

# Two Theorems on Hermitian Matrices

Sage Binder and Zilin Jiang

June 6, 2026

## 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

<b>1 Polynomial Factorization</b>	<b>4</b>
<b>2 Vector Normalization</b>	<b>5</b>
<b>3 Determinant, Invertability, and Eigenvalues</b>	<b>5</b>
<b>4 Quadratic Form</b>	<b>9</b>
<b>5 Leading Principal Submatrix</b>	<b>9</b>
<b>6 Hermitian Matrix</b>	<b>11</b>
6.1 Locale for Complex Hermitian Matrices . . . . .	12
<b>7 Matrix Conjugate</b>	<b>15</b>

<b>8</b>	<b>Block Matrix</b>	<b>17</b>
8.1	Schur's Formula . . . . .	23
<b>9</b>	<b>Positive Definite Matrix</b>	<b>26</b>
<b>10</b>	<b>Matrix-Vector Multiplication</b>	<b>28</b>
<b>11</b>	<b>Module Span</b>	<b>30</b>
<b>12</b>	<b>Module Homomorphism, Linear Combination, and Span</b>	<b>34</b>
<b>13</b>	<b>Vector Summation</b>	<b>37</b>
<b>14</b>	<b>Vector Space</b>	<b>41</b>
<b>15</b>	<b>Linear Map</b>	<b>45</b>
<b>16</b>	<b>Matrix as Linear Map</b>	<b>47</b>
<b>17</b>	<b>Matrix Isometry</b>	<b>48</b>
<b>18</b>	<b>Compression</b>	<b>48</b>
<b>19</b>	<b>Submatrix</b>	<b>49</b>
19.1	Submatrix of Injective Function on Indices, as Compression . . . . .	49
19.2	Submatrix of <i>pick</i> Function, as Compression . . . . .	50
<b>20</b>	<b>Schur Decomposition</b>	<b>52</b>
<b>21</b>	<b>Rayleigh Quotient</b>	<b>54</b>
<b>22</b>	<b>Courant-Fischer Minimax Theorem</b>	<b>56</b>
22.1	Max-Min Statement . . . . .	66
22.2	Min-Max Statement . . . . .	67
22.3	Theorem Statement . . . . .	70
<b>23</b>	<b>Cauchy Eigenvalue Interlacing Theorem</b>	<b>70</b>
23.1	Theorem Statement and Proof . . . . .	70
23.2	Principal Submatrix Corollaries (Using <i>pick</i> Function) . . . . .	74
23.3	Principal Submatrix Corollary (as Injective Function on Indices)	75
<b>24</b>	<b>Theorem Statements (Outside Locale)</b>	<b>76</b>
<b>25</b>	<b>Sylvester's Criterion Setup</b>	<b>78</b>

<b>26 Sylvester’s Criterion</b>	<b>78</b>
26.1 Forward Implication . . . . .	78
26.2 Reverse Implication . . . . .	86
26.3 Theorem Statement . . . . .	88
26.4 Theorem Statement (Outside Locale) . . . . .	89
<b>27 Code</b>	<b>89</b>
<b>theory</b> <i>Misc-Matrix-Results</i>	
<b>imports</b> <i>Commuting-Hermitian.Commuting-Hermitian</i>	
<i>BenOr-Kozen-Reif.More-Matrix</i>	
<i>Jordan-Normal-Form.Spectral-Radius</i>	
<i>Jordan-Normal-Form.DL-Rank-Submatrix</i>	
<i>Jordan-Normal-Form.Jordan-Normal-Form-Uniqueness</i>	
<i>Jordan-Normal-Form.VS-Connect</i>	
<i>QHLProver.Complex-Matrix</i>	
<i>Fishers-Inequality.Matrix-Vector-Extras</i>	
<i>Complex-Bounded-Operators.Extra-Jordan-Normal-Form</i>	
<i>Hermite-Lindemann.Misc-HLW</i>	
<b>begin</b>	
<b>hide-type (open)</b>	
<i>Finite-Cartesian-Product.vec</i>	
<i>Matrix-Legacy.vec</i>	
<i>Matrix-Legacy.mat</i>	
<b>hide-fact (open)</b>	
<i>Matrix-Legacy.mat-def</i>	
<i>Finite-Cartesian-Product.mat-def</i>	
<i>Finite-Cartesian-Product.row-def</i>	
<i>Matrix-Legacy.row-def</i>	
<i>Matrix-Legacy.row-def</i>	
<i>Matrix-Legacy.col-def</i>	
<i>Determinants.det-def</i>	
<i>Determinants.det-def</i>	
<i>Finite-Cartesian-Product.vec-def</i>	
<i>Matrix-Legacy.vec-def</i>	
<i>Linear-Algebra.adjoint-def</i>	
<b>hide-const (open)</b>	
<i>Matrix-Legacy.mat</i>	
<i>Finite-Cartesian-Product.mat</i>	
<i>Finite-Cartesian-Product.row</i>	
<i>Matrix-Legacy.row</i>	
<i>Matrix-Legacy.row</i>	
<i>Matrix-Legacy.col</i>	
<i>Determinants.det</i>	
<i>Determinants.det</i>	
<i>Finite-Cartesian-Product.vec</i>	

*Matrix-Legacy.vec*  
*Linear-Algebra.adjoint*

**no-notation** *fps-nth* (**infixl** \$ 75)

**unbundle** *no inner-syntax*  
**unbundle** *no vec-syntax*

**hide-const** (**open**) *Spectral-Radius.spectrum*  
**hide-fact** (**open**) *Spectral-Radius.spectrum-def*

## 1 Polynomial Factorization

**lemma** *idom-poly-factor-unique-aux*:

**fixes** *rs rs' :: 'a::idom list*

**assumes** *eq: (∏ a←rs. [:a, 1:]) = (∏ a←rs'. [:a, 1:])*

**shows** *mset rs ⊆# mset rs'*

**using** *eq*

**by** (*metis count-list-eq-length-filter count-mset poly-linear-exp-linear-factors poly-linear-exp-linear-factors-rev subseteq-mset-def*)

**lemma** *idom-poly-factor-unique*:

**fixes** *rs rs' :: 'a::idom list*

**assumes** *eq: (∏ a←rs. [:a, 1:]) = (∏ a←rs'. [:a, 1:])*

**shows** *mset rs = mset rs'*

**using** *idom-poly-factor-unique-aux[OF eq] idom-poly-factor-unique-aux[OF eq[symmetric]]*

**by** *order*

**lemma** *idom-poly-factor-unique'*:

**fixes** *rs rs' :: 'a::idom list*

**assumes** *eq: (∏ a←rs. [:- a, 1:]) = (∏ a←rs'. [:- a, 1:])*

**shows** *mset rs = mset rs'*

**proof** –

**have**  $(\prod a \leftarrow rs. [:- a, 1:]) = (\prod a \leftarrow (\text{map } \text{uminus } rs). [:a, 1:])$

**by** (*simp add: o-def pCons-def*)

**moreover have**  $(\prod a \leftarrow rs'. [:- a, 1:]) = (\prod a \leftarrow (\text{map } \text{uminus } rs'). [:a, 1:])$

**by** (*simp add: o-def pCons-def*)

**ultimately have**  $mset (\text{map } \text{uminus } rs) = mset (\text{map } \text{uminus } rs^\wedge)$

**using** *idom-poly-factor-unique[of map uminus rs map uminus rs'] eq* **by** *argo*

**moreover have**  $mset (\text{map } \text{uminus } (\text{map } \text{uminus } (as::'a \text{ list}))) = mset as$  **for**

*as* **by** *simp*

**ultimately show** *?thesis* **using** *mset-map* **by** *metis*

**qed**

## 2 Vector Normalization

**lemma** *vec-normalize-norm*:  $v \in \text{carrier-vec } n \implies v \neq 0_v \ n \implies \text{vec-norm } (\text{vec-normalize } v) = 1$   
**by** (*simp add: normalized-vec-norm vec-norm-def*)

