Hermite Normal Form

By Jose Divasón and Jesús Aransay*

February 6, 2026

Abstract

The Hermite Normal Form is a canonical matrix analogue of Reduced Echelon Form, but involving matrices over more general rings. In this work we formalise an algorithm to compute the Hermite Normal Form of a matrix by means of elementary row operations, taking advantage of the Echelon Form AFP entry. We have proven the correctness of such an algorithm and refined it to immutable arrays. Furthermore, we have also formalised the uniqueness of the Hermite Normal Form of a matrix. Code can be exported and some examples of execution involving $\mathbb{Z}$-matrices and $\mathbb{K}[x]$-matrices are presented as well.

Contents

---

*This research has been funded by the research grant FPI-UR-12 of the Universidad de La Rioja and by the project MTM2014-54151-P from Ministerio de Economía y Competitividad (Gobierno de España).

1 Hermite Normal Form

theory *Hermite*
imports
Echelon-Form.Echelon-Form-Inverse
Echelon-Form.Examples-Echelon-Form-Abstract
HOL−Computational-Algebra.Euclidean-Algorithm
begin

1.1 Some previous properties

1.1.1 Rings

subclass (**in** *bezout-ring-div*) *euclidean-ring*
⟨*proof*⟩

1.1.2 Polynomials

lemma *coeff-dvd-poly*: $[:\mathit{coeff}\ a\ (\mathit{degree}\ a):]\ \mathit{dvd}\ (a{::}'a{::}\{\mathit{field}\}\ \mathit{poly})$
⟨*proof*⟩

lemma *poly-dvd-antisym2*:
fixes $p\ q :: 'a{::}\mathit{field}\ \mathit{poly}$
assumes *dvd1*: $p\ \mathit{dvd}\ q$ **and** *dvd2*: $q\ \mathit{dvd}\ p$
shows $p\ \mathit{div}\ [:\mathit{coeff}\ p\ (\mathit{degree}\ p):] = q\ \mathit{div}\ [:\mathit{coeff}\ q\ (\mathit{degree}\ q):]$
⟨*proof*⟩

1.1.3 Units

lemma *unit-prod*:
assumes *finite S*
shows $\mathit{is\text{-}unit}\ (\mathit{prod}\ (\lambda i.\ U\ \$\ i\ \$\ i)\ S) = (\forall i{\in}S.\ \mathit{is\text{-}unit}\ (U\ \$\ i\ \$\ i))$
⟨*proof*⟩

1.1.4 Upper triangular matrices

lemma *is-unit-diagonal*:
fixes *U*::*'a*::{*comm-ring-1*, *algebraic-semidom*}^*'n*::{*finite*, *wellorder*}^*'n*::{*finite*, *wellorder*}
assumes *U*: *upper-triangular U*
and *det-U*: *is-unit* (*det U*)
shows ∀ *i*. *is-unit* (*U* $ *i* $ *i*)
⟨*proof*⟩

lemma *upper-triangular-mult*:
fixes *A*::*'a*::{*semiring-1*}^*'n*::{*mod-type*}^*'n*::{*mod-type*}
assumes *A*: *upper-triangular A*
and *B*: *upper-triangular B*
shows *upper-triangular* (*A**B*)
⟨*proof*⟩

lemma *upper-triangular-adjugate*:
fixes *A*::((*'a*::*comm-ring-1*,*'n*::{*wellorder*, *finite*}) *vec*, *'n*) *vec*
assumes *A*: *upper-triangular A*
shows *upper-triangular* (*adjugate A*)
⟨*proof*⟩

lemma *upper-triangular-inverse*:
fixes *A*::((*'a*::{*euclidean-semiring*,*comm-ring-1*},*'n*::{*wellorder*, *finite*}) *vec*, *'n*) *vec*
assumes *A*: *upper-triangular A*
and *inv-A*: *invertible A*
shows *upper-triangular* (*matrix-inv A*)
⟨*proof*⟩

lemma *upper-triangular-mult-diagonal*:
fixes *A*::((*'a*::{*semiring-1*},*'n*::{*wellorder*, *finite*}) *vec*, *'n*) *vec*
assumes *A*: *upper-triangular A*
and *B*: *upper-triangular B*
shows (*A**B*) $ *i* $ *i* = *A* $ *i* $ *i* * *B* $ *i* $ *i*
⟨*proof*⟩

1.1.5 More properties of mod type

lemma *add-left-neutral*:
fixes *a*::*'n*::*mod-type*
shows (*a* + *b* = *a*) = (*b* = *0*)
⟨*proof*⟩

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

1.1.6 Div and Mod

lemma *dvd-minus-eq-mod*:
fixes *c::′a::unique-euclidean-ring*
assumes $c \neq 0$ **and** $c\ dvd\ a - b$ **shows** $a\ mod\ c = b\ mod\ c$
⟨*proof*⟩

lemma *eq-mod-dvd-minus*:
fixes *c::′a::unique-euclidean-ring*
assumes $c \neq 0$ **and** $a\ mod\ c = b\ mod\ c$
shows $c\ dvd\ a - b$
⟨*proof*⟩

lemma *dvd-cong-not-eq-mod*:
fixes *c::′a::unique-euclidean-ring*
assumes $xa\ mod\ c \neq xb$ **and** $c\ dvd\ xa\ mod\ c - xb$ **and** $c \neq 0$
shows $xb\ mod\ c \neq xb$
⟨*proof*⟩

lemma *diff-mod-cong-0*:
fixes *c::′a::unique-euclidean-ring*
assumes $xa\ mod\ c \neq xb\ mod\ c$ **and** $c\ dvd\ xa\ mod\ c - xb\ mod\ c$ **shows** $c = 0$
⟨*proof*⟩

lemma *cong-diff-mod*:
fixes *c::′a::unique-euclidean-ring*
assumes $xa \neq xb$ **and** $c\ dvd\ xa - xb$ **and** $xa = xa\ mod\ c$ **shows** $xb \neq xb\ mod\ c$
⟨*proof*⟩

lemma *exists-k-mod*:
fixes *c::′a::unique-euclidean-ring*
shows $\exists\, k.\ a\ mod\ c = a + k * c$
⟨*proof*⟩

