ring)).
```

Note the slight difference between the classical and constructive definitions. In Lemma 26, we will show that the UniMath formulation implies classical. The definition of a leading entry is given as follows.

Definition 9. Leading entry. An entry of a vector is its leading entry if it is non-zero, and all previous entries are zero.

```
Definition is_leading_entry {F : fld} {n : nat} (v : Vector F n) (i_1 : ⟦ n ⟧ %stn)
    := (v i_1 != 0%ring)
      × (∏ i_2 : ⟦ n ⟧ %stn, i_2 < i_1 → (v i_2) = 0%ring).
```

We now attend to defining procedures for calculating the leading entry of a vector, that being the lowest index i such that $v_i \neq 0$. To cover the case for when there is no leading entry, equivalently when the vector considered is all-zero, we wrap some computations in the `maybe` type, which in UniMath gives a definition as the coproduct `just (stn n)` ⨿ `nothing`. For computational reasons, the leading entry procedure is formalized as the converse of selecting the highest index i such that $v_i \neq 0$; in UniMath, this is expressed as taking the dual element of that element.

Definition 10. Dual element.

```
Definition dualelement {n : nat} (i : ⟦ n⟧ ) : ⟦ n⟧ .
Proof.
  induction (natchoice0 n) as [H | H].
  - exact (make_stn n (n - 1 - i) (fromempty (dualelement_0_empty i H))).
  - exact (make_stn n (n - 1 - i) (dualelement_lt i n H)).
Defined.
```

Definition 11. Leading entry computation.

```
Definition leading_entry_compute_internal
    { n : nat } (v : Vector F n) (iter : ⟦ S n ⟧ %stn)
    : maybe (⟦ n ⟧ )%stn.
  Proof.
    destruct (leading_entry_compute_dual_internal
      (λ i : ⟦ n ⟧ %stn, v (dualelement i)) iter) as [s | ?].
    - exact (just (dualelement s)).
    - exact nothing.
  Defined.
```

Now, we formalize the underlying dual procedure of Definition 12 as follows.

Definition 12. Leading entry computation (the internal, dual version).

```
Definition leading_entry_compute_dual_internal
    { n : nat } (v : Vector F n) (iter : ⟦ S n ⟧%stn)
    : maybe (⟦ n ⟧%stn).
  Proof.
    destruct iter as [iter lt].
    induction iter.
    { exact nothing. }
    simpl in lt.
    destruct (fldchoice0 (v (iter,, lt))).
    - refine (IHiter _).
      apply (istransnatlth _ _ _ lt (natgthsnn n)).
    - exact (just (iter,, lt)).
  Defined.
```

And as customary, we provide equivalences between the two, expressed as equivalences on the index of the leading and dual leading entry, and as implication from having a leading entry to having the dual. In proving properties over inductively defined procedures over natural numbers, such as computing the leading entry in Definition 11, we use a few common proof techniques. We tend to state a induction hypothesis on all rows or columns greater than, or less than, a given separator for an internal partial version of a lemma used for proof purposes (for the interface to the programmer, we state another version of the lemma where the separator over which we internally recurse is set to the dimension of the given matrix or vector, thus fully carrying out the procedure in question). Then we set out to prove the induction step on the target row or column. To facilitate this, we make plentiful use of the `nat_rect_step` identity.

Lemma 13. Stepwise identity on the `nat_rect` induction principle.

```
Lemma nat_rect_step (P : nat → UU) (p0 : P 0) (IH : ∏ n, P n → P (S n)) n :
  nat_rect P p0 IH (S n) = IH n (nat_rect P p0 IH n).
```

The leading element computation is to satisfy the following two properties: it returns the leading element, or if none exists, the input vector contains all zero elements. We provide both directions of each respective implication, and provide the proofs and equivalences over both the leading entry procedure and its dual version. An example of these lemmata is the following.

Lemma 14. If the procedure in Definition 11 returns i on an input vector v, then i is the leading element of v.

```
Lemma leading_entry_compute_internal_correct1
    {n : nat} (v : Vector F n) (i : ⟦ n ⟧%stn)
    (eq : (leading_entry_compute_internal v (n,, natgthsnn n)) = (just i))
    : is_leading_entry v i.
```

Proving these lemmata involves straightforward but somewhat mechanical proofs by induction and case analysis.

Having thus formalized the notions of row echelon form and leading entry, we turn to the elimination procedure and operations on matrices. Clearing an entry by means of elementary row operations entails adding a multiple of a pivot row to a target row. In the following, the pivot element is referenced by indices k_i and k_j for row and column respectively, with the multiple $-A_{i,k_j}/A_{k_i,k_j}$ being selected such that $(A_{k_i} + A_i)_{k_j} = 0$. Note that we define the field inverse as a total function and handle the case that $A_{k_i,k_j} = 0$ upstream.

We first define a procedure for clearing an entry by means of left multiplication by an elementary matrix defined in Section 4. This corresponds to page 24 in [Str06].

Definition 15. Clearing one matrix entry as left multiplication by an elementary matrix.