## 3 Determinant, Invertability, and Eigenvalues

**definition** *eigvals-of* [*simp*]:

*eigvals-of*  $M$   $es \longleftrightarrow \text{char-poly } M = (\prod a \leftarrow es. [- a, 1:]) \wedge \text{length } es = \text{dim-row } M$

**lemma** *eigvals-of-mset-eq*:

*eigvals-of*  $(A :: 'a :: \text{idom mat})$   $es \implies \text{eigvals-of } A$   $es' \implies \text{mset } es = \text{mset } es'$   
**unfolding** *eigvals-of* **by** (*metis idom-poly-factor-unique'*)

**lemma** *det-is-prod-of-eigenvalues*:

**fixes**  $A :: \text{complex mat}$

**assumes** *square-mat*  $A$

**shows** *Determinant.det*  $A = (\prod e \leftarrow (\text{eigvals } A). e)$

**proof** –

**define**  $es$  **where**  $es \equiv \text{eigvals } A$

**define**  $n$  **where**  $n \equiv \text{dim-row } A$

**have**  $1$ :  $A \in \text{carrier-mat } n \ n$  **using** *assms n-def* **by** *fastforce*

**have**  $2$ : *char-poly*  $A = (\prod e \leftarrow es. [- e, 1:])$

**unfolding** *es-def eigvals-def*

**using**  $1$

**by** (*metis (mono-tags, lifting) 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** *Determinant.det*  $A = \text{Determinant.det } (Q * B * Q')$  **unfolding** *similar-mat-wit-def* **by** *metis*

**also have**  $\dots = \text{Determinant.det } Q * \text{Determinant.det } B * \text{Determinant.det } Q'$

**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**  $\dots = \text{Determinant.det } Q * \text{Determinant.det } B * 1 / (\text{Determinant.det } Q)$

**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**  $\dots = \text{Determinant.det } Q * (\prod e \leftarrow \text{diag-mat } B. e) * 1 / (\text{Determinant.det } Q)$

**by** (*metis \* det-upper-triangular list.map-ident similar-mat-witD(5)*)

**also have**  $\dots = (\prod e \leftarrow (\text{eigvals } A). e)$

**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*:

$(A :: (\text{complex mat})) \in \text{carrier-mat } n \ n \implies \text{eigvals-of } A \ \alpha \implies \text{Spectral-Radius.spectrum } A = \text{set } \alpha$   
**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** *spectrum-connect*:

$(A :: \text{complex mat}) \in \text{carrier-mat } n \ n \implies \text{Spectral-Radius.spectrum } A = \text{spectrum } A$   
**by** (*metis* *eigvals-of eigvals-of-spectrum eigvals-poly-length spectrum-def*)

**lemma** *spectrum-shift*:

**fixes**  $A :: \text{complex mat}$   
**assumes**  $\text{dim}: A \in \text{carrier-mat } n \ n$   
**shows**  $\text{spectrum } (A - r \cdot_m 1_m \ n) = \{\mu - r \mid \mu. \mu \in \text{spectrum } A\}$  (**is** *?lhs = ?rhs*)

**proof**

{ **let**  $?A' = A - r \cdot_m 1_m \ n$   
**fix**  $\mu$  **assume**  $\mu: \mu \in ?lhs$   
**then obtain**  $v$  **where**  $v: v \in \text{carrier-vec } n \ v \neq 0_v \ n \ ?A' \ *_v \ v = \mu \cdot_v \ v$   
**by** (*metis* (*no-types, lifting*) *carrier-mat-triv eigenvalue-def eigenvector-def index-minus-mat(2,3) index-one-mat(2,3) index-smult-mat(2,3) spectrum-eigenvalues*)  
**hence**  $?A' \ *_v \ v = A \ *_v \ v - r \cdot_v \ v$   
**using** *dim*  
**by** (*smt* (*verit, best*) *minus-mult-distrib-mat-vec one-carrier-mat one-mult-mat-vec smult-carrier-mat smult-mat-mult-mat-vec-assoc*)  
**hence**  $A \ *_v \ v = \mu \cdot_v \ v + r \cdot_v \ v$  **using**  $v(1,3)$  *dim* **by** *auto*  
**also have**  $\dots = (\mu + r) \cdot_v \ v$  **using**  $v(1)$  **by** (*simp add: add-smult-distrib-vec*)  
**finally have**  $\mu + r \in \text{spectrum } A$   
**using** *spectrum-connect[OF dim] v(1,2) dim*  
**by** (*metis* *Spectral-Radius.spectrum-def carrier-matD(1) eigenvalue-def eigenvector-def mem-Collect-eq*)  
**hence**  $\mu \in ?rhs$  **by** *force*  
}

**thus**  $?lhs \subseteq ?rhs$  **by** *blast*

**next**

{ **let**  $?A' = A - r \cdot_m 1_m \ n$   
**fix**  $\mu$  **assume**  $\mu: \mu \in ?rhs$   
**then obtain**  $\mu'$  **where**  $\mu': \mu = \mu' - r \ \mu' \in \text{spectrum } A$  **by** *blast*  
**then obtain**  $v$  **where**  $v: v \in \text{carrier-vec } n \ v \neq 0_v \ n \ A \ *_v \ v = \mu' \cdot_v \ v$   
**using** *spectrum-connect[OF dim] dim*  
**by** (*metis* *carrier-matD(1) eigenvalue-def eigenvector-def spectrum-eigenvalues*)  
**have**  $?A' \ *_v \ v = A \ *_v \ v - r \cdot_v \ v$

```

    using v(1) dim
  by (smt (verit, del-insts) minus-mult-distrib-mat-vec one-carrier-mat one-mult-mat-vec
      smult-carrier-mat smult-mat-mult-mat-vec-assoc)
  also have ... =  $\mu' \cdot_v v - r \cdot_v v$  using v(3) by auto
  also have ... =  $\mu \cdot_v v$  using dim v(1)  $\mu'(1)$  by (simp add: minus-smult-vec-distrib)
  finally have  $\mu \in \text{spectrum } ?A'$ 
    using v(1,2) spectrum-connect[OF dim]
    by (smt (verit) Spectral-Radius.spectrum-def carrier-mat-def eigenvalue-def
        eigenvector-def
            index-minus-mat(2,3) index-one-mat(2,3) index-smult-mat(2,3) mem-Collect-eq
            spectrum-connect)
    hence  $\mu \in ?lhs$  by blast
  }
  thus  $?rhs \subseteq ?lhs$  by blast
qed

```

**lemma** *trivial-kernel-imp-nonzero-eigenvalues:*  
**fixes**  $M :: 'a::\{\text{idom, ring-1-no-zero-divisors}\}$  *mat*  
**assumes** *square-mat*  $M$   
**assumes**  $\text{mat-kernel } M \subseteq \{0_v \text{ (dim-row } M)\}$   
**assumes** *eigenvalue*  $M$   $e$   
**shows**  $e \neq 0$   
**using** *assms* **by** (*metis (no-types, lifting) 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**  $M :: \text{complex mat}$   
**assumes** *square-mat*  $M$   
**assumes**  $\text{mat-kernel } M \subseteq \{0_v \text{ (dim-row } M)\}$   
**shows** *invertible-mat*  $M$   
**using** *assms* **by** (*metis carrier-matI det-0-iff-vec-prod-zero-field empty-iff invert-*  
*ible-det mat-kernelI singletonD square-mat.elims(2) subset-singletonD*)