1.2 Units, associated and congruent relations

context *semiring-1*
begin

definition $Units = \{x{::}'a.\ (\exists\, k.\ 1 = x * k)\}$

end

context *ring-1*
begin

definition $cong{::}'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where $cong\ a\ c\ b = (\exists\, k.\ (a - c) = b * k)$

lemma *cong-eq*: *cong a c b* = (*b dvd* (*a* − *c*))
⟨*proof*⟩

end

context *normalization-semidom*
begin

lemma *Units-eq*: *Units* = {*x*. *x dvd 1*} ⟨*proof*⟩

lemma *normalize-Units*: *x* ∈ *Units* ⟹ *normalize x* = *1*
⟨*proof*⟩

lemma *associated-eq*: (*normalize a* = *normalize b*) ⟷ (∃ *u*∈*Units*. *a* = *u*∗*b*)
⟨*proof*⟩

end

context *unique-euclidean-ring*
begin

definition *associated-rel* = {(*a*,*b*). *normalize a* = *normalize b*}

lemma *equiv-associated*:
shows *equiv UNIV associated-rel*
⟨*proof*⟩

definition *congruent-rel b* = {(*a*,*c*). *cong a c b*}

lemma *relf-congruent-rel*: *refl* (*congruent-rel b*)
⟨*proof*⟩

lemma *sym-congruent-rel*: *sym* (*congruent-rel b*)
⟨*proof*⟩

lemma *trans-congruent-rel*: *trans* (*congruent-rel b*)
⟨*proof*⟩

lemma *equiv-congruent*: *equiv UNIV* (*congruent-rel b*)
⟨*proof*⟩

end

1.3 Associates and residues functions

context *normalization-semidom*
begin

definition *ass-function* :: (′*a* ⇒ ′*a*) ⇒ *bool*

where *ass-function f*
= ((∀ *a*. *normalize a* = *normalize* (*f a*)) ∧ *pairwise* (λ*a b*. *normalize a* ≠ *normalize b*) (*range f*))

definition *Complete-set-non-associates S*
= (∃ *f*. *ass-function f* ∧ *f*‘*UNIV* = *S* ∧ (*pairwise* (λ*a b*. *normalize a* ≠ *normalize b*) *S*))

end

context *ring-1*
begin

definition *res-function* :: (′*a* ⇒ ′*a* ⇒ ′*a*) ⇒ *bool*
where *res-function f* = (∀ *c*. (∀ *a b*. *cong a b c* ⟷ *f c a* = *f c b*)
∧ *pairwise* (λ*a b*. ¬ *cong a b c*) (*range* (*f c*))
∧ (∀ *a*. ∃ *k*. *f c a* = *a* + *k*∗*c*))

definition *Complete-set-residues g*
= (∃ *f*. *res-function f* ∧ (∀ *c*. (*pairwise* (λ*a b*. ¬ *cong a b c*) (*f c*‘*UNIV*)) ∧ *g c* = *f c*‘*UNIV*))
end

lemma *ass-function-Complete-set-non-associates*:
assumes *f*: *ass-function f*
shows *Complete-set-non-associates* (*f*‘*UNIV*)
⟨*proof*⟩

lemma *in-Ass-not-associated*:
assumes *Ass-S*: *Complete-set-non-associates S*
and *x*: *x*∈*S* **and** *y*: *y*∈*S* **and** *x-not-y*: *x*≠*y*
shows *normalize x* ≠ *normalize y*
⟨*proof*⟩

lemma *ass-function-0*:
assumes *r*: *ass-function ass*
shows (*ass x* = *0*) = (*x* = *0*)
⟨*proof*⟩

lemma *ass-function-0*′:
assumes *r*: *ass-function ass*
shows (*ass x div x* = *0*) = (*x*=*0*)
⟨*proof*⟩

lemma *res-function-Complete-set-residues*:
assumes *f*: *res-function f*
shows *Complete-set-residues* (λ*c*. (*f c*)‘*UNIV*)
⟨*proof*⟩

lemma *in-Res-not-congruent*:
  assumes *res-g*: *Complete-set-residues g*
  and *x*: *x* ∈ *g b* **and** *y*: *y* ∈ *g b* **and** *x-not-y*: *x*≠*y*
  shows ¬ *cong x y b*
  ⟨*proof*⟩

1.3.1 Concrete instances in Euclidean rings

definition *ass-function-euclidean* (*p*::*'a*::{*normalization-euclidean-semiring*, *euclidean-ring*}) = *normalize p*
definition *res-function-euclidean b* (*n*::*'a*::{*euclidean-ring*}) = (*if b* = *0 then n else* (*n mod b*))

lemma *ass-function-euclidean*: *ass-function ass-function-euclidean*
  ⟨*proof*⟩

lemma *res-function-euclidean*:
  res-function (*res-function-euclidean* :: *'a* :: *unique-euclidean-ring* ⇒ -)
  ⟨*proof*⟩

1.3.2 Concrete case of the integer ring

definition *ass-function-int* (*n*::*int*) = *abs n*

lemma *ass-function-int*: *ass-function-int* = *ass-function-euclidean*
  ⟨*proof*⟩

lemma *ass-function-int-UNIV*: (*ass-function-int*‘ *UNIV*) = {*x*. *x*≥*0*}
  ⟨*proof*⟩

1.4 Definition of Hermite Normal Form

It is worth noting that there is not a single definition of Hermite Normal Form in the literature. For instance, some authors restrict their definitions to the case of square nonsingular matrices. Other authors just work with integer matrices. Furthermore, given a matrix A its Hermite Normal Form H can be defined to be upper triangular or lower triangular. In addition, the transformation from A to H can be made by means of elementary row operations or elementary column operations. In our case, we will work as general as possible, so our input will be any matrix (including nonsquare ones). The output will be an upper triangular matrix obtained by means of elementary row operations.