```
Definition gauss_clear_column_step
    {m n : nat}
    (k_i : ([[ m ]]%stn))
    (k_j : ([[ n ]]%stn))
    (i : (stn m))
    (mat : Matrix F m n)
    : Matrix F m n.
  Proof.
    destruct (stn_eq_or_neq i k_i) as [? | ?].
    - exact mat.
    - refine ((add_row_matrix k_i i _)%ring ** mat).
      exact (- ((mat i k_j) * fldmultinv' (mat k_i k_j)))%ring.
  Defined.
```

In addition, we define a procedure `gauss_clear_column_step'` that we prove is equivalent in output to Definition 15, but that is also computationally simpler.

Definition 16. Clearing one matrix entry as direct manipulation of target row.

```
Definition gauss_clear_column_step'
    {m n : nat}
    (k_i : ([[ m ]] %stn))
    (k_j : ([[ n ]] %stn))
    (i : (stn m))
    (mat : Matrix F m n)
    : Matrix F m n.
  Proof.
    destruct (stn_eq_or_neq i k_i) as [? | ?].
    - exact mat.
    - exact (gauss_add_row mat k_i i
        (- ((mat i k_j) * fldmultinv' (mat k_i k_j)))%ring).
  Defined.
```

This is a typical approach we use for multiple procedures. It is sufficient to show that the procedure is output-equivalent to multiplication by elementary matrices, while carrying out proofs under rewriting to a simpler procedure.

Having defined procedures for clearing an entry of a matrix, and proving the equivalence of the procedures, we define another procedure to carry out this operation recursively in order to clear a multi-element segment of a column below the pivot entry.

Definition 17 (Clearing of a column segment)**.**

```
Definition gauss_clear_column { m n : nat }
    (mat : Matrix F m n) (k_i : ([[ m ]] %stn))
    (k_j : ([[ n ]] %stn)) (row_sep : [[ S m ]] %stn)
    : Matrix F m n.
  Proof.
    destruct row_sep as [iter lt].
    induction iter as [ | iter gauss_clear_column_IH ].
    { exact mat. }
    destruct (natgthorleh iter k_i) as [gt | leh].
    2: {exact mat. }
    refine (gauss_clear_column_step k_i k_j (iter,, lt) _ ).
    refine (gauss_clear_column_IH _).
    refine (istransnatlth _ _ _ (natgthsnn iter) _).
    assumption.
  Defined.
```

This procedure recursively applies the column clearing step in Definition 15 for each index between k_i and `row_sep`. For computational reasons, an equality analogous to that between Definitions 15 and 16 was useful to prove properties over this procedure. We give a few invariants over Definition 17 in the following. In the following lemma,

we prove that considering any target row r, the output of the procedure is equal to the pointwise application of the step procedure (if the index is larger than the pivot element, and less than the iteration parameter). This will allow us to easily show that each element of the column segment is indeed cleared by our procedure.

Lemma 18 (Clearing of a column segment – row equals output of step function)**.**

```
Lemma gauss_clear_column_inv1
    { m n : nat } (k_i : (⟦ m ⟧ %stn))
    (k_j : stn n) (row_sep : ⟦ S m ⟧ %stn)
    (mat : Matrix F m n)
    : ∏ r : (⟦ m ⟧ %stn), r < row_sep → k_i < r →
      ((gauss_clear_column mat k_i k_j row_sep) r
      = (gauss_clear_column_step' k_i k_j r mat) r).
```

Proof. For readability, let $A =$ `mat`, $f(t, A) =$ `gauss_clear_column_step' n k_i k_j t mat`, and let $g(t, A) =$ `gauss_clear_column_old mat k_i k_j t`.

We want to show that $f(t, A)_r = g(t, A)_r$ if $r < t$ and $k_i < r$. We prove this by induction over t and case distinction on $r < t + 1$ or $r = t + 1$. In the first case, the equality follows from Definition 15 and the induction hypothesis. Assume $r = t + 1$. We have $g(t + 1, A)_r = f(t + 1, g(t, A))_r$ by `nat_rect_step`. Moreover, $f(t + 1, g(t, A))_r = f(t + 1, A)_r$ follows from the fact that $g(t, A) = A$ if $t < r$. □

Having defined the entry-wise and column-wise procedures for zeroing parts of a matrix, we turn to do this over multiple rows; this entails proceeding row-wise by selecting the first uncleared column, that is, one having a non-zero column entry at the current row of iteration.

Lemma 19 (Procedure for selecting uncleared column)**.**
We can calculate the left-most non-zero column segment of any $m \times n$ matrix A, observing only rows at index larger than an input parameter r.

