Rank-Nullity Theorem in Linear Algebra

By Jose Divasón and Jesús Aransay*

May 26, 2024

Abstract

In this contribution, we present some formalizations based on the HOL-Multivariate-Analysis session of Isabelle. Firstly, a generalization of several theorems of such library are presented. Secondly, some definitions and proofs involving Linear Algebra and the four fundamental subspaces of a matrix are shown. Finally, we present a proof of the result known in Linear Algebra as the "Rank-Nullity Theorem", which states that, given any linear map f from a finite dimensional vector space V to a vector space W, then the dimension of V is equal to the dimension of the kernel of f (which is a subspace of V) and the dimension of the range of f (which is a subspace of W). The proof presented here is based on the one given in [1]. As a corollary of the previous theorem, and taking advantage of the relationship between linear maps and matrices, we prove that, for every matrix A (which has associated a linear map between finite dimensional vector spaces), the sum of its null space and its column space (which is equal to the range of the linear map) is equal to the number of columns of A.

Contents

*This research has been funded by the research grant FPIUR12 of the Universidad de La Rioja.

1 Dual Order

theory *Dual-Order*
  imports *Main*
begin

1.1 Interpretation of dual wellorder based on wellorder

lemma *wf-wellorderI2*:
  assumes *wf*: *wf* $\{(x::'a::ord,\ y).\ y < x\}$
  assumes *lin*: *class.linorder* $(\lambda(x::'a)\ y::'a.\ y \leq x)$ $(\lambda(x::'a)\ y::'a.\ y < x)$
  shows *class.wellorder* $(\lambda(x::'a)\ y::'a.\ y \leq x)$ $(\lambda(x::'a)\ y::'a.\ y < x)$
  ⟨*proof*⟩

interpretation *dual-wellorder*: *wellorder* (≥)::('*a*::{*linorder*, *finite*}=>'*a*=>*bool*)
(>)
⟨*proof*⟩

1.2 Properties of the Greatest operator

lemma *dual-wellorder-Least-eq-Greatest*[*simp*]: *dual-wellorder.Least* = *Greatest*
  ⟨*proof*⟩

lemmas *GreatestI* = *dual-wellorder.LeastI*[*unfolded dual-wellorder-Least-eq-Greatest*]
lemmas *GreatestI2-ex* = *dual-wellorder.LeastI2-ex*[*unfolded dual-wellorder-Least-eq-Greatest*]

lemmas *GreatestI2-wellorder* = *dual-wellorder.LeastI2-wellorder*[*unfolded dual-wellorder-Least-eq-Greatest*]
lemmas *GreatestI-ex* = *dual-wellorder.LeastI-ex*[*unfolded dual-wellorder-Least-eq-Greatest*]
lemmas *not-greater-Greatest* = *dual-wellorder.not-less-Least*[*unfolded dual-wellorder-Least-eq-Greatest*]
lemmas *GreatestI2* = *dual-wellorder.LeastI2*[*unfolded dual-wellorder-Least-eq-Greatest*]
lemmas *Greatest-ge* = *dual-wellorder.Least-le*[*unfolded dual-wellorder-Least-eq-Greatest*]

end

2 Class for modular arithmetic

theory *Mod-Type*
imports
HOL−Library.Numeral-Type
HOL−Analysis.Cartesian-Euclidean-Space
Dual-Order
begin

2.1 Definition and properties

Class for modular arithmetic. It is inspired by the locale mod_type.

class *mod-type* = *times* + *wellorder* + *neg-numeral* +
fixes *Rep* :: *'a* => *int*
and *Abs* :: *int* => *'a*
assumes *type*: *type-definition Rep Abs* {*0..<int CARD* (*'a*)}
and *size1*: *1* < *int CARD* (*'a*)
and *zero-def*: *0* = *Abs 0*
and *one-def*: *1* = *Abs 1*
and *add-def*: *x* + *y* = *Abs* ((*Rep x* + *Rep y*) *mod* (*int CARD* (*'a*)))
and *mult-def*: *x* ∗ *y* = *Abs* ((*Rep x* ∗ *Rep y*) *mod* (*int CARD* (*'a*)))
and *diff-def*: *x* − *y* = *Abs* ((*Rep x* − *Rep y*) *mod* (*int CARD* (*'a*)))
and *minus-def*: − *x* = *Abs* ((− *Rep x*) *mod* (*int CARD* (*'a*)))
and *strict-mono-Rep*: *strict-mono Rep*
begin