Hence, given a complete set of nonassociates and a complete set of residues, H is said to be in Hermite Normal Form if:

1. H is in Echelon Form

2. The first nonzero element of a nonzero row belongs to the complete set of nonassociates
3. Let h be the first nonzero element of a nonzero row. Then each element above h belongs to the corresponding complete set of residues of h

A matrix H is the Hermite Normal Form of a matrix A if:

1. There exists an invertible matrix P such that $A = PH$
2. H is in Hermite Normal Form

The Hermite Normal Form is usually applied to integer matrices. As we have already said, there is no one single definition of it, so some authors impose different conditions. In the particular case of integer matrices, leading coefficients (the first nonzero element of a nonzero row) are usually required to be positive, but it is also possible to impose them to be negative since we would only have to multiply by -1.

In the case of the elements h_{ik} above a leading coefficient h_{ij}, some authors demand $0 \leq h_{ik} < h_{ij}$, other ones impose the conditions $h_{ik} \leq 0$ and $\mid h_{ik} \mid < h_{ij}$, and other ones $-\frac{h_{ij}}{2} < h_{ik} \leq \frac{h_{ij}}{2}$. More different options are also possible.

All the possibilities can be represented selecting a complete set of nonassociates and a complete set of residues. The algorithm to compute the Hermite Normal Form will be parameterised by functions which obtain the appropriate leading coefficient and the suitable elements above them. We can execute the algorithm with different functions to get exactly which Hermite Normal Form we want. Once we fix such a complete set of nonassociates and the corresponding complete set of residues, the Hermite Normal Form is unique.

1.4.1 Echelon form up to row k

We present the definition of echelon form up to a row k (not included).

definition *echelon-form-upt-row A k* =
(
(∀ *i. to-nat i* < *k* ∧ *is-zero-row i A* ⟶ ¬ (∃ *j. j*>*i* ∧ *to-nat j* < *k* ∧ ¬ *is-zero-row j A*)) ∧
(∀ *i j. i* < *j* ∧ *to-nat j* < *k* ∧ ¬ *is-zero-row i A* ∧ ¬ *is-zero-row j A* ⟶ (*LEAST n. A* $ *i* $ *n* ≠ *0*) < (*LEAST n. A* $ *j* $ *n* ≠ *0*))
)

lemma *echelon-form-upt-row-condition1-explicit*:
assumes *echelon-form-upt-row A k*
and *to-nat i* < *k* **and** *is-zero-row i A*

shows $\neg\ (\exists j.\ j>i \wedge \textit{to-nat}\ j < k \wedge \neg\ \textit{is-zero-row}\ j\ A)$
⟨*proof*⟩

lemma *echelon-form-upt-row-condition1-explicit′*:
assumes $\textit{echelon-form-upt-row}\ A\ k$
and $\textit{to-nat}\ i < k$ **and** $\textit{is-zero-row}\ i\ A$ **and** $i \leq j$ **and** $\textit{to-nat}\ j < k$
shows $\textit{is-zero-row}\ j\ A$
⟨*proof*⟩

lemma *echelon-form-upt-row-condition1-explicit-neg*:
assumes $\textit{echelon-form-upt-row}\ A\ k$
and *iA*: $\neg\ \textit{is-zero-row}\ i\ A$ **and** *ia-i*: $ia < i$
and *i*: $\textit{to-nat}\ i < k$
shows $\neg\ \textit{is-zero-row}\ ia\ A$
⟨*proof*⟩

lemma *echelon-form-upt-row-condition2-explicit*:
assumes $\textit{echelon-form-upt-row}\ A\ k$
and $ia < j$ **and** $\textit{to-nat}\ j < k$ **and** $\neg\ \textit{is-zero-row}\ ia\ A$ **and** $\neg\ \textit{is-zero-row}\ j\ A$
shows $(\textit{LEAST}\ n.\ A\ \$\ ia\ \$\ n \neq 0) < (\textit{LEAST}\ n.\ A\ \$\ j\ \$\ n \neq 0)$
⟨*proof*⟩

lemma *echelon-form-upt-row-intro*:
assumes$(\forall i.\ \textit{to-nat}\ i < k \wedge \textit{is-zero-row}\ i\ A \longrightarrow \neg\ (\exists j.\ i<j \wedge \textit{to-nat}\ j < k \wedge \neg\ \textit{is-zero-row}\ j\ A))$
and $(\forall i\ j.\ i < j \wedge \textit{to-nat}\ j < k \wedge \neg\ \textit{is-zero-row}\ i\ A \wedge \neg\ \textit{is-zero-row}\ j\ A \longrightarrow (\textit{LEAST}\ n.\ A\ \$\ i\ \$\ n \neq 0) < (\textit{LEAST}\ n.\ A\ \$\ j\ \$\ n \neq 0))$
shows $\textit{echelon-form-upt-row}\ A\ k$
⟨*proof*⟩

lemma *echelon-form-echelon-form-upt-row*: $\textit{echelon-form}\ A = \textit{echelon-form-upt-row}\ A\ (\textit{nrows}\ A)$
⟨*proof*⟩

1.4.2 Hermite Normal Form up to row k

Predicate to check if a matrix is in Hermite Normal form up to row k (not included).