**lemma** *trivial-kernel-imp-det-nz:*  
**fixes**  $M :: \text{complex mat}$   
**assumes** *square-mat*  $M$   
**assumes**  $\text{mat-kernel } M \subseteq \{0_v \text{ (dim-row } M)\}$   
**shows** *Determinant.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**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $B \in \text{carrier-mat } n \ n$   
**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  $A :: 'a::\text{conjugatable-ordered-field mat}$ 
  assumes square-mat  $A$ 
  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 = ( $A', P, Q$ )
     $\wedge$  similar-mat-wit  $A A' P Q$ 
     $\wedge$  upper-triangular  $A'$ 
     $\wedge$  diag-mat  $A' = es$ 
  using assms(1,3) unfolding eigvals-of square-mat.simps
  by (metis carrier-mat-triv old.prod.exhaust schur-decomposition)
  define  $B'$  where  $B' \equiv c \cdot_m A'$ 

  have B'-square: square-mat  $B'$  using A-decomp B'-def similar-mat-witD(5) by
fastforce
  have B-decomp: similar-mat-wit  $B B' P Q \wedge$  upper-triangular  $B'$ 
  proof -
    have upper-triangular  $B'$ 
    proof -
      { fix  $i j$  assume  $*$ :  $j < i \wedge i < \text{dim-row } B'$ 
        hence  $B'_{i,j} = c * A'_{i,j}$  using B'-def B'-square by auto
        also have  $\dots = 0$  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  $\dots = P * (c \cdot_m A') * Q$ 
      by (metis A-decomp similar-mat-wit-def similar-mat-wit-smult)
    also have  $\dots = P * B' * Q$  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  $B' = (\prod a \leftarrow \text{diag-mat } B'. [- a, 1:])$ 
  using char-poly-upper-triangular B-decomp B'-square by auto
  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**  $\text{diag-mat } B' = \text{map } (\lambda x. c * x) \text{ es}$   
**using**  $A\text{-decomp } B'\text{-def}$  **by**  $(\text{metis } \text{diag-mat-map } \text{similar-mat-witD}(5) \text{ smult-mat-def})$   
**ultimately show**  $?thesis$   
**using**  $\text{similar-mats-eigvals } B\text{-decomp } \text{assms}(2) \text{ assms}(3) \text{ char-poly-similar-similar-mat-def}$   
**by**  $\text{fastforce}$   
**qed**

**lemma**  $\text{neg-mat-eigvals}$ :  
**fixes**  $A :: \text{complex mat}$   
**assumes**  $\text{square-mat } A$   
**assumes**  $\text{eigvals-of } A \text{ es}$   
**shows**  $\text{eigvals-of } (-A) (\text{rev } (\text{map } (\lambda x. -x) \text{ es}))$   
**proof** –  
**have**  $\text{eigvals-of } A (\text{rev } \text{es})$   
**using**  $\text{assms}(2)$   
**unfolding**  $\text{eigvals-of}$   
**by**  $(\text{metis } \text{length-rev-prod-list-rev-rev-map})$   
**thus**  $?thesis$   
**using**  $\text{scale-eigvals}[of\ A\ -A\ -1\ \text{rev}\ \text{es}]$   
**by**  $(\text{metis } (\text{lifting}) \text{ ext } \text{assms}(1) \text{ carrier-mat-triv } \text{mult-commute-abs } \text{mult-minus1-right } \text{rev-map } \text{square-mat.elims}(2) \text{ uminus-mat})$   
**qed**

## 4 Quadratic Form

**definition**  $\text{quadratic-form} :: 'a \text{ mat} \Rightarrow 'a \text{ vec} \Rightarrow 'a::\{\text{conjugatable-ring}\}$  **where**  
 $\text{quadratic-form } M \ x \equiv \text{inner-prod } x \ (M *_{\nu} x)$

**abbreviation**  $QF \equiv \text{quadratic-form}$

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

## 5 Leading Principal Submatrix

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

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

**lemma**  $\text{leading-principal-submatrix-carrier}$ :  
 $m \geq n \implies A \in \text{carrier-mat } m \ m \implies \text{lps } A \ n \in \text{carrier-mat } n \ n$   
**proof** –  
**assume**  $*$ :  $m \geq n \ A \in \text{carrier-mat } m \ m$   
**let**  $?B = \text{lps } A \ n$   
**have**  $(\text{card } \{i. i < \text{dim-row } A \wedge i \in \{..<n\}\}) = n$   
**by**  $(\text{metis } *(1) *(2) \text{ Collect-conj-eq } \text{Collect-mem-eq } \text{card-lessThan } \text{carrier-matD}(1))$

*inf.absorb-iff2 lessThan-def lessThan-subset-iff*)  
**hence**  $\dim\text{-col } ?B = n \wedge \dim\text{-row } ?B = n$   
**unfolding** *leading-principal-submatrix-def submatrix-def*  
**using**  $*(2)$  **by** *auto*  
**thus** *?thesis* **by** *blast*  
**qed**

**lemma** *pick-n*:  
**assumes**  $i \leq n$   
**shows**  $\text{pick } \{..n\} i = i$   
**using** *assms*  
**proof**(*induct i*)  
**case**  $0$   
**then show** *?case* **by** *force*  
**next**  
**case** (*Suc i*)  
**hence**  $\text{Suc } i \in \{..n\}$  **by** *blast*  
**moreover from** *Suc* **have**  $\text{Suc } i > \text{pick } \{..n\} i$  **by** *simp*  
**moreover from** *Suc* **have**  $\forall i' < \text{Suc } i. \neg (i' \in \{..n\} \wedge i' > \text{pick } \{..n\} i)$   
**using** *Suc-leD not-less-eq* **by** *presburger*  
**ultimately have**  $\text{Suc } i = (\text{LEAST } a. a \in \{..n\} \wedge a > \text{pick } \{..n\} i)$   
**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**  $\text{pick } \{..<n\} i = i$   
**by** (*metis assms lessThan-Suc-atMost less-Suc-eq-le not0-implies-Suc not-less-zero pick-n*)

**lemma** *leading-principal-submatrix-index*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $k \leq n$   
**assumes**  $i < k$   
**assumes**  $j < k$   
**shows**  $(\text{lps } A \ k) \text{\$}\$(i,j) = A \text{\$}\$(i,j)$   
**proof** –  
**have**  $\bigwedge i. i < k \implies \text{pick } \{..<k\} i = i$  **by** (*simp add: pick-n-le*)  
**moreover have**  $\text{card } \{i. i < \dim\text{-row } A \wedge i \in \{..<k\}\} = k$   
**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**  $\text{card } \{j. j < \dim\text{-col } A \wedge j \in \{..<k\}\} = k$  **using** *assms(1)*  
**by force**  
**moreover have**  $(\text{mat } k \ k (\lambda(i, j). A \text{\$}\$(i,j))) \text{\$}\$(i,j) = A \text{\$}\$(i,j)$  **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**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $k_1 \leq k_2$   
**assumes**  $k_2 \leq n$   
**shows**  $\text{lps } A \ k_1 = \text{lps } (\text{lps } A \ k_2) \ k_1$  (**is**  $?lhs = ?rhs$ )  
**proof** –  
**have**  $\bigwedge i \ j. i < k_1 \implies j < k_1 \implies ?lhs\$\$(i,j) = ?rhs\$\$(i,j)$   
**by** (*smt (verit, best) assms dual-order.trans leading-principal-submatrix-carrier leading-principal-submatrix-index order.strict-trans2*)  
**moreover have**  $?lhs \in \text{carrier-mat } k_1 \ k_1$   
**by** (*meson assms leading-principal-submatrix-carrier order-trans*)  
**moreover have**  $?rhs \in \text{carrier-mat } k_1 \ k_1$   
**by** (*meson assms leading-principal-submatrix-carrier*)  
**ultimately show** *?thesis* **by** *auto*  
**qed**

