Two Theorems on Hermitian Matrices

Sage Binder and Zilin Jiang

March 17, 2025

Abstract

We formalize two results on Hermitian matrices. First, Sylvester's criterion: a Hermitian matrix is positive definite if and only if all its leading principal submatrices have positive determinant. Second, Cauchy's eigenvalue interlacing theorem: given a principal submatrix B of a Hermitian matrix A, the eigenvalues of B interlace those of A.

Our approach to Sylvester's criterion is fairly standard, and required us to formalize Schur's block matrix determinant formula, which gives a formula for the determinant of a block matrix (A, B, C, D) when A is invertible.

Our approach to Cauchy's eigenvalue interlacing theorem follows a proof given in a set of lecture notes by Dr. David Bindel [1]. This approach involved formalizing the Courant-Fischer minimax theorem (a theorem about the Rayleigh quotient, which we define in this entry). In our statement of the Courant-Fischer minimax theorem, we refer to the infimum and supremum instead of the minimum and maximum, as this simplifies the proof and is sufficient to prove Cauchy's eigenvalue interlacing theorem.

Contents

theory *Misc-Matrix-Results*
imports *Commuting-Hermitian.Commuting-Hermitian*
BenOr-Kozen-Reif.More-Matrix
Jordan-Normal-Form.Spectral-Radius
Jordan-Normal-Form.DL-Rank-Submatrix
Jordan-Normal-Form.Jordan-Normal-Form-Uniqueness
Jordan-Normal-Form.VS-Connect
QHLProver.Complex-Matrix
Fishers-Inequality.Matrix-Vector-Extras
Complex-Bounded-Operators.Extra-Jordan-Normal-Form
Hermite-Lindemann.Misc-HLW
begin

hide-type (**open**) *Matrix-Legacy.mat*
hide-const (**open**) *Matrix-Legacy.mat*
hide-fact (**open**) *Finite-Cartesian-Product.mat-def*
hide-const (**open**) *Finite-Cartesian-Product.mat*
hide-fact (**open**) *Matrix-Legacy.mat-def*
hide-const (**open**) *Finite-Cartesian-Product.row*
hide-fact (**open**) *Finite-Cartesian-Product.row-def*
hide-const (**open**) *Matrix-Legacy.row*
hide-fact (**open**) *Matrix-Legacy.row-def*

hide-const (**open**) *Matrix-Legacy.col*
hide-fact (**open**) *Matrix-Legacy.col-def*
hide-const (**open**) *Determinants.det*
hide-fact (**open**) *Determinants.det-def*
hide-type (**open**) *Finite-Cartesian-Product.vec*
hide-const (**open**) *Finite-Cartesian-Product.vec*
hide-fact (**open**) *Finite-Cartesian-Product.vec-def*
hide-type (**open**) *Matrix-Legacy.vec*
hide-const (**open**) *Matrix-Legacy.vec*
hide-fact (**open**) *Matrix-Legacy.vec-def*
hide-const (**open**) *Coset.order*
hide-fact (**open**) *Coset.order-def*
hide-const (**open**) *Linear-Algebra.adjoint*
hide-fact (**open**) *Linear-Algebra.adjoint-def*
hide-const (**open**) *Finite-Cartesian-Product.transpose*
hide-fact (**open**) *Finite-Cartesian-Product.transpose-def*
unbundle *no inner-syntax*
unbundle *no vec-syntax*
hide-const (**open**) *Missing-List.span*
hide-const (**open**)
dependent
independent
real-vector.representation
real-vector.subspace
span
real-vector.extend-basis
real-vector.dim
hide-const (**open**) *orthogonal*
no-notation *fps-nth* (**infixl** $ *75*)

1 Determinant, Invertible, and Eigenvalue Lemmas

definition *eigvals-of* [*simp*]:
eigvals-of M es $\longleftrightarrow$ *char-poly M* = ($\prod$ *a*←*es*. [:− *a*, *1*:]) ∧ *length es* = *dim-row M*

lemma *det-is-prod-of-eigenvalues*:
fixes *A* :: *complex mat*
assumes *square-mat A*
shows *det A* = ($\prod$ *e* ← (*eigvals A*). *e*)
proof−
define *es* **where** *es* ≡ *eigvals A*
define *n* **where** *n* ≡ *dim-row A*
have *1*: *A* ∈ *carrier-mat n n* **using** *assms n-def* **by** *fastforce*
have *2*: *char-poly A* = ($\prod$ *e* ← *es*. [:− *e*, *1*:])
unfolding *es-def eigvals-def*
by (*metis* (*mono-tags*, *lifting*) *1 eigvals-poly-length someI-ex*)

obtain Q Q' B **where** $*$: *similar-mat-wit A B Q Q'* $\wedge$ *upper-triangular B* $\wedge$ *diag-mat B* $=$ *es*
using *schur-decomposition*[*OF 1 2*] **by** (*metis surj-pair*)

then have *det A* $=$ *det* (*Q* $*$ *B* $*$ *Q'*) **unfolding** *similar-mat-wit-def* **by** *metis*
also have ... $=$ *det Q* $*$ *det B* $*$ *det Q'*
by (*smt* (*verit*, *ccfv-SIG*) $*$ *1 det-mult mult-carrier-mat similar-mat-witD2*(*5*) *similar-mat-witD2*(*6*) *similar-mat-witD2*(*7*))
also have ... $=$ *det Q* $*$ *det B* $*$ *1*/(*det Q*)
by (*smt* (*verit*, *ccfv-threshold*) $*$ *1 det-mult det-one div-by-0 helper mult-cancel-left1 n-def nonzero-mult-div-cancel-left similar-mat-witD*(*6*) *similar-mat-witD*(*7*) *similar-mat-witD2*(*1*))
also have ... $=$ *det Q* $*$ ($\prod$ *e* $\leftarrow$ *diag-mat B*. *e*) $*$ *1*/(*det Q*)
by (*metis* $*$ *det-upper-triangular list.map-ident similar-mat-witD*(*5*))
also have ... $=$ ($\prod$ *e* $\leftarrow$ (*eigvals A*). *e*)
by (*metis* (*no-types*, *lifting*) $*$ *es-def 1 Groups.mult-ac*(*2*) *class-field.zero-not-one det-mult det-one mult-cancel-left2 nonzero-mult-div-cancel-left similar-mat-witD*(*6*) *similar-mat-witD*(*7*) *similar-mat-witD2*(*2*))
finally show *?thesis* **.**
qed

lemma *eigvals-of-spectrum*:
(*A*::(*complex mat*)) $\in$ *carrier-mat n n* $\Longrightarrow$ *eigvals-of A* α $\Longrightarrow$ *spectrum A* $=$ *set* α
unfolding *eigvals-of*
using *eigenvalue-root-char-poly*[*of A n*]
by (*metis Spectral-Radius.spectrum-def equalityI linear-poly-root mem-Collect-eq root-poly-linear subsetI*)

lemma *trivial-kernel-imp-nonzero-eigenvalues*:
fixes *M* :: *'a*::{*idom*,*ring-1-no-zero-divisors*} *mat*
assumes *square-mat M*
assumes *mat-kernel M* $\subseteq$ {0_v (*dim-row M*)}
assumes *eigenvalue M e*
shows *e* $\neq$ *0*
by (*metis* (*no-types*, *lifting*) *assms carrier-matI carrier-vecD eigenvalue-def eigenvector-def empty-iff mat-kernelI singleton-iff smult-vec-zero square-mat.simps subset-singletonD*)

lemma *trivial-kernel-imp-invertible*:
fixes *M* :: *complex mat*
assumes *square-mat M*
assumes *mat-kernel M* $\subseteq$ {0_v (*dim-row M*)}
shows *invertible-mat M*
by (*metis assms*(*1*) *assms*(*2*) *carrier-matI det-0-iff-vec-prod-zero-field empty-iff invertible-det mat-kernelI singletonD square-mat.elims*(*2*) *subset-singletonD*)

lemma *trivial-kernel-imp-det-nz*:
fixes *M* :: *complex mat*

assumes *square-mat M*
assumes *mat-kernel M* $\subseteq \{0_v\ (dim\text{-}row\ M)\}$
shows *det M* $\neq$ *0*
using *trivial-kernel-imp-invertible*[*OF assms*(*1*) *assms*(*2*)]
using *invertible-det assms*(*1*) *square-mat.simps*
by *blast*

lemma *similar-mats-eigvals*:
assumes *A* $\in$ *carrier-mat n n*
assumes *B* $\in$ *carrier-mat n n*
assumes *similar-mat A B*
assumes *eigvals-of A es*
shows *eigvals-of B es*
using *assms* **unfolding** *eigvals-of*
by (*metis* (*no-types*) *char-poly-similar assms*(*1*−*3*) *carrier-matD*(*1*))

lemma *scale-eigvals*:
fixes *A* :: *complex mat*
assumes *A* $\in$ *carrier-mat n n*
assumes *B* = *c* $\cdot_m$ *A*
assumes *eigvals-of A es*
shows *eigvals-of B* (*map* ($\lambda x.\ c * x$) *es*)
proof−
obtain *A′ P Q* **where** *A-decomp*: *schur-decomposition A es* = (*A′*, *P*, *Q*)
∧ *similar-mat-wit A A′ P Q*
∧ *upper-triangular A′*
∧ *diag-mat A′* = *es*
using *assms*(*3*) **unfolding** *eigvals-of* **by** (*metis schur-decomposition assms*(*1*) *surj-pair*)
define *B′* **where** *B′* ≡ *c* $\cdot_m$ *A′*

have *B′-dim*: *B′* $\in$ *carrier-mat n n*
by (*metis A-decomp B′-def assms*(*1*) *similar-mat-witD2*(*5*) *smult-carrier-mat*)
have *B-decomp*: *similar-mat-wit B B′ P Q*
∧ *upper-triangular B′*
proof−
have *upper-triangular B′*
proof−
{ fix *i j* **assume** ∗: *j* < *i i* < *dim-row B′*
hence *B′*$$(*i*,*j*) = *c* ∗ *A′*$$(*i*,*j*)
by (*metis B′-def B′-dim carrier-matD*(*1*) *carrier-matD*(*2*) *dual-order.strict-trans1 index-smult-mat*(*1*) *index-smult-mat*(*2*) *index-smult-mat*(*3*) *le-simps*(*1*))
also have ... = *0* **using** *A-decomp* ∗ **unfolding** *upper-triangular-def* **by** (*simp add*: *B′-def*)
finally have *B′*$$(*i*,*j*) = *0* **.**
}
thus *?thesis* **by** *blast*
qed
moreover have *similar-mat-wit B B′ P Q*

proof −
have $B = c \cdot_m (P * A' * Q)$ **using** *A-decomp assms*(2) *similar-mat-witD2*(3)
by *blast*
also have ... $= P * (c \cdot_m A') * Q$
by (*metis A-decomp similar-mat-wit-def similar-mat-wit-smult*)
also have ... $= P * B' * Q$ **using** B'*-def* **by** *argo*
finally have $B = P * B' * Q$ **.**
thus *?thesis* **by** (*smt* (*verit*, *best*) *A-decomp B'-def assms*(2) *similar-mat-wit-smult*)
qed
ultimately show *?thesis* **by** *blast*
qed

hence *char-poly* $B' = (\prod a \leftarrow$*diag-mat* B'. [:− *a*, *1*:])
using *char-poly-upper-triangular B-decomp B'-dim* **by** *blast*
moreover have *length* (*diag-mat* B') = *dim-row* B'
by (*simp add*: *diag-mat-length*)
ultimately have *eigvals-of* B' (*diag-mat* B') **using** *eigvals-of* **by** *blast*
moreover have *diag-mat* B' = *map* ($\lambda x.\ c * x$) *es*
using *A-decomp* B'*-def*
by (*metis assms*(1) *diag-mat-map similar-mat-witD2*(5) *smult-mat-def*)
ultimately show *?thesis*
using *similar-mats-eigvals B-decomp assms*(2) *assms*(3) *char-poly-similar similar-mat-def*
by *fastforce*
qed

lemma *neg-mat-eigvals*:
fixes *A* :: *complex mat*
assumes $A \in$ *carrier-mat n n*
assumes *eigvals-of A es*
shows *eigvals-of* (−*A*) (*rev* (*map* ($\lambda x.\ -x$) *es*))
proof −
have *eigvals-of A* (*rev es*)
using *assms*(2)
unfolding *eigvals-of*
by (*metis length-rev prod-list.rev rev-map*)
thus *?thesis*
using *scale-eigvals*[*of A n* −*A* −*1 rev es*]
by (*metis assms*(1) *ext mult-minus1 rev-map uminus-mat*)
qed

2 Quadratic Form

definition *quadratic-form* :: *'a mat* ⇒ *'a vec* ⇒ *'a*::{*conjugatable-ring*} **where**
quadratic-form M x ≡ *inner-prod x* (*M* $*_v$ *x*)

abbreviation *QF* ≡ *quadratic-form*

lemma *hermitian-quadratic-form-real*:

fixes $A :: complex\ mat$
fixes $v :: complex\ vec$
assumes $A \in carrier\text{-}mat\ n\ n$
assumes $v \in carrier\text{-}vec\ n$
assumes $hermitian\ A$
shows $QF\ A\ v \in Reals$
proof−
have $conjugate\ (QF\ A\ v) = inner\text{-}prod\ (A *_v v)\ v$
by $(metis\ assms(1)\ assms(2)\ inner\text{-}prod\text{-}swap\ mult\text{-}mat\text{-}vec\text{-}carrier\ quadratic\text{-}form\text{-}def)$
also have $... = inner\text{-}prod\ v\ ((adjoint\ A) *_v v)$
by $(metis\ adjoint\text{-}def\text{-}alter\ assms(1)\ assms(2)\ assms(3)\ hermitian\text{-}def)$
also have $... = inner\text{-}prod\ v\ (A *_v v)$
using $assms(3)$ **by** $(simp\ add{:}\ hermitian\text{-}def)$
finally have $conjugate\ (QF\ A\ v) = QF\ A\ v$
by $(simp\ add{:}\ quadratic\text{-}form\text{-}def)$
thus $?thesis$ **by** $(simp\ add{:}\ Reals\text{-}cnj\text{-}iff)$
qed

declare
$quadratic\text{-}form\text{-}def[simp]$

3 Leading Principal Submatrix Lemmas

definition $leading\text{-}principal\text{-}submatrix :: 'a\ mat \Rightarrow nat \Rightarrow 'a\ mat$ **where**
$[simp]{:}\ leading\text{-}principal\text{-}submatrix\ A\ k = submatrix\ A\ \{..<k\}\ \{..<k\}$

abbreviation $lps \equiv leading\text{-}principal\text{-}submatrix$

lemma $leading\text{-}principal\text{-}submatrix\text{-}carrier$:
$m \geq n \Longrightarrow A \in carrier\text{-}mat\ m\ m \Longrightarrow lps\ A\ n \in carrier\text{-}mat\ n\ n$
proof−
assume $*$: $m \geq n\ A \in carrier\text{-}mat\ m\ m$
let $?B = lps\ A\ n$
have $(card\ \{i.\ i < dim\text{-}row\ A \wedge i \in \{..<n\}\}) = n$
by $(metis *(1) *(2)\ Collect\text{-}conj\text{-}eq\ Collect\text{-}mem\text{-}eq\ card\text{-}lessThan\ carrier\text{-}matD(1)$
$inf.absorb\text{-}iff2\ lessThan\text{-}def\ lessThan\text{-}subset\text{-}iff)$
hence $dim\text{-}col\ ?B = n \wedge dim\text{-}row\ ?B = n$
unfolding $leading\text{-}principal\text{-}submatrix\text{-}def\ submatrix\text{-}def$
using $*(2)$ **by** $auto$
thus $?thesis$ **by** $blast$
qed

lemma $pick\text{-}n$:
assumes $i \leq n$
shows $pick\ \{..n\}\ i = i$
using $assms$
proof$(induct\ i)$
case 0
then show $?case$ **by** $force$

next
case (*Suc i*)
hence *Suc i* ∈ {..*n*} **by** *blast*
moreover from *Suc* **have** *Suc i* > *pick* {..*n*} *i* **by** *simp*
moreover from *Suc* **have** ∀ *i'* < *Suc i*. ¬ (*i'*∈{..*n*} ∧ *i'* > *pick* {..*n*} *i*)
using *Suc-leD not-less-eq* **by** *presburger*
ultimately have *Suc i* = (*LEAST a*. *a*∈{..*n*} ∧ *a* > *pick* {..*n*} *i*)
by (*metis* (*no-types*, *lifting*) *LeastI linorder-not-less not-less-Least order.strict-iff-order*)
thus *?case* **by** (*metis DL-Missing-Sublist.pick.simps*(*2*))
qed

lemma *pick-n-le*:
assumes *i* < *n*
shows *pick* {..<*n*} *i* = *i*
by (*metis assms lessThan-Suc-atMost less-Suc-eq-le not0-implies-Suc not-less-zero pick-n*)

lemma *leading-principal-submatrix-index*:
assumes *A* ∈ *carrier-mat n n*
assumes *k* ≤ *n*
assumes *i* < *k*
assumes *j* < *k*
shows (*lps A k*)$$(*i*,*j*) = *A*$$(*i*,*j*)
proof−
have ⋀*i*. *i* < *k* ⟹ *pick* {..<*k*} *i* = *i* **by** (*simp add*: *pick-n-le*)
moreover have *card* {*i*. *i* < *dim-row A* ∧ *i* ∈ {..<*k*}} = *k*
by (*metis Collect-conj-eq Collect-mem-eq assms*(*1*) *assms*(*2*) *card-lessThan carrier-matD*(*1*) *inf.absorb-iff2 lessThan-def lessThan-subset-iff*)
moreover then have *card* {*j*. *j* < *dim-col A* ∧ *j* ∈ {..<*k*}} = *k* **using** *assms*(*1*) **by** *force*
moreover have (*mat k k* (λ(*i*, *j*). *A*$$(*i*,*j*)))$$(*i*,*j*) = *A*$$(*i*,*j*) **using** *assms*(*3*) *assms*(*4*) **by** *auto*
ultimately show *?thesis* **by** (*simp add*: *assms*(*3*) *assms*(*4*) *submatrix-def*)
qed

lemma *nested-leading-principle-submatrices*:
assumes *A* ∈ *carrier-mat n n*
assumes $k_1 \leq k_2$
assumes $k_2 \leq n$
shows *lps A* k_1 = *lps* (*lps A* k_2) k_1 (**is** *?lhs* = *?rhs*)
proof−
have ⋀*i j*. *i* < k_1 ⟹ *j* < k_1 ⟹ *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
by (*smt* (*verit*, *best*) *assms dual-order.trans leading-principal-submatrix-carrier leading-principal-submatrix-index order.strict-trans2*)
moreover have *?lhs* ∈ *carrier-mat* k_1 k_1
by (*meson assms leading-principal-submatrix-carrier order-trans*)
moreover have *?rhs* ∈ *carrier-mat* k_1 k_1
by (*meson assms leading-principal-submatrix-carrier*)
ultimately show *?thesis* **by** *auto*

qed

4 Submatrix Lemmas

lemma *submatrix-as-matrix-prod*:
fixes $A :: complex\ mat$
assumes $A \in carrier\text{-}mat\ n\ n$
assumes $I \subseteq \{..<n\}$
assumes $I \neq \{\}$
defines $m \equiv card\ I$
defines $B \equiv submatrix\ A\ I\ I$
defines $u\text{-}cols\text{-}inds \equiv map\ (pick\ I)\ [0..<m]$
defines $u\text{-}cols \equiv map\ ((!)\ (unit\text{-}vecs\ n))\ u\text{-}cols\text{-}inds$
defines $(Inm::complex\ mat) \equiv mat\text{-}of\text{-}cols\ n\ u\text{-}cols$
defines $(Inm'::complex\ mat) \equiv Inm^H$
shows $B = Inm' * A * Inm$
$Inm' * Inm = 1_m\ m$
$Inm \in carrier\text{-}mat\ n\ m$
$inj\text{-}on\ ((*_v)\ Inm)\ (carrier\text{-}vec\ m)$
proof−
have *u-cols-length*: $length\ u\text{-}cols = m$ **by** (*simp add*: *assms*(*7*) *u-cols-inds-def*)
thus *Inm-carr*: $Inm \in carrier\text{-}mat\ n\ m$ **unfolding** *Inm-def mat-of-cols-def* **by** *fastforce*
have *Inm'-carr*: $Inm' \in carrier\text{-}mat\ m\ n$ **using** *Inm-carr* **by** (*simp add*: *assms*(*9*))

let *?rhs* $= Inm' * A * Inm$

have *dim-A*: $dim\text{-}row\ A = n \wedge dim\text{-}col\ A = n$ **using** *assms*(*1*) **by** *simp*
hence *I*: $\{i.\ i < dim\text{-}row\ A \wedge i \in I\} = I \wedge \{i.\ i < dim\text{-}col\ A \wedge i \in I\} = I$ **using** *assms*(*2*) **by** *auto*
hence *B-carr*: $B \in carrier\text{-}mat\ m\ m$
using *dim-submatrix*[*of A I I*] **unfolding** *m-def B-def* **by** *auto*

have *m-leq-n*: $m \leq n$
using *assms*(*2*,*4*) *atLeast0LessThan subset-eq-atLeast0-lessThan-card* **by** *presburger*

have $\bigwedge i.\ i < m \Longrightarrow u\text{-}cols!i = unit\text{-}vec\ n\ (pick\ I\ i)$
unfolding *u-cols-def u-cols-inds-def unit-vecs-def*
by (*smt* (*verit*, *ccfv-SIG*) *arithmetic-simps*(*49*) *assms*(*2*,*4*) *atLeast-upt diff-zero length-map length-upt map-eq-conv map-nth nth-map nth-map-upt pick-in-set-le subsetD*)
hence *col-Inm-i*: $\bigwedge i.\ i < m \Longrightarrow col\ Inm\ i = unit\text{-}vec\ n\ (pick\ I\ i)$
by (*simp add*: *assms*(*8*) *u-cols-length*)

have *Inm'-is-trans*: $Inm' = Inm^T$
proof−
have $\bigwedge i\ j.\ i < n \wedge j < m \Longrightarrow Inm\$\$(i,j) \in \mathbb{R}$
proof−

fix i j **assume** $*$: $i < n \wedge j < m$
have $Inm\$\$(i,j) = (col\ Inm\ j)\$i$ **using** $*$ *Inm-carr* **by** *auto*
also have ... $= (unit\text{-}vec\ n\ (pick\ I\ j))\$i$ **using** *col-Inm-i* $*$ **by** *presburger*
also have ... $= 0 \vee$ ... $= 1$ **unfolding** *unit-vec-def* **using** $*$ **by** *auto*
finally show $Inm\$\$(i,j) \in \mathbb{R}$ **by** *fastforce*
qed
hence $\bigwedge i\ j.\ i < n \wedge j < m \Longrightarrow Inm^T\$\$(j,i) \in \mathbb{R}$ **using** *Inm-carr* **by** *auto*
hence $\bigwedge i\ j.\ i < n \wedge j < m \Longrightarrow conjugate\ (Inm^T\$\$(j,i)) = Inm^T\$\$(j,i)$
by (*simp add*: *Reals-cnj-iff*)
moreover have $\bigwedge i\ j.\ i < n \wedge j < m \Longrightarrow (conjugate\ Inm^T)\$\$(j,i) = conjugate\ (Inm^T\$\$(j,i))$
using *Inm-carr* **by** *auto*
ultimately show *?thesis* **using** *Inm-carr assms*(*9*) **by** *auto*
qed

have $Inm' * A * Inm \in carrier\text{-}mat\ m\ m$ **using** *Inm-carr Inm'-carr* **by** *fastforce*
moreover have $\bigwedge i\ j.\ i < m \wedge j < m \Longrightarrow B\$\$(i,j) = ?rhs\$\$(i,j)$
proof−
fix i j **assume** $*$: $i < m \wedge j < m$
hence *1*: $B\$\$(i,j) = A\$\$(pick\ I\ i,\ pick\ I\ j)$ **unfolding** *B-def submatrix-def* **using** *I m-def* **by** *auto*

have $col\ (A * Inm)\ j = A *_v (col\ Inm\ j)$ **using** $*$ *Inm-carr* **by** *auto*
also have ... $= A *_v unit\text{-}vec\ n\ (pick\ I\ j)$ **using** *col-Inm-i* $*$ **by** *presburger*
also have ... $= col\ A\ (pick\ I\ j)$
by (*metis* $*$ *assms*(*1*,*2*,*4*) *basic-trans-rules*(*31*) *lessThan-iff mat-unit-vec-col pick-in-set-le*)
finally have $col\ (A * Inm)\ j = col\ A\ (pick\ I\ j)$.
moreover have $(Inm' * (A * Inm))\$\$(i,j) = (row\ Inm'\ i) \cdot (col\ (A * Inm)\ j)$
using $*$ *Inm'-carr Inm-carr* **by** *auto*
ultimately have $?rhs\$\$(i,j) = (row\ Inm'\ i) \cdot col\ A\ (pick\ I\ j)$
using *Inm'-carr Inm-carr assms*(*1*) **by** *fastforce*
also have ... $= (col\ Inm\ i) \cdot col\ A\ (pick\ I\ j)$
using *Inm'-is-trans* $*$ *Inm-carr* **by** *auto*
also have ... $= (unit\text{-}vec\ n\ (pick\ I\ i)) \cdot col\ A\ (pick\ I\ j)$
using $*$ *col-Inm-i* **by** *presburger*
also have ... $= (col\ A\ (pick\ I\ j))\$(pick\ I\ i)$
by (*metis* $*$ *dim-A assms*(*2*) *basic-trans-rules*(*31*) *col-dim lessThan-iff m-def pick-in-set-le scalar-prod-left-unit*)
also have ... $= A\$\$(pick\ I\ i,\ pick\ I\ j)$ **using** $*$ *I m-def pick-le* **by** *auto*
finally show $B\$\$(i,j) = ?rhs\$\$(i,j)$ **using** *1* **by** *presburger*
qed
ultimately show $B = Inm' * A * Inm$ **by** (*metis B-carr carrier-matD*(*1*) *carrier-matD*(*2*) *eq-matI*)

show $Inm' * Inm = 1_m\ m$
proof
show $dim\text{-}row\ (Inm' * Inm) = dim\text{-}row\ (1_m\ m)$ **using** *Inm'-carr* **by** *auto*
show $dim\text{-}col\ (Inm' * Inm) = dim\text{-}col\ (1_m\ m)$ **using** *Inm-carr* **by** *auto*

fix *i j*
assume $i < dim\text{-}row\ (1_m\ m)$
hence *i*: $i < m$ **by** *simp*
assume $j < dim\text{-}col\ (1_m\ m)$
hence *j*: $j < m$ **by** *simp*

have $(Inm' * Inm)\$\$(i,j) = (row\ Inm'\ i) \cdot (col\ Inm\ j)$
by (*metis Inm'-carr Inm-carr j carrier-matD(1) carrier-matD(2) i index-mult-mat(1)*)
also have ... $= (col\ Inm\ i) \cdot (col\ Inm\ j)$ **using** *Inm'-is-trans Inm-carr i* **by** *auto*
also have ... $= unit\text{-}vec\ n\ (pick\ I\ i) \cdot unit\text{-}vec\ n\ (pick\ I\ j)$ **using** *col-Inm-i i j* **by** *presburger*
also have ... $= (if\ i = j\ then\ 1\ else\ 0)$
by (*metis* (*no-types*) *I assms(1) assms(4) card-pick-le carrier-matD(2) i index-unit-vec(1) j pick-le scalar-prod-right-unit*)
also have ... $= 1_m\ m\ \$\$\ (i,j)$ **by** (*simp add*: *i j*)
finally show $(Inm' * Inm)\$\$(i,j) = 1_m\ m\ \$\$\ (i,j)$.
qed
thus *inj-on* $((*_v)\ Inm)$ (*carrier-vec m*)
by (*smt* (*verit*, *best*) *Inm'-carr Inm-carr assoc-mult-mat-vec inj-onCI one-mult-mat-vec*)
qed

lemma *submatrix-as-matrix-prod-obt*:
fixes *A* :: *complex mat*
assumes $A \in carrier\text{-}mat\ n\ n$
assumes $I \subseteq \{..<n\}$
assumes $I \neq \{\}$
defines $m \equiv card\ I$
defines $B \equiv submatrix\ A\ I\ I$
obtains *Inm* **where** $B = Inm^H * A * Inm$
$Inm^H * Inm = 1_m\ m$
$Inm \in carrier\text{-}mat\ n\ m$
$inj\text{-}on\ ((*_v)\ Inm)\ (carrier\text{-}vec\ m)$
using *submatrix-as-matrix-prod assms* **by** *presburger*