lemma *size0*: *0* < *int CARD* (*'a*)
⟨*proof*⟩

lemmas *definitions* =
zero-def one-def add-def mult-def minus-def diff-def

lemma *Rep-less-n*: *Rep x* < *int CARD* (*'a*)
⟨*proof*⟩

lemma *Rep-le-n*: *Rep x* ≤ *int CARD* (*'a*)
⟨*proof*⟩

lemma *Rep-inject-sym*: *x* = *y* ⟷ *Rep x* = *Rep y*
⟨*proof*⟩

lemma *Rep-inverse*: $Abs\ (Rep\ x) = x$
⟨*proof*⟩

lemma *Abs-inverse*: $m \in \{0..<int\ CARD\ ('a)\} \Longrightarrow Rep\ (Abs\ m) = m$
⟨*proof*⟩

lemma *Rep-Abs-mod*: $Rep\ (Abs\ (m\ mod\ int\ CARD\ ('a))) = m\ mod\ int\ CARD\ ('a)$
⟨*proof*⟩

lemma *Rep-Abs-0*: $Rep\ (Abs\ 0) = 0$
⟨*proof*⟩

lemma *Rep-0*: $Rep\ 0 = 0$
⟨*proof*⟩

lemma *Rep-Abs-1*: $Rep\ (Abs\ 1) = 1$
⟨*proof*⟩

lemma *Rep-1*: $Rep\ 1 = 1$
⟨*proof*⟩

lemma *Rep-mod*: $Rep\ x\ mod\ int\ CARD\ ('a) = Rep\ x$
⟨*proof*⟩

lemmas *Rep-simps* =
Rep-inject-sym Rep-inverse Rep-Abs-mod Rep-mod Rep-Abs-0 Rep-Abs-1

2.2 Conversion between a modular class and the subset of natural numbers associated.

Definitions to make transformations among elements of a modular class and naturals

definition *to-nat* :: $'a => nat$
where $to\text{-}nat = nat \circ Rep$

definition *Abs'* :: $int => 'a$
where $Abs'\ x = Abs(x\ mod\ int\ CARD\ ('a))$

definition *from-nat* :: $nat \Rightarrow 'a$
where $from\text{-}nat = (Abs' \circ int)$

lemma *bij-Rep*: $bij\text{-}betw\ (Rep)\ (UNIV::'a\ set)\ \{0..<int\ CARD('a)\}$
⟨*proof*⟩

lemma *mono-Rep*: *mono Rep* ⟨*proof*⟩

lemma *Rep-ge-0*: $0 \le Rep\ x$ ⟨*proof*⟩

lemma *bij-Abs*: *bij-betw (Abs) {0..<int CARD('a)} (UNIV::'a set)*
⟨*proof*⟩

corollary *bij-Abs'*: *bij-betw (Abs') {0..<int CARD('a)} (UNIV::'a set)*
⟨*proof*⟩

lemma *bij-from-nat*: *bij-betw (from-nat) {0..<CARD('a)} (UNIV::'a set)*
⟨*proof*⟩

lemma *to-nat-is-inv*: *the-inv-into {0..<CARD('a)} (from-nat::nat=>'a) = (to-nat::'a=>nat)*
⟨*proof*⟩

lemma *bij-to-nat*: *bij-betw (to-nat) (UNIV::'a set) {0..<CARD('a)}*
⟨*proof*⟩

lemma *finite-mod-type*: *finite (UNIV::'a set)*
⟨*proof*⟩

subclass (**in** *mod-type*) *finite* ⟨*proof*⟩

lemma *least-0*: *(LEAST n. n ∈ (UNIV::'a set)) = 0*
⟨*proof*⟩

lemma *add-to-nat-def*: *x + y = from-nat (to-nat x + to-nat y)*
⟨*proof*⟩

lemma *to-nat-1*: *to-nat 1 = 1*
⟨*proof*⟩

lemma *add-def'*:
shows *x + y = Abs' (Rep x + Rep y)* ⟨*proof*⟩

lemma *Abs'-0*:
shows *Abs' (CARD('a))=(0::'a)* ⟨*proof*⟩

lemma *Rep-plus-one-le-card*:
assumes *a*: *a + 1 ≠ 0*
shows *(Rep a) + 1 < CARD ('a)*
⟨*proof*⟩

