A Formalization of Elementary Linear Algebra: Parts I and II

David M. Russinoff
david@russinoff.com

May 12, 2025

Outline

Part I:

- Matrix Algebra
- Determinants

Part II:

- Row reduction
- Invertibility
- Systems of Linear Equations

Part III: (2026?)

- Vector Spaces
- Linear Transformations
- Eigenvectors and Diagonalization

Outline

Part I: Matrices over a Commutative Ring

- Matrix Algebra
- Determinants

Part II: Matrices over a Field

- Row reduction
- Invertibility
- Systems of Linear Equations

Part III: (2026?)

- Vector Spaces
- Linear Transformations
- Eigenvectors and Diagonalization

THE DIRECTORY `books/projects/linear/`

- `ring`: Axioms of commutative ring R, lists of elements of R
- `rmat`: Matrices over R, algebraic operations, transpose
- `rdet`: Determinant of a matrix over R

- `field`: Axioms of field F, lists of elements of F
- `fmat`: Matrices over F, algebraic operations, transpose
- `fdet`: Determinant of a matrix over F
- `reduction`: Row reduction, inverse, linear equations

- `rational`: Matrices over $\mathbb{Q}$

The Commutative Ring R

```
(encapsulate (((rp *) => *)                          ;element recognizer
              ((r+ * *) => *) ((r* * *) => *)        ;operations
              ((r0) => *) ((r1) => *)                ;identities
              ((r- *) => *))                         ;additive inverse
  (local (defun rp (x) (rationalp x)))
  (local (defun r+ (x y) (+ x y)))
  (local (defun r* (x y) (* x y)))
  (local (defun r0 () 0))
  (local (defun r1 () 1))
  (local (defun r- (x) (- x)))
  ;; Closure:
  (defthm r+closed (implies (and (rp x) (rp y)) (rp (r+ x y))))
  (defthm r*closed (implies (and (rp x) (rp y)) (rp (r* x y))))
  ;; Commutativity

  ...
)
```

Informally, we refer to the ring R characterized by these axioms.

MATRICES OVER R

An `m×n` matrix over `R` is a list of `m` rlists of length `n`:

```
(defun rmatp (a m n)
  (if (zp m)
      (null a)
    (and (consp a)
         (rlistnp (car a) n)
         (rmatp (cdr a) (1- m) n))))

(defun row (i a) (nth i a))

(defun col (j a)
  (if (consp a)
      (cons (nth j (car a)) (col j (cdr a)))
    ()))

(defun entry (i j a) (nth j (nth i a)))
```

Matrix Operations

```
(defun rmat-add (a b)
  (if (consp a)
      (cons (rlist-add (car a) (car b))
            (rmat-add (cdr a) (cdr b)))
    ()))

(defun rmat-scalar-mul (c a)
  (if (consp a)
      (cons (rlist-scalar-mul c (car a))
            (rmat-scalar-mul c (cdr a)))
    ()))

(defun replace-row (a k r)
  (if (zp k)
      (cons r (cdr a))
    (cons (car a) (replace-row (cdr a) (1- k) r))))

(defun rmat-sum (a)
  (if (consp a)
      (r+ (rlist-sum (car a)) (rmat-sum (cdr a)))
    (r0)))
```

Transpose of a Matrix

The *transpose* of a matrix is the list of its columns:

```
(defun transpose-mat-aux (a j n)
  (if (and (natp j) (natp n) (< j n))
      (cons (col j a) (transpose-mat-aux a (1+ j) n))
    ()))

(defund transpose-mat (a) (transpose-mat-aux a 0 (len (car a))))

(defthm transpose-rmat-entry
  (implies (and (posp m) (posp n) (rmatp a m n)
                (natp j) (< j n) (natp i) (< i m))
           (equal (entry j i (transpose-mat a))
                  (entry i j a))))

(defthmd sum-rmat-transpose
  (implies (and (natp m) (natp n) (rmatp a m n))
           (equal (rmat-sum (transpose-mat a))
                  (rmat-sum a))))
```