```
Lemma select_uncleared_column_internal
    {m n : nat} (mat : Matrix F m n)
    (row_sep : ⟦ m ⟧ %stn) (col_iter : ⟦ S n ⟧ %stn) (p : n > 0)
    : coprod
      (exists_first_uncleared mat row_sep col_iter)
      (lower_left_zero mat row_sep col_iter).
```

For readability, the constructive definitions of properties `exists_first_uncleared` and `lower_left_zero` are left out. In essence, they entail finding pivot indices i, j (with $r \leq i$) such that i, j are the lowest indices for which $A_{i,j} \neq 0$, or evidence that for all i, j, $A_{i,j} = 0$. This can be done by using Definition 12 as a sub-procedure.

Now, we formalize the notion of clearing a row as first selecting a pivot entry, which will either be the leading element of the row to clear, or the first leading element

encountered on a later row, as found by the procedure in Definition 19. The non-internal version used in the following proof is as customary the internal version with the row separator parameter fixed to the value m (the length of a column).

Definition 20 (Row-wise procedure for echelon form)**.**
The procedure defined in Definition 17 outputs a matrix where for any given row r, the output matrix is identical to the result of calling the procedure in Definition 16 with r as the target parameter.

```
Definition gauss_clear_row
    { m n : nat }
    (mat : Matrix F m n)
    (row : ('[ m ] %stn))
    : Matrix F m n.
  Proof.
    destruct (natchoice0 n) as [contr_eq | p].
    { unfold Matrix, Vector; intros; apply fromstn0.
      rewrite contr_eq; assumption. }
    destruct (select_uncleared_column F mat row p) as [some | none].
    2: {exact mat. }
    set (piv_col := pr1 some).
    set (piv_row := pr1 (pr2 (pr2 some))).
    refine (gauss_clear_column _ row piv_col (m,, natlthnsn m)).
    exact (gauss_switch_row mat row piv_row).
  Defined.
```

Here, the notion of clearing a row i entails first finding the first column j with a non-zero entry at (i, j); if no such row exists (as certified by an invariant on the `select_uncleared_column` procedure), the matrix is in partial echelon form for all rows r with $r \geq i$, and the original input matrix is returned. The finishing switch row procedure is only algorithmically necessary if the selected pivot does not lie on our current row, but to avoid another case distinction we perform the "switch" regardless.

This procedure is defined in order to put a matrix in row echelon form. For fulfilling the first criteria given in Definition 20, we give the following defining lemma that shows the stepwise property that if the matrix is partially eliminated up until row m, the procedure will advance the elimination by one row, thus having the input matrix partially eliminated until row $m + 1$.

Lemma 21 (Clearing row – induction lemma)**.**
The procedure defined in Definition 22 takes an $m \times n$ matrix A, and outputs a matrix where, if each leading entry of the input matrix A is to right of all leading entries above up until the r:th row, with $r + 1 < n$, then the leading entry of row $r + 1$ in A', if not an empty row, is to the right of all previous rows. [This is proving the induction step for being in row echelon form given in Definition 8.]

```
Lemma gauss_clear_row_inv2
    { m n : nat } (mat : Matrix F m n) (p : n > 0)
    (row_sep : ([[ S m ]] %stn)) (p' : row_sep < m)
    : is_row_echelon_partial_1 mat p row_sep
    → is_row_echelon_partial_1 (gauss_clear_row mat (pr1 row_sep,, p' )) p
        (S (pr1 row_sep),, p' ).
```

Proof sketch. This proof is essentially a three part case distinction on the column index j when observing the echelon form of $A'_{i,j}$. If j is less than the leading entry of A_r, then $A'_{i,j}$ is zero by definition. If j is larger than the leading entry, we have again by definition a (partial) row echelon form. In the last case, j is equal to the leading entry of the row at index r. In that case, $A'_{i,j} = 0$ follows since for any row $i \geq r$, A_i is unchanged by the row clearing procedure. □

Having defined a procedure for clearing a row, the final elimination procedure is obtained by recursively clearing multiple rows. Starting with the uppermost row, locating the leading element if any, pivoting if there is a nearer leading element in a row below, clearing the associated column segment below, and proceeding likewise over successive rows, until we are either at the last row or have encountered an all-zero row.

Definition 22 (Clearing multiple rows)**.**

```
Definition gauss_clear_rows_up_to
    { m n : nat }
    (mat : Matrix F m n)
    (row_sep : ([[ S m ]] %stn))
    : (Matrix F m n).
  Proof.
    destruct row_sep as [row_sep row_sep_lt_n].
    induction row_sep as [ | row_sep gauss_clear_earlier_rows].
    {exact mat. }
    refine (gauss_clear_row _ (row_sep,, row_sep_lt_n)).
    refine (gauss_clear_earlier_rows _).
    exact (istransnatlth _ _ _ (natgthsnn row_sep) row_sep_lt_n ).
  Defined.
```

This is a standard recursive procedure applying Definition 20 repeatedly until all rows less than `row_sep` are cleared. We will show that this entails putting the input matrix

in (partial) row echelon form.

Lemma 23 (Clearing multiple rows – row echelon 1)**.**
The procedure defined in Definition 22 outputs a matrix where every leading entry has a higher index than the leading entry of any previous row. [This is the first criterion for being in row echelon form given in Definition 8.]