lemma *to-nat-plus-one-less-card*: *∀ a. a+1 ≠ 0 --> to-nat a + 1 < CARD('a)*
⟨*proof*⟩

corollary *to-nat-plus-one-less-card'*:
assumes *a+1 ≠ 0*
shows *to-nat a + 1 < CARD('a)* ⟨*proof*⟩

lemma *strict-mono-to-nat*: *strict-mono to-nat*

⟨*proof*⟩

lemma *to-nat-eq* [*simp*]: *to-nat* x = *to-nat* $y \longleftrightarrow x = y$
⟨*proof*⟩

lemma *mod-type-forall-eq* [*simp*]: $(\forall j::'a.\ (\textit{to-nat}\ j) < CARD('a) \longrightarrow P\ j) = (\forall a.\ P\ a)$
⟨*proof*⟩

lemma *to-nat-from-nat*:
assumes *t*:*to-nat* $j = k$
shows *from-nat* $k = j$
⟨*proof*⟩

lemma *to-nat-mono*:
assumes *ab*: $a < b$
shows *to-nat* a < *to-nat* b
⟨*proof*⟩

lemma *to-nat-mono'*:
assumes *ab*: $a \leq b$
shows *to-nat* $a \leq$ *to-nat* b
⟨*proof*⟩

lemma *least-mod-type*:
shows $0 \leq (n::'a)$
⟨*proof*⟩

lemma *to-nat-from-nat-id*:
assumes *x*: $x < CARD('a)$
shows *to-nat* ((*from-nat* x)::'a) = x
⟨*proof*⟩

lemma *from-nat-to-nat-id*[*simp*]:
shows *from-nat* (*to-nat* x) = x ⟨*proof*⟩

lemma *from-nat-to-nat*:
assumes *t*:*from-nat* $j = k$ **and** *j*: $j < CARD('a)$
shows *to-nat* $k = j$ ⟨*proof*⟩

lemma *from-nat-mono*:
assumes *i-le-j*: $i<j$ **and** *j*: $j < CARD('a)$
shows (*from-nat* i::'a) < *from-nat* j
⟨*proof*⟩

lemma *from-nat-mono'*:
assumes *i-le-j*: $i \leq j$ **and** $j < CARD\ ('a)$
shows (*from-nat* i::'a) $\leq$ *from-nat* j
⟨*proof*⟩

lemma *to-nat-suc*:
  assumes *to-nat* $(x)+1 < CARD\ ('a)$
  shows *to-nat* $(x + 1::'a) = (to\text{-}nat\ x) + 1$
⟨*proof*⟩

lemma *to-nat-le*:
  assumes $y < from\text{-}nat\ k$
  shows $to\text{-}nat\ y < k$
⟨*proof*⟩

lemma *le-Suc*:
  assumes *ab*: $a < (b::'a)$
  shows $a + 1 \leq b$
⟨*proof*⟩

lemma *le-Suc'*:
assumes *ab*: $a + 1 \leq b$
  and *less-card*: $(to\text{-}nat\ a) + 1 < CARD\ ('a)$
  shows $a < b$
⟨*proof*⟩

lemma *Suc-le*:
  assumes *less-card*: $(to\text{-}nat\ a) + 1 < CARD\ ('a)$
  shows $a < a + 1$
⟨*proof*⟩

lemma *Suc-le'*:
  fixes $a::'a$
  assumes $a + 1 \neq 0$
  shows $a < a + 1$ ⟨*proof*⟩

lemma *from-nat-not-eq*:
  assumes *a-eq-to-nat*: $a \neq to\text{-}nat\ b$
  and *a-less-card*: $a<CARD('a)$
  shows $from\text{-}nat\ a \neq b$
⟨*proof*⟩

lemma *Suc-less*:
  fixes $i::'a$
  assumes $i<j$
  and $i+1 \neq j$
  shows $i+1<j$ ⟨*proof*⟩