Matrix Multiplication

The product of an `m×n` matrix and an `n×p` matrix is an `m×p` matrix:

```
(defund rmat* (a b)
  (if (consp a)
      (cons (rdot-list (car a) (transpose-mat b))
            (rmat* (cdr a) b))
    ()))

(defthm rmatp-rmat*
  (implies (and (rmatp a m n) (rmatp b n p)
                (posp m) (posp n) (posp p))
           (rmatp (rmat* a b) m p)))

(defthmd rmat*-entry
  (implies (and (posp m) (posp n) (posp p)
                (rmatp a m n) (rmatp b n p)
                (natp i) (< i m) (natp j) (< j p))
           (equal (entry i j (rmat* a b))
                  (rdot (row i a) (col j b)))))
```

Identity Matrix

```
(defun runit (i n)
  (if (zp n) ()
    (if (zp i) (cons (r1) (rlistn0 (1- n)))
      (cons (r0) (runit (1- i) (1- n))))))

(defun id-rmat-aux (i n)
  (if (and (natp i) (natp n) (< i n))
      (cons (runit i n) (id-rmat-aux (1+ i) n))
    ()))
(defund id-rmat (n) (id-rmat-aux 0 n))

(defthmd id-rmat-right
  (implies (and (posp m) (posp n) (rmatp a m n))
           (equal (rmat* a (id-rmat n)) a)))

(defthmd id-rmat-left
  (implies (and (posp m) (posp n) (rmatp a m n))
           (equal (rmat* (id-rmat m) a) a)))
```

Associativity of Matrix Multiplication

If a, b, and c are matrices of dimensions $m \times n$, $n \times p$, and $p \times q$, then

$$((ab)c)_{ij} = \sum_{s=0}^{p-1} \sum_{r=0}^{n-1} a_{ir} b_{rs} c_{sj} = \sum_{r=0}^{n-1} \sum_{s=0}^{p-1} a_{ir} b_{rs} c_{sj} = (a(bc))_{ij}.$$

```
(defthmd rmat*-assoc-entry
  (implies (and  (posp m) (posp n) (posp p) (posp q)
                 (rmatp a m n) (rmatp b n p) (rmatp c p q)
                 (natp i) (< i m) (natp j) (< j q))
           (equal (entry i j (rmat* a (rmat* b c)))
                  (entry i j (rmat* (rmat* a b) c)))))

(defthmd rmat*-assoc
  (implies (and (rmatp a m n) (rmatp b n p) (rmatp c p q)
                (posp m) (posp n) (posp p) (posp q))
           (equal (rmat* a (rmat* b c))
                  (rmat* (rmat* a b) c))))
```

Definition of Determinant

Each permutation `p` in the symmetric group `(sym n)` contributes a term to the determinant of an `n×n` matrix `a`:

```
(defun rdet-prod (a p n)
  (if (zp n)
      (r1)
    (r* (rdet-prod a p (1- n))
        (entry (1- n) (nth (1- n) p) a))))

(defund rdet-term (a p n)
  (if (even-perm-p p n)
      (rdet-prod a p n)
    (r- (rdet-prod a p n))))

(defun rdet-sum (a l n)
  (if (consp l)
      (r+ (rdet-term a (car l) n) (rdet-sum a (cdr l) n))
    (r0)))

(defund rdet (a n) (rdet-sum a (slist n) n))
```

Properties of Determinant

The determinant is characterized by three properties:

```
(defthm rdet-id-rmat
  (implies (posp n) (equal (rdet (id-rmat n) n) (r1))))

(defthmd rdet-alternating
  (implies (and (rmatp a n n) (posp n)
                (natp i) (< i n) (natp j) (< j n) (not (= i j))
                (= (row i a) (row j a)))
           (equal (rdet a n) (r0))))

(defthm rdet-n-linear
  (implies (and (rmatp a n n) (posp n) (natp i) (< i n)
                (rlistnp x n) (rlistnp y n) (rp c))
           (equal (rdet (replace-row
                           a i (rlist-add (rlist-scalar-mul c x) y))
                        n)
                  (r+ (r* c (rdet (replace-row a i x) n))
                      (rdet (replace-row a i y) n)))))
```