5 Hermitian and Conjugate Lemmas

lemma *hermitian-is-square*: $hermitian\ A \Longrightarrow square\text{-}mat\ A$
by (*metis adjoint-dim-col hermitian-def square-mat.simps*)

lemma *hermitian-eigenvalues-real*:
assumes $(A::(complex\ mat)) \in carrier\text{-}mat\ n\ n$
assumes *hermitian A*
assumes *eigenvalue A e*
shows $e \in Reals$
using *cpx-sq-mat.hermitian-spectrum-real*[*OF - assms(1,2), of n n e*]
by (*metis Projective-Measurements.spectrum-def assms(1,3) cpx-sq-mat-axioms.intro cpx-sq-mat-def eigenvalue-imp-nonzero-dim eigenvalue-root-char-poly eigvals-poly-length fixed-carrier-mat.intro root-poly-linear*)

lemma *hermitian-spectrum-real*:
(*A*::(*complex mat*)) ∈ *carrier-mat n n* ⟹ *hermitian A* ⟹ *spectrum A* ⊆ *Reals*
by (*simp add*: *Spectral-Radius.spectrum-def hermitian-eigenvalues-real unfold-simps*(*2*))

lemma *leading-principal-submatrix-hermitian*:
assumes *A* ∈ *carrier-mat n n*
assumes *hermitian A*
assumes *k* ≤ *n*
shows *hermitian* (*lps A k*) (**is** *hermitian ?A'*)
proof−
have ⋀*i j*. *i* < *dim-row ?A'* ⟹ *j* < *dim-col ?A'* ⟹ *?A'*$$(*i*,*j*) = *conjugate*
(*?A'*$$(*j*,*i*))
by (*metis* (*no-types*, *lifting*) *adjoint-eval assms carrier-matD*(*1*) *carrier-matD*(*2*)
dual-order.strict-trans1 hermitian-def leading-principal-submatrix-carrier leading-principal-submatrix-index)
thus *?thesis*
by (*metis* (*no-types*, *lifting*) *adjoint-dim-col adjoint-dim-row adjoint-eval assms*(*1*)
assms(*3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *eq-matI hermitian-def leading-principal-submatrix-carrier*)
qed

lemma *conjugate-mat-dist*:
fixes *A B* :: *'a*::*conjugatable-ring mat*
assumes *A* ∈ *carrier-mat m n*
assumes *B* ∈ *carrier-mat n p*
shows (*conjugate A*) ∗ (*conjugate B*) = *conjugate* (*A* ∗ *B*)
by (*smt* (*z3*) *assms*(*1*) *assms*(*2*) *carrier-matD*(*1*) *carrier-matD*(*2*) *col-conjugate*
conjugate-scalar-prod dim-col dim-col-conjugate dim-row-conjugate eq-matI index-mult-mat(*1*)
index-mult-mat(*2*) *index-mult-mat*(*3*) *index-row*(*2*) *mat-index-conjugate row-conjugate*)

lemma *conjugate-mat-inv*:
fixes *A* :: *'a*::{*conjugatable-ring*,*semiring-1*} *mat*
assumes *A* ∈ *carrier-mat n n*
assumes *A'* ∈ *carrier-mat n n*
assumes *inverts-mat A A'*
shows *inverts-mat* (*conjugate A*) (*conjugate A'*)
proof−
have (*conjugate A*) ∗ (*conjugate A'*) = *conjugate* (*A* ∗ *A'*)
using *conjugate-mat-dist assms*(*1*) *assms*(*2*) **by** *blast*
also have ... = *conjugate* (*1*$_m$ *n*)
by (*metis assms*(*1*) *assms*(*3*) *carrier-matD*(*1*) *inverts-mat-def*)
also have ... = *1*$_m$ *n*
by (*metis* (*no-types*, *lifting*) *carrier-matI conjugate-id conjugate-mat-dist dim-col-conjugate*
dim-row-conjugate index-one-mat(*2*) *index-one-mat*(*3*) *left-mult-one-mat' right-mult-one-mat'*)
finally show *?thesis*
by (*metis index-mult-mat*(*2*) *index-one-mat*(*2*) *inverts-mat-def*)
qed

lemma *hermitian-mat-inv*:
assumes *A* ∈ *carrier-mat n n*

assumes $A' \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *inverts-mat A A'*
shows *hermitian A'*
proof−
have *inverts-mat* A^T A'^T
by (*metis assms*(1) *assms*(2) *assms*(4) *carrier-matD*(1) *carrier-matD*(2) *index-transpose-mat*(2) *inverts-mat-def inverts-mat-symm transpose-mult transpose-one*)
hence *inverts-mat* (*conjugate* A^T) (*conjugate* A'^T)
by (*metis assms*(1) *assms*(2) *conjugate-mat-inv transpose-carrier-mat*)
thus *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *adjoint-dim adjoint-dim-row adjoint-mult assms*(1) *assms*(2) *assms*(3) *assms*(4) *assoc-mult-mat carrier-matD*(2) *hermitian-def hermitian-one inverts-mat-def inverts-mat-symm right-mult-one-mat'*)
qed

lemma *hermitian-ij-ji*:
hermitian A
$\longleftrightarrow$ *square-mat A* $\wedge$ ($\forall$ *i j. i* $<$ *dim-row A* $\wedge$ *j* $<$ *dim-row A* $\longrightarrow$ *A*$$(*i*,*j*) = *conjugate* (*A*$$(*j*,*i*)))
by (*metis* (*no-types*, *lifting*) *adjoint-dim-col adjoint-dim-row adjoint-eval hermitian-def mat-eq-iff square-mat.simps*)

lemma *negative-hermitian*:
assumes *A* $\in$ *carrier-mat n n*
assumes *hermitian A*
shows *hermitian* (−*A*)
by (*metis assms hermitian-minus left-add-zero-mat minus-add-uminus-mat uminus-carrier-iff-mat zero-carrier-mat zero-hermitian*)

lemma *principal-submatrix-hermitian*:
assumes *A* $\in$ *carrier-mat n n*
assumes *hermitian A*
assumes *I* $\subseteq$ {..<*n*}
shows *hermitian* (*submatrix A I I*) (**is** *hermitian ?B*)
proof−
have *square-mat ?B*
by (*metis* (*full-types*) *assms*(1) *carrier-matD*(1) *carrier-matD*(2) *dim-submatrix*(1) *dim-submatrix*(2) *square-mat.elims*(1))
moreover {
fix *i j* **assume** ∗: *i* $<$ *dim-row ?B* $\wedge$ *j* $<$ *dim-row ?B*
then obtain *i' j'* **where** *?B*$$(*i*,*j*) = *A*$$(*i'*,*j'*) $\wedge$ *i'* = *pick I i* $\wedge$ *j'* = *pick I j*
unfolding *submatrix-def* **using** *assms*(1) *pick-le* **by** *auto*
moreover then have *?B*$$(*j*,*i*) = *A*$$(*j'*,*i'*)
unfolding *submatrix-def*
by (*metis* (*no-types*, *lifting*) *Collect-cong* ∗ *assms*(1) *carrier-matD*(1) *carrier-matD*(2) *case-prod-conv dim-submatrix*(1) *index-mat*(1))
ultimately have *?B*$$(*i*,*j*) = *conjugate* (*?B*$$(*j*,*i*))
by (*metis* ∗ *assms*(2) *dim-submatrix*(1) *hermitian-ij-ji pick-le*)

}
ultimately show *?thesis* **by** (*metis hermitian-ij-ji*)
qed

lemma *conjugate-dist-mult-mat*:
fixes $A :: 'a::$*conjugatable-ring mat*
assumes $A \in$ *carrier-mat m n* $B \in$ *carrier-mat n p*
shows *conjugate* $(A * B) =$ *conjugate A* $*$ *conjugate B*
(**is** *?lhs* $=$ *?rhs*)
proof−
have $\bigwedge i\ j.\ i < m \Longrightarrow j < p \Longrightarrow$ *?lhs*$$(i,j) =$ *?rhs*$$(i,j)$
by (*smt* (*verit*, *del-insts*) *assms carrier-matD*(*1*) *carrier-matD*(*2*) *col-conjugate conjugate-scalar-prod dim-col dim-col-conjugate dim-row-conjugate index-mult-mat*(*1*) *index-mult-mat*(*2*) *index-mult-mat*(*3*) *index-row*(*2*) *mat-index-conjugate row-conjugate*)

moreover have *?lhs* $\in$ *carrier-mat m p* **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *assms carrier-matD*(*2*) **by** *auto*
qed

lemma *conjugate-dist-add-mat*:
fixes $A :: 'a::$*conjugatable-ring mat*
assumes $A \in$ *carrier-mat m n* $B \in$ *carrier-mat m n*
shows *conjugate* $(A + B) =$ *conjugate A* $+$ *conjugate B*
(**is** *?lhs* $=$ *?rhs*)
proof−
have $\bigwedge i\ j.\ i < m \Longrightarrow j < n \Longrightarrow$ *?lhs*$$(i,j) =$ *?rhs*$$(i,j)$
using *assms assms conjugate-dist-add* **by** *fastforce*
moreover have *?lhs* $\in$ *carrier-mat m n* **using** *assms* **by** *auto*
ultimately show *?thesis* **using** *assms carrier-matD*(*2*) **by** *auto*
qed

lemma *mat-row-conj*:
assumes $A \in$ *carrier-mat m n*
assumes $i < m$
shows *conjugate* (*row A i*) $=$ *row* (*conjugate A*) *i*
using *assms*
unfolding *conjugate-mat-def*
by *auto*

lemma *conj-mat-vec-mult*:
fixes $A :: 'a::\{$*conjugate*,*conjugatable-ring*$\}$ *mat*
fixes $v :: 'a$ *vec*
assumes $A \in$ *carrier-mat n n*
assumes $v \in$ *carrier-vec n*
shows *conjugate* $(A *_v v) =$ (*conjugate A*) $*_v$ (*conjugate v*)
(**is** *?lhs* $=$ *?rhs*)
proof−
have $\bigwedge i.\ i < n \Longrightarrow$ *?lhs*$$i =$ *?rhs*$$i$
by (*metis assms carrier-matD*(*1*) *conjugate-sprod-vec dim-mult-mat-vec dim-row-conjugate*

index-mult-mat-vec mat-row-conj row-carrier-vec vec-index-conjugate)
moreover have *?lhs* ∈ *carrier-vec n* **using** *assms* **by** *force*
ultimately show *?thesis* **using** *assms*(*1*) **by** *auto*
qed

lemma *hermitian-row-col*:
assumes *A* ∈ *carrier-mat n n*
assumes *hermitian A*
assumes *i* < *n*
shows *row A i* = *conjugate* (*col A i*)
by (*metis adjoint-row assms carrier-matD*(*2*) *hermitian-def*)

lemma *hermitian-real-diag-decomp-eigvals*:
fixes *A* :: *complex mat*
assumes *A* ∈ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A es*
obtains *B U* **where**
real-diag-decomp A B U
diag-mat B = *es*
set es ⊆ *Reals*
B ∈ *carrier-mat n n*
U ∈ *carrier-mat n n*
proof−

have *es*: *char-poly A* = (∏ (*e* :: *complex*) ← *es*. [:− *e*, *1*:])
using *assms eigvals-poly-length* **by** *auto*
obtain *B U Q* **where** *us*: *unitary-schur-decomposition A es* = (*B*,*U*,*Q*)
by (*cases unitary-schur-decomposition A es*)
hence ∗: *similar-mat-wit A B U* (*adjoint U*) ∧ *diagonal-mat B*
∧ *diag-mat B* = *es* ∧ *unitary U* ∧ (∀ *i* < *n*. *B*$$(*i*, *i*) ∈ *Reals*)
using *hermitian-eigenvalue-real assms es* **by** *auto*
moreover then have *dim-row B* = *n* **using** *assms similar-mat-wit-dim-row*[*of A*] **by** *auto*
ultimately have *1*: *real-diag-decomp A B U* **using** *unitary-diagI*[*of A*]
unfolding *real-diag-decomp-def* **by** *simp*

from ∗ **have** *2*: *diag-mat B* = *es* **by** *blast*
from ∗ **have** *3*: *set es* ⊆ *Reals* **by** (*metis* ‹*dim-row B* = *n*› *diag-elems-real diag-elems-set-diag-mat*)
from ∗ **have** *4*: *B* ∈ *carrier-mat n n* **by** (*meson assms*(*1*) *similar-mat-witD2*(*5*))
from ∗ **have** *5*: *U* ∈ *carrier-mat n n* **by** (*meson assms*(*1*) *similar-mat-witD2*(*6*))

from *that* **show** *?thesis* **using** *1 2 3 4 5* **by** *blast*
qed

lemma *conjugate-vec-first*:
assumes *v* ∈ *carrier-vec n*
assumes *i* ≤ *n*

shows *conjugate* (*vec-first v i*) = *vec-first* (*conjugate v*) *i*
by (*smt* (*verit*, *ccfv-SIG*) *assms carrier-vecD dim-vec-conjugate dim-vec-first eq-vecI index-vec le-less less-trans vec-first-def vec-index-conjugate*)

lemma *conjugate-vec-last*: *i* ≤ *dim-vec v* ⟹ *conjugate* (*vec-last v i*) = *vec-last* (*conjugate v*) *i*
unfolding *vec-last-def* **by** *auto*

lemma *adjoint-is-conjugate-transpose*: A^H = *adjoint A*
by (*simp add*: *adjoint-def transpose-def cong-mat conjugate-mat-def*)

lemma *cscalar-prod-symm-conj*:
dim-vec (*x*::('*a*::{*comm-semiring-0*,*conjugatable-ring*} *vec*)) = *dim-vec* (*y*::'*a vec*)
⟹ *x* ·*c y* = *conjugate* (*y* ·*c x*)
by (*simp add*: *conjugate-scalar-prod scalar-prod-comm*)

6 Block Matrix Lemmas

lemma *block-mat-vec-mult*:
fixes *x*
assumes *A* ∈ *carrier-mat nr1 nc1*
assumes *B* ∈ *carrier-mat nr1 nc2*
assumes *C* ∈ *carrier-mat nr2 nc1*
assumes *D* ∈ *carrier-mat nr2 nc2*
assumes *M* = *four-block-mat A B C D*
assumes *x* ∈ *carrier-vec* (*nc1* + *nc2*)
defines x_1 ≡ *vec-first x nc1*
defines x_2 ≡ *vec-last x nc2*
shows *M* *$_v$ *x* = (*A* *$_v$ x_1 + *B* *$_v$ x_2) @$_v$ (*C* *$_v$ x_1 + *D* *$_v$ x_2)
by (*smt* (*verit*, *ccfv-threshold*) *assms four-block-mat-mult-vec vec-first-carrier vec-first-last-append vec-last-carrier*)

lemma *mat-vec-prod-leading-principal-submatrix*:
fixes *A* :: ('*a* :: *comm-ring*) *mat*
assumes *A* ∈ *carrier-mat* (*Suc n*) (*Suc n*)
assumes *x* ∈ *carrier-vec* (*Suc n*)
defines A_n ≡ *lps A n*
defines v_n ≡ *vec-first* (*col A n*) *n*
defines w_n ≡ *vec-first* (*row A n*) *n*
defines *a* ≡ *A* $$ (*n*, *n*)
defines x_n ≡ *vec-first x n*
defines *b* ≡ *x*$*n*
shows *A* *$_v$ *x* = (A_n *$_v$ x_n + *b* ·$_v$ v_n) @$_v$ (*vec 1* (λ*i*. (w_n · x_n) + *a* ∗ *b*)) (**is** *?lhs* = *?rhs*)
proof
have *dim-*x_n: *dim-vec* x_n = *n* **by** (*simp add*: *assms*(*7*))
have *dim-*w_n: *dim-vec* w_n = *n* **by** (*simp add*: *assms*(*5*))
have *dim-row-*A_n: *dim-row* A_n = *n*
by (*metis assms*(*1*) *assms*(*3*) *carrier-matD*(*1*) *le-add2 leading-principal-submatrix-carrier*

plus-1-eq-Suc)
have *dim-col-*A_n: *dim-col* $A_n = n$
by (*metis assms*(*1*) *assms*(*3*) *carrier-matD*(*2*) *le-add2 leading-principal-submatrix-carrier plus-1-eq-Suc*)

show *dims*: *dim-vec ?lhs* = *dim-vec ?rhs* **using** *assms*(*1*) v_n*-def* **by** *auto*
show $\bigwedge i.\ i < dim\text{-}vec\ ?rhs \Longrightarrow ?lhs\$i = ?rhs\$i$
proof−
fix *i* **assume** ∗: $i < dim\text{-}vec\ ?rhs$
hence *i*: $i < Suc\ n$ **using** *dims assms*(*1*) **by** *auto*
hence *dot*: $?lhs\$i = row\ A\ i \cdot x$ **using** ∗ *dims* **by** *fastforce*
have *row-j*: $\bigwedge j.\ j < Suc\ n \Longrightarrow (row\ A\ i)\$j = A\$\$(i,j)$ **using** *assms*(*1*) *i* **by** *force*

show $?lhs\$i = ?rhs\$i$
proof(*cases* $i < n$)
case *True*
have A_n: $\bigwedge j.\ j < n \Longrightarrow A\$\$(i,j) = A_n\$\$(i,j)$
by (*metis True assms*(*1*) *assms*(*3*) *le-add2 leading-principal-submatrix-index plus-1-eq-Suc*)
have A_n*-row*: $\bigwedge j.\ j < n \Longrightarrow A_n\$\$(i,j) = (row\ A_n\ i)\$j$
by (*metis True assms*(*1*) *assms*(*3*) *carrier-matD*(*1*) *carrier-matD*(*2*) *index-row*(*1*) *le-add2 leading-principal-submatrix-carrier plus-1-eq-Suc*)
have $?lhs\$i = (A_n *_v x_n + b \cdot_v v_n)\$i$
proof−
have $?lhs\$i = (\sum j{<}Suc\ n.\ A\$\$(i,j) * x\$j)$
unfolding *dot scalar-prod-def* **using** *row-j assms*(*2*) *atLeast0LessThan* **by** *force*
moreover have $(\sum j{<}n.\ A\$\$(i,j) * x\$j) = (\sum j{<}n.\ A_n\$\$(i,j) * x_n\$j)$
by (*smt* (*verit*) A_n *assms*(*7*) *index-vec lessThan-iff sum.cong vec-first-def*)
moreover have $(\sum j{<}n.\ A_n\$\$(i,j) * x_n\$j) = (A_n *_v x_n)\$i$
proof−
have $(A_n *_v x_n)\$i = (row\ A_n\ i) \cdot x_n$
by (*metis True assms*(*1*) *assms*(*3*) *carrier-matD*(*1*) *index-mult-mat-vec le-add2 leading-principal-submatrix-carrier plus-1-eq-Suc*)
moreover have $\bigwedge j.\ j < n \Longrightarrow A_n\$\$(i,j) * x_n\$j = (row\ A_n\ i)\$j * x_n\$j$
using A_n*-row* **by** *presburger*
ultimately show *?thesis*
unfolding *scalar-prod-def* **using** *atLeast0LessThan dim-*x_n **by** *fastforce*
qed
moreover have $(A\$\$(i,n) * x\$n) = (b \cdot_v v_n)\$i$
proof−
have $x\$n = b$ **unfolding** *b-def* **by** *blast*
moreover have $A\$\$(i,n) = v_n\$i$
using *assms*(*1*) *i* **by** (*simp add*: *True vec-first-def* v_n*-def*)
moreover have $(b \cdot_v v_n)\$i = b * (v_n\$i)$ **by** (*simp add*: *True* v_n*-def*)
ultimately show *?thesis* **by** (*simp add*: *mult.commute*)
qed

ultimately show *?thesis* **by** (*simp add*: *True assms*(*4*))
qed
thus *?thesis* **by** (*simp add*: *True i* v_n*-def*)
next
case *False*
hence $*$: $i = n$ **using** *i* **by** *linarith*
hence *?lhs*\$*i* = (*row A n*) $\cdot$ *x* **using** *dot* **by** *blast*
also have ... = $w_n \cdot x_n + a * b$
proof−
have *row A n* = w_n @$_v$ *vec-last* (*row A n*) *1*
by (*metis* $*$ w_n*-def assms*(*1*) *i row-carrier-vec semiring-norm*(*174*) *vec-first-last-append*)
moreover have (*vec-last* (*row A n*) *1*)\$*0* = (*row A n*)\$*n*
by (*metis* $*$ *False assms*(*1*) *calculation carrier-matD*(*2*) *dim-*w_n *gr0I i index-append-vec*(*1*) *index-append-vec*(*2*) *index-row*(*2*) *zero-less-diff*)
ultimately have *row A n* = w_n @$_v$ (*vec 1* (λ-. *A*\$\$(*n*,*n*)))
by (*smt* (*verit*, *best*) $*$ *One-nat-def carrier-vecD dim-vec eq-vecI i index-vec less-Suc0 row-j vec-last-carrier*)
moreover have *x* = x_n @$_v$ (*vec 1* (λ-. *b*))
proof
show $*$: *dim-vec x* = *dim-vec* (x_n @$_v$ (*vec 1* (λ-. *b*))) **by** (*simp add*: *assms*(*2*) *dim-*x_n)
have $\bigwedge i.\ i <$ *Suc n* $\Longrightarrow$ *x*\$*i* = (x_n @$_v$ (*vec 1* (λ-. *b*)))\$*i*
proof−
fix *i* **assume** *i* < *Suc n*
show *x*\$*i* = (x_n @$_v$ (*vec 1* (λ-. *b*)))\$*i*
apply (*cases i* = *n*)
apply (*simp add*: *append-vec-def assms*(*8*) *dim-*x_n)
apply (*simp add*: *append-vec-def*)
by (*smt* (*verit*, *best*) ‹*i* < *Suc n*› *dim-*x_n *index-vec less-antisym vec-first-def* x_n*-def*)
qed
thus $\bigwedge i.\ i <$ *dim-vec* (x_n @$_v$ *vec 1* (λ-. *b*)) $\Longrightarrow$ *x* \$ *i* = (x_n @$_v$ *vec 1* (λ-. *b*)) \$ *i*
by (*metis* $*$ *assms*(*2*) *carrier-vecD*)
qed
ultimately have (*row A n*) $\cdot$ *x* = ($w_n \cdot x_n$) + ((*vec 1* (λ-. *A*\$\$(*n*,*n*))) $\cdot$ (*vec 1* (λ-. *b*)))
by (*metis assms*(*5*) *assms*(*7*) *scalar-prod-append vec-carrier vec-first-carrier*)
moreover have ((*vec 1* (λ-. *A*\$\$(*n*,*n*))) $\cdot$ (*vec 1* (λ-. *b*))) = $a * b$
by (*simp add*: *a-def b-def scalar-prod-def*)
ultimately show *?thesis* **by** *argo*
qed
finally show *?thesis* **by** (*simp add*: $*$ v_n*-def*)
qed
qed
qed

lemma *vec-first-index*: $n \leq$ *dim-vec v* $\Longrightarrow$ $i < n \Longrightarrow$ *v*\$*i* = (*vec-first v n*)\$*i*

unfolding *vec-first-def* **by** *simp*

lemma *vec-last-index*:
$n \leq$ *dim-vec* $v \Longrightarrow i \in \{$*dim-vec* $v - m..<m\} \Longrightarrow v\$i =$ (*vec-last* $v\ m$)\$($i -$ (*dim-vec* $v - m$))
unfolding *vec-last-def* **by** *auto*

lemma *inner-prod-append*:
assumes $x \in$ *carrier-vec* (*dim-vec* (u @$_v$ v))
shows $x \cdot_c$ (u @$_v$ v) = (*vec-first* x (*dim-vec* u)) $\cdot_c$ u + (*vec-last* x (*dim-vec* v)) $\cdot_c$ v
(u @$_v$ v) $\cdot_c$ x = $u \cdot_c$ (*vec-first* x (*dim-vec* u)) + $v \cdot_c$ (*vec-last* x (*dim-vec* v))
proof−
define n **where** $n \equiv$ *dim-vec* (u @$_v$ v)
define n_u **where** $n_u \equiv$ *dim-vec* u
define n_v **where** $n_v \equiv$ *dim-vec* v

have *dims-add*: $n_u + n_v = n$ **by** (*simp add*: n_u*-def* n_v*-def n-def*)

have n_u*-prop*: $\bigwedge i.\ i < n_u \Longrightarrow$ *conjugate* (u @$_v$ v)\$$i$ = (*conjugate* u)\$$i$ **by** (*simp add*: n_u*-def*)
have n_v*-prop*: $\bigwedge i.\ i < n_v \Longrightarrow$ *conjugate* (u @$_v$ v)\$($i + n_u$) = (*conjugate* v)\$$i$
by (*simp add*: n_u*-def* n_v*-def*)

have n = *dim-vec* (*conjugate* (u @$_v$ v)) **by** (*simp add*: *n-def*)
hence $x \cdot_c$ (u @$_v$ v) = ($\sum i \in \{0..<n\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$)
unfolding *scalar-prod-def* **by** *blast*
hence $x \cdot_c$ (u @$_v$ v) =
($\sum i \in \{0..<n_u\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$)
+ ($\sum i \in \{n_u..<n\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$)
by (*smt* (*verit*, *best*) *bot-nat-0.extremum index-append-vec*(2) n_u*-def n-def nat-le-linear nless-le not-add-less1 sum.atLeastLessThan-concat*)
moreover have ($\sum i \in \{0..<n_u\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$) = (*vec-first* x (*dim-vec* u)) $\cdot_c$ u
proof−
have $*$: $\bigwedge i.\ i \in \{0..<n_u\} \Longrightarrow x\$i =$ (*vec-first* x n_u)\$$i$
by (*simp add*: *vec-first-def*)
have ($\sum i \in \{0..<n_u\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$) = ($\sum i \in \{0..<n_u\}$. $x\$i * ($*conjugate* u)\$$i$)
using n_u*-prop* **by** *simp*
also have ... = ($\sum i \in \{0..<n_u\}$. (*vec-first* x n_u)\$$i * ($*conjugate* u)\$$i$)
using $*$ **by** *auto*
also have ... = (*vec-first* x (*dim-vec* u)) $\cdot_c$ u
by (*metis* (*no-types*, *lifting*) *dim-vec-conjugate* n_u*-def scalar-prod-def sum.cong*)
finally show *?thesis* **.**
qed
moreover have ($\sum i \in \{n_u..<n\}$. $x\$i * ($*conjugate* ($u$ @$_v$ v))\$$i$) = (*vec-last* x (*dim-vec* v)) $\cdot_c$ v
proof−

have $*$: *vec-last* $(u \mathbin{@_v} v)$ $n_v = v$
by (*metis append-vec-eq carrier-vecI dims-add* n_u*-def n-def vec-first-carrier vec-first-last-append*)
have $\bigwedge i.\ i \in \{n_u..<n\} \Longrightarrow x\$i = (\textit{vec-last}\ x\ n_v)\$(i - (n - n_v))$
unfolding *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*
moreover have $\bigwedge i.\ i \in \{n_u..<n\}$
$\Longrightarrow (\textit{conjugate}\ (u \mathbin{@_v} v))\$i = (\textit{vec-last}\ (\textit{conjugate}\ (u \mathbin{@_v} v))\ n_v)\$(i - (n - n_v))$
unfolding *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*
ultimately have $(\sum i \in \{n_u..<n\}.\ x\$i * (\textit{conjugate}\ (u \mathbin{@_v} v))\$i)$
$= (\sum i \in \{n_u..<n\}.\ (\textit{vec-last}\ x\ n_v)\$(i - (n - n_v))$
$* (\textit{vec-last}\ (\textit{conjugate}\ (u \mathbin{@_v} v))\ n_v)\$(i - (n - n_v)))$
(**is** - = ($\sum i \in$ -. *?F i*))
by *force*
also have ... $= (\sum i \in \{0..<n-n_u\}.\ (\textit{vec-last}\ x\ n_v)\$((i + n_u) - (n - n_v))$
$* (\textit{vec-last}\ (\textit{conjugate}\ (u \mathbin{@_v} v))\ n_v)\$((i + n_u) - (n - n_v)))$
using *sum.shift-bounds-nat-ivl*[*of ?F 0* n_u n_v] *dims-add*
by (*metis* (*no-types*, *lifting*) *add.commute add-0 add-diff-cancel-left'*)
finally show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) $*$ *add-diff-cancel-left' add-diff-cancel-right' carrier-vecI conjugate-vec-last dim-vec-conjugate dims-add le-add2* n_v*-def n-def scalar-prod-def sum.cong*)
qed
ultimately show $x \cdot_c (u \mathbin{@_v} v) = (\textit{vec-first}\ x\ (\textit{dim-vec}\ u)) \cdot_c u + (\textit{vec-last}\ x\ (\textit{dim-vec}\ v)) \cdot_c v$
by *argo*

have n_u*-prop*: $\bigwedge i.\ i < n_u \Longrightarrow (u \mathbin{@_v} v)\$i = u\$i$ **by** (*simp add*: n_u*-def*)
have n_v*-prop*: $\bigwedge i.\ i < n_v \Longrightarrow (u \mathbin{@_v} v)\$(i + n_u) = v\$i$
by (*simp add*: n_u*-def* n_v*-def*)

have $n = \textit{dim-vec}\ (\textit{conjugate}\ (u \mathbin{@_v} v))$ **by** (*simp add*: *n-def*)
moreover have *dim-vec* (*conjugate x*) = *n* **using** *assms n-def* **by** *auto*
ultimately have $(u \mathbin{@_v} v) \cdot_c x = (\sum i \in \{0..<n\}.\ (u \mathbin{@_v} v)\$i * (\textit{conjugate}\ x)\$i)$
unfolding *scalar-prod-def* **by** *presburger*
hence $(u \mathbin{@_v} v) \cdot_c x =$
$(\sum i \in \{0..<n_u\}.\ (u \mathbin{@_v} v)\$i * (\textit{conjugate}\ x)\$i)$
$+ (\sum i \in \{n_u..<n\}.\ (u \mathbin{@_v} v)\$i * (\textit{conjugate}\ x)\$i)$
by (*smt* (*verit*, *best*) *bot-nat-0.extremum index-append-vec*(*2*) n_u*-def n-def nat-le-linear nless-le not-add-less1 sum.atLeastLessThan-concat*)
moreover have $(\sum i \in \{0..<n_u\}.\ (u \mathbin{@_v} v)\$i * (\textit{conjugate}\ x)\$i) = u \cdot_c (\textit{vec-first}\ x\ (\textit{dim-vec}\ u))$
proof−
have $*$: $\bigwedge i.\ i \in \{0..<n_u\} \Longrightarrow (\textit{conjugate}\ x)\$i = (\textit{vec-first}\ (\textit{conjugate}\ x)\ n_u)\$i$
by (*simp add*: *vec-first-def*)
have $(\sum i \in \{0..<n_u\}.\ (u \mathbin{@_v} v)\$i * (\textit{conjugate}\ x)\$i)$
$= (\sum i \in \{0..<n_u\}.\ u\$i * (\textit{conjugate}\ x)\$i)$
using n_u*-prop* **by** *simp*

also have ... = $(\sum i \in \{0..<n_u\}.\ u\$i * (vec\text{-}first\ (conjugate\ x)\ n_u)\$i)$
using $*$ **by** *force*
also have ... = $u \cdot c\ (vec\text{-}first\ x\ (dim\text{-}vec\ u))$
by (*metis* (*no-types*, *lifting*) *add.commute assms conjugate-vec-first dim-vec-first dims-add le-add2* n_u*-def n-def scalar-prod-def sum.cong*)
finally show *?thesis* **.**
qed
moreover have $(\sum i \in \{n_u..<n\}.\ (u\ @_v\ v)\$i * (conjugate\ x)\$i) = v \cdot c\ (vec\text{-}last\ x\ (dim\text{-}vec\ v))$
proof−
have $*$: $vec\text{-}last\ (u\ @_v\ v)\ n_v = v$
by (*metis append-vec-eq carrier-vecI dims-add* n_u*-def n-def vec-first-carrier vec-first-last-append*)
have $\bigwedge i.\ i \in \{n_u..<n\} \Longrightarrow (conjugate\ x)\$i = (vec\text{-}last\ (conjugate\ x)\ n_v)\$(i - (n - n_v))$
unfolding *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*
moreover have $\bigwedge i.\ i \in \{n_u..<n\}$
$\Longrightarrow (u\ @_v\ v)\$i = (vec\text{-}last\ (u\ @_v\ v)\ n_v)\$(i - (n - n_v))$
unfolding *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*
ultimately have $(\sum i \in \{n_u..<n\}.\ (u\ @_v\ v)\$i * (conjugate\ x)\$i)$
$= (\sum i \in \{n_u..<n\}.\ (vec\text{-}last\ (u\ @_v\ v)\ n_v)\$(i - (n - n_v))$
$* (vec\text{-}last\ (conjugate\ x)\ n_v)\$(i - (n - n_v)))$
(**is** - = ($\sum i \in$ -. *?F i*))
by *force*
also have ... = $(\sum i \in \{0..<n-n_u\}.\ (vec\text{-}last\ (u\ @_v\ v)\ n_v)\$((i + n_u) - (n - n_v))$
$* (vec\text{-}last\ (conjugate\ x)\ n_v)\$((i + n_u) - (n - n_v)))$
using *sum.shift-bounds-nat-ivl*[*of ?F 0* n_u n_v] *dims-add*
by (*metis* (*no-types*, *lifting*) *add.commute add-0 add-diff-cancel-left'*)
finally show *?thesis*
by (*smt* (*verit*, *best*) $*$ ‹*dim-vec* (*conjugate x*) = *n*› *add-diff-cancel-left' add-diff-cancel-right' conjugate-vec-last dim-vec-conjugate dim-vec-last dims-add le-add2 scalar-prod-def sum.cong*)
qed
ultimately show $(u\ @_v\ v) \cdot c\ x = u \cdot c\ (vec\text{-}first\ x\ (dim\text{-}vec\ u)) + v \cdot c\ (vec\text{-}last\ x\ (dim\text{-}vec\ v))$
by *argo*
qed