lemma *Greatest-is-minus-1*: $\forall\ a::'a.\ a \leq -1$
⟨*proof*⟩

lemma *a-eq-minus-1*: $\forall\ a::'a.\ a+1 = 0 \longrightarrow a = -1$

⟨*proof*⟩

lemma *forall-from-nat-rw*:
shows $(\forall x\in\{0..<CARD('a)\}.\ P\ (from\text{-}nat\ x::'a)) = (\forall x.\ P\ (from\text{-}nat\ x))$
⟨*proof*⟩

lemma *from-nat-eq-imp-eq*:
assumes *f-eq*: $from\text{-}nat\ x = (from\text{-}nat\ xa::'a)$
and *x*: $x<CARD('a)$ **and** *xa*: $xa<CARD('a)$
shows $x=xa$ ⟨*proof*⟩

lemma *to-nat-less-card*:
fixes $j::'a$
shows $to\text{-}nat\ j < CARD\ ('a)$
⟨*proof*⟩

lemma *from-nat-0*: $from\text{-}nat\ 0 = 0$
⟨*proof*⟩
lemma *to-nat-0*: $to\text{-}nat\ 0 = 0$ ⟨*proof*⟩
lemma *to-nat-eq-0*: $(to\text{-}nat\ x = 0) = (x = 0)$
⟨*proof*⟩

lemma *suc-not-zero*:
assumes $to\text{-}nat\ a + 1 \neq CARD('a)$
shows $a+1 \neq 0$
⟨*proof*⟩

lemma *from-nat-suc*:
shows $from\text{-}nat\ (j + 1) = from\text{-}nat\ j + 1$
⟨*proof*⟩

lemma *to-nat-plus-1-set*:
shows $to\text{-}nat\ a + 1 \in \{1..<CARD('a)+1\}$
⟨*proof*⟩

end

lemma *from-nat-CARD*:
shows $from\text{-}nat\ (CARD('a)) = (0::'a::\{mod\text{-}type\})$
⟨*proof*⟩

2.3 Instantiations

instantiation *bit0* **and** *bit1*:: (*finite*) *mod-type*
begin

definition $(Rep::'a\ bit0 => int)\ \ x = Rep\text{-}bit0\ x$

definition (*Abs*::*int* => *'a bit0*) *x* = *Abs-bit0' x*

definition (*Rep*::*'a bit1* => *int*) *x* = *Rep-bit1 x*
definition (*Abs*::*int* => *'a bit1*) *x* = *Abs-bit1' x*

instance
⟨*proof*⟩
end

end

3 Miscellaneous

theory *Miscellaneous*
imports
HOL−Analysis.Determinants
Mod-Type
HOL−Library.Function-Algebras
begin

context *Vector-Spaces.linear* **begin**
sublocale *vector-space-pair* ⟨*proof*⟩
end

hide-const (**open**) *Real-Vector-Spaces.linear*
abbreviation *linear* ≡ *Vector-Spaces.linear*

In this file, we present some basic definitions and lemmas about linear algebra and matrices.

3.1 Definitions of number of rows and columns of a matrix

definition *nrows* :: *'a^'columns^'rows* => *nat*
where *nrows A* = *CARD('rows)*

definition *ncols* :: *'a^'columns^'rows* => *nat*
where *ncols A* = *CARD('columns)*

definition *matrix-scalar-mult* :: *'a*::*ab-semigroup-mult* => *'a ^'n^'m* => *'a ^'n^'m*
(**infixl** *∗k 70*)
where *k ∗k A* ≡ (*χ i j. k ∗ A $ i $ j*)

3.2 Basic properties about matrices

lemma *nrows-not-0*[*simp*]:
shows *0* ≠ *nrows A* ⟨*proof*⟩

lemma *ncols-not-0*[*simp*]:
shows *0* ≠ *ncols A* ⟨*proof*⟩

lemma *nrows-transpose*: *nrows* (*transpose* *A*) = *ncols* *A*
⟨*proof*⟩

lemma *ncols-transpose*: *ncols* (*transpose* *A*) = *nrows* *A*
⟨*proof*⟩

lemma *finite-rows*: *finite* (*rows* *A*)
⟨*proof*⟩

lemma *finite-columns*: *finite* (*columns* *A*)
⟨*proof*⟩

lemma *transpose-vector*: *x* *v*∗ *A* = *transpose* *A* ∗*v* *x*
⟨*proof*⟩

lemma *transpose-zero*[*simp*]: (*transpose* *A* = *0*) = (*A* = *0*)
⟨*proof*⟩

3.3 Theorems obtained from the AFP

The following theorems and definitions have been obtained from the AFP http://isa-afp.org/browser_info/current/HOL/Tarskis_Geometry/Linear_Algebra2.html. I have removed some restrictions over the type classes.