Uniqueness of Determinant

The determinant is uniquely determined by those properties:

```
(encapsulate (((rdet0 * *) => *))
  (local (defun rdet0 (a n) (rdet a n)))
  (defthm rp-rdet0
    (implies (and (rmatp a n n) (posp n)) (rp (rdet0 a n))))
  (defthmd rdet0-n-linear
    ... )
  (defthmd rdet0-adjacent-equal
    (implies (and (rmatp a n n) (posp n)
                  (natp i) (< i (1- n))
                  (= (row i a) (row (1+ i) a)))
             (equal (rdet0 a n) (r0)))))

(defthmd rdet-unique
  (implies (rmatp a n n)
           (equal (rdet0 a n)
                  (r* (rdet a n) (rdet0 (id-rmat n) n)))))
```

Multiplicativity of Determinant

Multiplicativity is proved by functional instantiation:

```
(defun rdet-rmat* (a b n)
  (rdet (rmat* a b) n))

(defthm rdet-rmat*-rewrite
  (implies (and (rmatp a n n) (rmatp b n n) (posp n))
           (equal (rdet-rmat* a b n)
                  (r* (rdet a n) (rdet-rmat* (id-rmat n) b n))))
  :hints
  (("Goal" :use ((functional-instance rdet-unique
                   (rdet0 (lambda (a n) (rdet-rmat* a b n))))))))

(defthm rdet-multiplicative
  (implies (and (rmatp a n n) (rmatp b n n) (posp n))
           (equal (rdet (rmat* a b) n)
                  (r* (rdet a n) (rdet b n)))))
```

Cofactor Expansion of Determinant

Expansion with respect to column `j`:

```
(defund minor (i j a)
  (delete-col j (delete-row i a)))

(defund rdet-cofactor (i j a n)
  (if (evenp (+ i j))
      (rdet (minor i j a) (1- n))
    (r- (rdet (minor i j a) (1- n)))))

(defun expand-rdet-col-aux (a i j n)
  (if (zp i)
      (r0)
    (r+ (r* (entry (1- i) j a)
            (rdet-cofactor (1- i) j a n))
  (expand-rdet-col-aux a (1- i) j n))))

(defund expand-rdet-col (a j n)
  (expand-rdet-col-aux a n j n))
```

Correctness of Cofactor Expansion

For fixed `k`, `(expand-rdet-col a k n)` may be shown to satisfy the constraints on `(rdet0 a n)`. By functional instantiation of `rdet-unique`,

```
(defthmd expand-rdet-col-val
  (implies (and (rmatp a n n) (posp n) (> n 1)
                (natp k) (< k n))
           (equal (expand-rdet-col a k n)
                  (r* (rdet a n)
                      (expand-rdet-col (id-rmat n) k n)))))
```

But `(expand-rdet-col (id-rmat n) k n)` = 1. Thus,

```
(defthmd expand-rdet-col-rdet
  (implies (and (rmatp a n n) (posp n) (> n 1)
                (natp k) (< k n))
           (equal (expand-rdet-col a k n)
                  (rdet a n))))
```

Recursive Formulation of Determinant

```
(mutual-recursion
  (defund rdet-rec-cofactor (j a n)
    (if (zp n) ()
      (if (evenp j)
          (rdet-rec (minor 0 j a) (1- n))
        (r- (rdet-rec (minor 0 j a) (1- n))))))
  (defun expand-rdet-rec-aux (a j n)
    (if (zp j) (r0)
      (r+ (r* (entry 0 (1- j) a)
              (rdet-rec-cofactor (1- j) a n))
          (expand-rdet-rec-aux a (1- j) n))))
  (defund expand-rdet-rec (a n)
    (expand-rdet-rec-aux a n n))
  (defun rdet-rec (a n)
    (if (zp n) (r0)
      (if (= n 1) (entry 0 0 a)
        (expand-rdet-rec a n)))))

(defthmd rdet-rec-rdet
  (implies (and (rmatp a n n) (posp n))
           (equal (rdet-rec a n) (rdet a n))))
```