6.1 Schur's Formula

proposition *schur-formula*:
fixes *M* :: *'a*::*field mat*
assumes (*A*,*B*,*C*,*D*) = *split-block M r c*
assumes *r* < *dim-row M*
assumes *c* < *dim-col M*
assumes *square-mat M*
assumes *square-mat A*
assumes *inverts-mat A' A*

assumes *A′-dim*: $A' \in$ *carrier-mat r r*
shows *det M* = *det A* ∗ *det* (*D* − *C* ∗ *A′* ∗ *B*)
proof−
let *?r*$_M$ = *dim-row M*
let *?c*$_M$ = *dim-col M*
let *?r*$_A$ = *r*
let *?c*$_A$ = *c*
let *?r*$_B$ = *r*
let *?c*$_B$ = *?c*$_M$ − *?c*$_A$
let *?r*$_C$ = *?r*$_M$ − *?r*$_A$
let *?c*$_C$ = *c*
let *?r*$_D$ = *?r*$_M$ − *?r*$_A$
let *?c*$_D$ = *?c*$_M$ − *?c*$_A$
let *?I*$_A$ = *1*$_m$ *r*
let *?I*$_D$ = *1*$_m$ *?r*$_D$
let *?O*$_B$ = *0*$_m$ *?r*$_B$ *?c*$_B$
let *?O*$_C$ = *0*$_m$ *?r*$_C$ *?c*$_C$
let *?P* = *four-block-mat ?I*$_A$ *?O*$_B$ (*C* ∗ *A′*) *?I*$_D$
let *?Q* = *four-block-mat A ?O*$_B$ *?O*$_C$ (*D* − *C* ∗ *A′* ∗ *B*)
let *?R* = *four-block-mat ?I*$_A$ (*A′* ∗ *B*) *?O*$_C$ *?I*$_D$

have *M*: *M* = *four-block-mat A B C D*
using *Matrix.split-block*(*5*)[*of M r c A B C D*] **by** (*metis assms*(*1*−*3*) *le-simps*(*1*) *less-eqE*)

have *M-dim*: *M* ∈ *carrier-mat ?r*$_M$ *?c*$_M$
by *blast*
have *A-dim*: *A* ∈ *carrier-mat r r*
using *assms*(*1*)
unfolding *split-block-def*
by (*metis Pair-inject assms*(*5*) *carrier-mat-triv dim-row-mat*(*1*) *square-mat.elims*(*2*))
have *square*: *?r*$_M$ − *?r*$_A$ = *?c*$_M$ − *?c*$_A$ *r* = *c ?r*$_M$ = *?c*$_M$
using *M-dim assms*(*4*)
apply (*metis A-dim assms*(*1*) *assms*(*5*) *carrier-matD*(*1*) *dim-col-mat*(*1*) *prod.sel*(*1*) *split-block-def square-mat.elims*(*2*))
apply (*metis A-dim assms*(*1*) *carrier-matD*(*2*) *dim-col-mat*(*1*) *prod.sel*(*1*) *split-block-def*)
using *assms*(*4*) **by** *force*
have *C-A′-dim*: *C* ∗ *A′* ∈ *carrier-mat ?r*$_C$ *?c*$_C$
by (*smt* (*verit*) *A′-dim Pair-inject assms*(*1*) *carrier-matD*(*2*) *carrier-mat-triv dim-row-mat*(*1*) *index-mult-mat*(*2*) *index-mult-mat*(*3*) *split-block-def square*(*2*))
have *A′-B-dim*: *A′* ∗ *B* ∈ *carrier-mat ?r*$_B$ *?c*$_B$
by (*metis* (*no-types*, *lifting*) *A′-dim Pair-inject assms*(*1*) *carrier-matD*(*1*) *carrier-mat-triv dim-col-mat*(*1*) *index-mult-mat*(*2*) *index-mult-mat*(*3*) *split-block-def*)
have *D-min-C-A′-B-dim*: *D* − *C* ∗ *A′* ∗ *B* ∈ *carrier-mat ?c*$_D$ *?c*$_D$
by (*metis A′-B-dim C-A′-dim carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-mat-triv index-mult-mat*(*2*) *index-mult-mat*(*3*) *minus-carrier-mat square*(*1*))
have *P-dim*: *?P* ∈ *carrier-mat ?r*$_M$ *?c*$_M$
using *assms*(*2*) *square*(*3*) **by** *auto*

have *Q-dim*: $?Q \in$ *carrier-mat* $?r_M$ $?c_M$
by (*smt* (*verit*) *A-dim D-min-C-A′-B-dim P-dim carrier-matD*(*1*) *carrier-matD*(*2*) *carrier-mat-triv index-mat-four-block*(*2*) *index-mat-four-block*(*3*) *index-one-mat*(*3*) *square*(*2*) *square*(*3*))
have *R-dim*: $?R \in$ *carrier-mat* $?r_M$ $?c_M$
using *P-dim* **by** *fastforce*

have *M*: $M = ?P * ?Q * ?R$
proof−
have *B-dim*: $B \in$ *carrier-mat* $?r_B$ $?c_B$
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*) *less-imp-le-nat ordered-cancel-comm-monoid-diff-class.add-diff-inve*
split-block(*2*))
have *C-dim*: $C \in$ *carrier-mat* $?r_C$ r
by (*metis assms*(*1*) *assms*(*3*) *diff-add-inverse less-eqE less-or-eq-imp-le split-block*(*3*) *square*(*2*) *square*(*3*))
have *D-dim*: $D \in$ *carrier-mat* $?r_D$ $?c_D$
using *A-dim M square*(*2*) **by** *auto*

have *1*: $?I_A \in$ *carrier-mat* r r **by** *simp*
have *2*: $?O_B \in$ *carrier-mat* $?r_B$ $?c_B$ **by** *auto*
have *3*: $C * A' \in$ *carrier-mat* $?r_C$ r **using** *C-A′-dim square*(*2*) **by** *blast*
have *4*: $?I_D \in$ *carrier-mat* $?r_C$ $?c_B$ **using** *2 square*(*1*) **by** *auto*
have *6*: $?O_B \in$ *carrier-mat* r $?c_B$ **using** *2 square*(*2*) **by** *blast*
have *7*: $?O_C \in$ *carrier-mat* $?c_B$ r **using** *4 square*(*2*) **by** *auto*
have *8*: $(D - C * A' * B) \in$ *carrier-mat* $?c_B$ $?c_B$
using *4 D-min-C-A′-B-dim square*(*1*) *square*(*2*) **by** *auto*
have *a*: $(D - C * A' * B) \in$ *carrier-mat* $?r_D$ $?c_D$ **using** *8 square*(*1*) **by** *presburger*
have *b*: $?I_D \in$ *carrier-mat* $?c_B$ $?c_B$ **using** *2 square*(*1*) **by** *auto*
have *ass*: $(C * A') * A = C * (A' * A)$ **using** *A-dim A′-dim C-dim* **by** (*simp add*: *square*(*2*))
have $?P * ?Q =$ *four-block-mat* A $?O_B$ C $(D - C * A' * B)$
proof−
have $A = ?I_A * A + ?O_B * ?O_C$
using *A-dim square*(*2*) *square*(*3*) **by** *auto*
moreover have $?O_B = ?I_A * ?O_B + ?O_B * (D - C * A' * B)$
by (*smt* (*verit*, *ccfv-threshold*) *1 2 8 D-min-C-A′-B-dim carrier-matD*(*2*) *left-add-zero-mat left-mult-zero-mat right-mult-zero-mat*)
moreover have $C = (C * A') * A + ?I_D * ?O_C$
by (*metis A′-dim C-dim square*(*2*) *Matrix.right-add-zero-mat ass assms*(*6*) *carrier-matD*(*1*) *index-zero-mat*(*2*) *inverts-mat-def left-mult-one-mat′ right-mult-one-mat*)
moreover have $(D - C * A' * B) = ((C * A') * ?O_B) + ?I_D * (D - C * A' * B)$
by (*metis C-A′-dim D-min-C-A′-B-dim left-add-zero-mat left-mult-one-mat right-mult-zero-mat square*(*1*) *square*(*2*))
ultimately show *?thesis*
using *mult-four-block-mat*[*OF 1 2 3 4 A-dim 6 7 8*] **by** *argo*
qed
also have ... $* ?R =$ *four-block-mat* A B C D

proof −
have $A = A * ?I_A + ?O_B * C$
using *A-dim C-dim square*(*1*) **by** *force*
moreover have $B = A * (A' * B) + ?O_B * ?I_D$
by (*smt* (*verit*, *ccfv-threshold*) *inverts-mat-symm A-dim B-dim Matrix.right-add-zero-mat assms*(*6*) *assms*(*7*) *assoc-mult-mat b carrier-mat-triv index-mult-mat*(*2*) *inverts-mat-def left-mult-one-mat left-mult-zero-mat*)
moreover have $C = C * ?I_A + (D - C * A' * B) * ?O_C$
by (*metis 8 C-dim Matrix.right-add-zero-mat right-mult-one-mat right-mult-zero-mat square*(*1*) *square*(*2*))
moreover have $D = C * (A' * B) + (D - C * A' * B) * ?I_D$
proof −
have $C * A' * B \in$ *carrier-mat* $?r_D$ $?c_D$
using *B-dim C-A'-dim mult-carrier-mat square*(*2*) **by** *blast*
moreover have $C * (A' * B) = C * A' * B$ **using** *B-dim C-dim assms*(*7*) **by** *force*
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *8 D-dim left-add-zero-mat mat-minus-minus minus-r-inv-mat right-mult-one-mat square*(*2*) *square*(*3*))
qed
ultimately show *?thesis*
using *mult-four-block-mat*[*OF A-dim 2 C-dim a 1 A'-B-dim 7 b*] *A-dim square*(*2*) *square*(*3*)
by *force*
qed
also have ... $= M$ **unfolding** M **by** *simp*
finally show *?thesis* **by** *argo*
qed
hence *det* $M =$ *det* $?P *$ *det* $?Q *$ *det* $?R$
by (*smt* (*verit*, *best*) *det-mult P-dim Q-dim R-dim assms*(*4*) *mult-carrier-mat square-mat.elims*(*2*))
moreover have *det* $?P = 1$
using *det-four-block-mat-upper-right-zero*[*OF - - C-A'-dim*, *of* $?I_A$ $?O_B$ $?I_D$]
by (*simp add*: *square*)
moreover have *det* $?Q =$ *det* $A *$ *det* $(D - C * A' * B)$
using *det-four-block-mat-upper-right-zero*[*OF A-dim - - D-min-C-A'-B-dim*, *of* $?O_B$ $?O_C$]
by (*simp add*: *square*)
moreover have *det* $?R = 1$
using *det-four-block-mat-lower-left-zero*[*OF - A'-B-dim*, *of* $?I_A$ $?O_C$ $?I_D$]
by (*simp add*: *square*)
ultimately show *?thesis* **by** *fastforce*
qed

7 Positive Definite Lemmas

definition *positive-definite* **where**
positive-definite $M \longleftrightarrow$ *hermitian* M
$\wedge$ $(\forall x \in$ *carrier-vec* $(dim\text{-}col\ M).\ x \neq 0_v\ (dim\text{-}col\ M) \longrightarrow QF\ M\ x > 0)$

lemma *leading-principal-submatrix-positive-definite*:
fixes $A :: 'a::\{conjugatable\text{-}field,ord\}\ mat$
assumes $A \in carrier\text{-}mat\ n\ n$
assumes *positive-definite A*
assumes $k \leq n$
shows *positive-definite* $(lps\ A\ k)$
proof−
define B **where** $B \equiv lps\ A\ k$
have *B-carrier*: $B \in carrier\text{-}mat\ k\ k$
using *B-def assms*(1) *assms*(3) *leading-principal-submatrix-carrier* **by** *blast*
hence *B-dims*: *dim-row* $B = k \wedge$ *dim-col* $B = k$ **by** *simp*
{ fix $v :: 'a\ vec$
assume $*$: $v \in carrier\text{-}vec\ k\ v \neq 0_v\ k$
define w **where** $w \equiv vec\ n\ (\lambda i.\ if\ i < k\ then\ v\$i\ else\ 0)$
hence $w \neq 0_v\ n$
by (*smt* (*verit*) $*(1)$ $*(2)$ *assms*(3) *carrier-vecD dual-order.strict-trans1 eq-vecI index-vec index-zero-vec*(1) *index-zero-vec*(2))
hence $QF\ A\ w > 0$ **using** *assms*(1) *assms*(2) *positive-definite-def vec-carrier w-def* **by** *blast*
moreover have $QF\ A\ w = QF\ B\ v$
proof−
have 1: $\bigwedge i.\ i \in \{k..<n\} \Longrightarrow conjugate\ (w\$i) = 0$ **using** *w-def* **by** *simp*
have 2: $\bigwedge i.\ i \in \{0..<k\} \Longrightarrow w\$i = v\$i$ **using** *assms*(3) *w-def* **by** *auto*
hence 3: $\bigwedge i.\ i \in \{0..<k\} \Longrightarrow conjugate\ (w\$i) = conjugate\ (v\$i)$ **by** *presburger*
have 4: $\bigwedge i.\ i \in \{0..<k\} \Longrightarrow (A *_v w)\$i = (B *_v v)\$i$
proof−
fix i **assume** $**$: $i \in \{0..<k\}$
have $***$: $\bigwedge j.\ j \in \{0..<k\} \Longrightarrow (row\ A\ i)\$j = (row\ B\ i)\$j$
proof−
fix j **assume** $j \in \{0..<k\}$
moreover then have $(row\ A\ i)\$j = A\$\$(i,j)$
using $** * assms(1)\ assms(3)$ **by** *force*
ultimately show $(row\ A\ i)\$j = (row\ B\ i)\$j$
by (*metis* (*mono-tags*, *lifting*) $**$ *B-def B-dims assms*(1) *assms*(3) *atLeastLessThan-iff index-vec leading-principal-submatrix-index row-def*)
qed
have $****$: $row\ B\ i \in carrier\text{-}vec\ k \wedge dim\text{-}vec\ v = k$
using *B-carrier B-dims* $*$ **by** *auto*
have $(A *_v w)\$i = row\ A\ i \cdot w$ **using** $**$ *assms*(1) *assms*(3) **by** *force*
also have $... = (\sum j \in \{0..<n\}.\ (row\ A\ i)\$j * w\$j)$
by (*metis* (*no-types*, *lifting*) *dim-vec scalar-prod-def sum.cong w-def*)
also have $... = (\sum j \in \{0..<k\}.\ (row\ A\ i)\$j * w\$j) + (\sum j \in \{k..<n\}.\ (row\ A\ i)\$j * w\$j)$
by (*simp add*: *assms*(3) *sum.atLeastLessThan-concat*)
also have $... = (\sum j \in \{0..<k\}.\ (row\ A\ i)\$j * v\$j)$
using 1 2 **by** *auto*
also have $... = (\sum j \in \{0..<k\}.\ (row\ B\ i)\$j * v\$j)$
using $***$ **by** *fastforce*

```
    also have ... = (row B i) · v
      using B-carrier **** unfolding scalar-prod-def by blast
    also have ... = (B *v v)$i using ** B-carrier by auto
    finally show (A *v w)$i = (B *v v)$i .
  qed
  have 5: v ∈ carrier-vec k by (simp add: *(1))
  hence 6: B *v v ∈ carrier-vec k ∧ conjugate v ∈ carrier-vec k
      by (metis B-def Matrix.carrier-vec-conjugate assms(1) assms(3) leading-principal-submatrix-carrier mult-mat-vec-carrier)
  have QF A w = (A *v w) ·c w by simp
  also have ... = (∑ i ∈ {0..<n}. (A *v w)$i * conjugate (w$i))
    by (smt (verit, ccfv-SIG) atLeastLessThan-iff dim-vec dim-vec-conjugate scalar-prod-def sum.cong vec-index-conjugate w-def)
  also have ... = (∑ i ∈ {0..<k}. (A *v w)$i * conjugate (w$i))
    + (∑ i ∈ {k..<n}. (A *v w)$i * conjugate (w$i))
    by (simp add: assms(3) sum.atLeastLessThan-concat)
  also have ... = (∑ i ∈ {0..<k}. (A *v w)$i * conjugate (w$i)) using 1 by simp
  also have ... = (∑ i ∈ {0..<k}. (B *v v)$i * conjugate (v$i)) using 3 4 by force
  also have ... = (B *v v) ·c v
    by (smt (verit, best) 5 6 atLeastLessThan-iff carrier-vecD scalar-prod-def sum.cong vec-index-conjugate)
  finally show ?thesis using quadratic-form-def by force
 qed
 ultimately have QF B v > 0 by argo
}
thus ?thesis
 using assms(2) leading-principal-submatrix-hermitian
 unfolding positive-definite-def
 by (metis B-def B-dims assms(1) assms(3))
qed

lemma positive-definite-invertible:
 fixes M :: complex mat
 assumes positive-definite M
 shows invertible-mat M
proof−
 define n where n ≡ dim-row M
 have ⋀x. x ∈ carrier-vec n ⟹ x ≠ 0v n ⟹ M *v x ≠ 0v n
  using assms n-def positive-definite-def hermitian-is-square by force
 hence mat-kernel M ⊆ {0v n}
  unfolding mat-kernel-def n-def
  by (metis (mono-tags, lifting) assms hermitian-is-square mem-Collect-eq positive-definite-def singleton-iff square-mat.simps subsetI)
 thus ?thesis
  using trivial-kernel-imp-invertible n-def assms positive-definite-def hermitian-is-square
  by blast
qed
```

lemma *positive-definite-det-nz*:
fixes A :: *complex mat*
assumes *positive-definite A*
shows *det A* $\neq$ *0*
using *positive-definite-invertible*[*OF assms*] *invertible-det invertible-mat-def square-mat.simps*
by *blast*

end
theory *Sylvester-Criterion*
imports *Misc-Matrix-Results*

begin

8 Sylvester's Criterion Setup

definition *sylvester-criterion* :: ('*a*::{*comm-ring-1*,*ord*}) *mat* $\Rightarrow$ *bool* **where**
sylvester-criterion A $\longleftrightarrow$ ($\forall$ *k* $\in$ {*0*..*dim-row A*}. *Determinant.det* (*lps A k*) > *0*)

lemma *leading-principle-submatrix-sylvester*:
assumes $A \in$ *carrier-mat n n*
assumes $m \leq n$
assumes *sylvester-criterion A*
shows *sylvester-criterion* (*lps A m*)
using *nested-leading-principle-submatrices*
by (*smt* (*verit*, *del-insts*) *assms atLeastAtMost-iff carrier-matD*(*1*) *order.trans leading-principal-submatrix-carrier sylvester-criterion-def*)

lemma *sylvester-criterion-positive-det*:
assumes $A \in$ *carrier-mat n n*
assumes *sylvester-criterion A*
shows *det A* > *0*
proof−
have *A* = *lps A n*
unfolding *leading-principal-submatrix-def submatrix-def*
using *assms*(*1*) *pick-n-le*
by *auto*
thus *?thesis* **using** *assms* **unfolding** *sylvester-criterion-def* **by** *force*
qed

9 Sylvester's Criterion

9.1 Forward Implication

lemma *sylvester-criterion-forward*:
fixes A :: *complex mat*
assumes $A \in$ *carrier-mat n n*

assumes $x \in$ *carrier-vec n*
assumes *hermitian A*
assumes *sylvester-criterion A*
assumes $x \neq 0_v\ n$
shows *Re* (*QF A x*) > *0*
using *assms*
proof(*induction n arbitrary*: *A x*)
case *0*
then show *?case* **by** (*metis carrier-vecD eq-vecI not-less-zero zero-carrier-vec*)
next
case (*Suc n*)

have ∗: $\bigwedge k.\ k \leq$ *dim-row A* $\Longrightarrow$ *det* (*leading-principal-submatrix A k*) > *0*
using *Suc*(*5*) *atLeastAtMost-iff* **unfolding** *sylvester-criterion-def* **by** *blast*

define A_n **where** $A_n \equiv$ (*leading-principal-submatrix A n*)
define v_n **where** $v_n \equiv$ *vec-first* (*col A n*) *n*
define $v_n c$ **where** $v_n c \equiv$ *conjugate* v_n
define w_n **where** $w_n \equiv$ *vec-first* (*row A n*) *n*
define *a* **where** *a* ≡ *A* $$ (*n*, *n*)
define x_n **where** $x_n \equiv$ *vec-first x n*
define $x_n c$ **where** $x_n c =$ *conjugate* x_n
define *b* **where** *b* ≡ *x*$*n*
define *b-conj* **where** *b-conj* ≡ *conjugate b*

have *carrier-*A_n: $A_n \in$ *carrier-mat n n*
by (*metis* A_n*-def Suc.prems*(*1*) *le-add2 leading-principal-submatrix-carrier plus-1-eq-Suc*)
have *herm-*A_n: *hermitian* A_n
using *principal-submatrix-hermitian*[*of A Suc n* {..*n*}] A_n*-def*
unfolding *leading-principal-submatrix-def*
by (*metis Suc.prems*(*1*) *Suc.prems*(*3*) *dual-order.refl le-SucI lessThan-subset-iff principal-submatrix-hermitian*)

have (*col A n*) = *conjugate* (*row A n*)
by (*metis Suc.prems*(*1*) *Suc.prems*(*3*) *adjoint-col carrier-matD*(*1*) *hermitian-def lessI*)
hence *wn-vn-conj*: $w_n = v_n c$
by (*metis Suc.prems*(*1*) *conjugate-vec-first col-carrier-vec conjugate-id le-add2 lessI plus-1-eq-Suc* v_n*-def* $v_n c$*-def* w_n*-def*)

have *invertible-mat* A_n
by (*metis* ∗ A_n*-def Suc*(*2*) *carrier-*A_n *carrier-matD*(*1*) *invertible-det le-add2 less-irrefl plus-1-eq-Suc*)
then obtain $A_n{}'$ **where** *An'*: *inverts-mat* $A_n{}'$ A_n ∧ $A_n{}'$ ∈ *carrier-mat n n*
by (*metis* (*no-types*, *lifting*) *invertible-mat-def* A_n*-def Suc.prems*(*1*) *carrier-matD*(*1*) *carrier-matI index-mult-mat*(*3*) *index-one-mat*(*3*) *inverts-mat-def le-add2 leading-principal-submatrix-carrier plus-1-eq-Suc square-mat.simps*)

have *xn*: $x_n \in$ *carrier-vec n* **by** (*simp add*: x_n*-def*)

moreover have *An*: $A_n \in$ *carrier-mat n n*
using *leading-principal-submatrix-carrier*
by (*metis* A_n*-def Suc.prems*(*1*) *Suc-n-not-le-n linorder-linear*)
ultimately have *An-xn*: $A_n *_v x_n \in$ *carrier-vec n* **by** *fastforce*
have *vn*: $v_n \in$ *carrier-vec n* **by** (*simp add*: v_n*-def*)
hence *b-vn*: $b \cdot_v v_n \in$ *carrier-vec n* **by** *simp*

have $(A_n *_v x_n + b \cdot_v v_n) \in$ *carrier-vec n* **using** *An-xn b-vn* **by** *auto*

from *herm-*A_n **have** *hermitian*: *hermitian* $A_n{}'$
by (*metis hermitian-mat-inv An' An inverts-mat-symm*)
hence A_n*-inv-conj*: *conjugate* $A_n{}' = A_n{}^{\prime T}$
by (*metis conjugate-id hermitian-def adjoint-is-conjugate-transpose*)

have $**$: $(A_n *_v (x_n + b \cdot_v (A_n{}' *_v v_n))) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^{\prime T} *_v v_n c))$
$= (QF\ A_n\ x_n) + b * (x_n c \cdot v_n) + b\text{-}conj * (x_n \cdot v_n c) + (cmod\ b)\hat{}2 * ((A_n{}' *_v v_n) \cdot v_n c)$
(**is** *?lhs* = *?rhs*)
proof−
define E **where** $E \equiv ((A_n *_v x_n) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^{\prime T} *_v v_n c)))$
define F **where** $F \equiv ((b \cdot_v v_n) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^{\prime T} *_v v_n c)))$

have $A_n *_v (x_n + b \cdot_v (A_n{}' *_v v_n)) = (A_n *_v x_n) + b \cdot_v v_n$
(**is** *?lhs'* = -)
proof−
have *?lhs'* = $A_n *_v x_n + A_n *_v (b \cdot_v (A_n{}' *_v v_n))$
by (*meson An' carrier-*A_n *mult-add-distrib-mat-vec mult-mat-vec-carrier smult-carrier-vec vn xn*)
also have ... = $A_n *_v x_n + (b \cdot_v ((A_n * A_n{}') *_v v_n))$
by (*metis An' assoc-mult-mat-vec carrier-*A_n *mult-mat-vec mult-mat-vec-carrier vn*)
also have ... = $(A_n *_v x_n) + b \cdot_v v_n$
by (*metis An' carrier-*A_n *carrier-matD*(*1*) *inverts-mat-def inverts-mat-symm one-mult-mat-vec vn*)
finally show *?thesis* **.**
qed
hence *?lhs* = $((A_n *_v x_n) + b \cdot_v v_n) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^{\prime T} *_v v_n c))$ **by** *argo*
moreover have ... = $E + F$
unfolding *E-def F-def*
by (*metis An' An-xn Matrix.carrier-vec-conjugate add-carrier-vec add-scalar-prod-distrib mult-mat-vec-carrier smult-carrier-vec transpose-carrier-mat* $v_n c$*-def vn* $x_n c$*-def xn*)
moreover have $E = QF\ A_n\ x_n + b\text{-}conj * (x_n \cdot v_n c)$
proof−
have $E = ((A_n *_v x_n) \cdot x_n c) + ((A_n *_v x_n) \cdot (b\text{-}conj \cdot_v (A_n{}^{\prime T} *_v v_n c)))$
unfolding *E-def*
by (*metis An' An-xn Matrix.carrier-vec-conjugate mult-mat-vec-carrier scalar-prod-add-distrib smult-carrier-vec transpose-carrier-mat* $v_n c$*-def vn* $x_n c$*-def xn*)

moreover have $((A_n *_v x_n) \cdot x_n c) = QF\ A_n\ x_n$ **by** (*simp add*: $x_n c$-*def*)
moreover have $((A_n *_v x_n) \cdot (b\text{-}conj \cdot_v ({A_n}^{T} *_v v_n c))) = b\text{-}conj * (x_n \cdot v_n c)$
(**is** *?lhs* = -)
proof−
have *?lhs* = $b\text{-}conj * ((A_n *_v x_n) \cdot ({A_n}^{T} *_v v_n c))$ **using** *An An'* **by** *auto*
also have ... = $b\text{-}conj * ((A_n *_v x_n) \cdot (conjugate\ {A_n}' *_v v_n c))$
using A_n-*inv-conj* **by** *presburger*
also have ... = $b\text{-}conj * (((A_n' * A_n) *_v x_n) \cdot v_n c)$
by (*smt* (*verit*) *An An' conj-mat-vec-mult hermitian hermitian-def inner-prod-mult-mat-vec-right* $v_n c$-*def vn xn*)
also have ... = $b\text{-}conj * (x_n \cdot v_n c)$
by (*metis An' carrier-matD*(*1*) *inverts-mat-def one-mult-mat-vec xn*)
finally show *?thesis* **.**
qed
ultimately show *?thesis* **by** *argo*
qed
moreover have $F = b * (x_n c \cdot v_n) + (cmod\ b)\hat{}2 * ((A_n' *_v v_n) \cdot v_n c)$
proof−
have $F = (b \cdot_v v_n) \cdot (x_n c) + (b \cdot_v v_n) \cdot (b\text{-}conj \cdot_v ({A_n}^{T} *_v v_n c))$
unfolding *F-def*
by (*metis An' Matrix.carrier-vec-conjugate b-vn carrier-matD*(*2*) *carrier-vec-dim-vec dim-mult-mat-vec index-smult-vec*(*2*) *index-transpose-mat*(*2*) *scalar-prod-add-distrib* $x_n c$-*def xn*)
moreover have $(b \cdot_v v_n) \cdot (x_n c) = b * (v_n \cdot x_n c)$ **using** *vn* $x_n c$-*def xn* **by** *auto*
moreover have $(b \cdot_v v_n) \cdot (b\text{-}conj \cdot_v ({A_n}^{T} *_v v_n c)) = (cmod\ b)\hat{}2 * (({A_n}' *_v v_n) \cdot v_n c)$
(**is** *?lhs* = -)
proof−
have *?lhs* = $(cmod\ b)\hat{}2 * (v_n \cdot ({A_n}^{T} *_v v_n c))$
using *An' vn b-conj-def complex-norm-square* **by** *force*
also have ... = $(cmod\ b)\hat{}2 * (({A_n}' *_v v_n) \cdot v_n c)$
by (*metis* A_n-*inv-conj An' adjoint-def-alter conj-mat-vec-mult hermitian hermitian-def* $v_n c$-*def vn*)
finally show *?thesis* **.**
qed
ultimately show *?thesis* **by** (*metis conjugate-vec-sprod-comm vn* $x_n c$-*def xn*)
qed
ultimately show *?thesis* **by** *fastforce*
qed

let $?c_n = b \cdot_v ({A_n}' *_v v_n)$
have *cn*: $?c_n \in carrier\text{-}vec\ n$
by (*metis An' An-xn* ‹*invertible-mat* A_n› *carrier-vecD carrier-vec-dim-vec dim-mult-mat-vec index-mult-mat*(*3*) *index-one-mat*(*3*) *invertible-mat-def inverts-mat-def smult-carrier-vec square-mat.simps*)