## 6 Hermitian Matrix

**lemma** *hermitian-real-diag-decomp-eigvals*:  
**fixes**  $A :: \text{complex mat}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes** *hermitian*  $A$   
**assumes** *eigvals-of*  $A$  *es*  
**obtains**  $B \ U$  **where**  
*real-diag-decomp*  $A \ B \ U$   
*diag-mat*  $B = es$   
*set*  $es \subseteq \text{Reals}$   
 $B \in \text{carrier-mat } n \ n$   
 $U \in \text{carrier-mat } n \ n$   
**proof** –  
**have**  $es: \text{char-poly } A = (\prod (e :: \text{complex}) \leftarrow es. [:- e, 1:])$   
**using** *assms eigvals-poly-length* **by** *auto*  
**obtain**  $B \ U \ Q$  **where**  $us: \text{unitary-schur-decomposition } A \ es = (B, U, Q)$   
**by** (*cases unitary-schur-decomposition*  $A \ es$ )  
**hence**  $*$ : *similar-mat-wit*  $A \ B \ U$  (*adjoint*  $U$ )  $\wedge$  *diagonal-mat*  $B$   
 $\wedge$  *diag-mat*  $B = es \wedge$  *unitary*  $U \wedge (\forall i < n. B\$\$(i, i) \in \text{Reals})$   
**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**  $1: \text{real-diag-decomp } A \ B \ U$  **using** *unitary-diagI[of*  $A]$   
**unfolding** *real-diag-decomp-def* **by** *simp*  
  
**from**  $*$  **have**  $2: \text{diag-mat } B = es$  **by** *blast*  
**from**  $*$  **have**  $3: \text{set } es \subseteq \text{Reals}$  **by** (*metis*  $\langle \text{dim-row } B = n \rangle$  *diag-elems-real*  
*diag-elems-set-diag-mat*)  
**from**  $*$  **have**  $4: B \in \text{carrier-mat } n \ n$  **by** (*meson assms(1) similar-mat-witD2(5)*)  
**from**  $*$  **have**  $5: U \in \text{carrier-mat } n \ n$  **by** (*meson assms(1) similar-mat-witD2(6)*)

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

**lemma** *hermitian-eigvals-of-real*: *hermitian* ( $A :: \text{complex mat}$ )  $\implies$  *eigvals-of*  $A$  *es*  
 $\implies$  *set*  $es \subseteq \mathbb{R}$   
**using** *hermitian-real-diag-decomp-eigvals[of A - es] hermitian-square by blast*

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

**lemma** *hermitian-ij-ji-iff*:  
*hermitian*  $A$   
 $\iff$  *square-mat*  $A \wedge (\forall i j. i < \text{dim-row } A \wedge j < \text{dim-row } A \longrightarrow A\$(i,j) =$   
*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** *adjoint-is-conjugate-transpose*:  $A^H = \text{adjoint } A$   
**by** (*simp add: adjoint-def transpose-def cong-mat conjugate-mat-def*)

## 6.1 Locale for Complex Hermitian Matrices

**locale** *cmplx-herm-mat* = *complex-vec-space*  $n$  **for**  $n +$   
**fixes**  $A :: \text{complex mat}$   
**assumes**  $A\text{-dim}$ :  $A \in \text{carrier-mat } n \ n$   
**assumes**  $A\text{-herm}$ : *hermitian*  $A$   
**begin**

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

**definition** *eigenvalues<sub>0</sub>* :: *complex list* **where**  
*eigenvalues<sub>0</sub>*  $\equiv$  *eigvals*  $A$

**lemma** *eigenvalues<sub>0</sub>-exist*:  $\exists es. \text{eigvals-of } A \text{ es}$  **using**  $A\text{-dim}$  *eigvals-poly-length* **by**  
*auto*

**lemma** *eigenvalues*<sub>0</sub>: *eigvals-of A eigenvalues*<sub>0</sub>  
**using** *eigenvalues*<sub>0</sub>-*exist* **unfolding** *eigvals-of eigenvalues*<sub>0</sub>-*def eigvals-def* ..

**lemma** *eigenvalues*<sub>0</sub>-*real*: *set eigenvalues*<sub>0</sub>  $\subseteq \mathbb{R}$   
**using** *A-herm eigenvalues*<sub>0</sub> *hermitian-eigvals-of-real*  
**by** *blast*

**definition** *eigenvalues* :: *complex list where*  
*eigenvalues*  $\equiv$  *rev (sort-key Re eigenvalues*<sub>0</sub>)

**lemma** *eigenvalues*: *eigvals-of A eigenvalues*

**proof** –

**have** *mset eigenvalues* = *mset eigenvalues*<sub>0</sub> **by** (*simp add: eigenvalues-def*)

**thus** *?thesis*

**using** *eigenvalues-def eigvals-of eigenvalues*<sub>0</sub>

**by** (*metis (mono-tags, lifting) mset-eq-length mset-map prod-mset-prod-list*)

**qed**

**lemma** *eigenvalues-sorted*: *sorted-wrt ( $\geq$ ) eigenvalues*

**proof** –

**have**  $\forall \mu_1 \mu_2. \mu_1 \in \text{set } \text{eigenvalues}_0 \longrightarrow \mu_2 \in \text{set } \text{eigenvalues}_0 \longrightarrow (\mu_1 \leq \mu_2 \longleftrightarrow \text{Re } \mu_1 \leq \text{Re } \mu_2)$

**using** *eigenvalues-def eigenvalues*<sub>0</sub>-*real* **by** (*simp add: complex-is-Real-iff less-eq-complex-def subset-iff*)

**hence** *sorted-wrt ( $\leq$ ) (sort-key Re eigenvalues*<sub>0</sub>)

**by** (*smt (verit, ccfv-threshold) length-map nth-map nth-mem order-trans-rules(19) set-sort sorted-sort-key*

*sorted-wrt-iff-nth-less*)

**thus** *?thesis unfolding eigenvalues-def sorted-wrt-rev* .

**qed**

**lemma** *eigenvalues-unique*:

**assumes** *es*: *eigvals-of A es'*

**assumes** *es-sorted*: *sorted-wrt ( $\geq$ ) es'*

**shows** *es' = eigenvalues*

**proof** –

**have** *mset es' = mset eigenvalues* **using** *es eigenvalues eigvals-of-mset-eq* **by** *blast*

**moreover** **have** *length es' = length eigenvalues* **by** (*metis calculation size-mset*)

**moreover** **have** *sorted (rev (map Re es'))*

**using** *es-sorted* **by** (*metis less-eq-complex-def sorted-wrt-map-mono sorted-wrt-rev*)

**moreover** **have** *sorted (rev (map Re eigenvalues))*

**using** *eigenvalues* **by** (*simp add: eigenvalues-def rev-map*)

**ultimately** **have** *rev (map Re es') = rev (map Re eigenvalues)*

**by** (*metis mset-map mset-rev properties-for-sort*)

**hence** *map Re es' = map Re eigenvalues* **by** *force*

**moreover** **have** *set es'  $\subseteq \mathbb{R} \wedge$  set eigenvalues  $\subseteq \mathbb{R}$*

**using** *es A-herm eigenvalues hermitian-eigvals-of-real* **by** *blast*

**ultimately** **show** *es' = eigenvalues* **by** (*metis list.inj-map-strong of-real-Re sub-*

*set-code(1)*  
**qed**

**lemma** *eigenvalues-real*: *set eigenvalues*  $\subseteq \mathbb{R}$   
**using** *A-herm hermitian-eigvals-of-real eigenvalues(1)* **by** *blast*

**lemma** *eigenvalues-set-eq*: *set eigenvalues*<sub>0</sub> = *set eigenvalues*  
**by** (*simp add: eigenvalues-def*)

**lemma** *hermitian-eigenvalues-real*:  
**assumes** *e: eigenvalue A e*  
**shows** *e*  $\in$  *Reals*  
**using** *cpx-sq-mat.hermitian-spectrum-real[OF - A-dim A-herm, of n n e]*  
**using** *A-dim e*  
**by** (*metis Projective-Measurements.spectrum-def 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*: *spectrum A*  $\subseteq$  *Reals*  
**unfolding** *spectrum-connect[OF A-dim, symmetric]*  
**by** (*simp add: Spectral-Radius.spectrum-def hermitian-eigenvalues-real unfold-simps(2)*)

**lemma** *leading-principal-submatrix-hermitian*:  
**assumes** *k: k*  $\leq$  *n*  
**shows** *hermitian (lps A k)* (**is** *hermitian ?A'*)