```
Lemma gauss_clear_rows_up_to_inv1
    { m n : nat } (mat : Matrix F m n)
    (p : n > 0) (row_sep : ([[ S m ]]%stn))
    : is_row_echelon_partial_1
        (gauss_clear_rows_up_to mat row_sep) p row_sep.
```

The above follows essentially from the induction step proven in Lemma 21. That which follows now is a lemma that shows that our procedure in Definition 22 satisfies the second condition of row echelon form.

Lemma 24 (Clearing multiple rows – row echelon 2)**.**
The procedure defined in Definition 22 inputs a matrix A and outputs a matrix A' where for each pair of rows i_1, i_2 of A', if $i_1 < i_2$, and i_1 is an all-zero row, i_2 is too.

```
Lemma gauss_clear_rows_up_to_inv0
    { m n : nat } (mat : Matrix F m n) (row_sep : ([[ S m ]]%stn)) (p : n > 0)
  : ∏ i_1 : [[ m ]]%stn, i_1 < row_sep
  → (gauss_clear_rows_up_to mat row_sep) i_1 = const_vec 0%ring
  → ∏ i_2 : [[ m ]]%stn, i_1 < i_2
  → (gauss_clear_rows_up_to mat row_sep) i_2 = const_vec 0%ring.
```

Proof. Induction on `row_sep`. Case analysis as follows.
If `select_uncleared_column` returns none, we are done according to Definition 19; hence assume `select_uncleared_column` returns a column j s.t. there exists a row i and $A'_{i,j} \neq 0$. Let $t =$ `row_sep` for brevity. *Case 1 :* $i_1 < t$ follows essentially from the induction hypothesis. *Case 2 :* $i_1 = t$ follows by contradiction. □

Having shown that our procedure does put a given matrix in row echelon form, verifying Theorem 7, we furthermore show that our constructive statement implies the classical definition.

Lemma 25 (Row echelon form – constructive implies classical)**.**
If a matrix is in row echelon form in our setting, it is in row echelon form in the classical formulation too.

```
Lemma is_row_echelon_1_eq
    { m n : nat } (mat : Matrix F m n)
    : is_row_echelon mat
    → ∏ i_1 i_2 : ⟦ m ⟧ %stn, ∏ j_1 j_2 : ⟦ n ⟧ %stn,
       i_1 < i_2
    → is_leading_entry (mat i_1) j_1
    → is_leading_entry (mat i_2) j_2
    → j_1 < j_2.
```

This follows directly from Definition 11.

Row echelon form also implies upper triangularity, which is a notion we make use of in Section 6.

Lemma 26 (Row echelon implies upper triangular)**.**
Any $m \times n$ matrix A that is in row echelon form is also in upper triangular form.

```
Lemma row_echelon_to_upper_triangular
    { m n : nat }
    (mat : Matrix F m n)
    :  is_row_echelon mat
    → @ is_upper_triangular F _ _ mat.
```

Proof. Another induction proof with induction on `row_sep` on the internal, partial version of the lemma. Let t = `row_sep` for brevity. Assume that we for a given parameter t, with $t < n$, want to show that the notion of row echelon form *up to* a given row separator t implies a corresponding upper triangularity for every row *up to* the same separator. By the induction hypothesis and definition of upper triangularity, we have that the leading entry le of A_t satisfies $le \geq t$. By definition of row echelon form, we also have that $\forall j' \leq t$, $A_{t+1,j'} = 0$. Thus A is also (partially) upper triangular up until row $t+1$. The form in the signature above now follows from fixing $t = n$. □

6 Elimination applications

We prove a few of the plentiful corollaries resulting from having in the previous section defined an elimination procedure for reducing a matrix to row echelon form. Having such a procedure allows us to solve systems of equations on the form $Ax^T = b^T$ for x. In particular, this allows us to construct the inverse of any invertible square matrix by solving for $b = e_i$. In this section, definitions and proofs are stated over a fixed field F. We also consider only square matrices for the remainder of this section.

The main result in this module is proving the following theorem.

Theorem 34 (Invertibility)**.** *We can either produce the inverse of a given $n \times n$ matrix, or show that it is non-invertible.*

```
Definition matrix_inverse_or_non_invertible_stmt
  { n : nat } {F : fld}
  (A : Matrix F n n)
  := coprod (@ matrix_inverse F n A)
            (@ matrix_inverse F n A → empty).