have $A \in carrier\text{-}mat\ (Suc\ n)\ (Suc\ n)$

by (*simp add*: *Suc.prems*(*1*))
moreover have $x \in$ *carrier-vec* (*Suc n*)
by (*simp add*: *Suc.prems*(*2*))
ultimately have *Ax*: $A *_v x = (A_n *_v x_n + b \cdot_v v_n) @_v (vec\ 1\ (\lambda i.\ (w_n \cdot x_n) + a * b))$
(**is** - = - $@_v$ *?Ax-last*)
using *mat-vec-prod-leading-principal-submatrix*
unfolding A_n-*def a-def b-def* v_n-*def* w_n-*def* x_n-*def* **by** *blast*

hence *QF A x* = ... $\cdot c\ x$ **by** *force*
also have ... = $((A_n *_v x_n + b \cdot_v v_n) \cdot c\ x_n) + ((w_n \cdot x_n) + a * b) * b\text{-}conj$
proof−
have $x \in$ *carrier-vec* (*dim-vec* $((A_n *_v x_n + b \cdot_v v_n) @_v$ *?Ax-last*))
using *Suc.prems*(*2*) *vn* **by** *force*
moreover have $(A_n *_v x_n + b \cdot_v v_n) \cdot c\ (vec\text{-}first\ x\ (dim\text{-}vec\ (A_n *_v x_n + b \cdot_v v_n)))$
$= (A_n *_v x_n + b \cdot_v v_n) \cdot c\ x_n$
by (*simp add*: v_n-*def* x_n-*def*)
moreover have *dim-vec ?Ax-last* = *1* **by** *simp*
moreover have *?Ax-last* $\cdot c$ (*vec-last x 1*) = $(w_n \cdot x_n + a * b) * b\text{-}conj$
proof−
have *dim-vec ?Ax-last* = *1* **by** *simp*
moreover have (*vec-last x 1*)$0 = b$
by (*smt* (*verit*) *Suc.prems*(*2*) *add.commute add.right-neutral add-diff-cancel-right*′ *b-def carrier-vecD index-vec plus-1-eq-Suc vec-last-def zero-less-one-class.zero-less-one*)
moreover have *?Ax-last*$0 = (w_n \cdot x_n + a * b)$ **by** *simp*
moreover have *?Ax-last* $\cdot c$ (*vec-last x 1*) = *?Ax-last*$0 * conjugate$ ((*vec-last x 1*)$0)$
unfolding *scalar-prod-def* **by** *force*
ultimately show *?thesis* **using** *b-conj-def* **by** *presburger*
qed
ultimately show *?thesis* **by** (*simp add*: *inner-prod-append*(*2*))
qed
also have ... = $QF\ A_n\ x_n + ((b \cdot_v v_n) \cdot c\ x_n) + ((w_n \cdot x_n) * b\text{-}conj) + (a * b * b\text{-}conj)$
using *inner-prod-distrib-right*[*of* x_n *n* $A_n *_v x_n$ $b \cdot_v v_n$] *b-vn An-xn*
by (*simp add*: *ring-class.ring-distribs*(*2*) *xn*)
also have ... = $QF\ A_n\ x_n + ((b \cdot_v v_n) \cdot c\ x_n) + ((w_n \cdot x_n) * b\text{-}conj) + (a * (cmod\ b)\hat{}2)$
using *b-conj-def complex-norm-square* **by** *auto*
also have ... = $QF\ A_n\ x_n + b * (x_n c \cdot v_n) + b\text{-}conj * (x_n \cdot v_n c) + (a * (cmod\ b)\hat{}2)$
by (*metis conjugate-vec-sprod-comm inner-prod-smult-left mult.commute* $v_n c$-*def vn wn-vn-conj* $x_n c$-*def xn*)
also have ... = $(A_n *_v (x_n + b \cdot_v (A_n{}' *_v v_n))) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^T *_v v_n c))$
$- (cmod\ b)\hat{}2 * ((A_n{}' *_v v_n) \cdot v_n c) + (a * (cmod\ b)\hat{}2)$
using ** **by** *fastforce*
also have ... = $(A_n *_v (x_n + b \cdot_v (A_n{}' *_v v_n))) \cdot (x_n c + b\text{-}conj \cdot_v (A_n{}^T *_v$

```
v_n c))
      + (cmod b)^2 * (a - QF A_n' v_n)
    by (simp add: right-diff-distrib v_n c-def)
  also have ... = QF A_n (x_n + ?c_n) + (cmod b)^2 * (a - QF A_n' v_n)
  proof-
    have conjugate (A_n' *_v v_n) = (conjugate A_n' *_v v_n c)
      by (metis conj-mat-vec-mult adjoint-dim-col carrier-mat-triv carrier-vecD cn
dim-mult-mat-vec hermitian hermitian-def index-smult-vec(2) v_n c-def vn)
    thus ?thesis
      by (smt (verit, ccfv-threshold) A_n-inv-conj b-conj-def cn conjugate-add-vec
conjugate-smult-vec x_n c-def xn quadratic-form-def)
  qed
  finally have eq: QF A x = QF A_n (x_n + ?c_n) + (cmod b)^2 * (a - QF A_n'
v_n) .

  have x_n-c_n: (x_n + ?c_n) ∈ carrier-vec n using add-carrier-vec cn xn by blast
  have sylvester-criterion A_n
    using leading-principle-submatrix-sylvester
    by (metis A_n-def Suc.prems(1) Suc.prems(4) Suc-n-not-le-n linorder-linear)
  hence 1: Re (QF A_n (x_n + ?c_n)) > 0 if x_n + ?c_n ≠ 0_v n
    using Suc.IH[OF carrier-A_n x_n-c_n herm-A_n] that by metis
  have 2: x_n + ?c_n ≠ 0_v n if b = 0
  proof-
    have ?c_n = 0_v n
      by (metis that cn conjugate-square-eq-0-vec inner-prod-smult-left mult-eq-0-iff
smult-smult-assoc)
    hence *: x_n + ?c_n = x_n by (simp add: xn)
    show ?thesis
    proof(rule ccontr)
      assume ¬ x_n + ?c_n ≠ 0_v n
      hence x_n = 0_v n using * by argo
      hence ∀ i < n. x_n$i = 0 by fastforce
      moreover have ∀ i < n. x_n$i = x$i by (simp add: vec-first-def x_n-def)
      moreover have x$n = 0 using that unfolding b-def .
      ultimately have ∀ i < Suc n. x$i = 0 using less-Suc-eq by presburger
      thus False using Suc.prems(2,4,5) by auto
    qed
  qed
  have 3: a - QF A_n' v_n > 0
  proof-
    have det A = det A_n * (a - QF A_n' v_n)
    proof-
      let ?B = mat-of-cols n [v_n]
      let ?C = mat-of-rows n [conjugate v_n]
      let ?D = mat 1 1 (λ-. a)

      have (A_n, ?B, ?C, ?D) = split-block A n n
      proof-
        have A_n = mat n n (($$) A)
```

by (*metis* A_n*-def An An-xn Suc*(2) *carrier-matD*(2) *carrier-vecD dim-col-mat*(1) *dim-mult-mat-vec dim-row-mat*(1) *index-mat*(1) *le-add2 leading-principal-submatrix-index mat-eq-iff plus-1-eq-Suc*)
moreover have *?B* = *mat n* (*dim-col A* − *n*) (λ(*i*, *j*). *A* $$ (*i*, *j* + *n*))
(**is** *?lhs* = *?rhs*)
proof
show *dim-row ?lhs* = *dim-row ?rhs* **by** *simp*
show *dim-col ?lhs* = *dim-col ?rhs* **using** *Suc*(2) **by** *force*
show ⋀*i j*. *i* < *dim-row ?rhs* ⟹ *j* < *dim-col ?rhs* ⟹ *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
proof−
fix *i j* **assume** ∗: *i* < *dim-row ?rhs j* < *dim-col ?rhs*
hence *j* = *0* **using** *Suc*(2) **by** *auto*
thus *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
apply (*simp add*: v_n*-def vec-first-def mat-of-cols-def*)
using ∗(*1*) *Suc*(2) **by** *auto*
qed
qed
moreover have *?C* = *mat* (*dim-row A* − *n*) *n* (λ(*i*, *j*). *A* $$ (*i* + *n*, *j*))
(**is** *?lhs* = *?rhs*)
proof
show *dim-row ?lhs* = *dim-row ?rhs* **using** *Suc*(2) **by** *force*
show *dim-col ?lhs* = *dim-col ?rhs* **by** *simp*
show ⋀*i j*. *i* < *dim-row ?rhs* ⟹ *j* < *dim-col ?rhs* ⟹ *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
proof−
fix *i j* **assume** ∗: *i* < *dim-row ?rhs j* < *dim-col ?rhs*
hence *i* = *0* **using** *Suc*(2) **by** *auto*
moreover have *conjugate* (*vec n* (($) (*col A n*))) = (*vec n* (($) (*row A n*)))
using *Suc*(4)
unfolding *hermitian-def adjoint-def*
by (*metis* v_n*-def* v_n*c-def vec-first-def* w_n*-def wn-vn-conj*)
ultimately show *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
apply (*simp add*: v_n*-def vec-first-def mat-of-cols-def*)
using ∗(*2*) *Suc*(2) **by** (*simp add*: *mat-of-rows-def*)
qed
qed
moreover have *?D* = *mat* (*dim-row A* − *n*) (*dim-col A* − *n*) (λ(*i*, *j*). *A* $$ (*i* + *n*, *j* + *n*))
(**is** *?lhs* = *?rhs*)
proof
show *row*: *dim-row ?lhs* = *dim-row ?rhs* **using** *Suc*(2) **by** *fastforce*
show *col*: *dim-col ?lhs* = *dim-col ?rhs* **using** *Suc*(2) **by** *fastforce*
show ⋀*i j*. *i* < *dim-row ?rhs* ⟹ *j* < *dim-col ?rhs* ⟹ *?lhs*$$(*i*,*j*) = *?rhs*$$(*i*,*j*)
proof−
fix *i j* **assume** ∗: *i* < *dim-row ?rhs j* < *dim-col ?rhs*
hence *i* = *0* ∧ *j* = *0* **using** *Suc*(2) **by** *auto*

thus *?lhs$$(i,j) = ?rhs$$(i,j)*
apply (*simp add*: *a-def*)
using *col row* **by** *force*
qed
qed
ultimately show *?thesis* **unfolding** *split-block-def* **by** *metis*
qed
hence *det A = det* A_n ** det (?D − ?C * A_n' * ?B)*
using *schur-formula*[*of* A_n *?B ?C ?D A n n* A_n'] *An' Suc*(*4*) *herm-*A_n *hermitian-is-square*
by (*metis Suc*(*2*) *carrier-matD*(*1*) *carrier-matD*(*2*) *lessI*)
moreover have *det (?D − ?C * A_n' * ?B) = (a − QF* A_n' v_n)
proof−
have *?C * A_n' * ?B = mat 1 1* (λ-. *QF* A_n' v_n) (**is** *?lhs = ?rhs*)
proof
have *dim*: *?C * A_n' * ?B ∈ carrier-mat 1 1* **by** (*simp add*: *carrier-matI*)
thus *dim-row ?lhs = dim-row ?rhs dim-col ?lhs = dim-col ?rhs* **by** *auto*
have *col* (A_n' * *?B*) *0 =* A_n' $*_v$ v_n
by (*metis An' vn mat-vec-as-mat-mat-mult*)
moreover have *row ?C 0 = conjugate* v_n **using** v_n*c-def* w_n*-def wn-vn-conj* **by** *auto*
moreover have (*?C * (*A_n' * *?B))$$(0,0) = row ?C 0 · col* (A_n' * *?B) 0* **by** *simp*
moreover have *?C * (*A_n' * *?B) = ?C ** A_n' * *?B*
by (*metis An' assoc-mult-mat carrier-matI mat-of-cols-carrier*(*2*) *mat-of-rows-carrier*(*3*))
ultimately have (*?C * *A_n' * *?B)$$(0,0) = (conjugate* v_n*) ·* (A_n' $*_v$ v_n) **by** *argo*
also have *... =* (A_n' $*_v$ v_n) $\cdot_c$ v_n
by (*metis An' conjugate-vec-sprod-comm mult-mat-vec-carrier vn*)
also have *... = QF* A_n' v_n **by** *simp*
finally show ⋀*i j. i < dim-row ?rhs* ⟹ *j < dim-col ?rhs* ⟹ *?lhs$$(i,j) = ?rhs$$(i,j)*
by *fastforce*
qed
hence *?D − ?C * A_n' * ?B = mat 1 1* (λ-. *a − QF* A_n' v_n) **by** *fastforce*
thus *?thesis* **by** (*simp add*: *det-single*)
qed
ultimately show *?thesis* **by** *argo*
qed
moreover have *det A > 0* **using** *Suc.prems*(*1*,*4*) *sylvester-criterion-positive-det* **by** *blast*
moreover have *det* A_n *> 0* **using** *Suc*(*2*,*5*) **unfolding** A_n*-def sylvester-criterion-def* **by** *simp*
ultimately show *?thesis* **by** (*simp add*: *less-complex-def zero-less-mult-iff*)
qed
have *4*: (*cmod b*)^*2 > 0* **if** *b ≠ 0* **using** *that* **by** *force*

have *?case* **if** *b = 0*

```
  proof−
    have Re (QF A_n (x_n + ?c_n)) > 0 using that 1 2 by blast
    thus ?thesis unfolding eq by (simp add: that)
  qed
  moreover have ?case if b ≠ 0
  proof−
    have (cmod b)^2 * (a − QF A_n' v_n) > 0
      using 3 4[OF that] by (simp add: less-le square-nneg-complex)
    moreover have Re (QF A_n (x_n + ?c_n)) ≥ 0 using 1 carrier-A_n by fastforce
    ultimately show ?thesis unfolding eq by (simp add: less-complex-def)
  qed
  ultimately show ?case by blast
qed
```

9.2 Reverse Implication

```
lemma prod-list-gz:
  fixes l :: real list
  assumes ∀ x ∈ set l. x > 0
  shows prod-list l > 0
  using assms apply (induct l)
   apply fastforce
  by auto

lemma sylvester-criterion-reverse:
  fixes A :: complex mat
  assumes A ∈ carrier-mat n n
  assumes hermitian A
  assumes positive-definite A
  shows sylvester-criterion A
  unfolding sylvester-criterion-def
proof
  fix k assume k: k ∈ {0..dim-row A}
  let ?A' = lps A k
  have pd: positive-definite ?A'
    using assms(1,3) leading-principal-submatrix-positive-definite k by auto
  hence det-nz: det ?A' ≠ 0 using positive-definite-det-nz by blast
   have square: square-mat ?A' using pd hermitian-is-square positive-definite-def
by blast
  have A'-dim: ?A' ∈ carrier-mat k k
    using assms(1) k leading-principal-submatrix-carrier by auto

  have ∀ e ∈ set (map Re (eigvals ?A')). e > 0
  proof
    fix e assume e ∈ set (map Re (eigvals ?A'))
    then obtain e' where e': e' ∈ set (eigvals ?A') ∧ e = Re e'
      by auto
    moreover have e' > 0
    proof−
```

have $e' \in$ *spectrum ?A'*
by (*metis e' Projective-Measurements.spectrum-def Spectral-Radius.spectrum-def hermitian-square mem-Collect-eq pd positive-definite-def spectrum-eigenvalues*)
then obtain *x* **where** *x*: $x \in$ *carrier-vec k* $\wedge$ $x \neq 0_v$ *k* $\wedge$ *?A'* $*_v$ *x* $= e' \cdot_v x$
unfolding *spectrum-def eigenvalue-def eigenvector-def* **using** *A'-dim* **by** *auto*
hence $e' * (x \cdot_c x) > 0$ **using** *pd A'-dim* **unfolding** *positive-definite-def* **by** *fastforce*
moreover have $x \cdot_c x > 0$ **using** *conjugate-square-greater-0-vec x* **by** *blast*
ultimately show *?thesis* **by** (*simp add*: *less-complex-def zero-less-mult-iff*)
qed
ultimately show $e > 0$ **by** (*simp add*: *less-complex-def*)
qed
hence *prod-list* (*map Re* (*eigvals ?A'*)) > 0
using *prod-list-gz* **by** *blast*
moreover have *prod-list* (*eigvals ?A'*) = *prod-list* (*map Re* (*eigvals ?A'*))
proof−
have $\forall i <$ (*length* (*eigvals ?A'*)). (*eigvals ?A'*)!*i* = (*map Re* (*eigvals ?A'*))!*i*
proof *safe*
fix *i* **assume** $*$: $i <$ *length* (*eigvals ?A'*)
hence (*eigvals ?A'*)!*i* $\in$ *Reals*
by (*metis eigenvalue-root-char-poly eigvals-poly-length hermitian-eigenvalues-real hermitian-square linear-poly-root nth-mem pd positive-definite-def*)
thus (*eigvals ?A'*)!*i* = (*map Re* (*eigvals ?A'*))!*i* **using** $*$ **by** *auto*
qed
thus *?thesis*
by (*metis length-map map-nth-eq-conv of-real-hom.hom-prod-list*)
qed
ultimately show $0 <$ *det ?A'*
using *det-is-prod-of-eigenvalues*[*OF square*] **by** (*simp add*: *less-complex-def*)
qed

9.3 Theorem Statement

theorem *sylvester-criterion*:
fixes *A* :: *complex mat*
assumes *A* $\in$ *carrier-mat n n*
assumes *hermitian A*
shows *sylvester-criterion A* $\longleftrightarrow$ *positive-definite A*
proof
show *1*: *sylvester-criterion A* $\Longrightarrow$ *positive-definite A*
unfolding *positive-definite-def*
using *sylvester-criterion-forward*[*of A n*] *assms complex-is-Real-iff hermitian-quadratic-form-real less-complex-def*
by *simp*
show *2*: *positive-definite A* $\Longrightarrow$ *sylvester-criterion A*
using *sylvester-criterion-reverse*[*OF assms*(*1*,*2*)] **.**
qed

end
theory *Cauchy-Eigenvalue-Interlacing*
imports *Misc-Matrix-Results*

begin

10 Rayleigh Quotient Lemmas

definition *rayleigh-quotient-complex* (ϱ_c) **where**
$\varrho_c\ M\ x = (QF\ M\ x)\ /\ (x \cdot_c x)$

definition *rayleigh-quotient* (ϱ) **where**
$\varrho\ M\ x = Re\ (\varrho_c\ M\ x)$

declare
rayleigh-quotient-complex-def[*simp*]
rayleigh-quotient-def[*simp*]

lemma *rayleigh-quotient-negative*: $A \in$ *carrier-mat n n* $\Longrightarrow x \in$ *carrier-vec n* $\Longrightarrow$
$\varrho\ A\ x = -\ \varrho\ (-\ A)\ x$
by *auto*

lemma *rayleigh-quotient-complex-scale*:
fixes k :: *real*
assumes $A \in$ *carrier-mat n n*
assumes $v \in$ *carrier-vec n*
assumes $k \neq 0$
shows $\varrho_c\ A\ v = \varrho_c\ A\ (k \cdot_v v)$
proof−
have $\varrho_c\ A\ v = (k\hat{}2 * ((A *_v v) \cdot_c v))\ /\ (k\hat{}2 * (v \cdot_c v))$ **using** *assms*(*3*) **by** *simp*
also have ... $= (((k \cdot_v (A *_v v)) \cdot_c (k \cdot_v v)))\ /\ (k\hat{}2 * (v \cdot_c v))$
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *assms*(*2*) *more-arith-simps*(*11*) *mult-mat-vec-carrier power2-eq-square scalar-prod-smult-distrib smult-carrier-vec smult-scalar-prod-distrib vec-conjugate-real*)
also have ... $= (((k \cdot_v (A *_v v)) \cdot_c (k \cdot_v v)))\ /\ (((k \cdot_v v) \cdot_c (k \cdot_v v)))$
by (*simp add*: *power2-eq-square*)
also have ... $= ((((A *_v (k \cdot_v v))) \cdot_c (k \cdot_v v)))\ /\ (((k \cdot_v v) \cdot_c (k \cdot_v v)))$
by (*metis assms*(*1*) *assms*(*2*) *mult-mat-vec*)
also have ... $= \varrho_c\ A\ (k \cdot_v v)$ **by** *auto*
finally show *?thesis* **.**
qed

lemma *rayleigh-quotient-scale*:
fixes k :: *real*
assumes $A \in$ *carrier-mat n n*
assumes $v \in$ *carrier-vec n*
assumes $k \neq 0$
shows $\varrho\ A\ v = \varrho\ A\ (k \cdot_v v)$

by (*smt* (*verit*, *ccfv-SIG*) *rayleigh-quotient-complex-scale assms Groups.mult-ac*(*2*) *complex-norm-square conjugate-complex-def inner-prod-smult-left-right mult-divide-mult-cancel-left-if mult-mat-vec mult-mat-vec-carrier norm-of-real of-real-eq-0-iff power-eq-0-iff quadratic-form-def rayleigh-quotient-complex-def rayleigh-quotient-def*)

lemma *hermitian-rayleigh-quotient-real*:
fixes A :: *complex mat*
assumes $A \in$ *carrier-mat n n*
assumes $v \in$ *carrier-vec n*
assumes *hermitian A*
assumes $v \neq 0_v\ n$
shows $\varrho_c\ A\ v \in$ *Reals*
proof−
have *QF A v* ∈ *Reals*
using *hermitian-quadratic-form-real assms* **by** *blast*
moreover have *inner-prod v v* ∈ *Reals* **by** (*simp add*: *self-inner-prod-real*)
moreover have *inner-prod v v* ≠ *0* **using** *assms*(*2*,*4*) **by** *fastforce*
ultimately show *?thesis* **unfolding** *rayleigh-quotient-complex-def* **using** *Reals-divide* **by** *blast*
qed

11 Vector Summation Lemmas

lemma *complex-vec-norm-sum*:
fixes x :: *complex vec*
assumes $x \in$ *carrier-vec n*
shows *vec-norm x* = *csqrt* ((∑ *i* ∈ {..<*n*}. (*cmod* (*x*$*i*))^2))
proof−
have ∗: ⋀*i*. *i* ∈ {..<*n*} ⟹ (*conjugate x*)$*i* = *conjugate* (*x*$*i*)
using *assms* **by** *auto*
have ∗∗: ⋀*i*. *i* ∈ {..<*n*} ⟹ (*x*$*i*) ∗ *conjugate* (*x*$*i*) = (*cmod* (*x*$*i*))^2
using *mult-conj-cmod-square* **by** *blast*
have *vec-norm x* = *csqrt* (*x* ·*c x*)
by (*simp add*: *vec-norm-def*)
also have ... = *csqrt* (∑ *i* ∈ {..<*n*}. (*x*$*i*) ∗ (*conjugate x*)$*i*)
by (*metis assms atLeast0LessThan carrier-vecD dim-vec-conjugate scalar-prod-def*)
also have ... = *csqrt* (∑ *i* ∈ {..<*n*}. (*x*$*i*) ∗ *conjugate* (*x*$*i*))
by (*simp add*: ∗)
also have ... = *csqrt* ((∑ *i* ∈ {..<*n*}. (*cmod* (*x*$*i*))^2)) **using** ∗∗ **by** *fastforce*
finally show *?thesis* **.**
qed

lemma *inner-prod-vec-sum*:
assumes $v \in$ *carrier-vec n*
assumes $w \in$ *carrier-vec n*
assumes $B \subseteq$ *carrier-vec n*
assumes *finite B*
assumes v = *finsum-vec TYPE*('*a*::*conjugatable-ring*) *n* (λ*b*. *cs b* ·$_v$ *b*) *B*
shows *inner-prod w v* = (∑ *b* ∈ *B*. *cs b* ∗ *inner-prod w b*)

proof−
let *?vs* = $\lambda b.\ cs\ b \cdot_v b$
let *?f* = $\lambda i\ b.$ *if* $i \in \{..<n\}$ *then* (*?vs b*)$\$i * (conjugate\ w)\$i$ *else 0*
have *f*: $\bigwedge y.$ *finite* $\{x.\ ?f\ x\ y \neq 0\}$
proof−
fix *y*
have $\{x.\ ?f\ x\ y \neq 0\} \subseteq \{..<n\}$ **by** (*simp add*: *subset-eq*)
thus *finite* $\{x.\ ?f\ x\ y \neq 0\}$ **using** *finite-nat-iff-bounded* **by** *blast*
qed
have *vs*: *?vs* $\in B \to$ *carrier-vec n* **using** *assms*(*3*) **by** *force*
have *b-scale*: $\bigwedge i\ b.\ i \in \{..<n\} \Longrightarrow b \in B \Longrightarrow$ (*?vs b*)$\$i = cs\ b * b\$i$
using *assms*(*3*) **by** *auto*
have *assoc*: $\bigwedge i\ b.\ (cs\ b * b\$i) * (conjugate\ w)\$i = cs\ b * (b\$i * (conjugate\ w)\$i)$
using *Groups.mult-ac*(*1*) **by** *blast*

have *inner-prod w v* = $(\sum i \in \{..<n\}.\ v\$i * (conjugate\ w)\$i)$
unfolding *scalar-prod-def* **using** *atLeast0LessThan assms*(*2*) **by** *force*
moreover have $\bigwedge i.\ i \in \{..<n\} \Longrightarrow v\$i = (\sum b \in B.\ (?vs\ b)\$i)$
using *index-finsum-vec*[*OF assms*(*4*) - *vs*] **unfolding** *assms*(*5*) **by** *blast*
ultimately have *inner-prod w v* = $(\sum i \in \{..<n\}.\ (\sum b \in B.\ (?vs\ b)\$i) * (conjugate\ w)\$i)$
by *force*
also have ... = $(\sum i \in \{..<n\}.\ \sum b \in B.\ (?vs\ b)\$i * (conjugate\ w)\$i)$
by (*simp add*: *sum-distrib-right*)
also have ... = $(\sum i \in \{..<n\}.\ \sum b \in B.\ ?f\ i\ b)$
by *fastforce*
also have ... = *Sum-any* $(\lambda i.\ \sum b \in B.\ ?f\ i\ b)$
using *Sum-any.conditionalize*[*of* $\{..<n\}$ $\lambda i.\ (\sum b \in B.\ ?f\ i\ b)$]
by (*smt* (*verit*, *ccfv-SIG*) *Sum-any.cong finite-nat-iff-bounded subset-eq sum.neutral*)
also have ... = $(\sum b \in B.$ (*Sum-any* $(\lambda i.\ ?f\ i\ b)))$
using *Sum-any-sum-swap*[*OF assms*(*4*) *f*, *of* $\lambda x.\ x$] .
also have ... = $(\sum b \in B.\ (\sum i \in \{..<n\}.\ (?vs\ b)\$i * (conjugate\ w)\$i))$
proof−
have $\bigwedge b.\ b \in B \Longrightarrow (\sum i \in \{..<n\}.\ (?vs\ b)\$i * (conjugate\ w)\$i) =$ *Sum-any* $(\lambda i.\ ?f\ i\ b)$
using *Sum-any.conditionalize*[*of* $\{..<n\}$] **by** *blast*
thus *?thesis* **by** *fastforce*
qed
also have ... = $(\sum b \in B.\ (\sum i \in \{..<n\}.\ (cs\ b * b\$i) * (conjugate\ w)\$i))$
using *b-scale* **by** *simp*
also have ... = $(\sum b \in B.\ (\sum i \in \{..<n\}.\ cs\ b * (b\$i * (conjugate\ w)\$i)))$
using *assoc* **by** *force*
also have ... = $(\sum b \in B.\ cs\ b * (\sum i \in \{..<n\}.\ b\$i * (conjugate\ w)\$i))$
by (*simp add*: *sum-distrib-left*)
also have ... = $(\sum b \in B.\ cs\ b *$ *inner-prod w b*)
unfolding *scalar-prod-def* **using** *atLeast0LessThan assms*(*2*) **by** *force*
finally show *?thesis* .
qed

lemma *sprod-vec-sum*:
assumes $v \in$ *carrier-vec n*
assumes $w \in$ *carrier-vec n*
assumes $B \subseteq$ *carrier-vec n*
assumes *finite B*
assumes $v =$ *finsum-vec TYPE('a::{comm-ring}) n* $(\lambda b.\ cs\ b \cdot_v b)\ B$
shows $w \cdot v = (\sum b \in B.\ cs\ b * (w \cdot b))$
proof−
let *?vs* $= \lambda b.\ cs\ b \cdot_v b$
let *?f* $= \lambda i\ b.$ *if* $i \in \{..<n\}$ *then* $(?vs\ b)\$i * w\$i$ *else 0*
have *f*: $\bigwedge y.$ *finite* $\{x.\ ?f\ x\ y \neq 0\}$
proof−
fix *y*
have $\{x.\ ?f\ x\ y \neq 0\} \subseteq \{..<n\}$ **by** (*simp add*: *subset-eq*)
thus *finite* $\{x.\ ?f\ x\ y \neq 0\}$ **using** *finite-nat-iff-bounded* **by** *blast*
qed

have *vs*: $?vs \in B \to$ *carrier-vec n* **using** *assms*(*3*) **by** *force*
have *b-scale*: $\bigwedge i\ b.\ i \in \{..<n\} \Longrightarrow b \in B \Longrightarrow (?vs\ b)\$i = cs\ b * b\$i$
using *assms*(*3*) **by** *auto*
have *assoc*: $\bigwedge i\ b.\ (cs\ b * b\$i) * w\$i = cs\ b * (b\$i * w\$i)$
using *Groups.mult-ac*(*1*) **by** *blast*
have *B-dim*: $\bigwedge b.\ b \in B \Longrightarrow$ *dim-vec* $b = n$
using *assms*(*3*) **by** *fastforce*