**proof** –

**have**  $\bigwedge i j. i < \text{dim-row } ?A' \implies j < \text{dim-col } ?A' \implies ?A'_{(i,j)} = \text{conjugate } (?A'_{(j,i)})$

**using** *k A-dim A-herm*

**by** (*metis (no-types, lifting) adjoint-eval carrier-matD(1) carrier-matD(2)*

*dual-order.strict-trans1 hermitian-def leading-principal-submatrix-carrier leading-principal-submatrix-index*)

**thus** *?thesis*

**using** *A-dim k*

**by** (*metis (no-types, lifting) adjoint-dim-col adjoint-dim-row adjoint-eval carrier-matD(1) carrier-matD(2) eq-matI hermitian-def leading-principal-submatrix-carrier*)

**qed**

**lemma** *hermitian-mat-inv*:  
**assumes** *A'-dim: A'*  $\in$  *carrier-mat n n*  
**assumes** *inv: inverts-mat A A'*  
**shows** *hermitian A'*  
**using** *A-dim A-herm assms*  
**by** (*metis (no-types, lifting) ext adjoint-dim' adjoint-mult adjoint-one carrier-matD(1,2) hermitian-def invertible-det invertible-mat-def inverts-mat-def inverts-mat-sym square-mat.simps vec-space.det-nonzero-congruence*)

**lemma** *negative-hermitian*: *hermitian* ( $-A$ )  
**using** *A-dim A-herm*

by (metis hermitian-minus left-add-zero-mat minus-add-uminus-mat uminus-carrier-iff-mat zero-carrier-mat zero-hermitian)

**lemma** *principal-submatrix-hermitian*:

assumes  $I: I \subseteq \{..<n\}$

shows hermitian (submatrix  $A I I$ ) (is hermitian  $?B$ )

**proof** –

have square-mat  $?B$

using  $A$ -dim

by (metis (full-types) carrier-matD(1) carrier-matD(2) dim-submatrix(1) dim-submatrix(2) square-mat.elims(1))

moreover {

fix  $i j$  assume  $*$ :  $i < \text{dim-row } ?B \wedge j < \text{dim-row } ?B$

then obtain  $i' j'$  where  $?B\$\$(i,j) = A\$\$(i',j') \wedge i' = \text{pick } I i \wedge j' = \text{pick } I j$

unfolding submatrix-def using  $A$ -dim pick-le by auto

moreover then have  $?B\$\$(j,i) = A\$\$(j',i')$

unfolding submatrix-def

using  $A$ -dim

by (metis (no-types, lifting) Collect-cong \* carrier-matD(1) carrier-matD(2) case-prod-conv dim-submatrix(1) index-mat(1))

ultimately have  $?B\$\$(i,j) = \text{conjugate } (?B\$\$(j,i))$

using \*  $A$ -herm by (metis dim-submatrix(1) hermitian-ij-ji-iff pick-le)

}

ultimately show  $?thesis$  by (metis hermitian-ij-ji-iff)

qed

**lemma** *hermitian-row-col*:

assumes  $i < n$

shows row  $A i = \text{conjugate } (\text{col } A i)$

using  $A$ -dim  $A$ -herm assms by (metis adjoint-row carrier-matD(2) hermitian-def)

**lemma** *inner-prod-swap*:

assumes  $v: v \in \text{carrier-vec } n$

assumes  $w: w \in \text{carrier-vec } n$

shows  $(A *_v v) \cdot c w = v \cdot c (A *_v w)$

**proof** –

have  $(A *_v v) \cdot c w = v \cdot c (A^H *_v w)$

using assms by (metis adjoint-def-alter adjoint-is-conjugate-transpose  $A$ -dim)

also have  $\dots = v \cdot c (A *_v w)$

using  $A$ -herm assms by (simp add: adjoint-is-conjugate-transpose hermitian-def)

finally show  $?thesis$  .

qed

end

## 7 Matrix Conjugate

**lemma** *conjugate-mat-dist*:

fixes  $A B :: 'a::\text{conjugatable-ring mat}$

**assumes**  $A \in \text{carrier-mat } m \ n$   
**assumes**  $B \in \text{carrier-mat } n \ p$   
**shows**  $(\text{conjugate } A) * (\text{conjugate } B) = \text{conjugate } (A * B)$   
**using** *assms*  
**by** (*smt (z3) 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**  $A :: 'a::\{\text{conjugatable-ring, semiring-1}\} \text{ mat}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $A' \in \text{carrier-mat } n \ n$   
**assumes** *inverts-mat*  $A \ A'$   
**shows** *inverts-mat*  $(\text{conjugate } A) \ (\text{conjugate } A')$   
**proof** –  
**have**  $(\text{conjugate } A) * (\text{conjugate } A') = \text{conjugate } (A * A')$   
**using** *conjugate-mat-dist* *assms(1) assms(2)* **by** *blast*  
**also have**  $\dots = \text{conjugate } (1_m \ n)$   
**by** (*metis assms(1) assms(3) carrier-matD(1) inverts-mat-def*)  
**also have**  $\dots = 1_m \ n$   
**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** *conjugate-dist-mult-mat:*

**fixes**  $A :: 'a::\text{conjugatable-ring} \text{ mat}$   
**assumes**  $A \in \text{carrier-mat } m \ n \ B \in \text{carrier-mat } n \ p$   
**shows**  $\text{conjugate } (A * B) = \text{conjugate } A * \text{conjugate } B$   
**(is** *?lhs = ?rhs***)**  
**proof** –  
**have**  $\bigwedge i \ j. \ i < m \implies j < p \implies ?lhs\$(i,j) = ?rhs\$(i,j)$   
**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**  $?lhs \in \text{carrier-mat } m \ p$  **using** *assms* **by** *auto*  
**ultimately show** *?thesis* **using** *assms carrier-matD(2)* **by** *auto*  
**qed**

**lemma** *conjugate-dist-add-mat:*

**fixes**  $A :: 'a::\text{conjugatable-ring} \text{ mat}$   
**assumes**  $A \in \text{carrier-mat } m \ n \ B \in \text{carrier-mat } m \ n$   
**shows**  $\text{conjugate } (A + B) = \text{conjugate } A + \text{conjugate } B$   
**(is** *?lhs = ?rhs***)**  
**proof** –  
**have**  $\bigwedge i \ j. \ i < m \implies j < n \implies ?lhs\$(i,j) = ?rhs\$(i,j)$   
**using** *assms assms conjugate-dist-add* **by** *fastforce*

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

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

**lemma** *conj-mat-vec-mult*:  
fixes  $A :: 'a::\{\text{conjugate,conjugatable-ring}\} \ \text{mat}$   
fixes  $v :: 'a \ \text{vec}$   
assumes  $A \in \text{carrier-mat } n \ n$   
assumes  $v \in \text{carrier-vec } n$   
shows  $\text{conjugate } (A *_{v} v) = (\text{conjugate } A) *_{v} (\text{conjugate } v)$   
(is *?lhs = ?rhs*)

**proof** –  
have  $\bigwedge i. i < n \implies ?lhs\ \$i = ?rhs\ \$i$   
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  $?lhs \in \text{carrier-vec } n$  using *assms* by *force*  
ultimately show *?thesis* using *assms(1)* by *auto*  
qed

**lemma** *conjugate-vec-first*:  
assumes  $v \in \text{carrier-vec } n$   
assumes  $i \leq n$   
shows  $\text{conjugate } (\text{vec-first } v \ i) = \text{vec-first } (\text{conjugate } v) \ i$   
by (*smt (verit, cfv-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*:  $i \leq \text{dim-vec } v \implies \text{conjugate } (\text{vec-last } v \ i) = \text{vec-last } (\text{conjugate } v) \ i$   
unfolding *vec-last-def* by *auto*