definition $\textit{Hermite-upt-row}\ A\ k\ \textit{associates}\ \textit{residues} =$
(
$\textit{Complete-set-non-associates}\ \textit{associates} \wedge$
$\textit{Complete-set-residues}\ \textit{residues} \wedge$
$\textit{echelon-form-upt-row}\ A\ k \wedge$
$(\forall i.\ \textit{to-nat}\ i < k \wedge \neg\ \textit{is-zero-row}\ i\ A \longrightarrow A\ \$\ i\ \$\ (\textit{LEAST}\ n.\ A\ \$\ i\ \$\ n \neq 0) \in \textit{associates}) \wedge$
$(\forall i.\ \textit{to-nat}\ i < k \wedge \neg\ \textit{is-zero-row}\ i\ A \longrightarrow (\forall j<i.\ A\ \$\ j\ \$\ (\textit{LEAST}\ n.\ A\ \$\ i\ \$\ n \neq 0) \in \textit{residues}\ (A\ \$\ i\ \$\ (\textit{LEAST}\ n.\ A\ \$\ i\ \$\ n \neq 0))))$
)

The definition of Hermite Normal Form is now introduced:

definition *Hermite*::*'a*::{*bezout-ring-div*,*normalization-semidom*} *set* ⇒ (*'a* ⇒ *'a set*) ⇒
((*'a*, *'b*::{*mod-type*}) *vec*, *'c*::{*mod-type*}) *vec* ⇒ *bool*
where *Hermite associates residues A* = (
Complete-set-non-associates associates
∧ (*Complete-set-residues residues*)
∧ *echelon-form A*
∧ (∀ *i*. ¬ *is-zero-row i A* ⟶ *A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) ∈ *associates*)
∧ (∀ *i*. ¬ *is-zero-row i A* ⟶ (∀ *j*. *j*<*i* ⟶ *A* $ *j* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) ∈ *residues* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))))
)

lemma *Hermite-Hermite-upt-row*: *Hermite ass res A* = *Hermite-upt-row A* (*nrows A*) *ass res*
⟨*proof*⟩

lemma *Hermite-intro*:
assumes *Complete-set-non-associates associates*
and *Complete-set-residues residues*
and *echelon-form A*
and (∀ *i*. ¬ *is-zero-row i A* ⟶ *A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) ∈ *associates*)
and (∀ *i*. ¬ *is-zero-row i A* ⟶ (∀ *j*. *j*<*i* ⟶ *A* $ *j* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) ∈ *residues* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))))
shows *Hermite associates residues A*
⟨*proof*⟩

1.5 Definition of an algorithm to compute the Hermite Normal Form

The algorithm is parameterised by three functions:

- The function that computes de Bézout identity (necessary to compute the echelon form).
- The function that given an element, it returns its representative element in the associated equivalent class, which will be an element in the complete set of nonassociates.
- The function that given two elements a and b, it returns its representative element in the congruent equivalent class of b, which will be an element in the complete set of residues of b.

primrec *Hermite-reduce-above* :: *'a*::*unique-euclidean-ring*^*'cols*::*mod-type*^*'rows*::*mod-type*⇒*nat*
⇒*'rows*⇒*'cols*⇒ (*'a*⇒*'a*⇒*'a*) ⇒ *'a*^*'cols*::*mod-type*^*'rows*::*mod-type*
where *Hermite-reduce-above A 0 i j res* = *A*
| *Hermite-reduce-above A* (*Suc n*) *i j res* = (*let i'*=((*from-nat n*)::*'rows*);
Aij = *A* $ *i* $ *j*;

Ai′j = A $ i′ $ j
in
Hermite-reduce-above (row-add A i′ i (((res Aij (Ai′j)) − (Ai′j)) div Aij)) n i j res)

definition *Hermite-of-row-i ass res A i = (*
if is-zero-row i A
then A
else
let j = (LEAST n. A $ i $ n ≠ 0); Aij= (A $ i $ j);
A′ = mult-row A i ((ass Aij) div Aij)
in Hermite-reduce-above A′ (to-nat i) i j res)

definition *Hermite-of-upt-row-i A i ass res = foldl (Hermite-of-row-i ass res) A (map from-nat [0..<i])*

definition *Hermite-of A ass res bezout =*
(let A′= echelon-form-of A bezout in Hermite-of-upt-row-i A′ (nrows A) ass res)

1.6 Proving the correctness of the algorithm

1.6.1 The proof

lemma *Hermite-reduce-above-preserves*:
assumes *n*: *n≤to-nat a*
shows *(Hermite-reduce-above A n i j res) $ a $ b = A $ a $ b*
⟨*proof*⟩

lemma *Hermite-reduce-above-works*:
assumes *n*: *n ≤ to-nat i* **and** *a*: *to-nat a < n*
shows *(Hermite-reduce-above A n i j res) $ a $ b*
= row-add A a i ((res (Aij) (Aaj) − (Aaj)) div (Aij)) $ a $ b
⟨*proof*⟩

lemma *Hermite-of-row-preserves-below*:
assumes *i-a*: *i<a*
shows *(Hermite-of-row-i ass res A i) $ a $ b = A $ a $ b*
⟨*proof*⟩

lemma *Hermite-of-row-preserves-previous-cols*:
assumes *b*: *b<(LEAST n. A $ i $ n ≠ 0)*
and *not-zero-i-A*: *¬ is-zero-row i A*
and *e*: *echelon-form A*
shows *(Hermite-of-row-i ass res A i) $ a $ b = A $ a $ b*
⟨*proof*⟩

lemma *echelon-form-Hermite-of-condition1*:
fixes *res ass i A*
defines *M*: *M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A*