have $w \cdot v = (\sum i \in \{..<n\}.\ v\$i * w\$i)$
unfolding *scalar-prod-def* **using** *atLeast0LessThan*[*of n*] *assms*(*1*)
by (*metis* (*no-types*, *lifting*) *Groups.mult-ac*(*2*) *carrier-vecD sum.cong*)
moreover have $\bigwedge i.\ i \in \{..<n\} \Longrightarrow v\$i = (\sum b \in B.\ (?vs\ b)\$i)$
using *index-finsum-vec*[*OF assms*(*4*) - *vs*] **unfolding** *assms*(*5*) **by** *blast*
ultimately have $w \cdot v = (\sum i \in \{..<n\}.\ (\sum b \in B.\ (?vs\ b)\$i) * w\$i)$
by *force*
also have $... = (\sum i \in \{..<n\}.\ \sum b \in B.\ (?vs\ b)\$i * w\$i)$
by (*simp add*: *sum-distrib-right*)
also have $... = (\sum i \in \{..<n\}.\ \sum b \in B.\ ?f\ i\ b)$
by *fastforce*
also have $... =$ *Sum-any* $(\lambda i.\ \sum b \in B.\ ?f\ i\ b)$
using *Sum-any.conditionalize*[*of* $\{..<n\}$ $\lambda i.\ (\sum b \in B.\ ?f\ i\ b)$]
by (*smt* (*verit*, *ccfv-SIG*) *Sum-any.cong finite-nat-iff-bounded subset-eq sum.neutral*)
also have $... = (\sum b \in B.$ (*Sum-any* $(\lambda i.\ ?f\ i\ b)))$
using *Sum-any-sum-swap*[*OF assms*(*4*) *f*, *of* $\lambda x.\ x$] .
also have $... = (\sum b \in B.\ (\sum i \in \{..<n\}.\ (?vs\ b)\$i * w\$i))$
proof−
have $\bigwedge b.\ b \in B \Longrightarrow (\sum i \in \{..<n\}.\ (?vs\ b)\$i * w\$i) =$ *Sum-any* $(\lambda i.\ ?f\ i\ b)$
using *Sum-any.conditionalize*[*of* $\{..<n\}$] **by** *blast*
thus *?thesis* **by** *fastforce*
qed
also have $... = (\sum b \in B.\ (\sum i \in \{..<n\}.\ (cs\ b * b\$i) * w\$i))$
using *b-scale* **by** *simp*

also have ... = $(\sum b \in B.\ (\sum i \in \{..<n\}.\ cs\ b * (b\$i * w\$i)))$
using *assoc* **by** *force*
also have ... = $(\sum b \in B.\ cs\ b * (\sum i \in \{..<n\}.\ b\$i * w\$i))$
by (*simp add*: *sum-distrib-left*)
also have ... = $(\sum b \in B.\ cs\ b * (\sum i \in \{..<n\}.\ w\$i * b\$i))$
by (*metis* (*no-types*, *lifting*) *Groups.mult-ac*(2) *sum.cong*)
also have ... = $(\sum b \in B.\ cs\ b * (w \cdot b))$
unfolding *scalar-prod-def* **using** *atLeast0LessThan assms*(3) *B-dim* **by** *fast-force*
finally show *?thesis* **.**
qed

lemma *mat-vec-mult-sum*:
assumes $v \in$ *carrier-vec n*
assumes $A \in$ *carrier-mat n n*
assumes $B \subseteq$ *carrier-vec n*
assumes *finite B*
assumes $v =$ *finsum-vec TYPE*('a::*comm-ring*) *n* $(\lambda b.\ cs\ b \cdot_v b)$ *B*
shows $A *_v v =$ *finsum-vec TYPE*('a::*comm-ring*) *n* $(\lambda b.\ cs\ b \cdot_v (A *_v b))$ *B*
(**is** *?lhs* = *?rhs*)
proof−
have $\bigwedge i.\ i < n \Longrightarrow$ *?lhs*\$*i* = *?rhs*\$*i*
proof−
fix *i* **assume** ∗: $i < n$
let *?r* = *row A i*
have *?lhs*\$*i* = *?r* · *v* **using** ∗ *assms*(2) **unfolding** *mult-mat-vec-def* **by** *simp*
also have ... = $(\sum b \in B.\ (cs\ b * (?r \cdot b)))$
using *sprod-vec-sum*[*OF assms*(1) - *assms*(3) *assms*(4) *assms*(5)] *assms*(2)
by *fastforce*
also have ... = $(\sum b \in B.\ (cs\ b * ((A *_v b)\$i)))$
using ∗ *assms*(2) **unfolding** *mult-mat-vec-def* **by** *simp*
also have ... = $(\sum b \in B.\ ((cs\ b \cdot_v (A *_v b))\$i))$
using *assms*(2) ∗ **by** *force*
also have ... = (*finsum-vec TYPE*('a::*comm-ring*) *n* $(\lambda b.\ cs\ b \cdot_v (A *_v b))$ *B*)\$*i*
using *index-finsum-vec*[*OF assms*(4) ∗]
by (*smt* (*verit*, *best*) *Pi-I assms*(2) *carrier-matD*(1) *carrier-vec-dim-vec dim-mult-mat-vec index-smult-vec*(2) *sum.cong*)
finally show *?lhs*\$*i* = *?rhs*\$*i* **by** *blast*
qed
moreover have *?lhs* ∈ *carrier-vec n* **using** *assms*(1) *assms*(2) **by** *force*
moreover have *?rhs* ∈ *carrier-vec n*
by (*smt* (*verit*, *ccfv-SIG*) *Pi-iff assms*(2) *assms*(3) *finsum-vec-closed mult-mat-vec-carrier smult-carrier-vec subsetD*)
ultimately show *?thesis* **by** (*metis* (*no-types*, *lifting*) *carrier-vecD eq-vecI*)
qed

12 Module Span Lemmas

context *module*

begin

lemma *mk-coeffs-of-list*:
assumes $\alpha \in (set\ A \to carrier\ R)$
shows $\exists\, c \in \{0..<length\ A\} \to carrier\ R.\ \forall\, v \in set\ A.\ mk\text{-}coeff\ A\ c\ v = \alpha\ v$
using *assms*
proof(*induct length A arbitrary*: *A*)
case *0*
thus *?case* **by** *fastforce*
next
case (*Suc n*)
then obtain *a A'* **where** *a*: $A = A' @ [a]$ **by** (*metis length-Suc-conv-rev*)
hence $\alpha \in (set\ A' \to carrier\ R)$ **using** *Suc.prems* **by** *simp*
moreover from *a* **have** *len-A'*: $n = length\ A'$ **using** *Suc*(*2*) **by** *auto*
ultimately obtain *c'* **where** *c'*:
$c' \in \{0..<length\ A'\} \to carrier\ R \wedge (\forall\, v \in set\ A'.\ mk\text{-}coeff\ A'\ c'\ v = \alpha\ v)$
using *Suc.hyps* **by** *blast*

have *len-A'-A*: $length\ A' = length\ A - 1$ **using** *Suc*(*2*) *len-A'* **by** *presburger*
moreover have $[0..<length\ A] = [0..<length\ A - 1] @ [length\ A - 1]$
by (*metis Suc*(*2*) *calculation len-A' upt-Suc-append zero-order*(*1*))
ultimately have *A-A'-int*: $[0..<length\ A] = [0..<length\ A'] @ [length\ A']$ **by** *presburger*
show *?case*
proof(*cases a* $\in$ *set A'*)
case *True*
hence *A-A'*: *set A* = *set A'* **using** *a* **by** *auto*
define *c* **where** $c \equiv (\lambda i.\ if\ i \in \{0..<length\ A'\}\ then\ c'\ i\ else\ \mathbf{0})$
hence *c-carrier*: $c \in \{0..<length\ A\} \to carrier\ R$ **using** *c'* **by** *force*
moreover have $\forall\, v \in set\ A.\ mk\text{-}coeff\ A\ c\ v = \alpha\ v$
proof
fix *v* **assume** $*$: $v \in set\ A$
show *mk-coeff A c v* = $\alpha\ v$
proof(*cases v* = *a*)
case *True*
hence *find-indices v A* = (*find-indices v A'*) @ [*length A* − *1*]
proof−
from *A-A'-int* **have** *find-indices v A*
= (*filter* ($\lambda i.\ A\ !\ i = v$) $[0..<length\ A']$) @ (*filter* ($\lambda i.\ A\ !\ i = v$) [*length A'*])
unfolding *find-indices-def* **by** (*metis Suc.hyps*(*2*) *filter-append len-A'*)
moreover then have (*filter* ($\lambda i.\ A\ !\ i = v$) $[0..<length\ A']$) = (*find-indices v A'*)
using *a* **by** *auto*
moreover have *filter* ($\lambda i.\ A\ !\ i = v$) [*length A'*] = [*length A'*] **by** (*simp add*: *True a*)
ultimately show *?thesis* **using** *len-A'-A* **by** *argo*
qed
hence *map c* (*find-indices v A*) = (*map c* (*find-indices v A'*)) @ [*c* (*length*

A − 1)]
by *auto*
hence *foldr* (⊕) (*map c* (*find-indices v A*)) **0**
= *foldr* (⊕) (*map c* (*find-indices v A′*)) (**0** ⊕ *c* (*length A − 1*))
by (*simp add*: *c-def len-A′-A*)
also have ... = (*foldr* (⊕) (*map c* (*find-indices v A′*)) **0**) ⊕ *c* (*length A − 1*)
by (*metis R.sumlist-def R.zero-closed c-carrier atLeastLessThan-iff c-def calculation cring-simprules*(16) *len-A′-A less-irrefl-nat mk-coeff-carrier mk-coeff-def*)
finally have *mk-coeff A c v* = *mk-coeff A′ c v* ⊕ *c* (*length A − 1*)
unfolding *mk-coeff-def R.sumlist-def* **.**
moreover have *c* (*length A − 1*) = **0** **unfolding** *c-def* **using** *len-A′ a* **by** *force*
moreover have *mk-coeff A′ c v* ∈ *carrier R*
by (*smt* (*verit*) *Pi-iff c′ c-def mk-coeff-carrier*)
ultimately have *mk-coeff A c v* = *mk-coeff A′ c v* **by** *algebra*
moreover have *mk-coeff A′ c v* = *mk-coeff A′ c′ v*
unfolding *mk-coeff-def find-indices-def*
by (*metis* (*mono-tags*, *lifting*) *c-def list.map-cong mem-Collect-eq set-filter set-upt*)
ultimately show *?thesis* **by** (*metis* ∗ *A-A′ c′*)
next
case *v-neq-a*: *False*
hence *find-indices v A* = *find-indices v A′*
proof−
from *A-A′-int* **have** *find-indices v A*
= (*filter* (λ*i*. *A* ! *i* = *v*) [*0*..<*length A′*]) @ (*filter* (λ*i*. *A* ! *i* = *v*) [*length A′*])
unfolding *find-indices-def* **by** (*metis Suc.hyps*(2) *filter-append len-A′*)
moreover have (*filter* (λ*i*. *A* ! *i* = *v*) [*0*..<*length A′*]) = *find-indices v A′*
unfolding *find-indices-def*
by (*smt* (*verit*, *ccfv-SIG*) *a append.right-neutral calculation filter.simps*(1) *filter.simps*(2) *filter-cong find-indices-def find-indices-snoc nth-append-length v-neq-a*)
moreover have (*filter* (λ*i*. *A* ! *i* = *v*) [*length A′*]) = [] **using** *a v-neq-a* **by** *auto*
ultimately show *?thesis* **by** *force*
qed
hence *mk-coeff A c v* = *mk-coeff A′ c v* **unfolding** *mk-coeff-def* **by** *fastforce*
also have ... = *mk-coeff A′ c′ v*
unfolding *mk-coeff-def find-indices-def c-def*
by (*smt* (*verit*, *best*) *atLeastLessThan-upt list.map-cong mem-Collect-eq set-filter*)
also have ... = α *v* **using** *c′* ∗ *A-A′* **by** *blast*
finally show *?thesis* **.**
qed
qed
ultimately show *?thesis* **by** *blast*
next
case *False*
hence *A-A′*: *set A′* = *set A* − {*a*} **by** (*simp add*: *a*)

define c **where** $c \equiv (\lambda i.\ \mathit{if}\ i \in \{0..<\mathit{length}\ A'\}\ \mathit{then}\ c'\ i\ \mathit{else}\ \alpha\ a)$
hence $c \in \{0..<\mathit{length}\ A\} \to \mathit{carrier}\ R$ **using** *Suc.prems a c'* **by** *fastforce*
moreover have $\forall\, v \in \mathit{set}\ A.\ \mathit{mk\text{-}coeff}\ A\ c\ v = \alpha\ v$
proof
fix v **assume** $*$: $v \in \mathit{set}\ A$
show *mk-coeff A c v* $= \alpha\ v$
proof(*cases* $v = a$)
case *v-eq-a*: *True*
hence *filter* $(\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A] = [\mathit{length}\ A - 1]$
proof−
from *A-A'-int* **have** *filter* $(\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A]$
$= (\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A'])\ @\ (\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [\mathit{length}\ A'])$
by (*metis Suc.hyps*(2) *filter-append len-A'*)
moreover have $(\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A']) = []$
proof−
have $\bigwedge i.\ i < \mathit{length}\ A' \Longrightarrow A!i \neq v$ **by** (*metis False a nth-append nth-mem v-eq-a*)
thus *?thesis* **by** *fastforce*
qed
moreover have $(\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [\mathit{length}\ A']) = [\mathit{length}\ A']$ **by** (*simp add*: *a v-eq-a*)
ultimately show *?thesis* **by** (*metis Suc.hyps*(2) *diff-Suc-1 len-A' self-append-conv2*)
qed
hence *map c* $(\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A]) = [c\ (\mathit{length}\ A')]$
by (*metis len-A' Suc.hyps*(2) *diff-Suc-1 list.map*(1) *list.map*(2))
moreover have $c\ (\mathit{length}\ A') = \alpha\ v$ **by** (*simp add*: *v-eq-a c-def*)
ultimately have *mk-coeff A c v* $= \alpha\ v \oplus \mathbf{0}$ **unfolding** *mk-coeff-def find-indices-def* **by** *force*
moreover have $\alpha\ v \in \mathit{carrier}\ R$ **using** $*$ *Suc*(3) **by** *blast*
ultimately show *?thesis* **by** *auto*
next
case *v-neq-a*: *False*
hence *find-indices v A* = *find-indices v A'*
proof−
from *A-A'-int* **have** *find-indices v A*
$= (\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A'])\ @\ (\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [\mathit{length}\ A'])$
unfolding *find-indices-def* **by** (*metis Suc.hyps*(2) *filter-append len-A'*)
moreover have $(\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [0..<\mathit{length}\ A']) = \mathit{find\text{-}indices}\ v\ A'$
unfolding *find-indices-def*
by (*smt* (*verit*, *ccfv-SIG*) *a append.right-neutral calculation filter.simps*(1) *filter.simps*(2) *filter-cong find-indices-def find-indices-snoc nth-append-length v-neq-a*)
moreover have $(\mathit{filter}\ (\lambda i.\ A\ !\ i = v)\ [\mathit{length}\ A']) = []$ **using** *a v-neq-a* **by** *auto*
ultimately show *?thesis* **by** *force*
qed
hence *mk-coeff A c v* = *mk-coeff A' c v* **unfolding** *mk-coeff-def* **by** *fastforce*

```
      also have ... = mk-coeff A' c' v
        unfolding mk-coeff-def find-indices-def c-def
         by (smt (verit, best) atLeastLessThan-upt list.map-cong mem-Collect-eq
set-filter)
      also have ... = α v by (metis c' ∗ a insert-iff v-neq-a vec-space.append-insert)
       finally show ?thesis .
     qed
   qed
   ultimately show ?thesis by blast
  qed
qed

lemma span-list-span:
  assumes set A ⊆ carrier M
  shows span-list A = span (set A)
proof−
  have span-list A ⊆ span (set A)
  proof(rule subsetI)
    fix x assume x ∈ span-list A
   then obtain c where c: x = lincomb-list c A ∧ c ∈ {0..<length A} → carrier
R
      unfolding span-list-def by blast
    hence 1: lincomb-list c A = lincomb (mk-coeff A c) (set A)
      using lincomb-list-as-lincomb[OF assms(1)] by presburger
    have mk-coeff A c ∈ (set A) → carrier R by (simp add: c mk-coeff-carrier)
     hence 2: lincomb (mk-coeff A c) (set A) ∈ span (set A) unfolding span-def
by blast
    show x ∈ span (set A) using 1 2 c by argo
  qed
  moreover have span (set A) ⊆ span-list A
  proof(rule subsetI)
    fix x assume ∗: x ∈ span (set A)
    then obtain α where α: x = lincomb α (set A) ∧ α ∈ (set A → carrier R)
      using ∗ finite-span assms by auto
    define α' where α' = (λv. if v ∈ set A then α v else 0)
    hence α-α': ⋀v. v ∈ set A ⟹ α v = α' v by presburger
    hence x-α': x = lincomb α' (set A)
      using α
      unfolding lincomb-def
       by (smt (verit) M.add.finprod-cong' Pi-iff assms basic-trans-rules(31) car-
rier-is-submodule submoduleE(4))
    have 1: α' ∈ (set A → carrier R) by (simp add: Pi-cong α α'-def)
     then obtain c where c: c ∈ {0..<length A} → carrier R ∧ (∀ v ∈ set A.
mk-coeff A c v = α' v)
      using mk-coeffs-of-list by blast
    have mk-coeff A c = α' unfolding α'-def by (metis mk-coeff-0 c α-α')
    hence lincomb α' (set A) = lincomb-list c A
       using lincomb-list-as-lincomb[OF assms(1), of c] c by argo
    also have ... ∈ span-list A using c in-span-listI by blast
```

finally show $x \in$ *span-list* A **using** x-α' **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

end

13 Module Homomorphism Linear Combination and Span Lemmas

context *mod-hom*
begin

lemma *lincomb-list-distrib*:
assumes *set* $S \subseteq$ *carrier* M
assumes $\alpha \in \{..<$*length* $S\} \to$ *carrier* R
shows f (M.*lincomb-list* α S) = N.*lincomb-list* α (*map* f S)
using *assms*
proof(*induct length S arbitrary*: S α)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then obtain v S' **where** v: $S = v \# S'$ **by** (*metis length-Suc-conv*)
have *1*: $n =$ *length* S' **using** *Suc*(*2*) v **by** *auto*
have *2*: *set* $S' \subseteq$ *carrier* M **using** *Suc*(*3*) v **by** *auto*
have *3*: ($\alpha \circ$ *Suc*) $\in \{..<$*length* $S'\} \to$ *carrier* R **using** *1 Suc*(*4*) *Suc.hyps*(*2*) **by** *fastforce*

have *ih*: f (M.*lincomb-list* ($\alpha \circ$ *Suc*) S') = N.*lincomb-list* ($\alpha \circ$ *Suc*) (*map* f S')
using *Suc.hyps*(*1*)[*OF 1 2 3*] **.**

have $*$: M.*lincomb-list* α ($v \# S'$) = (α *0* $\odot_M$ v) $\oplus_M$ (M.*lincomb-list* ($\alpha \circ$ *Suc*) S')
using M.*lincomb-list-Cons* **.**
have $v \in$ *carrier* M **using** *Suc.prems*(*1*) v **by** *force*
moreover have α *0* $\in$ *carrier* R **using** *Suc*(*4*) *Suc.hyps*(*2*) **by** *auto*
ultimately have α *0* $\odot_M$ $v \in$ *carrier* M **by** *blast*
moreover have M.*lincomb-list* ($\alpha \circ$ *Suc*) $S' \in$ *carrier* M
by (*metis* (*no-types*, *lifting*) *1 2 M.lincomb-list-carrier Pi-iff Suc*(*4*) *Suc.hyps*(*2*) *Suc-less-eq atLeastLessThan-iff lessThan-iff o-apply*)
ultimately have f (M.*lincomb-list* α S) = (f (α *0* $\odot_M$ v)) $\oplus_N$ (f (M.*lincomb-list* ($\alpha \circ$ *Suc*) S'))
using *f-hom* $*$ v **unfolding** *module-hom-def* **by** *force*
also have ... = (α *0* $\odot_N$ f v) $\oplus_N$ (f (M.*lincomb-list* ($\alpha \circ$ *Suc*) S'))
by (*simp add*: ‹α *0* $\in$ *carrier* R› ‹$v \in$ *carrier* M›)
also have ... = (α *0* $\odot_N$ f v) $\oplus_N$ (N.*lincomb-list* ($\alpha \circ$ *Suc*) (*map* f S')) **using** *ih* **by** *argo*

also have ... = *N.lincomb-list* α (*map f S*)
using *N.lincomb-list-Cons* **by** (*simp add*: *v*)
finally show *?case* **.**
qed

lemma *lincomb-distrib*:
assumes *inj-on f S*
assumes $S \subseteq$ *carrier M*
assumes $\alpha \in S \to$ *carrier R*
assumes $\forall v \in S.\ \alpha\ v = \beta\ (f\ v)$
assumes *finite S*
shows *f* (*M.lincomb* α *S*) = *N.lincomb* β (*f*‘*S*)
proof−
let *?α′* = $(\lambda v.\ (\alpha\ v) \odot_M v)$
let *?β′* = $(\lambda v.\ (\beta\ v) \odot_N v)$
have ∗: *?α′* ∈ *S* → *carrier M* **using** *assms*(*2*,*3*) **by** *auto*
have *f* (*M.lincomb* α *S*) = *f* (*finsum M ?α′ S*) **using** *M.lincomb-def* **by** *presburger*
also have ... = $(\bigoplus_N a \in S.\ f\ ((\alpha\ a) \odot_M a))$ **using** *hom-sum*[*OF assms*(*2*) ∗] **.**
also have ... = $(\bigoplus_N a \in S.\ (\alpha\ a) \odot_N (f\ a))$
proof−
have $\forall a \in S.\ a \in$ *carrier M* **using** *assms*(*2*) **by** *fastforce*
moreover have $\forall a \in S.\ \alpha\ a \in$ *carrier R* **using** *assms*(*3*) **by** *blast*
ultimately show *?thesis*
using *f-hom* **unfolding** *module-hom-def* **by** (*simp add*: *N.M.add.finprod-cong′*)
qed
also have ... = $(\bigoplus_N a \in S.\ (\beta\ (f\ a)) \odot_N (f\ a))$
by (*smt* (*verit*, *del-insts*) *N.M.add.finprod-cong′ M.summands-valid PiE Pi-I assms*(*2*−*4*) *basic-trans-rules*(*31*) *f-im f-smult*)
also have ... = $(\bigoplus_N a \in (f`S).\ (?\beta'\ a))$
by (*smt* (*verit*, *best*) *assms*(*1*,*4*) *M.summands-valid N.M.add.finprod-cong′ N.M.add.finprod-reindex PiE Pi-I assms*(*2*) *assms*(*3*) *assms*(*4*) *basic-trans-rules*(*31*) *f-im f-smult imageE*)
also have ... = *N.lincomb* β (*f*‘*S*) **using** *N.lincomb-def* **by** *presburger*
finally show *?thesis* **.**
qed

lemma *lincomb-distrib-obtain*:
assumes *inj-on f S*
assumes $S \subseteq$ *carrier M*
assumes $\alpha \in S \to$ *carrier R*
assumes $\forall v \in S.\ \alpha\ v = \beta\ (f\ v)$
assumes *finite S*
obtains β **where** $(\forall v \in S.\ \alpha\ v = \beta\ (f\ v)) \wedge f\ (M.lincomb\ \alpha\ S) = N.lincomb\ \beta\ (f`S)$
proof−
obtain β **where** β: $\forall v \in S.\ \alpha\ v = \beta\ (f\ v)$
proof−
let *?β* = $\lambda y.\ \alpha\ (THE\ x.\ x \in S \wedge f\ x = y)$

```
    have ∀ v ∈ S. α v = ?β (f v)
    proof
      fix v assume v ∈ S
      then have v = (THE x. x ∈ S ∧ f x = f v)
        using assms(1) by (simp add: inj-on-def the-equality)
      thus α v = ?β (f v) by argo
    qed
    thus ?thesis using that by fast
  qed
  thus ?thesis using lincomb-distrib that assms by blast
qed

lemma image-span-list:
  assumes set vs ⊆ carrier M
  shows f`(M.span-list vs) = N.span-list (map f vs) (is ?lhs = ?rhs)
proof−
  have ?lhs ⊆ ?rhs
  proof(rule subsetI)
    fix w assume w ∈ ?lhs
    then obtain v where v: v ∈ M.span-list vs ∧ f v = w by blast
    then obtain α where α: v = M.lincomb-list α vs ∧ α ∈ {..<length vs} →
carrier R
      unfolding M.span-list-def by fastforce
   hence f v = N.lincomb-list α (map f vs) using lincomb-list-distrib[OF assms(1)]
v by blast
    thus w ∈ ?rhs using v α unfolding N.span-list-def by force
  qed
  moreover have ?rhs ⊆ ?lhs
  proof(rule subsetI)
    fix w assume w ∈ ?rhs
    then obtain α where α: w = N.lincomb-list α (map f vs) ∧ α ∈ {..<length
vs} → carrier R
      unfolding N.span-list-def by fastforce
    hence w = f (M.lincomb-list α vs)
      using lincomb-list-distrib[OF assms] by presburger
    thus w ∈ ?lhs using α unfolding M.span-list-def by fastforce
  qed
  ultimately show ?thesis by blast
qed

lemma image-span:
  assumes finite vs
  assumes vs ⊆ carrier M
  shows f`(M.span vs) = N.span (f`vs)
proof−
  obtain vs-list where vs-list: set vs-list = vs using assms(1) finite-list by blast
  have f`vs = set (map f vs-list) using vs-list by simp
  hence N.span (f`vs) = N.span-list (map f vs-list)
    by (metis N.span-list-span M.sum-simp assms(2) f-im image-subset-iff)
```

moreover have *M.span vs = M.span-list vs-list*
using *M.span-list-span vs-list assms(2)* **by** *presburger*
ultimately show *?thesis* **using** *image-span-list assms vs-list* **by** *presburger*
qed

end

14 Linear Map Lemmas

lemma (**in** *linear-map*) *inj-image-lin-indpt*:
assumes *inj-on T* (*carrier V*)
assumes *S* ⊆ *carrier V*
assumes *V.module.lin-indpt S*
assumes *finite S*
shows *W.module.lin-indpt* (*T‘S*)
proof(*rule ccontr*)
assume ¬ *W.module.lin-indpt* (*T‘S*)
then obtain *B β b* **where** *B*: *finite B*
∧ *B* ⊆ *T‘S*
∧ (*β* ∈ (*B* → *carrier K*))
∧ (*lincomb β B* = **0**$_W$)
∧ (*b*∈*B*)
∧ (*β b* ≠ **0**$_K$)
using *W.module.lin-dep-def* **by** *auto*
define *A* **where** *A* ≡ {*a* ∈ *S*. *T a* ∈ *B*}
define *α* **where** *α* ≡ (*λv*. *β* (*T v*))
have *1*: *inj-on T A*
by (*metis* (*no-types*, *lifting*) *assms*(*1*,*2*) *inj-on-subset A-def mem-Collect-eq subsetI*)
have *2*: *A* ⊆ *carrier V* **using** *A-def assms*(*2*) **by** *blast*
have *3*: *α* ∈ *A* → *carrier K*
proof
fix *x* **assume** *x* ∈ *A*
moreover then have *T x* ∈ *carrier W* **using** *2* **by** *blast*
ultimately show *α x* ∈ *carrier K* **unfolding** *α-def* **using** *B A-def* **by** *blast*
qed
have *4*: ∀ *v*∈*A*. *α v* = *β* (*T v*) **using** *α-def* **by** *blast*
have *5*: *finite A* **using** *A-def assms*(*4*) **by** *force*
have *B* = *T‘A* **unfolding** *A-def* **using** *B* **by** *blast*
hence *lincomb β B* = *T* (*V.module.lincomb α A*) **using** *lincomb-distrib*[*OF 1 2 3 4 5*] **by** *argo*
hence *T* (*V.module.lincomb α A*) = **0**$_W$ **using** *B* **by** *argo*
moreover have *T* (**0**$_V$) = **0**$_W$ **by** *auto*
ultimately have ∗: (*V.module.lincomb α A*) = **0**$_V$ **using** *assms*(*1*) **by** (*simp add*: *2 3 inj-onD*)
moreover obtain *a* **where** *T a* = *b* ∧ *a* ∈ *A* **using** *B* ‹*B* = *T* ‘ *A*› **by** *blast*
moreover then have *α a* ≠ **0**$_K$ **by** (*simp add*: *B α-def*)
moreover have *A* ⊆ *S* **using** *A-def* **by** *blast*
ultimately show *False* **using** *assms*(*3*) *5 3 V.module.lin-dep-def* **by** *blast*

qed

lemma *linear-map-mat*:
assumes $A \in$ *carrier-mat n m*
shows *linear-map class-ring* (*module-vec TYPE*('*a*::{*field*,*ring-1*}) *m*) (*module-vec TYPE*('*a*) *n*) ((∗$_v$) *A*)
(**is** *linear-map ?K ?V ?W ?T*)
proof−
have *vectorspace ?K ?V* **using** *VS-Connect.vec-vs*[*of m*] **by** *blast*
moreover have *vectorspace ?K ?W* **using** *VS-Connect.vec-vs*[*of n*] **by** *blast*
moreover have *mod-hom ?K ?V ?W ?T*
proof−
have *V*: *module ?K ?V* **by** (*simp add*: *vec-module*)
moreover have *W*: *module ?K ?W* **by** (*simp add*: *vec-module*)
moreover have *?T* ∈ *LinearCombinations.module-hom ?K ?V ?W*
proof−
have *?T* ∈ *carrier ?V* → *carrier ?W* **by** (*metis Pi-I assms mult-mat-vec-carrier vec-space.cV*)
moreover have ∀ $v_1 \in$ *carrier ?V*. ∀ $v_2 \in$ *carrier ?V*. *?T* (v_1 + v_2) = *?T* v_1 + *?T* v_2
by (*metis module-vec-def assms monoid-record-simps*(*1*) *mult-add-distrib-mat-vec*)
moreover have ∀ $\alpha \in$ *carrier ?K*. ∀ $v \in$ *carrier ?V*. *?T* (α ·$_v$ v) = α ·$_v$ (*?T* v)
by (*metis assms mult-mat-vec vec-space.cV*)
ultimately show *?thesis* **unfolding** *module-vec-def module-hom-def* **by** *force*
qed
ultimately show *?thesis*
unfolding *mod-hom-def mod-hom-axioms-def* **by** *blast*
qed
ultimately show *?thesis* **unfolding** *linear-map-def* **by** *blast*
qed

15 Courant-Fischer Minimax Theorem

We follow the proof given in this set of lecture notes by Dr. David Bindel: https://www.cs.cornell.edu/courses/cs6210/2019fa/lec/2019-11-04.pdf.

definition *sup-defined* :: '*a*::*preorder set* ⇒ *bool* **where**
sup-defined S ⟷ *S* ≠ {} ∧ *bdd-above S*

definition *inf-defined* :: '*a*::*preorder set* ⇒ *bool* **where**
inf-defined S ⟷ *S* ≠ {} ∧ *bdd-below S*

locale *hermitian-mat* = *complex-vec-space n* **for** *n* +
fixes *A* :: *complex mat*
assumes *dim-is*: *A* ∈ *carrier-mat n n*
assumes *is-herm*: *hermitian A*
begin

definition *dimensional* :: *complex vec set* ⇒ *nat* ⇒ *bool* **where**
dimensional $\mathcal{V}$ *k* ⟷ (∃ *vs*. $\mathcal{V}$ = *span vs* ∧ *card vs* = *k* ∧ *vs* ⊆ *carrier-vec n* ∧ *lin-indpt vs*)

lemma *dimensional-n*: *dimensional* $\mathcal{V}$ *k* ⟹ $\mathcal{V}$ ⊆ *carrier-vec n*
using *hermitian-mat.dimensional-def hermitian-mat-axioms* **by** *auto*

lemma *dimensional-n-vec*: ⋀*v*. *v* ∈ $\mathcal{V}$ ⟹ *dimensional* $\mathcal{V}$ *k* ⟹ *v* ∈ *carrier-vec n*
using *dimensional-n* **by** *fast*