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

## 8 Block Matrix

**lemma** *block-mat-vec-mult*:  
fixes  $x$   
assumes  $A \in \text{carrier-mat } nr1 \ nc1$   
assumes  $B \in \text{carrier-mat } nr1 \ nc2$

```

assumes  $C \in \text{carrier-mat } nr2 \ nc1$ 
assumes  $D \in \text{carrier-mat } nr2 \ nc2$ 
assumes  $M = \text{four-block-mat } A \ B \ C \ D$ 
assumes  $x \in \text{carrier-vec } (nc1 + nc2)$ 
defines  $x_1 \equiv \text{vec-first } x \ nc1$ 
defines  $x_2 \equiv \text{vec-last } x \ nc2$ 
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  $A :: ('a :: \text{comm-ring}) \text{ mat}$ 
assumes  $A \in \text{carrier-mat } (Suc \ n) \ (Suc \ n)$ 
assumes  $x \in \text{carrier-vec } (Suc \ n)$ 
defines  $A_n \equiv \text{lps } A \ n$ 
defines  $v_n \equiv \text{vec-first } (\text{col } A \ n) \ n$ 
defines  $w_n \equiv \text{vec-first } (\text{row } A \ n) \ n$ 
defines  $a \equiv A \ \$\$ \ (n, \ n)$ 
defines  $x_n \equiv \text{vec-first } x \ n$ 
defines  $b \equiv x \$n$ 
shows  $A *_v x = (A_n *_v x_n + b \cdot_v v_n) @_v (\text{vec } 1 \ (\lambda i. (w_n \cdot x_n) + a * b))$  (is
?lhs = ?rhs)

```

**proof**

```

have dim-xn:  $\text{dim-vec } x_n = n$  by (simp add: assms(7))
have dim-wn:  $\text{dim-vec } w_n = n$  by (simp add: assms(5))
have dim-row-An:  $\text{dim-row } A_n = n$ 
by (metis assms(1) assms(3) carrier-matD(1) le-add2 leading-principal-submatrix-carrier
plus-1-eq-Suc)
have dim-col-An:  $\text{dim-col } A_n = n$ 
by (metis assms(1) assms(3) carrier-matD(2) le-add2 leading-principal-submatrix-carrier
plus-1-eq-Suc)

```

**show** *dims*:  $\text{dim-vec } ?lhs = \text{dim-vec } ?rhs$  **using** *assms(1) v<sub>n</sub>-def* **by** *auto*

**show**  $\bigwedge i. i < \text{dim-vec } ?rhs \implies ?lhs \$i = ?rhs \$i$

**proof**–

```

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

```

**show**  $?lhs \$i = ?rhs \$i$

**proof** (*cases*  $i < n$ )

**case** *True*

**have**  $A_n$ :  $\bigwedge j. j < n \implies A \$\$ (i, j) = A_n \$\$ (i, j)$

**by** (*metis True assms(1) assms(3) le-add2 leading-principal-submatrix-index*
*plus-1-eq-Suc*)

**have**  $A_n$ -*row*:  $\bigwedge j. j < n \implies A_n \$\$ (i, j) = (\text{row } A_n \ i) \$j$

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<sub>n</sub> \*<sub>v</sub> x<sub>n</sub> + b ·<sub>v</sub> v<sub>n</sub>)*i*

**proof**–

have ?lhs*i* = (∑ j<Suc n. A*i,j* \* x*j*)

unfolding dot scalar-prod-def using row-j assms(2) atLeast0LessThan by

*force*

**moreover** have (∑ j<n. A*i,j* \* x*j*) = (∑ j<n. A<sub>n</sub>*i,j* \* x<sub>n</sub>*j*)

by (smt (verit) A<sub>n</sub> assms(7) index-vec lessThan-iff sum.cong vec-first-def)

**moreover** have (∑ j<n. A<sub>n</sub>*i,j* \* x<sub>n</sub>*j*) = (A<sub>n</sub> \*<sub>v</sub> x<sub>n</sub>)*i*

**proof**–

have (A<sub>n</sub> \*<sub>v</sub> x<sub>n</sub>)*i* = (row A<sub>n</sub> i) · x<sub>n</sub>

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 ∧j. j < n ⇒ A<sub>n</sub>*i,j* \* x<sub>n</sub>*j* = (row A<sub>n</sub> i)*j* \* x<sub>n</sub>*j*

using A<sub>n</sub>-row by presburger

ultimately show ?thesis

unfolding scalar-prod-def using atLeast0LessThan dim-x<sub>n</sub> by fastforce

**qed**

**moreover** have (A*i,n* \* x*n*) = (b ·<sub>v</sub> v<sub>n</sub>)*i*

**proof**–

have x*n* = b unfolding b-def by blast

**moreover** have A*i,n* = v<sub>n</sub>*i*

using assms(1) i by (simp add: True vec-first-def v<sub>n</sub>-def)

**moreover** have (b ·<sub>v</sub> v<sub>n</sub>)*i* = b \* (v<sub>n</sub>*i*) by (simp add: True v<sub>n</sub>-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<sub>n</sub>-def)

**next**

case False

**hence** \*: i = n using i by linarith

**hence** ?lhs*i* = (row A n) · x using dot by blast

**also** have ... = w<sub>n</sub> · x<sub>n</sub> + a \* b

**proof**–

have row A n = w<sub>n</sub> @<sub>v</sub> vec-last (row A n) 1

by (metis \* w<sub>n</sub>-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<sub>n</sub> grOI i index-append-vec(1) index-append-vec(2) index-row(2) zero-less-diff)

**ultimately** have row A n = w<sub>n</sub> @<sub>v</sub> (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<sub>n</sub> @<sub>v</sub> (vec 1 (λ-. b))

**proof**

show \*: dim-vec x = dim-vec (x<sub>n</sub> @<sub>v</sub> (vec 1 (λ-. b))) by (simp add: assms(2) dim-x<sub>n</sub>)

```

have  $\bigwedge i. i < \text{Suc } n \implies x \$ i = (x_n \text{ @}_v (\text{vec } 1 (\lambda-. b))) \$ i$ 
proof –
  fix  $i$  assume  $i < \text{Suc } n$ 
  show  $x \$ i = (x_n \text{ @}_v (\text{vec } 1 (\lambda-. b))) \$ i$ 
    apply (cases  $i = n$ )
    apply (simp add: append-vec-def assms(8) dim-xn)
    apply (simp add: append-vec-def)
    by (smt (verit, best)  $\langle i < \text{Suc } n \rangle$  dim-xn index-vec less-antisym
vec-first-def xn-def)
  qed
  thus  $\bigwedge i. i < \text{dim-vec } (x_n \text{ @}_v \text{vec } 1 (\lambda-. b)) \implies x \$ i = (x_n \text{ @}_v \text{vec } 1 (\lambda-. b)) \$ i$ 
    by (metis * assms(2) carrier-vecD)
  qed
  ultimately have  $(\text{row } A \ n) \cdot x = (w_n \cdot x_n) + ((\text{vec } 1 (\lambda-. A \$ \$ (n, n))) \cdot (\text{vec } 1 (\lambda-. b)))$ 
    by (metis assms(5) assms(7) scalar-prod-append vec-carrier vec-first-carrier)
  moreover have  $((\text{vec } 1 (\lambda-. A \$ \$ (n, n))) \cdot (\text{vec } 1 (\lambda-. b))) = a * b$ 
    by (simp add: a-def b-def scalar-prod-def)
  ultimately show ?thesis by argo
  qed
  finally show ?thesis by (simp add: * vn-def)
  qed
qed
qed

```

**lemma** *vec-first-index*:  $n \leq \text{dim-vec } v \implies i < n \implies v \$ i = (\text{vec-first } v \ n) \$ i$   
**unfolding** *vec-first-def* **by** *simp*

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

**lemma** *inner-prod-append*:  
**assumes**  $x \in \text{carrier-vec } (\text{dim-vec } (u \text{ @}_v v))$   
**shows**  $x \cdot c (u \text{ @}_v v) = (\text{vec-first } x (\text{dim-vec } u)) \cdot c u + (\text{vec-last } x (\text{dim-vec } v)) \cdot c v$   
 $(u \text{ @}_v v) \cdot c x = u \cdot c (\text{vec-first } x (\text{dim-vec } u)) + v \cdot c (\text{vec-last } x (\text{dim-vec } v))$