Classical Adjoint

The *adjoint* of an `n×n` matrix is defined to satisfy

```
(defthmd adjoint-rmat-entry
  (implies (and (rmatp a n n) (natp n) (> n 1)
                (natp i) (< i n) (natp j) (< j n))
           (and (rmatp (adjoint-rmat a n) n n)
                (equal (entry i j (adjoint-rmat a n))
                       (rdet-cofactor j i a n)))))
```

A consequence of `expand-rdet-col-rdet`:

```
(defthmd rmat*-adjoint-rmat
  (implies (and (rmatp a n n) (natp n) (> n 1))
           (equal (rmat* a (adjoint-rmat a n))
                  (rmat-scalar-mul (rdet a n) (id-rmat n)))))
```

THE FIELD *F*

```
(encapsulate (((fp *) => *)                          ;element recognizer
              ((f+ * *) => *) ((f* * *) => *)        ;operations
              ((f0) => *) ((f1) => *)                ;identities
              ((f- *) => *) ((f/ *) => *))           ;inverses
  (local (defun fp (x) (rationalp x)))
  (local (defun f+ (x y) (+ x y)))
  (local (defun f* (x y) (* x y)))
  (local (defun f0 () 0))
  (local (defun f1 () 1))
  (local (defun f- (x) (- x)))
  (local (defun f/ (x) (/ x)))
  ;; Closure:
  ...
  ;; Multiplicative inverse:
  (defthm fpf/
    (implies (and (fp x) (not (equal x (f0))))
             (fp (f/ x))))
  (defthm f*inv
    (implies (and (fp x) (not (equal x (f0))))
             (equal (f* x (f/ x)) (f1)))))
```

Reduced Row-Echelon Form

A *reduced row-echelon* matrix, recognized by the predicate `row-echelon-p`, satisfies the following:

(1) Every all-zero row is preceded by every nonzero row;

(2) The first nonzero entry of each nonzero row is 1, and every other entry in the same column is 0;

(3) The column of the leading 1 in the `i`th nonzero row is an increasing function of `i`.

Elementary Row Operations

(1) Multiply row `k` by a scalar `c`:

```
(defund ero1 (a c k)
  (replace-row a k (flist-scalar-mul c (nth k a))))
```

(2) Add a scalar multiple of row `j` to row `k`, where `j` $\neq$ `k`:

```
(defund ero2 (a c j k)
  (replace-row a k
               (flist-add (flist-scalar-mul c (nth j a))
                          (nth k a))))
```

(3) Interchange rows `j` and `k`, where `j` $\neq$ `k`:

```
(defund ero3 (a j k)
  (replace-row (replace-row a k (nth j a)) j (nth k a)))
```

Conversion to Reduced Row-Echelon Form

The function `row-reduce` applies a sequence of elementary row operations to convert a matrix to RRE form:

```
(defthmd row-echelon-p-row-reduce
  (implies (and (natp m) (natp n) (fmatp a m n))
           (row-echelon-p (row-reduce a))))
```

Row rank of a matrix:

```
(defun row-rank (a) (num-nonzero-rows (row-reduce a)))
```

Encoding Elementary Row Operations

Each elementary row op is encoded as a list, e.g., multiplication of row 3 by -2 is represented by `(1 -2 3)`.

`(apply-row-op op a)` applies an encoded `op` to matrix `a`

`(apply-row-ops ops a)` applies a list `ops` to matrix `a`

`(row-reduce-ops a)` is the list of ops required to reduce `a`

```
(defthmd apply-row-reduce-ops
  (implies (and (fmatp a m n) (posp m) (posp n))
           (equal (apply-row-ops (row-reduce-ops a) a)
                  (row-reduce a))))
```

Row Reduction as Matrix Multiplication

The *elementary matrix* corresponding to a row op:

```
(defund elem-mat (op m) (apply-row-op op (id-fmat m)))
```