lemma *vector-scalar-matrix-ac*:
fixes *k* :: *'a*::{*field*} **and** *x* :: *'a*::{*field*}^*'n* **and** *A* :: *'a*^*'m*^*'n*
shows *x* *v*∗ (*k* ∗*k* *A*) = *k* ∗*s* (*x* *v*∗ *A*)
⟨*proof*⟩

lemma *transpose-scalar*: *transpose* (*k* ∗*k* *A*) = *k* ∗*k* *transpose* *A*
⟨*proof*⟩

lemma *scalar-matrix-vector-assoc*:
fixes *A* :: *'a*::{*field*}^*'m*^*'n*
shows *k* ∗*s* (*A* ∗*v* *v*) = *k* ∗*k* *A* ∗*v* *v*
⟨*proof*⟩

lemma *matrix-scalar-vector-ac*:
fixes *A* :: *'a*::{*field*}^*'m*^*'n*
shows *A* ∗*v* (*k* ∗*s* *v*) = *k* ∗*k* *A* ∗*v* *v*
⟨*proof*⟩

definition
is-basis :: (*'a*::{*field*}^*'n*) *set* => *bool* **where**
is-basis *S* ≡ *vec.independent* *S* ∧ *vec.span* *S* = *UNIV*

lemma *card-finite*:
assumes *card* *S* = *CARD*(*'n*::*finite*)

shows *finite S*
⟨*proof*⟩

lemma *independent-is-basis*:
fixes B :: ($'a$::{*field*}$\hat{}'n$) *set*
shows *vec.independent* $B \wedge$ *card* $B = CARD('n) \longleftrightarrow$ *is-basis* B
⟨*proof*⟩

lemma *basis-finite*:
fixes B :: ($'a$::{*field*}$\hat{}'n$) *set*
assumes *is-basis* B
shows *finite* B
⟨*proof*⟩

Here ends the statements obtained from AFP: http://isa-afp.org/browser_info/current/HOL/Tarskis_Geometry/Linear_Algebra2.html which have been generalized.

3.4 Basic properties involving span, linearity and dimensions

context *finite-dimensional-vector-space*
begin

This theorem is the reciprocal theorem of *local.independent* $?B \Longrightarrow$ *finite* $?B \wedge$ *card* $?B =$ *local.dim* (*local.span* $?B$)

lemma *card-eq-dim-span-indep*:
assumes *dim* (*span* A) $=$ *card* A **and** *finite* A
shows *independent* A
⟨*proof*⟩

lemma *dim-zero-eq*:
assumes *dim-A*: *dim* $A = 0$
shows $A = \{\} \vee A = \{0\}$
⟨*proof*⟩

lemma *dim-zero-eq'*:
assumes A: $A = \{\} \vee A = \{0\}$
shows *dim* $A = 0$
⟨*proof*⟩

lemma *dim-zero-subspace-eq*:
assumes *subs-A*: *subspace* A
shows (*dim* $A = 0$) $=$ ($A = \{0\}$)
⟨*proof*⟩

lemma *span-0-imp-set-empty-or-0*:
assumes *span* $A = \{0\}$
shows $A = \{\} \vee A = \{0\}$ ⟨*proof*⟩

end

context *Vector-Spaces.linear*
begin

lemma *linear-injective-ker-0*:
  shows *inj f* = ({*x*. *f x* = *0*} = {*0*})
  ⟨*proof*⟩

end

lemma *snd-if-conv*:
  shows *snd* (*if P then* (*A*,*B*) *else* (*C*,*D*))=(*if P then B else D*) ⟨*proof*⟩