$ i $ (LEAST n. A $ i $ n ≠ 0))
defines *H*: *H ≡ Hermite-reduce-above M (to-nat i) i (LEAST n. A $ i $ n ≠ 0) res*
assumes *e*: *echelon-form A*
and *a*: *ass-function ass*
and *not-zero-iA*: *¬ is-zero-row i A*
and *zero-ia-H*: *is-zero-row ia H*
and *ia-j*: *ia < j*
shows *is-zero-row j H*
⟨*proof*⟩

lemma *row-zero-A-imp-row-zero-H*:
fixes *res ass i A*
defines *M*: *M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A $ i $ (LEAST n. A $ i $ n ≠ 0))*
defines *H*: *H ≡ Hermite-reduce-above M (to-nat i) i (LEAST n. A $ i $ n ≠ 0) res*
assumes *e*: *echelon-form A*
and *not-zero-iA*: *¬ is-zero-row i A*
and *zero-j-A*: *is-zero-row j A*
shows *is-zero-row j H*
⟨*proof*⟩

lemma *Hermite-reduce-above-Least-eq-le*:
fixes *res ass i A*
defines *M*: *M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A $ i $ (LEAST n. A $ i $ n ≠ 0))*
defines *H*: *H ≡ Hermite-reduce-above M (to-nat i) i (LEAST n. A $ i $ n ≠ 0) res*
assumes *i-ia*: *i<ia*
and *not-zero-ia-H*: *¬ is-zero-row ia H*
shows *(LEAST n. A $ ia $ n ≠ 0) = (LEAST n. H $ ia $ n ≠ 0)*
⟨*proof*⟩

lemma *Hermite-reduce-above-Least-eq*:
fixes *res ass i A*
defines *M*: *M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A $ i $ (LEAST n. A $ i $ n ≠ 0))*
defines *H*: *H ≡ Hermite-reduce-above M (to-nat i) i (LEAST n. A $ i $ n ≠ 0) res*
assumes *a*: *ass-function ass*
and *not-zero-iA*: *¬ is-zero-row i A*
shows *(LEAST n. A $ i $ n ≠ 0) = (LEAST n. H $ i $ n ≠ 0)*
⟨*proof*⟩

lemma *Hermite-reduce-above-Least-eq-ge*:

fixes *res ass i A*
defines *M*: *M* ≡ *mult-row A i* (*ass* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*)) *div A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))
defines *H*: *H* ≡ *Hermite-reduce-above M* (*to-nat i*) *i* (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) *res*
assumes *e*: *echelon-form A*
and *not-zero-iA*: ¬ *is-zero-row i A*
and *not-zero-ia-A*: ¬ *is-zero-row ia A*
and *not-zero-ia-H*: ¬ *is-zero-row ia H*
and *ia-less-i*: *ia* < *i*
shows (*LEAST n*. *H* $ *ia* $ *n* ≠ *0*) = (*LEAST n*. *A* $ *ia* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *Hermite-reduce-above-Least*:
fixes *res ass i A*
defines *M*: *M* ≡ *mult-row A i* (*ass* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))
div A $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))
defines *H*: *H* ≡ *Hermite-reduce-above M* (*to-nat i*) *i* (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) *res*
assumes *e*: *echelon-form A*
and *a*: *ass-function ass*
and *not-zero-iA*: ¬ *is-zero-row i A*
and *not-zero-ia-A*: ¬ *is-zero-row ia A*
and *not-zero-ia-H*: ¬ *is-zero-row ia H*
shows (*LEAST n*. *H* $ *ia* $ *n* ≠ *0*) = (*LEAST n*. *A* $ *ia* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *echelon-form-Hermite-of-condition2*:
fixes *res ass i A*
defines *M*: *M* ≡ *mult-row A i* (*ass* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*)) *div A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*))
defines *H*: *H* ≡ *Hermite-reduce-above M* (*to-nat i*) *i* (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) *res*
assumes *e*: *echelon-form A*
and *a*: *ass-function ass*
and *not-zero-iA*: ¬ *is-zero-row i A*
and *ia-less-j*: *ia* < *j*
and *not-zero-ia-H*: ¬ *is-zero-row ia H*
and *not-zero-j-H*: ¬ *is-zero-row j H*
shows (*LEAST n*. *H* $ *ia* $ *n* ≠ *0*) < (*LEAST n*. *H* $ *j* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *echelon-form-Hermite-of-row*:
assumes *a*: *ass-function ass*
and *res-function res*
and *e*: *echelon-form A*
shows *echelon-form* (*Hermite-of-row-i ass res A i*)

⟨*proof*⟩

lemma *echelon-form-fold-Hermite-of-row-i*:
  assumes *e*: *echelon-form A* **and** *a*: *ass-function ass* **and** *r*: *res-function res*
  shows *echelon-form* (*foldl* (*Hermite-of-row-i ass res*) *A* (*map from-nat* [*0*..<*k*]))
⟨*proof*⟩

lemma *echelon-form-Hermite-of-upt-row-i*:
  assumes *e*: *echelon-form A* **and** *a*: *ass-function ass* **and** *r*: *res-function res*
  shows *echelon-form* (*Hermite-of-upt-row-i A k ass res*)
  ⟨*proof*⟩

lemma *echelon-form-Hermite-of*:
 fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}^*'cols*::{*mod-type*}^*'rows*::{*mod-ty*
  assumes *a*: *ass-function ass*
  and *r*: *res-function res*
  and *b*: *is-bezout-ext bezout*
  shows *echelon-form* (*Hermite-of A ass res bezout*)
  ⟨*proof*⟩

lemma *in-ass-Hermite-of-row*:
  assumes *a*: *ass-function ass*
  and *res-function res*
  and *not-zero-i-A*: ¬ *is-zero-row i A*
  shows (*Hermite-of-row-i ass res A i*) $ *i* $ (*LEAST n*. (*Hermite-of-row-i ass res A i*) $ *i* $ *n* ≠ *0*) ∈ *range ass*
⟨*proof*⟩

lemma *Hermite-of-upt-row-preserves-below*:
  assumes *i*: *to-nat a*≥*k*
  shows *Hermite-of-upt-row-i A k ass res* $ *a* $ *b* = *A* $ *a* $ *b*
  ⟨*proof*⟩