Theorem matrix_inverse_or_non_invertible { n : nat }
    (A : Matrix F n n)
    : @ matrix_inverse_or_non_invertible_stmt _ _ A.
```

The procedure in Definition 22 will put an input matrix in row echelon form by means of elementary row operations equivalent to multiplication by invertible matrices. This implies the output matrix is upper triangular. In the case that the output matrix has a zero element in its diagonal, we will show that this implies non-invertibility. In the other case, we can construct the inverse as per the below signature.

Theorem 27 (Back substitution)**.**
There exists a procedure, such that for every upper triangular matrix A of dimensions $n \times n$ with all non-zero diagonal, and every length n vector b, this procedure returns x such that $Ax^T = b^T$.

```
Definition back_sub_stmt
  { n : nat } {F : fld}
  (mat : Matrix F n n)
  (vec : Vector F n)
  (ut : @ is_upper_triangular F _ _ mat)
  (df : @ diagonal_all_nonzero F _ mat)
  := Σ f : (Matrix F n n → Vector F n → Vector F n),
    (@ matrix_mult F _ _ mat _ (col_vec (f mat vec))) = (col_vec vec).

Lemma back_sub_inv0
    { n : nat }
    (mat : Matrix F n n) (vec : Vector F n)
    (ut : @ is_upper_triangular F _ _ mat)
    (df : @ diagonal_all_nonzero F _ mat)
    : back_sub_stmt mat vec ut df.
```

Showing that an invertible upper triangular matrix must have only non-zero elements throughout the main diagonal can be done in multiple ways. We proceed slightly differently from [Str06] by not explicitly considering row, column or null spaces, but constructing witnesses through elementary methods involving previously shown properties on products of invertible or triangular matrices. In our case, we construct such evidence by carrying out elimination on the transpose of the reduced (upper triangular) matrix.

Lemma 28 (Invertibility implies non-zero diagonal)**.**
An invertible, upper triangular matrix has necessarily no zero elements in its diagonal.

```
Lemma invertible_upper_triangular_to_diagonal_all_nonzero
    {n : nat }
    (A : Matrix F n n)
    (p : @ is_upper_triangular F n n A)
    (p' : @ matrix_inverse F n A)
    : (@ diagonal_all_nonzero F n A).
```

Proof Sketch. Essentially, we are using the following lemma to differentiate between invertible and non-invertible matrices.

```
Lemma zero_row_to_non_invertibility { n : nat } (A : Matrix F n n)
      (i : ⟦ n ⟧ %stn) (zero_row : A i = (const_vec 0%ring)) :
  (@ matrix_inverse F n A) → empty.
```

As a test for invertibility, we can carry out another round of elimination on the transpose of the output matrix of Definition 22. If the procedure in Definition 22 returns a matrix A', the product of a sequence of invertible matrices $E_n...E_2E_1$ by the transpose $(A')^T$ of A' must also be invertible. We can define such a sequence $E_n...E_2E_1(A')^T$ such that if A is invertible, $E_n...E_2E_1(A')^T$ has all non-zero diagonal, and if A' has a non-zero element in its diagonal, $E_n...E_2E_1(A')^T$ has an all-zero row. Indeed, multiplication by the sequence $E_n...E_2E_1$ corresponds to the procedure Elimination given in Definition 22. □

Thus operating from an upper triangular, invertible matrix A, its inverse A^{-1} can be found by a back-substitution formula. Working from the last element to the first, with i in the range $n-1$ down to 1, the element x_i of a given solution to the equation $Ax^T = b^T$ is calculated as $x_i = (b_i - \sum_{j=i+1}^{n} A_{i,j}x_j)/A_{i,i}$, with $x_n = b_n/A_{i,j}$. In UniMath, we adapt the summation bounds for computational simplicity.

Definition 29 (Back substitution step)**.**

```
Definition back_sub_step { n : nat } ( iter : (⟦ n ⟧ )%stn )
    (mat : Matrix F n n) (b : Vector F n) (vec : Vector F n) : Vector F n.
  Proof.
    intros i.
    destruct (nat_eq_or_neq iter i) as [? | ?].
    - exact (((vec i) * fldmultinv' (mat i i))
          - ((∑ (mat i ^ b) - (b i)* (mat i i))
          * (fldmultinv' (mat i i))))%ring.
    - exact (b i).
  Defined.
```

To prove correctness of the inductive procedure, we use a few properties of the stepwise update, of which the following is important.

Lemma 30 (Back substitution step invariant. The back substitution procedure provides pointwise solution)**.**
Let A be an $n \times n$, upper triangular matrix with no zero elements on its diagonal, and let b be a vector of length n. Let $x' =$ (`back_sub_step iter mat x b`). Then, we have $(Ax'^T)_i = b_i$.