3.5 Basic properties about matrix multiplication

lemma *row-matrix-matrix-mult*:
  fixes $A::'a::\{comm\text{-}ring\text{-}1\}^{\frown}'n^{\frown}'m$
  shows (*P* \$ *i*) *v*∗ *A* = (*P* ∗∗ *A*) \$ *i*
  ⟨*proof*⟩

corollary *row-matrix-matrix-mult'*:
  fixes $A::'a::\{comm\text{-}ring\text{-}1\}^{\frown}'n^{\frown}'m$
  shows (*row i P*) *v*∗ *A* = *row i* (*P* ∗∗ *A*)
  ⟨*proof*⟩

lemma *column-matrix-matrix-mult*:
  shows *column i* (*P*∗∗*A*) = *P* ∗*v* (*column i A*)
  ⟨*proof*⟩

lemma *matrix-matrix-mult-inner-mult*:
  shows (*A* ∗∗ *B*) \$ *i* \$ *j* = *row i A* · *column j B*
  ⟨*proof*⟩

lemma *matrix-vmult-column-sum*:
  fixes $A::'a::\{field\}^{\frown}'n^{\frown}'m$
  shows ∃*f*. *A* ∗*v* *x* = *sum* (λ*y*. *f y* ∗*s* *y*) (*columns A*)
⟨*proof*⟩

3.6 Properties about invertibility

lemma *matrix-inv*:
  assumes *invertible M*
  shows *matrix-inv-left*: *matrix-inv M* ∗∗ *M* = *mat 1*
    and *matrix-inv-right*: *M* ∗∗ *matrix-inv M* = *mat 1*
  ⟨*proof*⟩

In the library, *matrix-inv ?A* = (*SOME A'*. *?A* ∗∗ *A'* = *mat* (*1*::*?'a*) ∧ *A'* ∗∗ *?A* = *mat* (*1*::*?'a*)) allows the use of non squary matrices. The following lemma can be also proved fixing A

lemma *matrix-inv-unique*:
  fixes *A*::*'a*::{*semiring-1*}^*'n*^*'n*
  assumes *AB*: *A* ** *B* = *mat 1* **and** *BA*: *B* ** *A* = *mat 1*
  shows *matrix-inv A* = *B*
  ⟨*proof*⟩

lemma *matrix-vector-mult-zero-eq*:
  assumes *P*: *invertible P*
  shows ((*P****A*)**v x* = *0*) = (*A* **v x* = *0*)
⟨*proof*⟩

lemma *independent-image-matrix-vector-mult*:
  fixes *P*::*'a*::{*field*}^*'n*^*'m*
  assumes *ind-B*: *vec.independent B* **and** *inv-P*: *invertible P*
  shows *vec.independent* (((**v*) *P*)‘ *B*)
⟨*proof*⟩

lemma *independent-preimage-matrix-vector-mult*:
fixes *P*::*'a*::{*field*}^*'n*^*'n*
assumes *ind-B*: *vec.independent* (((**v*) *P*)‘ *B*) **and** *inv-P*: *invertible P*
shows *vec.independent B*
⟨*proof*⟩

3.7 Properties about the dimension of vectors

lemma *dimension-vector*[*code-unfold*]: *vec.dimension TYPE*(*'a*::{*field*}) *TYPE*(*'rows*::{*mod-type*})=*CARD*(*'r*
⟨*proof*⟩

3.8 Instantiations and interpretations

Functions between two real vector spaces form a real vector

instantiation *fun* :: (*real-vector*, *real-vector*) *real-vector*
begin

definition *scaleR-fun a f* = ($\lambda i.\ a *_R f\ i$)

instance
  ⟨*proof*⟩
end

instantiation *vec* :: (*type*, *finite*) *equal*
begin
definition *equal-vec* :: (*'a*, *'b*::*finite*) *vec* => (*'a*, *'b*::*finite*) *vec* => *bool*
  where *equal-vec x y* = (∀ *i*. *x*$*i* = *y*$*i*)
instance