lemma *not-zero-Hermite-reduce-above*:
  fixes *ass i A*
  defines *M*: *M*≡(*mult-row A i* (*ass* (*A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*)) *div A* $ *i* $ (*LEAST n*. *A* $ *i* $ *n* ≠ *0*)))
  assumes *not-zero-a-A*: ¬ *is-zero-row a A*
  and *not-zero-i-A*: ¬ *is-zero-row i A*
  and *e*: *echelon-form A*
  and *a*: *ass-function ass*
  and *n*: *n* ≤ *to-nat i*
  shows ¬ *is-zero-row a* (*Hermite-reduce-above M n i* (*LEAST n*. *A* $ *i* $ *n* ≠ *0*) *res*)
⟨*proof*⟩

lemma *Least-Hermite-of-row-i*:
assumes *i*: ¬ *is-zero-row i A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*
shows (*LEAST n. Hermite-of-row-i ass res A i $ i $ n ≠ 0*) = (*LEAST n. A $ i $ n ≠ 0*)
⟨*proof*⟩

lemma *Least-Hermite-of-row-i2*:
assumes *i*: ¬ *is-zero-row i A* **and** *k*: ¬ *is-zero-row k A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*
shows (*LEAST n. Hermite-of-row-i ass res A k $ i $ n ≠ 0*) = (*LEAST n. A $ i $ n ≠ 0*)
⟨*proof*⟩

lemma *Hermite-of-row-i-works*:
fixes *i A ass*
defines *n*:*n* ≡(*LEAST n. A $ i $ n ≠ 0*)
defines *M*:*M* ≡ (*mult-row A i* (*ass* (*A $ i $ n*) *div A $ i $ n*))
assumes *ai*: *a* < *i*
and *i*: ¬ *is-zero-row i A*
shows *Hermite-of-row-i ass res A i $ a $ b* =
row-add M a i ((*res* (*M $ i $ n*) (*M $ a $ n*)
− *M $ a $ n*) *div M $ i $ n*) *$ a $ b*
⟨*proof*⟩

lemma *Hermite-of-row-i-works2*:
fixes *i A ass*
defines *n*:*n* ≡(*LEAST n. A $ i $ n ≠ 0*)
defines *M*:*M* ≡ (*mult-row A i* (*ass* (*A $ i $ n*) *div A $ i $ n*))
assumes *i*: ¬ *is-zero-row i A*
shows *Hermite-of-row-i ass res A i $ i $ b* = *M $ i $ b*
⟨*proof*⟩

lemma *Hermite-of-upt-row-preserves-nonzero-rows-ge*:
assumes *i*: ¬ *is-zero-row i A* **and** *i2*: *to-nat i≥k*
shows ¬ *is-zero-row i* (*Hermite-of-upt-row-i A k ass res*)
⟨*proof*⟩

lemma *Hermite-of-upt-row-i-Least-ge*:
assumes *i*: ¬ *is-zero-row i A*
and *i2*: *to-nat i*≥*k*
shows (*LEAST n. Hermite-of-upt-row-i A k ass res* $ *i* $ *n* ≠ *0*) = (*LEAST n. A* $ *i* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *Hermite-of-upt-row-i-Least*:
assumes *iA*: ¬ *is-zero-row i A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*
and *r*: *res-function res*
and *k*: *k*≤*nrows A*
shows (*LEAST n. Hermite-of-upt-row-i A k ass res* $ *i* $ *n* ≠ *0*) = (*LEAST n. A* $ *i* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *Hermite-of-upt-row-preserves-nonzero-rows*:
assumes *i*: ¬ *is-zero-row i A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*
and *r*: *res-function res*
and *k*: *k*≤*nrows A*
shows ¬ *is-zero-row i* (*Hermite-of-upt-row-i A k ass res*)
⟨*proof*⟩

lemma *Hermite-of-upt-row-i-in-range*:
fixes *k ass res*
assumes *not-zero-i-A*: ¬ *is-zero-row i A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*
and *r*: *res-function res*
and *k*: *to-nat i*<*k*
and *k2*: *k*≤*nrows A*
shows *Hermite-of-upt-row-i A k ass res* $ *i* $ (*LEAST n. A* $ *i* $ *n* ≠ *0*) ∈ *range ass*
⟨*proof*⟩

lemma *Hermite-of-upt-row-preserves-zero-rows-ge*:
assumes *i*: *is-zero-row i A*
and *k*: *k* ≤ *nrows A*
and *ik*: *to-nat i*≥*k*
shows *is-zero-row i* (*Hermite-of-upt-row-i A k ass res*)
⟨*proof*⟩

lemma *Hermite-of-upt-row-preserves-zero-rows*:
fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}^*'cols*::{*mod-type*}^*'rows*::{*mod-ty*
assumes *i*: *is-zero-row i A*
and *e*: *echelon-form A* **and** *a*: *ass-function ass* **and** *r*: *res-function res* **and** *k*: *k*
≤ *nrows A*
shows *is-zero-row i* (*Hermite-of-upt-row-i A k ass res*)
⟨*proof*⟩

lemma *Hermite-of-preserves-zero-rows*:
fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}^*'cols*::{*mod-type*}^*'rows*::{*mod-t*
assumes *i*: *is-zero-row i* (*echelon-form-of A bezout*)
and *a*: *ass-function ass*
and *r*: *res-function res*
and *b*: *is-bezout-ext bezout*
shows *is-zero-row i* (*Hermite-of A ass res bezout*)
⟨*proof*⟩

lemma *Hermite-of-Least*:
fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}^*'cols*::{*mod-type*}^*'rows*::{*mod-t*
assumes *i*: ¬ *is-zero-row i* (*Hermite-of A ass res bezout*)
and *a*: *ass-function ass*
and *r*: *res-function res*
and *b*: *is-bezout-ext bezout*
shows (*LEAST n. Hermite-of A ass res bezout* $ *i* $ *n* ≠ *0*) = (*LEAST n.*
(*echelon-form-of A bezout*) $ *i* $ *n* ≠ *0*)
⟨*proof*⟩