```
Lemma back_sub_step_inv0 { n : nat }
    (iter : ⟦ n ⟧ %stn) (mat : Matrix F n n)
    (x : Vector F n) (b : Vector F n)
    (p: @ is_upper_triangular F n n mat)
    (p' : (mat iter iter != 0)%ring)
    : (mat ** (col_vec (back_sub_step iter mat x b))) iter = (col_vec b) iter.
```

This can be proven by case analysis on the iteration variable, in accordance to Definition 29. The equivalence is given by Lemma 41, and in the second case, additionally using Lemma 37. We have shown that our procedure assures the property of $(Ax^T)_i = b_i^T$ when applied to a given row i. Having defined the step procedure, we define an inductive method that provides the solution for every entry.

Definition 31 (Back substitution iteration)**.**

```
Definition back_sub_internal
    { n : nat }
    (mat : Matrix F n n)
    (b : Vector F n) (vec : Vector F n)
    (iter : ⟦ S n ⟧ %stn)
    : Vector F n.
  Proof.
    destruct iter as [iter p].
    induction iter as [ | m IHn] .
    - exact b.
    - refine (back_sub_step (dualelement (m,, p)) mat (IHn _) vec).
      apply (istransnatlth _ _ _ (natgthsnn m) p).
  Defined.
```

This is a standard iteration procedure with one twist to the previously defined ones in that we make the inner procedure call over the dual element of the iteration variable. Showing that the inductive procedure provides a correct solution to the systems of equations (that is, proving Theorem 27) is now fairly simple given Lemma 30.

Having a procedure for solving system of equations on form $Ax^T = b^T$, we define the following procedure for computing the (right) inverse of A.

Definition 32 (Construction of matrix inverse by Definition 31)**.**

```
Definition upper_triangular_right_inverse_construction
    { n : nat }
    (mat : Matrix F n n)
    := transpose (λ  i : (stn n), (back_sub F (mat) ((@ identity_matrix F n) i))).
```

The above is accompanied by a corresponding proof of correctness, given below.

Lemma 33 (Correctness of Definition 32)**.**
For any $n \times n$ matrix A that is upper triangular and with all non-zero elements on its diagonal, we can construct its right inverse A_R^{-1} with Definition 32.

```
Lemma right_inverse_construction_correct
    { n : nat } (mat : Matrix F n n)
    (ut : @ is_upper_triangular F _ _ mat)
    (df: @ diagonal_all_nonzero F n mat)
    : (mat ** ((upper_triangular_right_inverse_construction mat)))
      = (@ identity_matrix F n).
```

Proof. This follows straightforwardly from the fact that $\forall i : [\![n]\!]$, $\mathbf{y} : F^n$, `mat ** ( back_sub mat y)` $= y$, if mat is upper triangular with all non-zero diagonal. There is some mechanical work involved in showing that `col_vec` is injective in the UniMath setting, which then allows us to show the equivalence that $Ax = b$ gives; for all rows i, $\sum_j (A_{i,j} \cdot x_j) = b_i$; thus we can construct the inverse by seeking x_j satisfying $(Ax_j) = e_j$ (noting that the identity matrix is symmetric), for all $j : [\![n]\!]$. Correctness now follows from the definition of the inverse construction (Definition 32). □

Tying together the results of having defined an elimination procedure and constructing the inverse of a matrix under certain conditions, we can prove the following stronger property of being able to either give the inverse of a matrix, or show that is is not invertible.

Theorem 34 (Invertibility)**.** *We can either produce the inverse of a given $n \times n$ matrix, or show that it is non-invertible.*

```
Definition matrix_inverse_or_non_invertible_stmt
  { n : nat } {F : fld}
  (A : Matrix F n n)
  := coprod (@ matrix_inverse F n A)
            (@ matrix_inverse F n A → empty).

Theorem matrix_inverse_or_non_invertible { n : nat }
    (A : Matrix F n n)
    : @ matrix_inverse_or_non_invertible_stmt _ _ A.
```

Proof. Putting a matrix A into row echelon (and consequentially upper triangular) form can be accomplished by left multiplication of a suitably constructed invertible matrix B. If the product BA has a zero element in its diagonal, we are done, as BA is not invertible by Lemma 28, and since B is invertible, A can't be (Lemma 6).

Thus BA is upper triangular with all non-zero diagonal. This means we can construct a right inverse C of BA such that $BAC = I$ (Lemma 33). Since $BAC = ACB$ (for any left-right invertible B), this gives us the right inverse of A, $A_R^{-1} = CB$. It remains to show that right inverse implies left: $BAC = I \implies CBA = I$, without the prior that A is two-side invertible. □

Lemma 35 (Left inverse implies right)**.**
Having the left inverse B_L^{-1} of a matrix A, we can give the right inverse B_R^{-1} of A.