**proof** –

```

define  $n$  where  $n \equiv \text{dim-vec } (u \text{ @}_v v)$ 
define  $n_u$  where  $n_u \equiv \text{dim-vec } u$ 
define  $n_v$  where  $n_v \equiv \text{dim-vec } v$ 

```

**have** *dims-add*:  $n_u + n_v = n$  **by** (*simp add: n<sub>u</sub>-def* *n<sub>v</sub>-def* *n-def*)

**have** *n<sub>u</sub>-prop*:  $\bigwedge i. i < n_u \implies \text{conjugate } (u \text{ @}_v v) \$ i = (\text{conjugate } u) \$ i$  **by** (*simp add: n<sub>u</sub>-def*)

**have** *n<sub>v</sub>-prop*:  $\bigwedge i. i < n_v \implies \text{conjugate } (u \text{ @}_v v) \$ (i + n_u) = (\text{conjugate } v) \$ i$

**by** (*simp add: n<sub>u</sub>-def n<sub>v</sub>-def*)

**have**  $n = \text{dim-vec } (\text{conjugate } (u @_v v))$  **by** (*simp add: n-def*)

**hence**  $x \cdot c (u @_v v) = (\sum i \in \{0..<n\}. x\$i * (\text{conjugate } (u @_v v))\$i)$

**unfolding** *scalar-prod-def by blast*

**hence**  $x \cdot c (u @_v v) =$

$(\sum i \in \{0..<n_u\}. x\$i * (\text{conjugate } (u @_v v))\$i)$

$+ (\sum i \in \{n_u..<n\}. x\$i * (\text{conjugate } (u @_v v))\$i)$

**by** (*smt (verit, best) bot-nat-0.extremum index-append-vec(2) n<sub>u</sub>-def n-def nat-le-linear nless-le not-add-less1 sum.atLeastLessThan-concat*)

**moreover have**  $(\sum i \in \{0..<n_u\}. x\$i * (\text{conjugate } (u @_v v))\$i) = (\text{vec-first } x (\text{dim-vec } u)) \cdot c u$

**proof**–

**have**  $*$ :  $\bigwedge i. i \in \{0..<n_u\} \implies x\$i = (\text{vec-first } x n_u)\$i$

**by** (*simp add: vec-first-def*)

**have**  $(\sum i \in \{0..<n_u\}. x\$i * (\text{conjugate } (u @_v v))\$i) = (\sum i \in \{0..<n_u\}. x\$i * (\text{conjugate } u)\$i)$

**using** *n<sub>u</sub>-prop by simp*

**also have**  $\dots = (\sum i \in \{0..<n_u\}. (\text{vec-first } x n_u)\$i * (\text{conjugate } u)\$i)$

**using**  $*$  **by** *auto*

**also have**  $\dots = (\text{vec-first } x (\text{dim-vec } u)) \cdot c u$

**by** (*metis (no-types, lifting) dim-vec-conjugate n<sub>u</sub>-def scalar-prod-def sum.cong*)

**finally show** *?thesis* .

**qed**

**moreover have**  $(\sum i \in \{n_u..<n\}. x\$i * (\text{conjugate } (u @_v v))\$i) = (\text{vec-last } x (\text{dim-vec } v)) \cdot c v$

**proof**–

**have**  $*$ :  $\text{vec-last } (u @_v v) n_v = v$

**by** (*metis append-vec-eq carrier-vecI dims-add n<sub>u</sub>-def n-def vec-first-carrier vec-first-last-append*)

**have**  $\bigwedge i. i \in \{n_u..<n\} \implies x\$i = (\text{vec-last } x n_v)\$(i - (n - n_v))$

**unfolding** *vec-last-def using assms dims-add less-diff-conv2 n-def by simp*

**moreover have**  $\bigwedge i. i \in \{n_u..<n\}$

$\implies (\text{conjugate } (u @_v v))\$i = (\text{vec-last } (\text{conjugate } (u @_v v)) n_v)\$(i - (n - n_v))$

**unfolding** *vec-last-def using assms dims-add less-diff-conv2 n-def by simp*

**ultimately have**  $(\sum i \in \{n_u..<n\}. x\$i * (\text{conjugate } (u @_v v))\$i)$

$= (\sum i \in \{n_u..<n\}. (\text{vec-last } x n_v)\$(i - (n - n_v))$

$* (\text{vec-last } (\text{conjugate } (u @_v v)) n_v)\$(i - (n - n_v)))$

$(\text{is } - = (\sum i \in -. ?F i))$

**by** *force*

**also have**  $\dots = (\sum i \in \{0..<n-n_u\}. (\text{vec-last } x n_v)\$((i + n_u) - (n - n_v))$

$* (\text{vec-last } (\text{conjugate } (u @_v v)) n_v)\$((i + n_u) - (n - n_v)))$

**using** *sum.shift-bounds-nat-ivl[of ?F 0 n<sub>u</sub> n<sub>v</sub>] 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<sub>v</sub>-def n-def scalar-prod-def sum.cong*)

**qed**  
**ultimately show**  $x \cdot c (u @_v v) = (\text{vec-first } x (\text{dim-vec } u)) \cdot c u + (\text{vec-last } x (\text{dim-vec } v)) \cdot c v$   
**by** *argo*

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

**have**  $n = \text{dim-vec } (\text{conjugate } (u @_v v))$  **by** (*simp add: n-def*)  
**moreover have**  $\text{dim-vec } (\text{conjugate } x) = n$  **using** *assms n-def* **by** *auto*  
**ultimately have**  $(u @_v v) \cdot c x = (\sum i \in \{0..<n\}. (u @_v v)\$i * (\text{conjugate } x)\$i)$   
**unfolding** *scalar-prod-def* **by** *presburger*  
**hence**  $(u @_v v) \cdot c x =$   
 $(\sum i \in \{0..<n_u\}. (u @_v v)\$i * (\text{conjugate } x)\$i)$   
 $+ (\sum i \in \{n_u..<n\}. (u @_v v)\$i * (\text{conjugate } x)\$i)$   
**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**  $(\sum i \in \{0..<n_u\}. (u @_v v)\$i * (\text{conjugate } x)\$i) = u \cdot c (\text{vec-first } x (\text{dim-vec } u))$

**proof**–

**have**  $*$ :  $\bigwedge i. i \in \{0..<n_u\} \implies (\text{conjugate } x)\$i = (\text{vec-first } (\text{conjugate } x) n_u)\$i$   
**by** (*simp add: vec-first-def*)  
**have**  $(\sum i \in \{0..<n_u\}. (u @_v v)\$i * (\text{conjugate } x)\$i)$   
 $= (\sum i \in \{0..<n_u\}. u\$i * (\text{conjugate } x)\$i)$   
**using**  $n_u\text{-prop}$  **by** *simp*  
**also have**  $\dots = (\sum i \in \{0..<n_u\}. u\$i * (\text{vec-first } (\text{conjugate } x) n_u)\$i)$   
**using**  $*$  **by** *force*  
**also have**  $\dots = u \cdot c (\text{vec-first } x (\text{dim-vec } u))$   
**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**  $(\sum i \in \{n_u..<n\}. (u @_v v)\$i * (\text{conjugate } x)\$i) = v \cdot c (\text{vec-last } x (\text{dim-vec } v))$