lemma *in-associates-Hermite-of*:
fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}^*'cols*::{*mod-type*}^*'rows*::{*mod-t*
assumes *a*: *ass-function ass*
and *r*: *res-function res*
and *b*: *is-bezout-ext bezout*
and *i*: ¬ *is-zero-row i* (*Hermite-of A ass res bezout*)
shows *Hermite-of A ass res bezout* $ *i* $ (*LEAST n. Hermite-of A ass res bezout*
$ *i* $ *n* ≠ *0*) ∈ *range ass*
⟨*proof*⟩

lemma *Hermite-of-row-i-range-res*:
assumes *ji*: *j*<*i* **and** *not-zero-i-A*: ¬ *is-zero-row i A* **and** *r*: *res-function res*
shows *Hermite-of-row-i ass res A i* $ *j* $ (*LEAST n. A* $ *i* $ *n* ≠ *0*)
∈ *range* (*res* (*Hermite-of-row-i ass res A i* $ *i* $ (*LEAST n. A* $ *i* $ *n* ≠ *0*)))
⟨*proof*⟩

lemma *Hermite-of-upt-row-i-in-range-res*:
fixes *k ass res*
assumes *not-zero-i-A*: ¬ *is-zero-row i A*
and *e*: *echelon-form A*
and *a*: *ass-function ass*

and *r*: *res-function res*
and *k*: *to-nat i<k*
and *k2*: *k≤nrows A*
and *j*: *j<i*
shows *Hermite-of-upt-row-i A k ass res $ j $ (LEAST n. A $ i $ n ≠ 0)*
∈ range (res (Hermite-of-upt-row-i A k ass res $ i $ (LEAST n. A $ i $ n ≠ 0)))
⟨*proof*⟩

lemma *in-residues-Hermite-of*:
fixes *A::′a::{bezout-ring-div,normalization-semidom,unique-euclidean-ring}^′cols::{mod-type}^′rows::{mod-ty*
assumes *a*: *ass-function ass*
and *r*: *res-function res*
and *b*: *is-bezout-ext bezout*
and *i*: *¬ is-zero-row i (Hermite-of A ass res bezout)*
and *ji*: *j < i*
shows *Hermite-of A ass res bezout $ j $ (LEAST n. Hermite-of A ass res bezout $ i $ n ≠ 0)*
∈ range (res (Hermite-of A ass res bezout $ i $ (LEAST n. Hermite-of A ass res bezout $ i $ n ≠ 0)))
⟨*proof*⟩

lemma *Hermite-Hermite-of*:
assumes *a*: *ass-function ass*
and *r*: *res-function res*
and *b*: *is-bezout-ext bezout*
shows *Hermite (range ass) (λc. range (res c)) (Hermite-of A ass res bezout)*
⟨*proof*⟩

1.6.2 Proving that the Hermite Normal Form is computed by means of elementary operations

lemma *invertible-Hermite-reduce-above*:
assumes *n*: *n ≤ to-nat i*
shows *∃P. invertible P ∧ Hermite-reduce-above A n i j res = P ** A*
⟨*proof*⟩

lemma *invertible-Hermite-of-row-i*:
assumes *a*: *ass-function ass*
shows *∃P. invertible P ∧ Hermite-of-row-i ass res A i = P ** A*
⟨*proof*⟩

lemma *invertible-Hermite-of-upt-row-i*:
assumes *a*: *ass-function ass*
shows *∃P. invertible P ∧ Hermite-of-upt-row-i A k ass res = P ** A*

⟨*proof*⟩

lemma *invertible-Hermite-of*:
 fixes *A*::*'a*::{*bezout-ring-div*,*normalization-semidom*,*unique-euclidean-ring*}$^{\frown}$*'cols*::{*mod-type*}$^{\frown}$*'rows*::{*mod-t*
  assumes *a*: *ass-function ass*
  and *b*: *is-bezout-ext bezout*
  shows ∃*P*. *invertible P* ∧ *Hermite-of A ass res bezout* = *P* ∗∗ *A*
⟨*proof*⟩

1.6.3 The final theorem

lemma *Hermite*:
  assumes *a*: *ass-function ass*
  and *r*: *res-function res*
  and *b*: *is-bezout-ext bezout*
  shows ∃*P*. *invertible P* ∧ (*Hermite-of A ass res bezout*) = *P* ∗∗ *A* ∧
  Hermite (*range ass*) (λ*c*. *range* (*res c*)) (*Hermite-of A ass res bezout*)
  ⟨*proof*⟩

1.7 Proving the uniqueness of the Hermite Normal Form

lemma *diagonal-least-nonzero*:
 fixes *H* :: ((*'a* :: {*bezout-ring-div*, *normalization-euclidean-semiring*, *unique-euclidean-ring*}, *'b* :: *mod-type*) *vec*, *'b*) *vec*
  assumes *H*: *Hermite associates residues H*
  and *inv-H*: *invertible H* **and** *up-H*: *upper-triangular H*
  shows (*LEAST n*. *H* $ *i* $ *n* ≠ *0*) = *i*
⟨*proof*⟩

lemma *diagonal-in-associates*:
 fixes *H* :: ((*'a* :: {*bezout-ring-div*, *normalization-euclidean-semiring*, *unique-euclidean-ring*}, *'b* :: *mod-type*) *vec*, *'b*) *vec*
  assumes *H*: *Hermite associates residues H*
  and *inv-H*: *invertible H* **and** *up-H*: *upper-triangular H*
  shows *H* $ *i* $ *i* ∈ *associates*
⟨*proof*⟩

lemma *above-diagonal-in-residues*:
 fixes *H* :: ((*'a* :: {*bezout-ring-div*, *normalization-euclidean-semiring*, *unique-euclidean-ring*}, *'b* :: *mod-type*) *vec*, *'b*) *vec*
  assumes *H*: *Hermite associates residues H*
  and *inv-H*: *invertible H* **and** *up-H*: *upper-triangular H*
  and *j-i*: *j*<*i*
  shows *H* $ *j* $ (*LEAST n*. *H* $ *i* $ *n* ≠ *0*) ∈ *residues* (*H* $ *i* $ (*LEAST n*. *H* $ *i* $ *n* ≠ *0*))
⟨*proof*⟩