Row reduction is equivalent to multiplication by a product of elementary matrices:

```
(defund row-ops-mat (ops m)
  (if (consp ops)
      (fmat* (row-ops-mat (cdr ops) m) (elem-mat (car ops) m))
    (id-fmat m)))

(defund row-reduce-mat (a)
  (row-ops-mat (row-reduce-ops a) (len a)))

(defthmd row-ops-mat-row-reduce
  (implies (and (fmatp a m n) (posp m) (posp n))
           (equal (fmat* (row-reduce-mat a) a)
                  (row-reduce a))))
```

Invertibility of an n×n Matrix

```
(defund invertiblep (a n) (= (row-rank a) n))

(defund inverse-mat (a) (row-reduce-mat a))

(defthmd invertiblep-sufficient
  (implies (and (fmatp a n n) (posp n) (invertiblep a n))
           (let ((p (inverse-mat a)))
               (and (equal (fmat* a p) (id-fmat n))
                    (equal (fmat* p a) (id-fmat n))))))

(defthmd invertiblep-necessary
  (implies (and (fmatp a n n) (fmatp b n n) (posp n)
                (= (fmat* a b) (id-fmat n)))
           (invertiblep a n)))

(defthmd invertiblep-fdet-not-zero
  (implies (and (fmatp a n n) (posp n))
           (iff (invertiblep a n))
                (not (equal (fdet a n) (f0)))))
```

Simultaneous Systems of Linear Equations

Given an `m×n` matrix `a` with `(entry i j a)` = $a_{\mathtt{i},\mathtt{j}}$ and a vector `b` = $(b_0 \ldots b_{\mathtt{m}-1})$, find a vector `x` = $(x_0 \ldots x_{\mathtt{n}-1})$ such that for $0 \leq \mathtt{i} < \mathtt{m}$,

$$a_{\mathtt{i},0}x_0 + \ldots + a_{\mathtt{i},\mathtt{n}-1}x_{\mathtt{n}-1} = b_{\mathtt{i}}.$$

As a matrix equation,

```
(defund solutionp (x a b)
  (equal (fmat* a (col-mat x))
         (col-mat b)))
```

SOLVABILITY

Let

```
bc = (col-mat b)
xc = (col-mat x)
p = (row-reduce-mat a)
ar = (fmat* p a)
br = (fmat* p bc).
```

Then `(fmat* a xc) = bc` $\Leftrightarrow$ `(fmat* ar xc) = br`.

Let `q = (row-rank a)`.

A solution exists $\Leftrightarrow$ the last $\texttt{m} - \texttt{q}$ entries of `br` are 0.

SOLVABILITY

Algorithmic definition of solvability:

```
(defun find-nonzero (br q m)
  (if (and (natp q) (natp m) (< q m))
      (if (= (entry (1- m) 0 br) (f0))
          (find-nonzero br q (1- m))
        (1- m))
    ()))

(defun solvablep (a b)
  (null (find-nonzero (fmat* (row-reduce-mat a) (col-mat b))
                      (row-rank a)
                      (len a))))
```

Uniquely Solvable Case

If `(solvablep a b)` and `(row-rank a)` = `n`, then the solution is unique:

```
(defthmd linear-equations-unique-solution-case
  (let* ((br (fmat* (row-reduce-mat a) (col-mat b)))
         (bq (first-rows n br)))
    (implies (and (fmatp a m n) (posp m) (posp n)
                  (flistnp b m) (flistnp x n)
                  (solvablep a b) (= (row-rank a) n))
             (iff (solutionp x a b)
                  (equal x (col 0 bq))))))
```

Cramer's Rule

An alternative method of solution in the case of an invertible square coefficient matrix:

```
(defthmd cramer
  (implies (and (fmatp a n n) (natp n) (> n 1)
                (invertiblep a n)
                (flistnp b n) (flistnp x n)
                (solutionp x a b)
                (natp i) (< i n))
           (equal (nth i x)
                  (f* (fdet (replace-col a i b) n)
                      (f/ (fdet a n))))))
```

Future Work

Continuation of linear algebra:

- Vector spaces and linear transformations
- Polynomials and factorization
- Eigenvectors and diagonalization

Algebraic number theory:

- Algebraic extensions and number fields
- Galois groups
- Quadratic fields and Hilbert's 10th problem