⟨*proof*⟩
end

interpretation *matrix*: *vector-space* $((*k))$::*'a*::{*field*}=>*'a*^*'cols*^*'rows*=>*'a*^*'cols*^*'rows*
⟨*proof*⟩

end

4 Fundamental Subspaces

theory *Fundamental-Subspaces*
imports
Miscellaneous
begin

4.1 The fundamental subspaces of a matrix

4.1.1 Definitions

definition *left-null-space* :: *'a*::{*semiring-1*}^*'n*^*'m* => (*'a*^*'m*) *set*
where *left-null-space* A = {*x*. *x v*∗ A = *0*}

definition *null-space* :: *'a*::{*semiring-1*}^*'n*^*'m* => (*'a*^*'n*) *set*
where *null-space* A = {*x*. A ∗*v* *x* = *0*}

definition *row-space* :: *'a*::{*field*}^*'n*^*'m*=>(*'a*^*'n*) *set*
where *row-space* A = *vec.span* (*rows* A)

definition *col-space* :: *'a*::{*field*}^*'n*^*'m*=>(*'a*^*'m*) *set*
where *col-space* A = *vec.span* (*columns* A)

4.1.2 Relationships among them

lemma *left-null-space-eq-null-space-transpose*: *left-null-space* A = *null-space* (*transpose* A)
⟨*proof*⟩

lemma *null-space-eq-left-null-space-transpose*: *null-space* A = *left-null-space* (*transpose* A)
⟨*proof*⟩

lemma *row-space-eq-col-space-transpose*:
fixes A::*'a*::{*field*}^*'columns*^*'rows*
shows *row-space* A = *col-space* (*transpose* A)
⟨*proof*⟩

lemma *col-space-eq-row-space-transpose*:
fixes A::*'a*::{*field*}^*'n*^*'m*
shows *col-space* A = *row-space* (*transpose* A)

⟨*proof*⟩

4.2 Proving that they are subspaces

lemma *subspace-null-space*:
fixes $A::'a::\{field\}$^$'n$^$'m$
shows *vec.subspace* (*null-space* A)
⟨*proof*⟩

lemma *subspace-left-null-space*:
fixes $A::'a::\{field\}$^$'n$^$'m$
shows *vec.subspace* (*left-null-space* A)
⟨*proof*⟩

lemma *subspace-row-space*:
shows *vec.subspace* (*row-space* A) ⟨*proof*⟩

lemma *subspace-col-space*:
shows *vec.subspace* (*col-space* A) ⟨*proof*⟩

4.3 More useful properties and equivalences

lemma *col-space-eq*:
fixes $A::'a::\{field\}$^$'m::\{finite, wellorder\}$^$'n$
shows *col-space* $A = \{y.\ \exists x.\ A *v\ x = y\}$
⟨*proof*⟩

corollary *col-space-eq'*:
fixes $A::'a::\{field\}$^$'m::\{finite, wellorder\}$^$'n$
shows *col-space* A = *range* $(\lambda x.\ A *v\ x)$
⟨*proof*⟩

lemma *row-space-eq*:
fixes $A::'a::\{field\}$^$'m$^$'n::\{finite, wellorder\}$
shows *row-space* $A = \{w.\ \exists y.$ (*transpose* A) $*v\ y = w\}$
⟨*proof*⟩

lemma *null-space-eq-ker*:
fixes f::($'a$::*field*^$'n$) => ($'a$^$'m$)
assumes *lf*: *Vector-Spaces.linear* $(*s)$ $(*s)$ f
shows *null-space* (*matrix* f) $= \{x.\ f\ x = 0\}$
⟨*proof*⟩

lemma *col-space-eq-range*:
fixes f::($'a$::*field*^$'n::\{finite, wellorder\}$) ⇒ ($'a$^$'m$)
assumes *lf*: *Vector-Spaces.linear* $(*s)$ $(*s)$ f
shows *col-space* (*matrix* f) = *range* f
⟨*proof*⟩

lemma *null-space-is-preserved*:
  fixes $A::'a::\{field\}^{\frown}'cols^{\frown}'rows$
  assumes *P*: *invertible P*
  shows *null-space* $(P**A)$ = *null-space A*
  ⟨*proof*⟩

lemma *row-space-is-preserved*:
  fixes $A::'a::\{field\}^{\frown}'cols^{\frown}'rows::\{finite, wellorder\}$
    and $P::'a::\{field\}^{\frown}'rows::\{finite, wellorder\}^{\frown}'rows::\{finite, wellorder\}$
  assumes *P*: *invertible P*
  shows *row-space* $(P**A)$ = *row-space A*
⟨*proof*⟩
end