The uniqueness of the Hermite Normal Form is proven following the proof presented in the book Integral Matrices (1972) by Morris Newman.

lemma *Hermite-unique*:

fixes *K*::*'a*::{*bezout-ring-div*,*normalization-euclidean-semiring*,*unique-euclidean-ring*}^*'n*::*mod-type*^*'n*::*mod-*
assumes *A-PH*: *A* = *P* ∗∗ *H*
and *A-QK*: *A* = *Q* ∗∗ *K*
and *inv-A*: *invertible A*
and *inv-P*: *invertible P*
and *inv-Q*: *invertible Q*
and *H*: *Hermite associates residues H*
and *K*: *Hermite associates residues K*
shows *H* = *K*
⟨*proof*⟩

1.8 Examples of execution

value[*code*] *let A* = *list-of-list-to-matrix* ([[*37*,*8*,*6*],[*5*,*4*,−*8*],[*3*,*24*,−*7*]])::*int*^*3*^*3*
in matrix-to-list-of-list (*Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2*)

value[*code*] *let A* = *list-of-list-to-matrix* ([[[:*3*,*4*,*5*:],[:−*2*,*1*:]],[[:−*1*,*0*,*2*:],[:*0*,*1*,*4*,*1*:]]])::*real poly*^*2*^*2*
in matrix-to-list-of-list (*Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2*)

end

2 Hermite Normal Form refined to immutable arrays

theory *Hermite-IArrays*
imports
Hermite
Echelon-Form.*Echelon-Form-IArrays*
begin

2.1 Definition of the algorithm over immutable arrays

primrec *Hermite-reduce-above-iarrays* :: *'a*::*unique-euclidean-ring iarray iarray* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ (*'a*⇒*'a*⇒*'a*) ⇒ *'a iarray iarray*
where *Hermite-reduce-above-iarrays A 0 i j res* = *A*
| *Hermite-reduce-above-iarrays A* (*Suc n*) *i j res* = (*let i'*=*n*;
Aij = *A* !! *i* !! *j*;
Ai'j = *A* !! *i'* !! *j*
in
Hermite-reduce-above-iarrays (*row-add-iarray A i' i* (((*res Aij* (*Ai'j*)) − (*Ai'j*)) *div Aij*)) *n i j res*)

definition *Hermite-of-row-i-iarray ass res A i* = (
if is-zero-iarray (*A* !! *i*)
then A
else

let j = least-non-zero-position-of-vector (A !! i); Aij= (A !! i !! j);
A′ = mult-row-iarray A i ((ass Aij) div Aij)
in Hermite-reduce-above-iarrays A′ i i j res)

definition *Hermite-of-upt-row-i-iarrays A i ass res = foldl (Hermite-of-row-i-iarray ass res) A [0..<i]*

definition *Hermite-of-iarrays A ass res bezout =*
(let A′= echelon-form-of-iarrays A bezout
in Hermite-of-upt-row-i-iarrays A′ (nrows-iarray A) ass res)

2.2 Proving the equivalence between definitions of both representations

lemma *matrix-to-iarray-Hermite-reduce-above*:
fixes *A*::*′a*::{*unique-euclidean-ring*}^*′cols*::{*mod-type*}^*′rows*::{*mod-type*}
assumes *n<nrows A*
shows *matrix-to-iarray (Hermite-reduce-above A n i j res)*
= Hermite-reduce-above-iarrays (matrix-to-iarray A) n (to-nat i) (to-nat j) res
⟨*proof*⟩

lemma *matrix-to-iarray-Hermite-of-row-i*[*code-unfold*]:
fixes *A*::*′a*::{*unique-euclidean-ring*}^*′cols*::{*mod-type*}^*′rows*::{*mod-type*}
shows *matrix-to-iarray (Hermite-of-row-i ass res A i)*
= Hermite-of-row-i-iarray ass res (matrix-to-iarray A) (to-nat i)
⟨*proof*⟩

lemma *matrix-to-iarray-Hermite-of-upt-row-i*:
fixes *A*::*′a*::{*unique-euclidean-ring*}^*′cols*::{*mod-type*}^*′rows*::{*mod-type*}
assumes *i*: *i≤nrows A*
shows *matrix-to-iarray (Hermite-of-upt-row-i A i ass res)*
= Hermite-of-upt-row-i-iarrays (matrix-to-iarray A) i ass res
⟨*proof*⟩

lemma *matrix-to-iarray-Hermite-of*[*code-unfold*]:
shows *matrix-to-iarray (Hermite-of A ass res bezout)*
= Hermite-of-iarrays (matrix-to-iarray A) ass res bezout
⟨*proof*⟩

2.3 Examples of execution using immutable arrays

value[*code*] *let A = list-of-list-to-matrix ([[37,8,6],[5,4,−8],[3,24,−7]])::int^3^3*
in matrix-to-iarray (Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2)

value[*code*] *let A = IArray[IArray[37,8,6::int],IArray[5,4,−8],IArray[3,24,−7]]*
in (Hermite-of-iarrays A ass-function-euclidean res-function-euclidean euclid-ext2)

```
value[code] let A = list-of-list-to-matrix ([[[:3,4,5:],[:−2,1:]],[[:−1,0,2:],[:0,1,4,1:]]])::real
poly^2^2
  in matrix-to-iarray (Hermite-of A ass-function-euclidean res-function-euclidean
euclid-ext2)

end
```