Note here that we refer to the Inf and Sup rather than the Min and Max.

definition *rayleigh-min*:
rayleigh-min $\mathcal{V}$ = *Inf* {*ϱ A v* | *v*. *v* ≠ 0_v *n* ∧ *v* ∈ $\mathcal{V}$ ∧ *vec-norm v* = *1*}

definition *rayleigh-max*:
rayleigh-max $\mathcal{V}$ = *Sup* {*ϱ A v* | *v*. *v* ≠ 0_v *n* ∧ *v* ∈ $\mathcal{V}$ ∧ *vec-norm v* = *1*}

definition *maximin* :: *nat* ⇒ *real* **where**
maximin k = *Sup* {*rayleigh-min* $\mathcal{V}$ | $\mathcal{V}$. *dimensional* $\mathcal{V}$ *k*}

definition *minimax* :: *nat* ⇒ *real* **where**
minimax k = *Inf* {*rayleigh-max* $\mathcal{V}$ | $\mathcal{V}$. *dimensional* $\mathcal{V}$ (*n* − *k* + *1*)}

definition *maximin-defined* **where**
maximin-defined k ⟷ *sup-defined* {*rayleigh-min* $\mathcal{V}$ | $\mathcal{V}$. *dimensional* $\mathcal{V}$ *k*}

definition *minimax-defined* **where**
minimax-defined k ⟷ *inf-defined* {*rayleigh-max* $\mathcal{V}$ | $\mathcal{V}$. *dimensional* $\mathcal{V}$ (*n* − *k* + *1*)}

end

locale *courant-fischer* = *hermitian-mat n* **for** *n* +
fixes Λ *U* :: *complex mat*
fixes *es* :: *complex list*
assumes *eigvals*: *eigvals-of A es*
assumes *eigvals-sorted*: *sorted-wrt* (≥) *es*
assumes *A-decomp*: *real-diag-decomp A* Λ *U*
∧ *diag-mat* Λ = *es*
∧ *set es* ⊆ *Reals*
∧ *U* ∈ *carrier-mat n n*
∧ Λ ∈ *carrier-mat n n*
begin

sublocale *conjugatable-vec-space TYPE*(*complex*) *n* **.**

lemma *dim*: *local.dim* = *n*
by (*simp add*: *dim-is-n*)

lemma *fin-dim*: *fin-dim* **by** *simp*

lemma *gr-n-lin-dpt*:
assumes $B \subseteq$ *carrier-vec n*
assumes *card* $B >$ *local.dim*
shows *lin-dep B*
using *li-le-dim*(*2*)[*of B*] *dim fin-dim assms* **by** *linarith*

lemma *rayleigh-kx*:
assumes $v \in$ *carrier-vec n*
assumes $k \neq 0$
assumes $v \neq 0_v\ n$
shows $\varrho\ A\ (k \cdot_v v) = \varrho\ A\ v$
proof−
let *?v'* $= (k \cdot_v v)$
have $(A *_v$ *?v'*$) \cdot c$ *?v'* $= ($*cmod k*$)\hat{}2 * ($*QF A v*$)$
by (*smt* (*verit*, *ccfv-SIG*) *dim-is assms*(*1*) *carrier-vecD cring-simprules*(*11*) *dim-vec-conjugate index-smult-vec*(*2*) *inner-prod-smult-right mult-conj-cmod-square mult-mat-vec mult-mat-vec-carrier quadratic-form-def scalar-prod-smult-left*)
moreover have *?v'* $\cdot c$ *?v'* $= ($*cmod k*$)\hat{}2 * (v \cdot c\ v)$
by (*metis assms*(*1*) *cring-simprules*(*11*) *dim-vec-conjugate index-smult-vec*(*2*) *inner-prod-smult-right mult-conj-cmod-square scalar-prod-smult-left*)
ultimately show $\varrho\ A$ *?v'* $= \varrho\ A\ v$ **by** (*simp add*: *assms*(*2*))
qed

lemma *unit-vec-rayleigh-formula*:
assumes *unit-v*: *vec-norm v* $= 1$
assumes *v-dim*: $v \in$ *carrier-vec n*
shows $\varrho\ A\ v = (\sum j \in \{..<n\}.\ es!j * ($*cmod* $((U^H *_v v)\$j))\hat{}2)$
proof−
have *U-*Λ: *unitary U* $\wedge$ *unitary* U^H $A = U * \Lambda * U^H$ $U^H \in$ *carrier-mat n n*
apply (*metis A-decomp adjoint-is-conjugate-transpose real-diag-decomp-def unitarily-equiv-def unitary-adjoint unitary-diag-def*)
apply (*metis A-decomp adjoint-is-conjugate-transpose real-diag-decomp-def similar-mat-wit-def unitarily-equiv-def unitary-diag-def*)
by (*simp add*: *A-decomp*)
have*diagonal-mat* Λ **using** *A-decomp* **unfolding** *real-diag-decomp-def* **by** *fast-force*
hence Λ*-diag-mult*: $\bigwedge x\ i.\ x \in$ *carrier-vec n* $\Longrightarrow i \in \{..<n\} \Longrightarrow (\Lambda *_v x)\$i = (\Lambda\$\$(i,i) * x\$i)$
using *A-decomp diagonal-mat-mult-vec* **by** *blast*
have Λ*-diag-eigenvals*: $\bigwedge i.\ i \in \{..<n\} \Longrightarrow \Lambda\$\$(i,i) = es!i$
using *A-decomp*
unfolding *diag-mat-def*
by (*smt* (*verit*, *del-insts*) *carrier-matD*(*1*) *diff-zero length-map length-upt lessThan-iff nth-map nth-upt semiring-norm*(*50*))

define *x* **where** $x \equiv U^H *_v v$
hence *x-dim*: $x \in$ *carrier-vec n*

using *U-Λ*(*3*) *mult-mat-vec-carrier v-dim* **by** *blast*
hence *x-conj-dim*: *conjugate x* ∈ *carrier-vec n* **by** *simp*
have *x-norm*: *vec-norm x* = *1* **using** *U-Λ unit-v unitary-vec-norm v-dim x-def*
by *presburger*

have ∗: ⋀*i*. *i* ∈ {..<*n*} ⟹ (*conjugate x*)$*i* = *conjugate* (*x*$*i*)
unfolding *conjugate-vec-def* **using** *x-dim* **by** *auto*

have *v* ·*c v* = *1* **using** *unit-v csqrt-eq-1* **unfolding** *vec-norm-def* **by** *blast*
hence *QF A v* / *Complex-Matrix.inner-prod v v* = *QF A v* **by** *simp*
hence *ϱ A v* = *complex-of-real* (*Re* (*QF A v*))
unfolding *rayleigh-quotient-def* **by** *simp*
also have ... = *QF A v*
using *hermitian-quadratic-form-real*[*OF dim-is v-dim is-herm*] **by** *simp*
also have ... = *inner-prod v* ((*U* ∗ Λ ∗ *U*^{H}) ∗$_v$ *v*)
unfolding *quadratic-form-def* **using** *U-Λ* **by** *blast*
also have ... = *inner-prod v* (*U* ∗$_v$ ((Λ ∗ *U*^{H}) ∗$_v$ *v*))
by (*smt* (*verit*, *best*) *A-decomp More-Matrix.carrier-vec-conjugate assoc-mat-mult-vec*′
carrier-dim-vec mat-vec-mult-assoc transpose-carrier-mat v-dim)
also have ... = *inner-prod* (*U*^{H} ∗$_v$ *v*) ((Λ ∗ *U*^{H}) ∗$_v$ *v*)
by (*metis A-decomp More-Matrix.carrier-vec-conjugate adjoint-def-alter adjoint-is-conjugate-transpose mult-carrier-mat mult-mat-vec-carrier transpose-carrier-mat v-dim*)
also have ... = (Λ ∗$_v$ *x*) ·*c x*
by (*metis A-decomp More-Matrix.carrier-vec-conjugate carrier-vecD mat-vec-mult-assoc transpose-carrier-mat v-dim x-def*)
also have ... = (∑*j* ∈ {..<*n*}. (Λ ∗$_v$ *x*)$*j* ∗ (*conjugate x*)$*j*)
unfolding *inner-prod-def scalar-prod-def* **using** *atLeast0LessThan x-dim* **by**
auto
also have ... = (∑*j* ∈ {..<*n*}. (Λ$$(*j*,*j*) ∗ *x*$*j*) ∗ (*conjugate x*)$*j*)
using Λ*-diag-mult x-dim* **by** *auto*
also have ... = (∑*j* ∈ {..<*n*}. (*es*!*j* ∗ *x*$*j*) ∗ (*conjugate x*)$*j*)
using Λ*-diag-eigenvals* **by** *simp*
also have ... = (∑*j* ∈ {..<*n*}. (*es*!*j* ∗ *x*$*j*) ∗ *conjugate* (*x*$*j*))
using ∗ **by** *simp*
also have ... = (∑*j* ∈ {..<*n*}. *es*!*j* ∗ (*cmod* (*x*$*j*))^2)
by (*smt* (*verit*) *cring-simprules*(*11*) *mult-conj-cmod-square of-real-mult of-real-sum*
sum.cong)
finally show *ϱ A v* = (∑*j* ∈ {..<*n*}. *es*!*j* ∗ (*cmod* (*x*$*j*))^2)
using *of-real-eq-iff* **by** *blast*
qed

lemma *rayleigh-bdd-below*′:
assumes *k* ≤ *n*
shows ∃ *m*. ∀ *v* ∈ *carrier-vec n*. *v* ≠ *0*$_v$ *n* ⟶ *ϱ A v* ≥ *m*
proof−
define *m* **where** *m* ≡ *Min* (*Re* ‘ *set es*)
have ⋀*v*. *v* ∈ *carrier-vec n* ⟹ *v* ≠ *0*$_v$ *n* ⟹ *ϱ A v* ≥ *m*
proof−

fix v :: *complex vec*
assume $*$: $v \in$ *carrier-vec n* $v \neq 0_v\ n$
define v' **where** $v' \equiv$ *vec-normalize v*
have v': *vec-norm* $v' = 1 \wedge v' \in$ *carrier-vec n*
using *normalized-vec-norm*[*of v n*] **unfolding** *vec-norm-def v'-def*
using $*$ *csqrt-1 normalized-vec-dim* **by** *presburger*
have $\varrho\ A\ v = \varrho\ A\ v'$
by (*metis* $*(1)$ *normalize-zero rayleigh-kx v' v'-def vec-eq-norm-smult-normalized vec-norm-zero*)
also have ... $= (\sum i \in \{..<n\}.\ es!i * (cmod\ ((U^H *_v\ v')\$i))\hat{}2)$
using *unit-vec-rayleigh-formula* $*$ v' **by** *blast*
also have ... $\geq m$
proof−
have *vec-norm* $(U^H *_v\ v') = 1$
by (*metis v' A-decomp Complex-Matrix.unitary-def adjoint-dim-row adjoint-is-conjugate-transpose carrier-matD*(2) *real-diag-decomp-def unitary-adjoint unitary-diagD*(3) *unitary-vec-norm*)
moreover have *vec-norm* $(U^H *_v\ v') =$ *csqrt* $(\sum i \in \{..<n\}.$ (*cmod* $((U^H *_v\ v')\$i))\hat{}2)$
by (*metis complex-vec-norm-sum A-decomp carrier-dim-vec carrier-matD*(2) *dim-mult-mat-vec dim-row-conjugate index-transpose-mat*(2))
ultimately have *norm*: $(\sum i \in \{..<n\}.$ (*cmod* $((U^H *_v\ v')\$i))\hat{}2) = 1$
by (*metis Re-complex-of-real one-complex.sel*(1) *one-power2 power2-csqrt*)

have *finite* (*Re* ‘ *set es*) **by** *simp*
hence $\forall x \in$ *Re* ‘ *set es*. $m \leq x$ **using** *Min-le m-def* **by** *blast*
moreover have $\forall i \in \{..<n\}.\ \exists x \in$ *Re* ‘ *set es*. $x = es!i$
by (*metis A-decomp carrier-matD*(1) *diag-mat-length image-eqI lessThan-iff nth-mem of-real-Re subsetD*)
ultimately have $\bigwedge i.\ i \in \{..<n\} \Longrightarrow m \leq es!i$
by (*metis Im-complex-of-real Re-complex-of-real less-eq-complex-def*)
hence *ineq*:
$\bigwedge i.\ i \in \{..<n\} \Longrightarrow\ m * (cmod\ ((U^H *_v\ v')\$i))\hat{}2 \leq es!i * (cmod\ ((U^H *_v\ v')\$i))\hat{}2$
by (*metis conjugate-square-positive mult-conj-cmod-square mult-right-mono of-real-hom.hom-mult*)

have $m \leq m * (\sum i \in \{..<n\}.$ (*cmod* $((U^H *_v\ v')\$i))\hat{}2)$ **using** *norm* **by** *auto*
also have ... $= (\sum i \in \{..<n\}.\ m * (cmod\ ((U^H *_v\ v')\$i))\hat{}2)$
by (*simp add*: *mult-hom.hom-sum*)
also have ... $\leq (\sum i \in \{..<n\}.\ es!i * (cmod\ ((U^H *_v\ v')\$i))\hat{}2)$
by (*smt* (*verit*, *best*) *of-real-sum sum-mono ineq*)
finally show *?thesis*
by (*metis Im-complex-of-real Re-complex-of-real less-eq-complex-def*)
qed
finally show $\varrho\ A\ v \geq m$ **by** (*simp add*: *less-eq-complex-def*)
qed
thus *?thesis* **by** *blast*

qed

lemma *rayleigh-bdd-below*:
assumes *dimensional* $\mathcal{V}$ k
assumes $k \leq n$
shows $\exists\, m.\ \forall\, v \in \mathcal{V}.\ v \neq 0_v\ n \longrightarrow \varrho\ A\ v \geq m$
using *assms*
unfolding *dimensional-def*
by (*meson span-is-subset2 subsetI rayleigh-bdd-below' span-closed span-mem*)

lemma *rayleigh-min-exists*:
assumes *dimensional* $\mathcal{V}$ k
assumes $k \leq n$
shows $\forall\, x \in \{\varrho\ A\ v \mid v.\ v \neq 0_v\ n \wedge v \in \mathcal{V} \wedge \textit{vec-norm}\ v = 1\}.\ \textit{rayleigh-min}\ \mathcal{V} \leq x$
using *rayleigh-bdd-below*[*OF assms*(*1*) *assms*(*2*)]
unfolding *rayleigh-min*
by (*smt* (*verit*) *bdd-below.I cInf-lower mem-Collect-eq*)

lemma *courant-fischer-unit-rayleigh-helper2*:
assumes *dimensional* $\mathcal{V}$ $(k + 1)$
shows $\exists\, v.\ \textit{vec-norm}\ v = 1 \wedge v \in \mathcal{V} \wedge v \neq 0_v\ n \wedge \varrho\ A\ v \leq es!k$
proof−
have *suc-k-leq-n*: $k + 1 \leq n$
using *assms*(*1*) **unfolding** *dimensional-def* **using** *gr-n-lin-dpt*
by (*metis dim fin-dim li-le-dim*(*2*))
obtain v **where** v:
$v \in \textit{carrier-vec}\ n\ \textit{vec-norm}\ v = 1\ (\forall\, j < k.\ (U^H *_v v)\$j = 0) \wedge v \neq 0_v\ n \wedge v \in \mathcal{V}$
proof−
let *?k-kernel* $= \{v \in \textit{carrier-vec}\ n.\ (\forall\, j < k.\ (U^H *_v v)\$j = 0)\}$
obtain v **where** v: $v \in$ *?k-kernel* $\cap\ \mathcal{V} \wedge v \neq 0_v\ n$
proof−
obtain $B_\mathcal{V}$ **where** $B_\mathcal{V}$: $\mathcal{V} = \textit{span}\ B_\mathcal{V} \wedge \textit{card}\ B_\mathcal{V} = k + 1 \wedge B_\mathcal{V} \subseteq \textit{carrier-vec}\ n \wedge \textit{lin-indpt}\ B_\mathcal{V}$
using *assms* **unfolding** *dimensional-def* **by** *blast*
obtain B_k **where** B_k: $\textit{span}\ B_k \subseteq$ *?k-kernel* $\wedge\ \textit{card}\ B_k = n - k \wedge B_k \subseteq \textit{carrier-vec}\ n \wedge \textit{lin-indpt}\ B_k$
proof−
obtain M **where** M: $M * U^H = 1_m\ n \wedge U^H * M = 1_m\ n \wedge M \in \textit{carrier-mat}\ n\ n$
by (*metis A-decomp Complex-Matrix.unitary-def adjoint-dim-row adjoint-is-conjugate-transpose carrier-matD*(*2*) *mat-mult-left-right-inverse real-diag-decomp-def unitary-adjoint unitary-diagD*(*3*) *unitary-simps*(*1*))
define B_k **where** $B_k = \textit{set}\ (\textit{drop}\ k\ (\textit{cols}\ M))$
hence $B_k \subseteq \textit{set}\ (\textit{cols}\ M)$ **by** (*meson set-drop-subset*)
hence *lin-indpt*: *lin-indpt* B_k
by (*metis M A-decomp More-Matrix.carrier-vec-conjugate det-zero-low-rank distinct-cols-id idom-vec.lin-dep-cols-imp-det-0 lin-indpt-id linorder-not-le one-carrier-mat*

rank-mat-mul-right supset-ld-is-ld transpose-carrier-mat vec-space.lin-indpt-full-rank)

have *1*: *span* $B_k \subseteq$ *?k-kernel*
proof
fix v **assume** $*$: $v \in$ *span* B_k
then obtain *cs* **where** $v =$ *lincomb cs* B_k
by (*metis* (*no-types*, *lifting*) M ‹$B_k \subseteq$ *set* (*cols* M)› *carrier-matD*(*1*) *cols-dim fin-dim finite-in-span li-le-dim*(*1*) *lin-indpt order-trans*)
hence *sum*: $U^H *_v v =$ *finsum-vec TYPE*(*complex*) n (λb. (*cs* b) $\cdot_v$ (U^H $*_v$ b)) B_k
using *mat-vec-mult-sum*[*of* v n U^H B_k *cs*] **unfolding** *lincomb-def*
by (*metis A-decomp* B_k*-def List.finite-set M More-Matrix.carrier-vec-conjugate* ‹$B_k \subseteq$ *set* (*cols* M)› ‹$v =$ *lincomb cs* B_k› *basic-trans-rules*(*23*) *carrier-matD*(*1*) *cols-dim finsum-vec lincomb-closed transpose-carrier-mat*)

have $\bigwedge i.\ i < k \Longrightarrow (U^H *_v v)\$i = 0$
proof−
fix i **assume** $*$: $i < k$
have *cs*: (λb. *cs* b $\cdot_v$ (U^H $*_v$ b)) $\in B_k \to$ *carrier-vec* n
proof
fix x **assume** $x \in B_k$
thus *cs* x $\cdot_v$ (U^H $*_v$ x) $\in$ *carrier-vec* n
by (*metis A-decomp carrier-dim-vec carrier-matD*(*2*) *dim-mult-mat-vec dim-row-conjugate index-smult-vec*(*2*) *index-transpose-mat*(*2*))
qed
have B_k: $\bigwedge b.\ b \in B_k \Longrightarrow (U^H *_v b)\$i = 0$
proof−
fix b **assume** $**$: $b \in B_k$
then obtain j' **where** $b =$ *drop* k (*cols* M)$!j' \wedge j' < n - k$
by (*metis* B_k*-def* M *carrier-matD*(*2*) *cols-length in-set-conv-nth length-drop*)
then obtain j **where** j: $b =$ (*cols* M)$!j \wedge j \geq k \wedge j < n$
by (*metis Groups.add-ac*(*2*) M *add-leD1 carrier-matD*(*2*) *cols-length le-add2 less-diff-conv nth-drop suc-k-leq-n*)

have $i < n \wedge j < n \wedge i \neq j$ **using** $*$ j **by** *simp*
hence $0 = (1_m\ n)\$\$(i,j)$ **by** *simp*
also have ... $= (U^H * M)\$\$(i,j)$
using M **by** *argo*
also have ... = *row* U^H $i \cdot$ *col* M j
by (*metis* $*$ M *add-leD1 carrier-matD*(*1*) *carrier-matD*(*2*) *dual-order.strict-trans1 index-mult-mat*(*1*) *index-mult-mat*(*2*) j *suc-k-leq-n*)
also have ... = *row* U^H $i \cdot b$
using j M **by** *auto*
also have ... $= (U^H *_v b)\$i$
by (*metis* $*$ *A-decomp carrier-matD*(*2*) *dim-row-conjugate dual-order.strict-trans dual-order.strict-trans1 index-mult-mat-vec index-transpose-mat*(*2*) j)
finally show $(U^H *_v b)\$i = 0$ **by** *presburger*
qed

```
          have (U^H *_v v)$i = (finsum-vec TYPE(complex) n (λb. (cs b) ·_v (U^H
*_v b)) B_k)$i
            using sum by simp
          also have ... = (∑ b ∈ B_k. ((cs b) ·_v (U^H *_v b))$i)
           using index-finsum-vec[of B_k i n (λb. (cs b) ·_v (U^H *_v b))] cs * B_k-def
suc-k-leq-n
            by fastforce
          also have ... = (∑ b ∈ B_k. (cs b) * ((U^H *_v b))$i)
            using * A-decomp suc-k-leq-n by force
          also have ... = (∑ b ∈ B_k. (cs b) * 0)
            using B_k by fastforce
          also have ... = 0
            by simp
          finally show (U^H *_v v)$i = 0 .
        qed
        thus v ∈ ?k-kernel
            by (smt (verit) * M ‹B_k ⊆ set (cols M)› basic-trans-rules(23) car-
rier-matD(1) cols-dim mem-Collect-eq span-closed)
      qed
      have 2: card B_k = n − k
      proof−
        have distinct (cols M)
            by (metis A-decomp M More-Matrix.carrier-vec-conjugate conjugat-
able-vec-space.distinct-cols-id lin-indpt-full-rank lin-indpt-id linorder-not-le non-distinct-low-rank
one-carrier-mat rank-mat-mul-right transpose-carrier-mat)
        hence card B_k = length (drop k (cols M))
          using B_k-def distinct-card distinct-drop by blast
        moreover have length (cols M) = n using M by force
        ultimately show ?thesis by force
      qed
      have 3: B_k ⊆ carrier-vec n
        by (metis M ‹B_k ⊆ set (cols M)› basic-trans-rules(23) carrier-matD(1)
cols-dim)

      from 1 2 3 lin-indpt that show ?thesis by blast
    qed
    define B where B = B_V ∪ B_k
    have B_V ∩ B_k ≠ {} ⟹ ?thesis
    proof−
      assume *: B_V ∩ B_k ≠ {}
      have 0_v n ∉ B_V by (simp add: B_V vs-zero-lin-dep)
      moreover have 0_v n ∉ B_k by (simp add: B_k vs-zero-lin-dep)
      ultimately obtain b where b: b ∈ B_V ∩ B_k ∧ b ≠ 0_v n using * by blast
      moreover then have b ∈ V by (simp add: B_V span-mem)
      moreover have b ∈ ?k-kernel
        by (metis (no-types, lifting) B_k IntE Set.basic-monos(7) b in-own-span)
      ultimately show ?thesis using that by fast
    qed
```

moreover have $B_{\mathcal{V}} \cap B_k = \{\} \Longrightarrow$ *?thesis*
proof−
assume ∗: $B_{\mathcal{V}} \cap B_k = \{\}$
hence *card* $(B_{\mathcal{V}} \cup B_k) =$ *card* $B_{\mathcal{V}} +$ *card* B_k
by (*simp add*: $B_{\mathcal{V}}$ B_k *card-Un-disjnt disjnt-def li-le-dim*(*1*))
hence *card* $B > n$ **using** *B-def* $B_{\mathcal{V}}$ B_k *suc-k-leq-n* **by** *presburger*
moreover have $B \subseteq$ *carrier-vec* n **unfolding** *B-def* **using** $B_{\mathcal{V}}$ B_k **by** *blast*
ultimately have *lin-dep*: *lin-dep* B **using** *gr-n-lin-dpt dim* **by** *presburger*
obtain *cs cs*′ **where**
eq: *lincomb cs* $B_{\mathcal{V}} =$ *lincomb cs*′ $B_k \wedge (\exists\, b \in B_{\mathcal{V}}.\ cs\ b \neq 0) \wedge (\exists\, b \in B_k.$ *cs*′ $b \neq 0$)
proof−
obtain *cs* **where** *cs*: *lincomb cs* $B = 0_v\ n \wedge (\exists\, b \in B.\ cs\ b \neq 0)$
by (*metis lin-dep* ‹$B \subseteq$ *carrier-vec* n› ‹$n <$ *card* B› *bot-nat-0.extremum-strict card.infinite finite-lin-indpt2*)
define *cs*′ **where** *cs*′ $= (\lambda v.\ -\ cs\ v)$
have $\bigwedge i.\ i < n \Longrightarrow$ (*lincomb cs* $B_{\mathcal{V}}$)$\$i =$ (*lincomb cs*′ B_k)$\$i$
proof−
fix *i*
assume ∗∗: $i < n$
have *0* $=$ (*lincomb cs B*)$\$i$
by (*simp add*: ∗∗ *cs*)
also have ... $= (\sum b \in B.\ (cs\ b \cdot_v b)\$i)$
by (*smt* (*verit*, *ccfv-SIG*) ∗∗ *R.add.finprod-cong*′ ‹$B \subseteq$ *carrier-vec* n› *carrier-vecD index-smult-vec*(*1*) *lincomb-index smult-carrier-vec summands-valid*)
also have ... $= (\sum b \in B_{\mathcal{V}}.\ (cs\ b \cdot_v b)\$i) + (\sum b \in B_k.\ (cs\ b \cdot_v b)\$i)$
by (*simp add*: $B_{\mathcal{V}}$ B_k *B-def fin-dim-li-fin sum.union-disjoint* ∗)
finally have $(\sum b \in B_{\mathcal{V}}.\ (cs\ b \cdot_v b)\$i) = -\ (\sum b \in B_k.\ (cs\ b \cdot_v b)\$i)$
by (*simp add*: *eq-neg-iff-add-eq-0*)
hence (*lincomb cs* $B_{\mathcal{V}}$)$\$i = (\sum b \in B_k.\ -\ ((cs\ b) \cdot_v b)\$i)$
by (*smt* (*verit*, *best*) ∗∗ $B_{\mathcal{V}}$ *R.add.finprod-cong*′ *carrier-vecD index-smult-vec*(*1*) *lincomb-index smult-carrier-vec sum-negf summands-valid*)
moreover have $\bigwedge v.\ v \in$ *carrier-vec* $n \Longrightarrow -\ (v\$i) = (-v)\$i$
by (*simp add*: ∗∗)
moreover have $\bigwedge b.\ (-\ (cs\ b) \cdot_v b)\$i = ((\ -\ cs\ b) \cdot_v b)\$i$
by *blast*
ultimately have (*lincomb cs* $B_{\mathcal{V}}$)$\$i = (\sum b \in B_k.\ ((-\ cs\ b) \cdot_v b)\$i)$
by (*smt* (*verit*, *del-insts*) B_k *R.add.finprod-cong*′ *local.vec-neg smult-carrier-vec smult-l-minus summands-valid*)
also have ... $=$ (*lincomb cs*′ B_k)$\$i$
by (*smt* (*verit*, *best*) ∗∗ B_k *R.add.finprod-cong*′ *carrier-vecD cs*′*-def index-smult-vec*(*1*) *lincomb-index smult-carrier-vec summands-valid*)
finally show (*lincomb cs* $B_{\mathcal{V}}$)$\$i =$ (*lincomb cs*′ B_k)$\$i$ **by** *blast*
qed
hence *1*: (*lincomb cs* $B_{\mathcal{V}}$) $=$ (*lincomb cs*′ B_k)
by (*metis* (*no-types*, *lifting*) $B_{\mathcal{V}}$ B_k *carrier-vecD eq-vecI lincomb-closed*)
have *2*: $\exists\, b \in B_{\mathcal{V}}.\ cs\ b \neq 0$
proof(*rule ccontr*)
assume ¬ $(\exists\, b{\in}B_{\mathcal{V}}.\ cs\ b \neq 0)$