5 Rank Nullity Theorem of Linear Algebra

theory *Dim-Formula*
  imports *Fundamental-Subspaces*
begin

context *vector-space*
begin

5.1 Previous results

Linear dependency is a monotone property, based on the monotonocity of linear independence:

lemma *dependent-mono*:
  assumes *d*:*dependent A*
  and *A-in-B*: $A \subseteq B$
  shows *dependent B*
  ⟨*proof*⟩

Given a finite independent set, a linear combination of its elements equal to zero is possible only if every coefficient is zero:

lemma *scalars-zero-if-independent*:
  assumes *fin-A*: *finite A*
  and *ind*: *independent A*
  and *sum*: $(\sum x \in A.\ scale\ (f\ x)\ x) = 0$
  shows $\forall x \in A.\ f\ x = 0$
  ⟨*proof*⟩

end

context *finite-dimensional-vector-space*
begin

In an finite dimensional vector space, every independent set is finite, and thus

$[\![$*finite* A; *local.independent* A; $(\sum x{\in}A.\ f\ x *s\ x) = (0{::}'b)]\!]$
$\Longrightarrow \forall\, x{\in}A.\ f\ x = (0{::}'a)$

holds:

corollary *scalars-zero-if-independent-euclidean*:
  assumes *ind*: *independent A*
    and *sum*: $(\sum x{\in}A.\ scale\ (f\ x)\ x) = 0$
  shows $\forall\, x \in A.\ f\ x = 0$
  ⟨*proof*⟩

end

The following lemma states that every linear form is injective over the elements which define the basis of the range of the linear form. This property is applied later over the elements of an arbitrary basis which are not in the basis of the nullifier or kernel set (*i.e.*, the candidates to be the basis of the range space of the linear form).

Thanks to this result, it can be concluded that the cardinal of the elements of a basis which do not belong to the kernel of a linear form f is equal to the cardinal of the set obtained when applying f to such elements.

The application of this lemma is not usually found in the pencil and paper proofs of the "rank nullity theorem", but will be crucial to know that, being f a linear form from a finite dimensional vector space V to a vector space V', and given a basis B of *ker f*, when B is completed up to a basis of V with a set W, the cardinal of this set is equal to the cardinal of its range set:

context *vector-space*
begin

lemma *inj-on-extended*:
  assumes *lf*: *Vector-Spaces.linear scaleB scaleC f*
  and *f*: *finite C*
  and *ind-C*: *independent C*
  and *C-eq*: $C = B \cup W$
  and *disj-set*: $B \cap W = \{\}$
  and *span-B*: $\{x.\ f\ x = 0\} \subseteq span\ B$
  shows *inj-on f W*
  — The proof is carried out by reductio ad absurdum
⟨*proof*⟩
end

5.2 The proof

Now the rank nullity theorem can be proved; given any linear form *f*, the sum of the dimensions of its kernel and range subspaces is equal to the dimension of the source vector space.

The statement of the “rank nullity theorem for linear algebra”, as well as its proof, follow the ones on [1]. The proof is the traditional one found in the literature. The theorem is also named “fundamental theorem of linear algebra” in some texts (for instance, in [2]).

context *finite-dimensional-vector-space*
begin

theorem *rank-nullity-theorem*:
assumes *l*: *Vector-Spaces.linear scale scaleC f*
shows *dimension* = *dim* {*x*. *f x* = *0*} + *vector-space.dim scaleC* (*range f*)
⟨*proof*⟩

end

5.3 The rank nullity theorem for matrices

The proof of the theorem for matrices is direct, as a consequence of the “rank nullity theorem”.

lemma *rank-nullity-theorem-matrices*:
fixes *A*::*'a*::{*field*}^*'cols*::{*finite*, *wellorder*}^*'rows*
shows *ncols A* = *vec.dim* (*null-space A*) + *vec.dim* (*col-space A*)
⟨*proof*⟩

end

References

[1] S. Axler. *Linear Algebra Done Right*. Springer, 2nd edition, 1997.

[2] M. S. Gockenbach. *Finite Dimensional Linear Algebra*. CRC Press, 2010.