```
Lemma left_inverse_implies_right { n : nat } (A B: Matrix F n n)
    : (B ** A) = (@ identity_matrix F n)
    → (@ matrix_right_inverse F n n A).
```

Proof. Assume that $BA = I$, with A, B being $n \times n$ matrices over a field. By lemmata 23 and 28, there exists an $n \times n$ matrix C, such that CA is upper triangular with all non-zero diagonal. Thus our left inverse of A can also be expressed as $B = BC^{-1}C$, and $(CA)_L^{-1} = (BC)_R^{-1}$. Now, by the uniqueness of left inverses, and C being a left-right invertible matrix, it follows that $A_R^{-1} = BC_R^{-1}C = BC_L^{-1}C = B$. □

7 Vectors and auxiliary material

In this chapter, we outline some of the additional machinery necessary to formalize material presented in previous sections. Some of the material may appear simple from a classical mathematical standpoint, but required some work proving in an explicit constructive setting, in part due to care taken in presenting reusable interfaces for proofs and definitions to other authors working on UniMath. Firstly, a number of identities on sums over semirings were beneficial in formalizing Gaussian elimination. In addition we develop theories of standard basis vectors, establish various constructive lemmata pertaining to injectivity and weak equality of vectors and matrices with one as dimension, and prove a few special cases of sums over mostly zero functions and vectors ("pulse" functions) that only take a non-zero value at one or two points. This is useful when proving properties cover multiplication by elementary matrices, which have one or two non-zero elements per row. For the remainder of this section, the definitions and proofs are given over a fixed semiring R.

A useful identity on sums is the `iterop_fun_step`, allowing us to unroll the iterated application of an operator on a vector by one step, thus facilitating proofs by induction.

Lemma 36 (Stepwise identity on iterated function application)**.**

```
Definition iterop_fun_step' (lunax : islunit op unel) {m} (xs:stn m → X) (x:X) :
    iterop_fun (append_vec xs x) = op (iterop_fun xs) x.
```

From the identity in 36 and by working on induction on n, the length of each vector, we are able to derive a number of useful identities on sums, which we now present. Let f, g be functions $[\![n]\!] \mapsto R$, or interchangeably, vectors of length n with elements in R.

Lemma 37 (Sums of zero elements)**.**
If $f(i) = 0$ for all $0 < i \leq n$, then $\sum_{i=1}^{n} f = 0$.

```
Lemma zero_function_sums_to_zero:
    ∏ (n : nat)
      (f : (⟦ n ⟧)%stn → R),
    (λ i : (⟦ n ⟧)%stn, f i) = const_vec 0%rig →
    (∑ (λ i : (⟦ n ⟧)%stn, f i) ) = 0%rig.
```

This is easily proven by induction and the definition of the additive identity in a semiring.

Lemma 38 (Left distributivity of sums)**.**
$s \sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} sf(i)$

```
Lemma sum_is_ldistr :
    ∏ (n : nat) (vec : Vector R n) (s : R),
    op2 s (∑ vec) = ∑ ((const_vec s ) ^ vec).
```

This is again proven by induction on n in addition to vector appendation identities on `iterop_fun`. A corresponding lemma for right distributivity is also provided.

Lemma 39 (Sum equality 1)**.**
$\sum_{i=1}^{n} f(i) + \sum_{i=1}^{n} g(i) = \sum_{i=1}^{n} (f(i) + g(i))$

```
Lemma rigsum_add :
    ∏ (n : nat) (f1 f2 : (⟦ n ⟧)%stn → R),
    op1 (∑ (λ i: (⟦ n ⟧)%stn, f1 i)) (∑ (λ i : (⟦ n ⟧)%stn, f2 i))
    = ∑ (λ i: (⟦ n ⟧)%stn, op1 (f1 i) (f2 i)).
```

While mechanically slightly more involved, it can be proven in the same way as the previous lemma.

Lemma 40 (Interchanging order of summation)**.**
$\sum_{i=1}^{n} f(i) \sum_{i=1}^{m} g(i) = \sum_{i=1}^{m} g(i) \sum_{i=1}^{n} f(i)$

```
Lemma interchange_sums :
  ∏ (m n : nat)
    (f : (⟦ n ⟧)%stn → (⟦ m ⟧)%stn → R),
  ∑ (λ i: (⟦ m ⟧)%stn, ∑ (λ j : (⟦ n ⟧)%stn, f j i) )
  = ∑ (λ j: (⟦ n ⟧)%stn, ∑ (λ i : (⟦ m ⟧)%stn, f j i)).
```

Another proof by induction; this one requires a few more steps, noting in addition that in the definition of `iterop_fun`, the base case is addition by zero.

Lemma 41 (Sum equality 2)**.**
$\sum_{i=1}^{n} f(i) = \sum_{i=1, i\neq j}^{n} f(i) + f(j)$