hence $*$: $\forall\, b \in B_{\mathcal{V}}.\ cs\ b = 0$ **by** *blast*
hence $\forall\, i < n.\ (lincomb\ cs\ B_{\mathcal{V}})\$i = 0$ **by** (*simp add*: $B_{\mathcal{V}}$ *vec-space.lincomb-index*)
hence $lincomb\ cs\ B_{\mathcal{V}} = 0_v\ n$ **using** $B_{\mathcal{V}}$ **by** *auto*
moreover have *lin-indpt* B_k **using** B_k **by** *blast*
ultimately have $\forall\, b \in B_k.\ cs'\ b = 0$
by (*metis* (*no-types*, *lifting*) *1* B_k *fin fin-dim lin-dep-def order.refl*)
hence $\forall\, b \in B_k.\ cs\ b = 0$ **unfolding** *cs′-def* **by** *fastforce*
hence $\forall\, b \in B.\ cs\ b = 0$ **using** $*$ **unfolding** *B-def* **by** *blast*
thus *False* **using** *cs* **by** *blast*
qed
have *3*: $\exists\, b \in B_k.\ cs'\ b \neq 0$
proof(*rule ccontr*)
assume $\neg\ (\exists\, b \in B_k.\ cs'\ b \neq 0)$
hence $*$: $\forall\, b \in B_k.\ cs'\ b = 0$ **by** *blast*
hence $\forall\, i < n.\ (lincomb\ cs'\ B_k)\$i = 0$ **by** (*simp add*: B_k *vec-space.lincomb-index*)
hence $lincomb\ cs'\ B_k = 0_v\ n$ **using** B_k **by** *auto*
moreover have *lin-indpt* $B_{\mathcal{V}}$ **using** $B_{\mathcal{V}}$ **by** *blast*
ultimately have $\forall\, b \in B_{\mathcal{V}}.\ cs\ b = 0$
by (*metis 1* $B_{\mathcal{V}}$ *fin-dim li-le-dim*(*1*) *lin-dep-def subsetI*)
hence $\forall\, b \in B.\ cs\ b = 0$ **using** $*$ **unfolding** *B-def cs′-def* **by** *fastforce*
thus *False* **using** *cs* **by** *blast*
qed
from *1 2 3 that* **show** *?thesis* **by** *algebra*
qed
define v **where** $v \equiv lincomb\ cs\ B_{\mathcal{V}}$
define w **where** $w \equiv lincomb\ cs'\ B_k$
have $\bigwedge i.\ i < n \Longrightarrow v\$i = w\$i$ **using** *eq v-def w-def* **by** *argo*
moreover have $\bigwedge i.\ i < n \Longrightarrow w\$i = (\sum b \in B_k.\ (cs'\ b \cdot_v b)\$i)$
by (*smt* (*verit*, *best*) B_k *carrier-vecD index-smult-vec*(*1*) *lincomb-index smult-carrier-vec sum.cong summands-valid w-def*)
moreover have $v \in \mathcal{V}$ **by** (*simp add*: $B_{\mathcal{V}}$ *in-spanI li-le-dim*(*1*) *v-def*)
moreover have $w \in$ *?k-kernel*
proof−
have $w \in carrier\text{-}vec\ n$ **by** (*metis* $B_{\mathcal{V}}$ *eq lincomb-closed w-def*)
moreover have $U^H \in carrier\text{-}mat\ n\ n$ **by** (*simp add*: *A-decomp*)
moreover have $B_k \subseteq carrier\text{-}vec\ n$ **by** (*simp add*: B_k)
moreover have *finite* B_k **by** (*simp add*: B_k *fin*)
ultimately have $U^H *_v w = finsum\text{-}vec\ TYPE(complex)\ n\ (\lambda b.\ cs'\ b \cdot_v (U^H *_v b))\ B_k$
using *mat-vec-mult-sum*[*of w n* U^H B_k] *lincomb-def w-def* **by** *fastforce*
thus *?thesis* **using** B_k ‹*finite* B_k› *w-def* **by** *blast*
qed
moreover have $v \neq 0_v\ n$ **by** (*metis* $B_{\mathcal{V}}$ *eq fin fin-dim lin-dep-crit order.refl v-def*)
ultimately show *?thesis* **using** *eq that v-def w-def* **by** *auto*
qed
ultimately show *?thesis* **by** *blast*
qed
moreover define v' **where** $v' \equiv vec\text{-}normalize\ v$

ultimately have $v' \in$ *carrier-vec* $n \wedge$ *vec-norm* $v' = 1$
using *normalized-vec-norm vec-norm-def* **by** *force*
moreover then have v': $v' = (1 \ / \ \textit{vec-norm}\ v) \cdot_v v$
by (*metis div-by-1 one-smult-vec v'-def vec-normalize-def*)
moreover have $v' \neq 0_v\ n$ **by** (*metis calculation*(1) *field.one-not-zero vec-norm-zero*)
moreover have $v' \in \mathcal{V}$
proof−
have $v \in \mathcal{V}$ **using** v **by** *blast*
thus *?thesis* **using** v' *assms* **unfolding** *dimensional-def* **by** *auto*
qed
moreover have $\forall j < k.\ (U^H *_v v')\$j = 0$
proof *clarify*
fix j **assume** $*$: $j < k$
have $(U^H *_v v')\$j = \ (U^H *_v ((1 \ / \ \textit{vec-norm}\ v) \cdot_v v))\$j$
using v' **by** *blast*
also have ... $= ((1 \ / \ \textit{vec-norm}\ v) \cdot_v (U^H *_v v))\$j$
by (*metis A-decomp More-Matrix.carrier-vec-conjugate* ‹$v' \in$ *carrier-vec* $n \wedge$ *vec-norm* $v' = 1$› *mult-mat-vec smult-carrier-vec transpose-carrier-mat* v')
also have ... $= (1 \ / \ \textit{vec-norm}\ v) * ((U^H *_v v)\$j)$
using $*$ *A-decomp suc-k-leq-n* **by** *auto*
also have ... $= (1 \ / \ \textit{vec-norm}\ v) * 0$ **using** $*$ v **by** *auto*
finally show $(U^H *_v v')\$j = 0$ **by** *algebra*
qed
ultimately show *?thesis* **using** *that* **by** *blast*
qed
then obtain *weights* **where** *weights*: $(\forall i \in \{k..<n\}.\ \textit{weights}\ i \geq 0)$
$\wedge\ (\sum i \in \{k..<n\}.\ \textit{weights}\ i) = 1 \wedge \varrho\ A\ v = (\sum i \in \{k..<n\}.\ \textit{weights}\ i * es!i)$
proof−
define *weights* **where** *weights* $\equiv \lambda i.\ (\textit{cmod}\ ((U^H *_v v)\$i))\hat{}2$
hence *1*: $(\forall i \in \{k..<n\}.\ \textit{weights}\ i \geq 0)$ **by** *auto*

have $\bigwedge i.\ i \in \{..<k\} \Longrightarrow (U^H *_v v)\$i = 0$ **using** v **by** *blast*
hence $*$: $\bigwedge i.\ i \in \{..<k\} \Longrightarrow \textit{weights}\ i = 0$ **using** *weights-def* **by** *auto*
hence $**$: $\bigwedge i.\ i \in \{..<k\} \Longrightarrow \textit{weights}\ i * es!i = 0$ **by** (*simp add*: *weights-def*)

have $(\sum i \in \{..<n\}.\ \textit{weights}\ i) = (\sum i \in \{..<k\}.\ \textit{weights}\ i) + (\sum i \in \{k..<n\}.\ \textit{weights}\ i)$
by (*smt* (*verit*, *ccfv-threshold*) $*$ *atLeast0LessThan atLeastLessThan-iff le-eq-less-or-eq linorder-not-le sum.atLeastLessThan-concat sum.not-neutral-contains-not-neutral zero-order*(1))
also have ... $= (\sum i \in \{k..<n\}.\ \textit{weights}\ i)$
by (*simp add*: $*$)
finally have *weights*: $(\sum i \in \{..<n\}.\ \textit{weights}\ i) = (\sum i \in \{k..<n\}.\ \textit{weights}\ i)$ **.**

have $(\sum i \in \{..<n\}.\ \textit{weights}\ i * es!i) = (\sum i \in \{..<k\}.\ \textit{weights}\ i * es!i)$
$+ (\sum i \in \{k..<n\}.\ \textit{weights}\ i * es!i)$
by (*smt* (*verit*, *ccfv-threshold*) $**$ *atLeast0LessThan atLeastLessThan-iff le-eq-less-or-eq linorder-not-le sum.atLeastLessThan-concat sum.not-neutral-contains-not-neutral zero-order*(1))
also have ... $= (\sum i \in \{k..<n\}.\ \textit{weights}\ i * es!i)$

by (*simp add*: ∗∗)
finally have *weights-es*: $(\sum i \in \{..<n\}.\ weights\ i * es!i) = (\sum i \in \{k..<n\}.\ weights\ i * es!i)$ **.**

have $\varrho\ A\ v = (\sum i \in \{..<n\}.\ weights\ i * es!i)$
using *unit-vec-rayleigh-formula v weights-def* **by** *algebra*
hence *2*: $\varrho\ A\ v = (\sum i \in \{k..<n\}.\ weights\ i * es!i)$ **using** *weights-es* **by** *argo*

have *norm*: *vec-norm* $(U^H *_v v) =$ *vec-norm v*
by (*metis A-decomp More-Matrix.carrier-vec-conjugate adjoint-is-conjugate-transpose real-diag-decompD(1) transpose-carrier-mat unitary-adjoint unitary-diagD(3) unitary-vec-norm v(1)*)
hence *vec-norm* $(U^H *_v v) = 1$ **using** *v(2)* **by** *argo*
moreover have $U^H *_v v \in$ *carrier-vec n*
by (*metis A-decomp adjoint-dim-row adjoint-is-conjugate-transpose carrier-matD(2) carrier-vec-dim-vec dim-mult-mat-vec*)
ultimately have $(\sum i \in \{..<n\}.\ weights\ i) = 1$
unfolding *weights-def* **by** (*metis complex-vec-norm-sum norm Re-complex-of-real csqrt-eq-1 one-complex.sel(1) v(2)*)
hence *3*: $(\sum i \in \{k..<n\}.\ weights\ i) = 1$ **using** *weights* **by** *presburger*

show *?thesis*
by (*metis 1 2 3 that*[*of weights*] *Im-complex-of-real Re-complex-of-real less-eq-complex-def of-real-hom.hom-one of-real-hom.hom-zero of-real-sum*)
qed
have *length es* $= n$
using *A-decomp*
unfolding *diag-mat-def*
by (*metis A-decomp carrier-matD(1) diag-mat-length*)
hence $\bigwedge i.\ i \in \{k..<n\} \Longrightarrow es!i \leq es!k$
using *eigvals-sorted*
by (*metis atLeastLessThan-iff le-eq-less-or-eq sorted-wrt-iff-nth-less verit-eq-simplify(6)*)
then have $\bigwedge i.\ i \in \{k..<n\} \Longrightarrow weights\ i * es!i \leq weights\ i * es!k$
by (*meson weights atLeastLessThan-iff mult-left-mono*)
then have $(\sum i \in \{k..<n\}.\ weights\ i * es!i) \leq (\sum i \in \{k..<n\}.\ weights\ i * es!k)$
by (*meson sum-mono*)
also have $... = (\sum i \in \{k..<n\}.\ weights\ i) * es!k$
by (*metis ideal.scale-sum-left*)
also have $... = es!k$ **using** *weights* **by** *auto*
finally have $\varrho\ A\ v \leq es!k$
using *weights* **by** *presburger*
thus $\exists v.$ *vec-norm* $v = 1 \wedge v \in \mathcal{V} \wedge v \neq 0_v\ n \wedge \varrho\ A\ v \leq es!k$ **using** *v* **by** *blast*
qed

lemma *courant-fischer-unit-rayleigh-helper3*:
assumes $n > 0$
assumes $k < n$
assumes *eigvals-of A es*
defines *es-R* $\equiv$ *map Re es*

shows $\exists \mathcal{V}$. *dimensional* $\mathcal{V}$ $(k + 1) \wedge (\forall v.\ v \neq 0_v\ n \wedge v \in \mathcal{V} \wedge$ *vec-norm* $v = 1$ $\longrightarrow$ *es-R* ! $k \leq \varrho\ A\ v)$
proof−
let *?geq* $= \lambda v.\ \varrho\ A\ v \geq$ *es-R*!k
let *?P* $= \lambda \mathcal{V}\ v.\ v \neq 0_v\ n \wedge v \in \mathcal{V} \wedge$ *vec-norm* $v = 1$
let *?Q* $= \lambda \mathcal{V}.$ *dimensional* $\mathcal{V}$ $(k + 1)$

have *inverts-mat U* (*adjoint U*)
using *A-decomp*
unfolding *real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def*
by (*metis Complex-Matrix.unitary-def*)
hence *U-invertible*: *invertible-mat U*
by (*metis A-decomp Complex-Matrix.unitary-def carrier-matD*(*1*) *carrier-matD*(*2*) *invertible-mat-def square-mat.simps unitaryD2*)

have *unitary U*
using *A-decomp*
unfolding *real-diag-decomp-def unitary-diag-def unitarily-equiv-def*
by *blast*
hence *U-ortho*: *corthogonal-mat U* **using** *unitary-is-corthogonal A-decomp* **by** *auto*

have *set es* $\subseteq$ *Reals* **by** (*simp add*: *A-decomp*)
hence *es-R*: *map complex-of-real es-R* $=$ *es*
proof−
{ fix i **assume** $*$: $i <$ *length es*
hence *es*!$i \in$ *Reals* **using** *A-decomp* **by** *auto*
hence *complex-of-real* (*es-R*!i) $=$ *es*!i **by** (*simp add*: $*$ *es-R-def*)
}
thus *?thesis* **by** (*simp add*: *es-R-def map-nth-eq-conv*)
qed

let *?*$\mathcal{V}$*-basis* $=$ *set* (*take* $(k + 1)$ (*cols U*))
define $\mathcal{V}$ **where** $\mathcal{V} \equiv$ *span ?*$\mathcal{V}$*-basis*
have *lin-indpt ?*$\mathcal{V}$*-basis*
using *lin-indpt-subset-cols*[*of U ?*$\mathcal{V}$*-basis*] *A-decomp U-invertible*
by (*meson in-set-takeD subset-code*(*1*))
moreover have $\mathcal{V}$*-basis-card*: *card ?*$\mathcal{V}$*-basis* $= k + 1$
proof−
have *distinct* (*cols U*)
by (*metis A-decomp U-invertible invertible-det nat-less-le non-distinct-low-rank vec-space.det-rank-iff*)
hence *distinct* (*take* $(k + 1)$ (*cols U*))
using *distinct-take* **by** *blast*
thus *?thesis*
by (*metis A-decomp Suc-leI add.commute add-diff-cancel-right' append-take-drop-id assms*(*2*) *carrier-matD*(*2*) *cols-length diff-diff-cancel distinct-card length-append length-drop plus-1-eq-Suc*)
qed

ultimately have *1*: *?Q* $\mathcal{V}$
unfolding *dimensional-def*
by (*metis A-decomp* $\mathcal{V}$*-def carrier-matD*(*1*) *cols-dim in-set-takeD subset-code*(*1*))
moreover have *2*: $\forall v$. *?P* $\mathcal{V}$ *v* $\longrightarrow$ *?geq v*
proof *clarify*
fix *v* :: *complex vec*
assume $*$: $v \neq 0_v\ n$
assume $**$: $v \in \mathcal{V}$
assume $***$: *vec-norm v* = *1*

have *set* (*cols U*) $\subseteq$ *carrier-vec n* **using** *A-decomp cols-dim* **by** *blast*
hence $\mathcal{V}$*-basis-dim*: *?*$\mathcal{V}$*-basis* $\subseteq$ *carrier-vec n* **by** (*meson in-set-takeD subset-code*(*1*))
hence *v-dim*: *v* $\in$ *carrier-vec n* **using** $**$ $\mathcal{V}$*-def span-closed* **by** *blast*

define *x* **where** $x \equiv U^H *_v v$
have *x-dim*: *x* $\in$ *carrier-vec n*
by (*metis A-decomp More-Matrix.carrier-vec-conjugate mult-mat-vec-carrier transpose-carrier-mat v-dim x-def*)
have *x-norm*: *vec-norm x* = *1*
by (*metis* $***$ *A-decomp More-Matrix.carrier-vec-conjugate adjoint-is-conjugate-transpose real-diag-decomp-def transpose-carrier-mat unitarily-equiv-def unitary-adjoint unitary-diag-def unitary-vec-norm v-dim x-def*)
have *weights*: $(\sum j \in \{..<n\}.\ (cmod\ (x\$j))\hat{}2) = 1$
by (*metis atLeast0LessThan carrier-vecD cpx-vec-length-square of-real-eq-1-iff power-one vec-norm-sq-cpx-vec-length-sq x-dim x-norm*)

have *ineq*: $\bigwedge j.\ j \in \{..<n\} \Longrightarrow es!k * (cmod\ (x\$j))\hat{}2 \leq es!j * (cmod\ (x\$j))\hat{}2$
proof−
fix *j*
assume $*$: $j \in \{..<n\}$
show $es!k * (cmod\ (x\$j))\hat{}2 \leq es!j * (cmod\ (x\$j))\hat{}2$
proof(*cases j* $\leq$ *k*)
case *True*
hence $es!k \leq es!j$
by (*metis A-decomp antisym-conv1 assms*(*2*) *carrier-matD*(*1*) *diag-mat-length eigvals-sorted nless-le sorted-wrt-nth-less*)
thus *?thesis* **by** (*simp add*: *less-eq-complex-def mult-right-mono*)
next
case *False*
hence *j-gr-k*: *j* > *k* **by** *simp*
have *inner-prod* (*col U j*) *v* = *0*
proof−
have *j*: *j* < *dim-col U* **using** $*$ *A-decomp* **by** *blast*
have $\forall b \in$ *?*$\mathcal{V}$*-basis*. $\exists i.\ i <$ *dim-col U* $\wedge$ *b* = *col U i* $\wedge$ *i* $\neq$ *j*
proof
fix *b* **assume** $*$: *b* $\in$ *?*$\mathcal{V}$*-basis*
then obtain *i* **where** *b* = (*cols U*)!*i* $\wedge$ *i* $\leq$ *k*

by (*metis A-decomp U-invertible* $\mathcal{V}$*-basis-card add.commute distinct-card distinct-take in-set-conv-nth invertible-det less-Suc-eq-le nat-less-le nth-take plus-1-eq-Suc vec-space.det-rank-iff vec-space.non-distinct-low-rank*)
moreover then have $i \neq j$ **using** *j-gr-k* **by** *force*
ultimately show $\exists i{<}$*dim-col U*. $b =$ *col U i* $\wedge$ $i \neq j$
by (*meson cols-nth j j-gr-k le-simps*(1) *order-le-less-trans*)
qed
hence *basis-ortho*: $\forall\, b \in ?\mathcal{V}$*-basis*. *inner-prod* (*col U j*) $b = 0$
using *j-gr-k corthogonal-matD*[*OF U-ortho - j*] **by** *fast*

obtain *cs* **where** *lincomb cs* $?\mathcal{V}$*-basis* $= v$
using $**$ $\mathcal{V}$*-def* $\mathcal{V}$*-basis-dim finite-in-span* **by** *blast*
hence *inner-prod* (*col U j*) $v = (\sum b{\in}?\mathcal{V}$*-basis*. *cs b* $*$ *inner-prod* (*col U j*) *b*)
by (*smt* (*verit*, *best*) *A-decomp List.finite-set R.add.finprod-cong*$'$ $\mathcal{V}$*-basis-dim carrier-matD*(1) *col-dim finsum-vec inner-prod-vec-sum lincomb-def v-dim*)
thus *?thesis* **using** *basis-ortho* **by** *fastforce*
qed
hence *conjugate* (*col U j*) $\cdot$ $v = 0$
by (*metis A-decomp carrier-matD*(1) *col-dim conjugate-vec-sprod-comm v-dim*)
hence *row* U^H $j \cdot v = 0$
by (*metis* $*$ *adjoint-dim-row adjoint-is-conjugate-transpose adjoint-row carrier-vecD dim-mult-mat-vec lessThan-iff x-def x-dim*)
hence $x\$j = 0$
unfolding *x-def*
by (*metis* $*$ *carrier-vecD dim-mult-mat-vec index-mult-mat-vec lessThan-iff x-def x-dim*)
thus *?thesis* **by** *fastforce*
qed
qed

have *es-R*$!k =$ *es*$!k$
by (*metis A-decomp assms*(2) *carrier-matD*(1) *diag-mat-length es-R length-map nth-map*)
also have ... $=$ *es*$!k$ $*$ $(\sum j \in \{..{<}n\}.$ $(cmod\ (x\$j))\hat{}2)$
by (*simp add*: *weights*)
also have ... $= (\sum j \in \{..{<}n\}.$ *es*$!k$ $*$ $(cmod\ (x\$j))\hat{}2)$
by (*simp add*: *mult-hom.hom-sum*)
also have ... $\leq (\sum j \in \{..{<}n\}.$ *es*$!j$ $*$ $(cmod\ (x\$j))\hat{}2)$
by (*meson ineq sum-mono*)
also have ... $= \varrho\ A\ v$
using *unit-vec-rayleigh-formula A-decomp* $***$ *v-dim x-def* **by** *fastforce*
finally show *?geq v* **by** (*simp add*: *less-eq-complex-def*)
qed
ultimately show *?thesis* **by** *blast*
qed

theorem *courant-fischer-maximin*:
assumes $n > 0$
assumes $k < n$
shows $es!k = maximin\ (k + 1)$
$maximin\text{-}defined\ (k + 1)$
proof−
have *inverts-mat U* (*adjoint U*)
using *A-decomp*
unfolding *real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def*
by (*metis Complex-Matrix.unitary-def*)
hence *U-invertible*: *invertible-mat U*
by (*metis A-decomp Complex-Matrix.unitary-def carrier-matD*(*1*) *carrier-matD*(*2*) *invertible-mat-def square-mat.simps unitaryD2*)

have *unitary U*
using *A-decomp*
unfolding *real-diag-decomp-def unitary-diag-def unitarily-equiv-def*
by *blast*
hence *U-ortho*: *corthogonal-mat U* **using** *unitary-is-corthogonal A-decomp* **by** *auto*

define *es-R* **where** *es-R* ≡ *map Re es*
have *set es* ⊆ *Reals* **by** (*simp add*: *A-decomp*)
hence *es-R*: *map complex-of-real es-R* = *es*
proof−
{ fix *i* **assume** ∗: *i* < *length es*
hence *es*!*i* ∈ *Reals* **using** *A-decomp* **by** *auto*
hence *complex-of-real* (*es-R*!*i*) = *es*!*i* **by** (*simp add*: ∗ *es-R-def*)
}
thus *?thesis* **by** (*simp add*: *es-R-def map-nth-eq-conv*)
qed
hence *es-R-i*: ⋀*i*. *i* ∈ {..<*n*} ⟹ *es-R*!*i* = *Re* (*es*!*i*)
using *dim-is eigvals eigvals-poly-length es-R-def* **by** *simp*
hence *es-R-k*: *es-R*!*k* = *Re* (*es*!*k*) **by** (*simp add*: *assms*(*2*))

let *?leq* = λ*v*. *ϱ A v* ≤ *es-R*!*k*
let *?geq* = λ*v*. *ϱ A v* ≥ *es-R*!*k*
let *?P* = λ$\mathcal{V}$ *v*. *v* ≠ 0_v *n* ∧ *v* ∈ $\mathcal{V}$ ∧ *vec-norm v* = *1*
let *?Q* = λ$\mathcal{V}$. *dimensional* $\mathcal{V}$ (*k* + *1*)
let $?S_\varrho$ = λ$\mathcal{V}$. {*ϱ A v* |*v*. *?P* $\mathcal{V}$ *v*}

have *1*: ⋀$\mathcal{V}$. *?Q* $\mathcal{V}$ ⟹ (∃ *v*. *?P* $\mathcal{V}$ *v* ∧ *?leq v*)
by (*metis Re-complex-of-real courant-fischer.courant-fischer-unit-rayleigh-helper2 courant-fischer-axioms es-R-k less-eq-complex-def*)

have *2*: ∃$\mathcal{V}$. *?Q* $\mathcal{V}$ ∧ (∀ *v*. *?P* $\mathcal{V}$ *v* ⟶ *?geq v*)
using *courant-fischer-unit-rayleigh-helper3*[*OF assms*(*1*) *assms*(*2*) *eigvals*]
unfolding *es-R-def*
by *blast*

from *2* **obtain** $\mathcal{V}'$ **where** $\mathcal{V}'$: *?Q* $\mathcal{V}' \wedge (\forall\, v.$ *?P* $\mathcal{V}'$ $v \longrightarrow$ *?geq* $v)$ **by** *blast*
from *this 1* **obtain** v' **where** v': *?P* $\mathcal{V}'$ $v' \wedge$ *?leq* v' **by** *presburger*
moreover have *all-v-geq*: $\forall\, v.$ *?P* $\mathcal{V}'$ $v \longrightarrow$ *?geq* v **using** $\mathcal{V}'$ **by** *blast*
ultimately have ϱ A $v' =$ *es-R*!k **by** *fastforce*
hence *es-R*!$k \in$ *?*S_ϱ $\mathcal{V}'$ **using** v' **by** *fastforce*
moreover have $\forall\, x \in$ *?*S_ϱ $\mathcal{V}'.$ $x \geq$ *es-R*!k **using** *all-v-geq* **by** *blast*
ultimately have *es-R*!$k =$ *Inf* (*?*S_ϱ $\mathcal{V}'$) **by** (*smt* (*verit*) *rayleigh-bdd-below cInf-eq-minimum*)
moreover have $\bigwedge\mathcal{V}.$ *?Q* $\mathcal{V} \Longrightarrow$ *Inf* (*?*S_ϱ $\mathcal{V}$) $\leq$ *es-R*!k
proof−
fix $\mathcal{V}$
assume $*$: *?Q* $\mathcal{V}$
then obtain v **where** v: *?P* $\mathcal{V}$ $v \wedge$ *?leq* v **using** *1* **by** *presburger*
then have ϱ A $v \in$ *?*S_ϱ $\mathcal{V}$ **by** *blast*
then have *Inf* (*?*S_ϱ $\mathcal{V}$) $\leq$ ϱ A v
using *rayleigh-min-exists*[*OF* $*$] *assms*(*2*) *rayleigh-min courant-fischer-axioms* **by** *auto*
thus *Inf* (*?*S_ϱ $\mathcal{V}$) $\leq$ *es-R*!k **using** v **by** *linarith*
qed
ultimately have $*$: *es-R*!$k \in$ {*Inf* (*?*S_ϱ $\mathcal{V}$) | $\mathcal{V}$. *?Q* $\mathcal{V}$} $\wedge$ ($\forall\, x \in$ {*Inf* (*?*S_ϱ $\mathcal{V}$) | $\mathcal{V}$. *?Q* $\mathcal{V}$}. $x \leq$ *es-R*!k)
using $\mathcal{V}'$ **by** *blast*
hence *Sup* {*Inf* (*?*S_ϱ $\mathcal{V}$) | $\mathcal{V}$. *?Q* $\mathcal{V}$} = *es-R*!k **by** (*meson cSup-eq-maximum*)
moreover have *Sup* {*Inf* (*?*S_ϱ $\mathcal{V}$) | $\mathcal{V}$. *?Q* $\mathcal{V}$} = *maximin* (k + *1*)
unfolding *maximin-def rayleigh-min dimensional-def* **by** *blast*
ultimately show *es*!k = *maximin* (k + *1*)
by (*metis A-decomp assms*(*2*) *carrier-matD*(*1*) *diag-mat-length es-R length-map nth-map*)
show *maximin-defined* (k + *1*)
using $*$ **unfolding** *maximin-defined-def sup-defined-def rayleigh-min bdd-above-def* **by** *blast*
qed

end

lemma *courant-fischer-maximin*:
fixes A :: *complex mat*
assumes $n > 0$
assumes $k < n$
assumes $A \in$ *carrier-mat* n n
assumes *hermitian* A
assumes *eigvals-of* A *es*
assumes *sorted-wrt* ($\geq$) *es*
shows *es*!k = *hermitian-mat.maximin* n A (k + *1*) *hermitian-mat.maximin-defined* n A (k + *1*)
proof−
obtain Λ U **where** *real-diag-decomp* A Λ U
$\wedge$ *diag-mat* $\Lambda =$ *es*

```
    ∧ set es ⊆ Reals
    ∧ U ∈ carrier-mat n n
    ∧ Λ ∈ carrier-mat n n
   by (metis hermitian-real-diag-decomp-eigvals assms(3−5))
  then interpret cf: courant-fischer A n Λ U es
   using assms by unfold-locales
  show es!k = hermitian-mat.maximin n A (k + 1)
   using cf.courant-fischer-maximin(1)[OF assms(1) assms(2)] .
  show hermitian-mat.maximin-defined n A (k + 1)
   using cf.courant-fischer-maximin(2)[OF assms(1) assms(2)] .
qed

lemma maximin-minimax:
  fixes A :: complex mat
  assumes A ∈ carrier-mat n n
  assumes hermitian A
  assumes k < n
  shows hermitian-mat.maximin n (−A) (n − k) = − hermitian-mat.minimax n
A (k + 1)
    hermitian-mat.maximin-defined n (−A) (n − k) ⟹ hermitian-mat.minimax-defined
n A (k + 1)
proof−
  interpret hm: hermitian-mat n A using assms by unfold-locales
  interpret hm′: hermitian-mat n −A
   using assms by (simp add: hermitian-mat.intro negative-hermitian)

  define P where P ≡ λv V. v ≠ 0_v n ∧ v ∈ V ∧ vec-norm v = 1
  define Q where Q ≡ λV. hm′.dimensional V (n − k)

  have Inf {Sup {ϱ A v |v. P v V} |V. Q V} = − Sup (uminus‘{Sup {ϱ A v |v. P
v V} |V. Q V})
   using Inf-real-def .
  moreover have ∗: uminus‘{Sup {ϱ A v |v. P v V} |V. Q V} = {− Sup {ϱ A v
|v. P v V} |V. Q V}
   by blast
  moreover have ∗∗: {− Sup {ϱ A v |v. P v V} |V. Q V} = {Inf {ϱ (−A) v |v.
P v V} |V. Q V}
  proof−
   have ⋀V. Q V ⟹ − Sup {ϱ A v |v. P v V} = Inf {ϱ (−A) v |v. P v V}
   proof−
    fix V assume ∗: Q V
    have Inf {ϱ (−A) v |v. P v V} = − Sup (uminus‘{ϱ (−A) v |v. P v V})
     using Inf-real-def by fast
    moreover have uminus‘{ϱ (−A) v |v. P v V} = {− ϱ (−A) v |v. P v V} by
blast
    moreover have {− ϱ (−A) v |v. P v V} = {ϱ A v |v. P v V}
    proof−
     have ⋀v. P v V ⟹ − ϱ (− A) v = ϱ A v
     by (metis ∗ P-def Q-def assms(1) hm.dimensional-n-vec rayleigh-quotient-negative)
```

thus *?thesis* **by** *metis*
qed
ultimately show $-$ *Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}$ = *Inf* $\{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}$ **by** *presburger*
qed
thus *?thesis* **by** *force*
qed
ultimately have $-$ *Inf* $\{$*Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$ = *Sup* $\{$*Inf* $\{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$
by *auto*
moreover have $n - (k + 1) + 1 = n - k$ **using** *assms*(*3*) **by** *fastforce*
ultimately show *hermitian-mat.maximin n* $(-A)$ $(n - k)$ = $-$ *hermitian-mat.minimax n A* $(k + 1)$
by (*simp add*: *hm.minimax-def hm′.maximin-def hm′.rayleigh-min hm.rayleigh-max P-def Q-def*)

show *hermitian-mat.minimax-defined n A* $(k + 1)$ **if** *hermitian-mat.maximin-defined n* $(-A)$ $(n - k)$
proof$-$
have $\{$*Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\} \neq \{\}$
using *that*
unfolding *hm′.maximin-defined-def sup-defined-def hm′.rayleigh-min P-def Q-def*
by *fast*
moreover have *bdd-below* $\{$*Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$
proof$-$
have *bdd-above* $\{$*Inf* $\{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$
using *that*
unfolding *hm′.maximin-defined-def sup-defined-def hm′.rayleigh-min P-def Q-def*
by *argo*
hence *bdd-above* $\{-$ *Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$ **using** $**$ **by** *argo*
hence *bdd-below* $\{$*Sup* $\{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}.\ Q\ \mathcal{V}\}$ **by** (*smt* (*verit*, *best*) $*$ *bdd-above-uminus*)
thus *?thesis* **by** *blast*
qed
moreover have $n - k = n - (k + 1) + 1$ **using** *assms*(*3*) **by** *simp*
ultimately show *?thesis*
unfolding *hm.minimax-defined-def inf-defined-def hm.rayleigh-max P-def Q-def* **by** *algebra*
qed
qed

lemma *courant-fischer-minimax*:
fixes A :: *complex mat*
assumes $n > 0$
assumes $k < n$
assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*

```
  assumes eigvals-of A es
  assumes sorted-wrt (≥) es
  shows es!k = hermitian-mat.minimax n A (k + 1)
       hermitian-mat.minimax-defined n A (k + 1)
proof−
  define A′ where A′ ≡ − A
  define es′ where es′ = rev (map (λx. −x) es)
  have A′-hermitian: hermitian A′ using negative-hermitian A′-def assms by blast
  moreover have eigvals-of A′ es′
    using neg-mat-eigvals A′-def assms(3) assms(5) es′-def by blast
  moreover have A′-dim: A′ ∈ carrier-mat n n by (simp add: A′-def assms(3))
  moreover have sorted-wrt (≥) es′
  proof−
    let ?l = (map (λx. −x) es)
    { fix i j assume i < j j < length ?l
      moreover then have es!i ≥ es!j using assms(6) sorted-wrt-iff-nth-less[of
(≥) es] by force
     ultimately have ?l!i ≤ ?l!j by simp
    }
    hence sorted-wrt (≤) ?l by (metis sorted-wrt-iff-nth-less)
    thus ?thesis by (simp add: sorted-wrt-rev es′-def)
  qed
  ultimately have ∗: es′!(n − k − 1) = hermitian-mat.maximin n A′ (n − k)
     ∧ hermitian-mat.maximin-defined n A′ (n − k)
    using courant-fischer-maximin[of n n − k − 1 A′ es′] assms by (simp add:
Suc-diff-Suc)
  moreover have es!k = − es′!(n − k − 1)
  proof−
   have n − k − 1 < length es using assms(2) assms(3) assms(5) eigvals-poly-length
by force
    moreover have length es = n
     using assms(3) assms(5) by auto
    ultimately have es!k = (rev es)!(n − k − 1)
    using rev-nth[of n − k − 1 es] by (simp add: Suc-diff-Suc assms(2) le-simps(1))
    also have ... = rev (map (λx. −x) (map (λx. −x) es))!(n − k − 1)
     by simp
    also have ... = − es′!(n − k − 1)
      by (metis ‹n − k − 1 < length es› es′-def length-map length-rev nth-map
rev-map)
    finally show ?thesis .
  qed
  ultimately show es!k = hermitian-mat.minimax n A (k + 1)
     hermitian-mat.minimax-defined n A (k + 1)
    apply (simp add: A′-def assms(2) assms(3) assms(4) maximin-minimax(1))
   using A′-def ∗ assms(2) assms(3) assms(4) maximin-minimax(2) by blast
qed
```