```
Lemma rigsum_dni {n : nat} (f : ⟦ S n ⟧ %stn → R) (j : ⟦ S n ⟧ %stn ) :
  ∑ f = op1 (∑ (f ∘ dni j)) (f j).
```

First note that any sum over a finite set of rig elements $\sum_{i=1}^{n} f(i)$ can be expressed as $\sum_{i=1}^{m} f(i) + \sum_{i=m+1}^{n} f(i)$. This can be shown by induction over n; $n = 0$, this is a variant of a zero-sum lemma, e.g. as shown in Lemma 37. Proving this for $n + 1$, we want to show $\sum_{i=1}^{n+1} f(i) = \sum_{i=1}^{m} + \sum_{m+1}^{n+1}$. Applying Lemma 36, this is equivalent to showing $\sum_{i=1}^{n+1} f(i) = \sum_{i=1}^{m} f(i) + \sum_{m+1}^{n} + f(n+1)$ (in UniMath, this is slightly more tricky due to manipulations needed to have well typed decomposition of sums into left and right parts).

Lemma 42 (Sum of function non-zero at one point)**.**
$\forall i_1 i_2 < n,\ i_1 \neq i_2 \implies f(i_1) = 0 \implies \sum_{i=1} f(i) = f(i_2)$.

```
Lemma pulse_function_sums_to_point { n : nat }
  (f : ⟦ n ⟧ %stn → R) (i : ⟦ n ⟧ %stn)
  (f_pulse_function : is_pulse_function i f)
  : ∑ f = f i.
```

This follows from Lemma 41 and Lemma 37. In the following, let e_i refer to the standard basis vector with unit element at index i, else 0. Let $v_1 \cdot v_2$ refer to the pointwise product of two vectors v_1, v_2.

Lemma 43 (Sum of function non-zero at two points)**.** $\forall i_1\ i_2\ i_3 < n,\ i_1 \neq i_2 \wedge i_1 \neq i_3 \implies f(i_1) = 0 \implies \sum_{i=1}^{n} f(i) = f(i_2) + f(i_3)$.

```
Lemma two_pulse_function_sums_to_points { n : nat }
    (f : ⟦ n ⟧ %stn → R)
    (i : ⟦ n ⟧ %stn) (j : ⟦ n ⟧ %stn) (ne_i_j : i ≠ j)
    (X : ∏ (k: ⟦ n ⟧ %stn), (k ≠ i) → (k ≠ j) → (f k = 0%rig))
  : (∑ f = f i + f j)%rig.
```

First showing $f = f(i)e_i + f(j)e_j$ by functional extensionality; the pointwise equality follows simply by case analysis and definition of standard basis vector.

8 Conclusions

In this paper we have presented a formalization of Gaussian elimination over the minimal UniMath kernel programmed in Coq. This entailed formalizing, among other topics, the following material.

- A re-usable framework for elementary row operations.
- Procedures for Gaussian elimination.
- Applications in solving systems of linear equations and constructing matrix inverses.

The background theory necessary to facilitate such formalization included the following.

- Auxiliary materials on vectors. Injectivities, weak equalities, equalities on natural numbers, standard basis vectors; the concept of leading entries; properties of sums over semirings, pointwise sums and products of vectors.
- Properties of matrices over semirings and fields. Associativity, distributivity, identity matrices, transposes, triangular matrices, left inverses and right inverses, elementary matrices and properties of the product of matrices in those categories.

The total material has a scope of roughly 6000 lines of code, and can be found in a companion of this paper. [1]

We hope that our formalization of important results such Gaussian elimination, procedures for solving systems of linear equations and material on invertibility of matrices will prove useful to current and future authors working on UniMath. Our work in setting up row operations and providing a framework for Gaussian elimination in particular is something we hope can be used to formalize Smith normal form factorization in the future, covering some of the material in [CCDMS16].

References

[CCDMS16] Guillaume Cano, Cyril Cohen, Maxime Dénès, Anders Mörtberg, and Vincent Siles. "Formalized linear algebra over Elementary Divisor Rings in Coq". In: *Log. Methods Comput. Sci.* 12.2 (2016). DOI: 10.2168/LMCS-12(2:7)2016.

[Gra18] Daniel R. Grayson. "An introduction to univalent foundations for mathematicians". In: *Bull. Amer. Math. Soc. (N.S.)* 55.4 (2018), pp. 427–450. ISSN: 0273-0979. DOI: 10.1090/bull/1616.

[1] https://github.com/Skantz/UniMath/tree/elimination/UniMath/Algebra/Elimination

[Str06] Gilbert Strang. *Linear algebra and its applications.* 4th ed. Brooks/-Cole, 2006.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.

[VAG+] Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. *UniMath — a computer-checked library of univalent mathematics*. available at https://unimath.org.

[Voe10] Vladimir Voevodsky. *Univalent Foundations Project*. Available from https://www.math.ias.edu/vladimir/publications. a modified version of an NSF grant application. Oct. 2010.