15.1 Theorem Statement

theorem *courant-fischer*:
fixes A :: *complex mat*
assumes $n > 0$
assumes $k < n$
assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A es*
assumes *sorted-wrt* $(\geq)$ *es*
shows *es*$!k$ = *hermitian-mat.minimax n A* $(k + 1)$
es$!k$ = *hermitian-mat.maximin n A* $(k + 1)$
hermitian-mat.minimax-defined n A $(k + 1)$
hermitian-mat.maximin-defined n A $(k + 1)$
using *assms courant-fischer-minimax* **apply** *algebra*
using *assms courant-fischer-maximin* **apply** *algebra*
using *assms courant-fischer-minimax* **apply** *algebra*
using *assms courant-fischer-maximin* **by** *algebra*

16 Cauchy Eigenvalue Interlacing Theorem

We follow the proof given in this set of lecture notes by Dr. David Bindel: https://www.cs.cornell.edu/courses/cs6210/2019fa/lec/2019-11-04.pdf

16.1 Theorem Statement and Proof

theorem *cauchy-eigval-interlacing*:
fixes A W :: *complex mat*
assumes $n > 0$
assumes $j < n$
assumes $m \leq n$
assumes $m > 0$
assumes $j < m$

assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $W \in$ *carrier-mat n m*
assumes $W^H * W = 1_m\ m$
assumes *inj-on* $(\lambda v.\ W *_v v)$ $(carrier\text{-}vec\ m)$

defines $B \equiv W^H * A * W$
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β
shows $\alpha!(n-m+j) \leq \beta!j$ $\beta!j \leq \alpha!j$
proof−
interpret A: *hermitian-mat n A* **using** *assms hermitian-mat-def* **by** *presburger*

interpret B: *hermitian-mat* m B
proof *unfold-locales*
show $B \in$ *carrier-mat* m m **unfolding** *B-def* **using** *assms* **by** *fastforce*
show *hermitian* B
proof−
have $B^H = (W^H * A * W)^H$ **unfolding** *B-def* **by** *blast*
also have ... = $W^H * A * (W^H)^H$
by (*smt* (*verit*, *ccfv-threshold*) *assms adjoint-dim′ adjoint-is-conjugate-transpose adjoint-mult assoc-mult-mat hermitian-def mult-carrier-mat*)
also have ... = $W^H * A * W$ **by** (*simp add*: *transpose-conjugate*)
also have ... = B **unfolding** *B-def* **by** *blast*
finally show *?thesis* **by** (*simp add*: *adjoint-is-conjugate-transpose hermitian-def*)
qed
qed

have α: $\forall k < n.\ \alpha!k$ = *hermitian-mat.minimax* n A $(k + 1)$
$\wedge\ \alpha!k$ = *hermitian-mat.maximin* n A $(k + 1)$
$\wedge$ *hermitian-mat.minimax-defined* n A $(k + 1)$
$\wedge$ *hermitian-mat.maximin-defined* n A $(k + 1)$
using *courant-fischer assms* **by** *presburger*

let *?lhs* = $\lambda\mathcal{V}.\ \{\varrho\ B\ v\ |v.\ v \neq 0_v\ m \wedge v \in \mathcal{V} \wedge$ *vec-norm* $v = 1\}$
let *?rhs* = $\lambda\mathcal{V}.\ \{\varrho\ A\ v\ |v.\ v \neq 0_v\ n \wedge v \in (*_v)\ W\ `\ \mathcal{V} \wedge$ *vec-norm* $v = 1\}$
have *rayleigh-sets-eq*: $\bigwedge\mathcal{V}\ k.$ *B.dimensional* $\mathcal{V}\ k \Longrightarrow$ *?lhs* $\mathcal{V}$ = *?rhs* $\mathcal{V}$
proof−
fix $\mathcal{V}\ k$ **assume** $*$: *B.dimensional* $\mathcal{V}\ k$
have *?lhs* $\mathcal{V} \subseteq$ *?rhs* $\mathcal{V}$
proof
let *?lhs* = *?lhs* $\mathcal{V}$
let *?rhs* = *?rhs* $\mathcal{V}$
fix x **assume** $**$: $x \in$ *?lhs*
then obtain v **where** v: $v \neq 0_v\ m \wedge v \in \mathcal{V} \wedge$ *vec-norm* $v = 1 \wedge x = \varrho\ B\ v$ **by** *blast*
let *?c* = 1 / (*vec-norm* $(W *_v v)$)
define v' **where** $v' \equiv$ *?c* $\cdot_v$ $(W *_v v)$
have $\varrho_c\ B\ v = ((B *_v v) \cdot c\ v)$ / $(v \cdot c\ v)$ **by** *simp*
also have ... = $((W^H *_v (A * W *_v v)) \cdot c\ v)$ / $(v \cdot c\ v)$
unfolding *assms*(*9*)
by (*smt* (*verit*, *best*) *assms* $*$ *A.dim-is B.dimensional-n-vec More-Matrix.carrier-vec-conjugate assoc-mult-mat assoc-mult-mat-vec carrier-vecD dim-mult-mat-vec hermitian-def index-mult-mat*(*2*) *mult-carrier-mat transpose-carrier-mat v*)
also have ... = $((A * W *_v v) \cdot c\ (W *_v v))$ / $(v \cdot c\ v)$
by (*metis* (*no-types*, *lifting*) *assms*(*6*,*10*) $*$ *A.dim-is B.hermitian-mat-axioms cscalar-prod-conjugate-transpose hermitian-mat.dimensional-n-vec mult-carrier-mat mult-mat-vec-carrier v*)
also have ... = $((A * W *_v v) \cdot c\ (W *_v v))$ / $(v \cdot c\ ((1_m\ m) *_v v))$
using $*$ v *B.dimensional-n-vec* **by** *force*
also have ... = $((A *_v (W *_v v)) \cdot c\ (W *_v v))$ / $(v \cdot c\ (W^H *_v (W *_v v)))$

by (*metis* ∗ *assms*(6,10,11) *B.dimensional-n-vec adjoint-dim′ adjoint-is-conjugate-transpose assoc-mult-mat-vec v*)
also have ... = ((*A* ∗$_v$ (*W* ∗$_v$ *v*)) ·*c* (*W* ∗$_v$ *v*)) / ((*W* ∗$_v$ *v*) ·*c* ((*W* ∗$_v$ *v*)))
by (*metis* ∗ *assms*(10) *B.dimensional-n-vec adjoint-def-alter adjoint-is-conjugate-transpose mult-mat-vec-carrier v*)
also have ... = ϱ$_c$ *A* (*W* ∗$_v$ *v*) **by** *simp*
finally have ϱ *B v* = ϱ *A* (*W* ∗$_v$ *v*) **by** *fastforce*
also have ... = ϱ *A v′*
unfolding *v′-def*
by (*smt* (*verit*, *best*) *assms*(6,10,11) ∗ *rayleigh-quotient-scale B.hermitian-mat-axioms adjoint-is-conjugate-transpose div-by-1 hermitian-mat.dimensional-n-vec inner-prod-mult-mat-vec-right mult-mat-vec-carrier of-real-hom.hom-one one-mult-mat-vec v vec-norm-def*)
finally have *x* = ϱ *A v′* **using** *v* **by** *fast*
moreover have *vec-norm v′* = *1*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(10,11) ∗ *B.dimensional-n-vec adjoint-dim′ adjoint-is-conjugate-transpose assoc-mult-mat-vec carrier-matD*(*1*) *cscalar-prod-adjoint dim-mult-mat-vec div-by-1 one-mult-mat-vec scalar-vec-one v v′-def vec-norm-def*)
moreover have *v′* ∈ (∗$_v$) *W* ‘ $\mathcal{V}$
by (*smt* (*z3*) *assms*(10,11) ∗ *B.dimensional-n-vec Setcompr-eq-image adjoint-is-conjugate-transpose div-by-1 inner-prod-mult-mat-vec-left mem-Collect-eq one-mult-mat-vec one-smult-vec v v′-def vec-norm-def*)
moreover have *v′* ≠ 0$_v$ *n*
by (*metis A.add.one-closed calculation*(*2*) *one-neq-zero vec-norm-zero*)
ultimately show *x* ∈ *?rhs* **by** *blast*
qed
moreover have *?rhs* $\mathcal{V}$ ⊆ *?lhs* $\mathcal{V}$
proof
let *?lhs* = *?lhs* $\mathcal{V}$
let *?rhs* = *?rhs* $\mathcal{V}$
fix *x* **assume** ∗∗: *x* ∈ *?rhs*
then obtain *v′* **where** *v′*: *v′* ≠ 0$_v$ *n* ∧ *v′* ∈ (∗$_v$) *W* ‘ $\mathcal{V}$ ∧ *vec-norm v′* = *1* ∧ *x* = ϱ *A v′* **by** *blast*
then obtain *u* **where** *u*: *W* ∗$_v$ *u* = *v′* ∧ *u* ≠ 0$_v$ *m* ∧ *u* ∈ $\mathcal{V}$
by (*smt* (*verit*, *ccfv-threshold*) *assms*(6,10,11) *B.dimensional-n B.smult-l-null Setcompr-eq-image* ∗ *adjoint-is-conjugate-transpose assms*(15) *assms*(7) *csqrt-1 inner-prod-mult-mat-vec-left inner-prod-smult-left-right mem-Collect-eq mult-eq-0-iff one-mult-mat-vec one-neq-zero power2-csqrt subset-eq vec-norm-def*)
hence *u-m*: *u* ∈ *carrier-vec m* **using** ∗ *B.dimensional-n-vec* **by** *blast*
let *?c* = *1* / *vec-norm u*
define *v* **where** *v* ≡ *?c* ·$_v$ *u*
have ϱ$_c$ *B v* = ((*B* ∗$_v$ *v*) ·*c v*) / (*v* ·*c v*) **by** *simp*
also have ... = ((*W*^{H} ∗$_v$ (*A* ∗ *W* ∗$_v$ *v*)) ·*c v*) / (*v* ·*c v*)
unfolding *assms*(*9*)
by (*smt* (*verit*, *ccfv-threshold*) *assms*(6,10,11,13) *u-m A.dim-is B.smult-one adjoint-is-conjugate-transpose adjoint-mult assms*(15) *assms*(4) *assms*(7) *cscalar-prod-conjugate-transpose divide-self-if hermitian-def inner-prod-mult-mat-vec-right mult-carrier-mat mult-mat-vec-carrier one-mult-mat-vec u v′ v-def vec-norm-def*)
also have ... = ((*A* ∗ *W* ∗$_v$ *v*) ·*c* (*W* ∗$_v$ *v*)) / (*v* ·*c v*)
by (*smt* (*verit*, *ccfv-threshold*) *assms*(6,10,11) ∗ *A.dim-is B.dimensional-n-vec*

Complex-Matrix.adjoint-adjoint adjoint-dim′ adjoint-is-conjugate-transpose assms(7) *carrier-matD*(1) *carrier-vecD cscalar-prod-adjoint dim-mult-mat-vec index-mult-mat*(2) *index-smult-vec*(2) *u v-def*)
also have ... = $((A * W *_v v) \cdot c (W *_v v)) / (v \cdot c ((1_m m) *_v v))$
by (*simp add*: *u-m v-def*)
also have ... = $((A *_v (W *_v v)) \cdot c (W *_v v)) / (v \cdot c (W^H *_v (W *_v v)))$
by (*metis assms*(6,10,11) *B.smult-one adjoint-dim′ adjoint-is-conjugate-transpose assoc-mult-mat-vec div-by-1 inner-prod-mult-mat-vec-left one-mult-mat-vec u u-m v′ v-def vec-norm-def*)
also have ... = $((A *_v (W *_v v)) \cdot c (W *_v v)) / ((W *_v v) \cdot c ((W *_v v)))$
by (*metis assms*(10) *B.smult-closed u-m v-def adjoint-def-alter adjoint-is-conjugate-transpose mult-mat-vec-carrier*)
also have ... = $\varrho_c\ A\ (W *_v v)$ **by** *simp*
finally have $\varrho\ B\ v = \varrho\ A\ (W *_v v)$ **by** *fastforce*
also have ... = $\varrho\ A\ (W *_v (?c \cdot_v u))$ **using** *v-def* **by** *blast*
also have ... = $\varrho\ A\ (?c \cdot_v (W *_v u))$ **by** (*metis assms*(10) *mult-mat-vec u u-m*)
also have ... = $\varrho\ A\ v'$
by (*metis assms*(10,11) *u-m adjoint-is-conjugate-transpose div-by-1 inner-prod-mult-mat-vec-left one-mult-mat-vec one-smult-vec u v′ vec-norm-def*)
finally have $x = \varrho\ B\ v$ **using** v' **by** *presburger*
moreover have *vec-norm v* = *1*
by (*metis carrier-dim-vec csqrt-1 normalized-vec-norm u u-m v-def vec-norm-def vec-normalize-def*)
moreover then have $v \neq 0_v\ m$ **using** *vec-norm-zero* **by** *force*
moreover have $v \in \mathcal{V}$
by (*metis assms*(10,11) *B.smult-one adjoint-is-conjugate-transpose div-by-1 inner-prod-mult-mat-vec-right one-mult-mat-vec u u-m v′ v-def vec-norm-def*)
ultimately show *x* ∈ *?lhs* **by** *blast*
qed
ultimately show *?lhs* $\mathcal{V}$ = *?rhs* $\mathcal{V}$ **by** *blast*
qed

let *?lhs-min* = $\lambda k.\ \{A.rayleigh\text{-}min\ ((\lambda v.\ W *_v v)\ \mathcal{V})\ |\ \mathcal{V}.\ B.dimensional\ \mathcal{V}\ k\}$
let *?rhs-min* = $\lambda k.\ \{A.rayleigh\text{-}min\ \mathcal{V}\ |\ \mathcal{V}.\ A.dimensional\ \mathcal{V}\ k\}$
let *?lhs-max* = $\lambda k.\ \{A.rayleigh\text{-}max\ ((\lambda v.\ W *_v v)\ \mathcal{V})\ |\ \mathcal{V}.\ B.dimensional\ \mathcal{V}\ k\}$
let *?rhs-max* = $\lambda k.\ \{A.rayleigh\text{-}max\ \mathcal{V}\ |\ \mathcal{V}.\ A.dimensional\ \mathcal{V}\ k\}$
have *rayleigh-sets-eq′*: $\bigwedge k.$ *?lhs-min k* ⊆ *?rhs-min k* $\bigwedge k.$ *?lhs-max k* ⊆ *?rhs-max k*
proof−
fix *k*
have ∗: $\{(\lambda v.\ W *_v v)\ \mathcal{V}\ |\ \mathcal{V}.\ B.dimensional\ \mathcal{V}\ k\} \subseteq \{\mathcal{V}\ |\ \mathcal{V}.\ A.dimensional\ \mathcal{V}\ k\}$
proof(*rule subsetI*)
fix $\mathcal{V}$ **assume** ∗: $\mathcal{V} \in \{(\lambda v.\ W *_v v)\ \mathcal{V}\ |\ \mathcal{V}.\ B.dimensional\ \mathcal{V}\ k\}$
have *A.dimensional* $\mathcal{V}$ *k*
proof−
let *?f* = $\lambda v.\ W *_v v$
obtain $\mathcal{V}'$ **where** $\mathcal{V}'$: $\mathcal{V}$ = *?f* $\mathcal{V}'$ ∧ *B.dimensional* $\mathcal{V}'$ *k* **using** ∗ **by** *blast*

```
      then obtain vs' where vs': V' = B.span vs'
          ∧ card vs' = k
          ∧ vs' ⊆ carrier-vec m
          ∧ B.lin-indpt vs'
        unfolding B.dimensional-def by presburger
      define vs where vs = ?f`vs'

      interpret W: linear-map
          class-ring
          (module-vec TYPE(complex) m)
          (module-vec TYPE(complex) n)
          (λv. W *v v)
        using linear-map-mat[OF assms(10)] .

      have inj: inj-on ?f (carrier B.V) using assms(12) by fastforce
      have 1: vs' ⊆ carrier B.V using B.cV vs' by argo
      have V = A.span vs using W.image-span by (simp add: B.li-le-dim(1) V'
vs' vs-def)
      moreover have card vs = k
      proof−
        have card vs' = k using vs' by blast
        moreover have vs' ⊆ carrier B.V by (simp add: vs')
        ultimately show ?thesis by (metis inj card-image subset-inj-on vs-def)
      qed
      moreover have vs ⊆ carrier-vec n
      proof(rule subsetI)
        fix v assume v ∈ vs
        then obtain v' where v' ∈ vs' ∧ ?f v' = v using vs-def by blast
        thus v ∈ carrier-vec n using assms(10) mult-mat-vec-carrier vs' by blast
      qed
      moreover have A.lin-indpt vs
        using W.inj-image-lin-indpt[OF inj] vs' vs-def B.fin-dim-li-fin by auto
      ultimately show ?thesis unfolding A.dimensional-def by blast
    qed
    thus V ∈ {V | V. A.dimensional V k} by blast
  qed

  from ∗ show ?lhs-min k ⊆ ?rhs-min k by blast
  from ∗ show ?lhs-max k ⊆ ?rhs-max k by blast
qed

have β!j = B.maximin (j + 1)
  using courant-fischer(2)[OF assms(4,5) B.dim-is B.is-herm assms(14,15)] .
also have ... = Sup {A.rayleigh-min ((λv. W *v v) V) | V. B.dimensional V (j
+ 1)}
  unfolding B.maximin-def B.rayleigh-min A.rayleigh-min
  by (metis (no-types, lifting) rayleigh-sets-eq)
also have ... ≤ α!j
proof−
```

let *?lhs* = {*A.rayleigh-min* ((λ*v*. *W* ∗$_v$ *v*) *V*) | *V*. *B.dimensional V* (*j* + *1*)}
let *?rhs* = {*A.rayleigh-min V* | *V*. *A.dimensional V* (*j* + *1*)}
have α_j: α!*j* = *A.maximin* (*j* + *1*)
using *courant-fischer*(*2*)[*OF assms*(*1*,*2*,*6*,*7*,*8*,*9*)] **by** *argo*
hence *Re*-α_j: *Re* (α!*j*) = *Sup ?rhs*
by (*simp add*: *A.maximin-def image-Collect*)

hence *?lhs* ⊆ *?rhs* **using** *rayleigh-sets-eq*′(*1*) **by** *presburger*
moreover have *?lhs* ≠ {}
proof−
let *?vs* = *set* (*take* (*j* + *1*) (*unit-vecs m*))
have *B.dimensional* (*B.span ?vs*) (*j* + *1*)
proof−
have *card ?vs* = *j* + *1*
by (*metis B.unit-vecs-distinct distinct-take B.unit-vecs-length Groups.add-ac*(*2*) *One-nat-def Suc-leI add-0-right add-Suc-right assms*(*5*) *diff-zero distinct-card length-take length-upt take-upt*)
moreover have *B.lin-indpt ?vs*
by (*meson B.basis-def B.supset-ld-is-ld B.unit-vecs-basis set-take-subset*)
moreover have *?vs* ⊆ *carrier-vec m*
by (*meson in-set-takeD subset-code*(*1*) *unit-vecs-carrier*)
ultimately show *?thesis* **unfolding** *B.dimensional-def* **by** *blast*
qed
thus *?thesis* **by** *blast*
qed
moreover have *bdd-above ?rhs*
using α
by (*simp add*: *A.maximin-defined-def assms*(*2*) *sup-defined-def*)
ultimately have *Sup ?lhs* ≤ *Sup ?rhs* **by** (*meson cSup-subset-mono*)
thus *?thesis* **using** *Re*-α_j α_j *less-eq-complex-def* **by** *force*
qed
finally show β!*j* ≤ α!*j* **.**

let *?j*′ = *n* − *m* + *j*
have *j*′*-le*: *?j*′ < *n* **using** *assms* **by** *linarith*
hence *j*′*-eq*: *n* − *?j*′ + *1* = *m* − *j* + *1* **using** *assms*(*3*) **by** *force*

have α!*?j*′ = *A.minimax* (*?j*′ + *1*) **using** *courant-fischer*(*1*)[*OF assms*(*1*) *j*′*-le assms*(*6*,*7*,*8*,*9*)] **.**
also have ... ≤ *Inf* {*A.rayleigh-max* ((λ*v*. *W* ∗$_v$ *v*) *V*) | *V*. *B.dimensional V* (*n* − *?j*′)}
proof−
let *?rhs* = {*A.rayleigh-max* ((λ*v*. *W* ∗$_v$ *v*) *V*) | *V*. *B.dimensional V* (*n* − *?j*′)}
let *?lhs* = {*A.rayleigh-max V* | *V*. *A.dimensional V* (*n* − *?j*′)}

have *?rhs* ⊆ *?lhs* **using** *rayleigh-sets-eq*′(*2*) **by** *presburger*
moreover have *?rhs* ≠ {}
proof−
let *?vs* = *set* (*take* (*n* − *?j*′) (*unit-vecs m*))

```
      have B.dimensional (B.span ?vs) (n − ?j′)
      proof−
        have card ?vs = n − ?j′
       by (metis B.unit-vecs-distinct distinct-take B.unit-vecs-length Groups.add-ac(2)
One-nat-def add-Suc-right diff-zero distinct-card length-take length-upt take-upt diff-add-inverse2
diff-le-self j′-eq)
        moreover have B.lin-indpt ?vs
          by (meson B.basis-def B.supset-ld-is-ld B.unit-vecs-basis set-take-subset)
        moreover have ?vs ⊆ carrier-vec m
          by (meson in-set-takeD subset-code(1) unit-vecs-carrier)
        ultimately show ?thesis unfolding B.dimensional-def by blast
      qed
      thus ?thesis by blast
    qed
    moreover have bdd-below ?lhs
    proof−
      have n − m + j < n using j′-le by blast
      moreover then have n − ((n − m + j) + 1) + 1 = n − (n − m + j) by
linarith
      ultimately show ?thesis
        using α
        unfolding A.minimax-defined-def inf-defined-def
        by (smt (verit, del-insts) Collect-cong)
    qed
    ultimately have Inf ?lhs ≤ Inf ?rhs
      using cInf-superset-mono[of ?rhs ?lhs] by fast
    moreover have n − (n − m + j) = n − (n − m + j + 1) + 1 using assms
by linarith
    ultimately show ?thesis by (simp add: less-eq-complex-def A.minimax-def)
  qed
  also have ... = Inf {A.rayleigh-max ((λv. W ∗v v) V) | V. B.dimensional V (m
− j)}
    using assms(3) by force
  also have ... = B.minimax (j + 1)
  proof−
    have m − j = m − (j + 1) + 1 by (simp add: Suc-diff-Suc assms(5))
    thus ?thesis
      unfolding B.minimax-def B.rayleigh-max A.rayleigh-max
      by (metis (mono-tags, lifting) rayleigh-sets-eq)
  qed
  also have ... = β!j
    using courant-fischer(1)[OF assms(4,5) B.dim-is B.is-herm assms(14,15)] by
argo
  finally show α!(n − m + j) ≤ β!j .
qed

corollary cauchy-eigval-interlacing-alt:
  fixes A W :: complex mat
  assumes n > 0
```

assumes $j < n$
assumes $m \leq n$
assumes $m > 0$
assumes $j < m$

assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $W \in$ *carrier-mat n m*
assumes $W^H * W = 1_m\ m$
assumes *inj-on* $(\lambda v.\ W *_v v)$ *(carrier-vec m)*

defines $B \equiv W^H * A * W$
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β

shows $\beta!j \in \{\alpha!(n-m+j)..\alpha!j\}$
using *cauchy-eigval-interlacing assms* **by** *presburger*

16.2 Principal Submatrix Corollaries

corollary *ps-eigval-interlacing*:
fixes A :: *complex mat*
fixes k
assumes $n > 0$
assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $I \subseteq \{..<n\}$
assumes $I \neq \{\}$
defines $B \equiv$ *submatrix A I I*
defines $m \equiv$ *card I*
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β

assumes $j < m$
shows $\alpha!(n-m+j) \leq \beta!j\ \beta!j \leq \alpha!j$
proof−
have *0*: $n > 0$ **using** *assms(1)* **.**
have *1*: $j < n$
by (*metis assms(6,12) card-lessThan card-mono finite-lessThan m-def not-le not-less-iff-gr-or-eq order.strict-trans*)
have *2*: $m \leq n$
using *assms(6,9) atLeast0LessThan subset-eq-atLeast0-lessThan-card* **by** *presburger*

have *3*: $m > 0$ **using** *assms(12)* **by** *linarith*
have *4*: $j < m$ **using** *assms(12)* **.**
have *5*: $A \in$ *carrier-mat n n* **using** *assms(2)* **.**
have *6*: *hermitian A* **using** *assms(3)* **.**
have *7*: *eigvals-of A* α **using** *assms(4)* **.**
have *8*: *sorted-wrt* $(\geq)$ α **using** *assms(5)* **.**

obtain *Inm* **where** *Inm*:
$B = Inm^H * A * Inm$
$Inm^H * Inm = 1_m\ m$
$Inm \in$ *carrier-mat n m*
inj-on $((*_v)\ Inm)$ *(carrier-vec m)*
using *submatrix-as-matrix-prod-obt assms* **by** *blast*
hence *9*: *eigvals-of* $(Inm^H * A * Inm)$ β **using** *assms(10)* **by** *blast*
have *10*: *sorted-wrt* $(\lambda x\ y.\ y \leq x)$ β **by** (*simp add*: *assms(11)*)

have $*$: $\beta!j \in \{\alpha!(n-m+j)..\alpha!j\}$
using *cauchy-eigval-interlacing-alt*[*OF 0 1 2 3 4 5 6 7 8 Inm(3,2,4) 9 10*] **.**

from $*$ **show** $\alpha!(n-m+j) \leq \beta!j$ **by** *presburger*
from $*$ **show**$\beta!j \leq \alpha!j$ **by** *presburger*
qed

corollary *ps-eigval-interlacing-alt*:
fixes *A* :: *complex mat*
fixes *k*
assumes *n* > *0*
assumes *A* ∈ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $I \subseteq \{..<n\}$
assumes $I \neq \{\}$
defines $B \equiv$ *submatrix A I I*
defines $m \equiv$ *card I*
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β

assumes $j < m$
shows $\beta!j \in \{\alpha!(n-m+j)..\alpha!j\}$
using *ps-eigval-interlacing assms* **by** *presburger*

16.3 Leading Principal Submatrix Corollaries

corollary *lps-eigval-interlacing*:
fixes *A* :: *complex mat*
fixes *k*
assumes *n* > *0*

assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $0 < m$
assumes $m \leq n$
defines $B \equiv$ *lps A m*
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β

assumes $j < m$
shows $\alpha!(n-m+j) \leq \beta!j$ $\beta!j \leq \alpha!j$
using *ps-eigval-interlacing*(*1*)[*OF assms*(*1*,*2*,*3*,*4*,*5*), *of* {..<*m*}] *assms* **apply** *auto*[*1*]
using *ps-eigval-interlacing*(*2*)[*OF assms*(*1*,*2*,*3*,*4*,*5*), *of* {..<*m*}] *assms* **by** *auto*

corollary *lps-eigval-interlacing-alt*:
fixes *A* :: *complex mat*
fixes *k*
assumes $n > 0$
assumes $A \in$ *carrier-mat n n*
assumes *hermitian A*
assumes *eigvals-of A* α
assumes *sorted-wrt* $(\geq)$ α

assumes $0 < m$
assumes $m \leq n$
defines $B \equiv$ *lps A m*
assumes *eigvals-of B* β
assumes *sorted-wrt* $(\geq)$ β

assumes $j < m$
shows $\beta!j \in \{\alpha!(n-m+j)..\alpha!j\}$
using *lps-eigval-interlacing assms* **by** *presburger*

end

References

[1] D. Bindel. Lecture notes. https://www.cs.cornell.edu/courses/cs6210/2019fa/lec/2019-11-04.pdf, 2019. CS6210 at Cornell University.