**proof**–

**have**  $*$ :  $\text{vec-last } (u @_v v) n_v = v$   
**by** (*metis append-vec-eq carrier-vecI dims-add n\_u-def n-def vec-first-carrier*  
*vec-first-last-append*)  
**have**  $\bigwedge i. i \in \{n_u..<n\} \implies (\text{conjugate } x)\$i = (\text{vec-last } (\text{conjugate } x) n_v)\$(i - (n - n_v))$   
**unfolding** *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*  
**moreover have**  $\bigwedge i. i \in \{n_u..<n\}$   
 $\implies (u @_v v)\$i = (\text{vec-last } (u @_v v) n_v)\$(i - (n - n_v))$   
**unfolding** *vec-last-def* **using** *assms dims-add less-diff-conv2 n-def* **by** *simp*  
**ultimately have**  $(\sum i \in \{n_u..<n\}. (u @_v v)\$i * (\text{conjugate } x)\$i)$   
 $= (\sum i \in \{n_u..<n\}. (\text{vec-last } (u @_v v) n_v)\$(i - (n - n_v))$   
 $* (\text{vec-last } (\text{conjugate } x) n_v)\$(i - (n - n_v)))$

```

    (is - = (∑ i ∈ -. ?F i))
  by force
  also have ... = (∑ i ∈ {0..<n-nu}. (vec-last (u @v v) nv)$((i + nu) - (n
- nv))
    * (vec-last (conjugate x) nv)$((i + nu) - (n - nv)))
  using sum.shift-bounds-nat-ivl[of ?F 0 nu nv] 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) · c x = u · c (vec-first x (dim-vec u)) + v · c (vec-last
x (dim-vec v))
    by argo
  qed

```

## 8.1 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' ∈ carrier-mat r r
  shows Determinant.det M = Determinant.det A * Determinant.det (D - C *
A' * B)

```

**proof**–

```

  let ?rM = dim-row M
  let ?cM = dim-col M
  let ?rA = r
  let ?cA = c
  let ?rB = r
  let ?cB = ?cM - ?cA
  let ?rC = ?rM - ?rA
  let ?cC = c
  let ?rD = ?rM - ?rA
  let ?cD = ?cM - ?cA
  let ?IA = 1m r
  let ?ID = 1m ?rD
  let ?OB = 0m ?rB ?cB
  let ?OC = 0m ?rC ?cC
  let ?P = four-block-mat ?IA ?OB (C * A') ?ID
  let ?Q = four-block-mat A ?OB ?OC (D - C * A' * B)
  let ?R = four-block-mat ?IA (A' * B) ?OC ?ID

```

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

**have**  $M\text{-dim}$ :  $M \in \text{carrier-mat } ?r_M \ ?c_M$   
**by** *blast*  
**have**  $A\text{-dim}$ :  $A \in \text{carrier-mat } r \ r$   
**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**  $M\text{-dim}$  *assms(4)*  
**apply** (metis  $A\text{-dim}$  *assms(1) assms(5) carrier-matD(1) dim-col-mat(1)* *prod.sel(1) split-block-def square-mat.elims(2)*)  
**apply** (metis  $A\text{-dim}$  *assms(1) carrier-matD(2) dim-col-mat(1) prod.sel(1)* *split-block-def*)  
**using** *assms(4)* **by** *force*  
**have**  $C\text{-}A'\text{-dim}$ :  $C * A' \in \text{carrier-mat } ?r_C \ ?c_C$   
**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**  $A'\text{-}B\text{-dim}$ :  $A' * B \in \text{carrier-mat } ?r_B \ ?c_B$   
**by** (metis (*no-types, lifting*)  $A'\text{-dim}$  *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**  $D\text{-min-}C\text{-}A'\text{-}B\text{-dim}$ :  $D - C * A' * B \in \text{carrier-mat } ?c_D \ ?c_D$   
**by** (metis  $A'\text{-}B\text{-dim}$   $C\text{-}A'\text{-dim}$  *carrier-matD(1) carrier-matD(2) carrier-mat-triv index-mult-mat(2) index-mult-mat(3) minus-carrier-mat square(1)*)  
**have**  $P\text{-dim}$ :  $?P \in \text{carrier-mat } ?r_M \ ?c_M$   
**using** *assms(2) square(3)* **by** *auto*  
**have**  $Q\text{-dim}$ :  $?Q \in \text{carrier-mat } ?r_M \ ?c_M$   
**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**  $R\text{-dim}$ :  $?R \in \text{carrier-mat } ?r_M \ ?c_M$   
**using**  $P\text{-dim}$  **by** *fastforce*

**have**  $M$ :  $M = ?P * ?Q * ?R$

**proof**–

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

**have**  $1$ :  $?I_A \in \text{carrier-mat } r \ r$  **by** *simp*

**have**  $2$ :  $?O_B \in \text{carrier-mat } ?r_B \ ?c_B$  **by** *auto*

**have**  $3$ :  $C * A' \in \text{carrier-mat } ?r_C \ r$  **using**  $C\text{-}A'\text{-dim}$  *square(2)* **by** *blast*

**have** 4:  $?I_D \in \text{carrier-mat } ?r_C ?c_B$  **using** 2 *square(1)* **by** *auto*  
**have** 6:  $?O_B \in \text{carrier-mat } r ?c_B$  **using** 2 *square(2)* **by** *blast*  
**have** 7:  $?O_C \in \text{carrier-mat } ?c_B r$  **using** 4 *square(2)* **by** *auto*  
**have** 8:  $(D - C * A' * B) \in \text{carrier-mat } ?c_B ?c_B$   
**using** 4 *D-min-C-A'-B-dim square(1) square(2)* **by** *auto*  
**have** a:  $(D - C * A' * B) \in \text{carrier-mat } ?r_D ?c_D$  **using** 8 *square(1)* **by**  
*presburger*  
**have** b:  $?I_D \in \text{carrier-mat } ?c_B ?c_B$  **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**  $?P * ?Q = \text{four-block-mat } A ?O_B C (D - C * A' * B)$   
**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**  $\dots * ?R = \text{four-block-mat } A B C D$   
**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**  $C * A' * B \in \text{carrier-mat } ?r_D ?c_D$   
**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 Determinant.det M = Determinant.det ?P * Determinant.det ?Q * De-
terminant.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 Determinant.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 Determinant.det ?Q = Determinant.det A * Determinant.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 Determinant.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

```

## 9 Positive Definite Matrix

**definition** *positive-definite* :: 'a::{ord,conjugatable-field} mat  $\Rightarrow$  bool **where**  
*positive-definite* M  $\longleftrightarrow$  hermitian M  
 $\wedge (\forall x \in \text{carrier-vec } (\text{dim-col } M). x \neq 0_v (\text{dim-col } M) \longrightarrow QF M x > 0)$

**lemma** *leading-principal-submatrix-positive-definite*:

**fixes** A :: 'a::{conjugatable-field,ord} mat

**assumes** A  $\in$  carrier-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-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-vec k v  $\neq$  0\_v k

**define** w **where** w  $\equiv$  vec n ( $\lambda i. \text{if } i < k \text{ then } v\$i \text{ else } 0$ )

**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:  $\bigwedge i. i \in \{k..<n\} \implies \text{conjugate } (w\$i) = 0$  using *w-def* by *simp*

have 2:  $\bigwedge i. i \in \{0..<k\} \implies w\$i = v\$i$  using *assms(3)* *w-def* by *auto*

hence 3:  $\bigwedge i. i \in \{0..<k\} \implies \text{conjugate } (w\$i) = \text{conjugate } (v\$i)$  by *presburger*

have 4:  $\bigwedge i. i \in \{0..<k\} \implies (A *_v w)\$i = (B *_v v)\$i$

proof–

fix *i* assume \*\*:  $i \in \{0..<k\}$

have \*\*\*:  $\bigwedge j. j \in \{0..<k\} \implies (\text{row } A \ i)\$j = (\text{row } B \ i)\$j$

proof–

fix *j* assume  $j \in \{0..<k\}$

moreover then have  $(\text{row } A \ i)\$j = A\$\$(i,j)$

using \*\* \* *assms(1)* *assms(3)* by *force*

ultimately show  $(\text{row } A \ i)\$j = (\text{row } B \ i)\$j$

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 \*\*\*\*:  $\text{row } B \ i \in \text{carrier-vec } k \wedge \text{dim-vec } v = k$

using *B-carrier* *B-dims* \* by *auto*

have  $(A *_v w)\$i = \text{row } A \ i \cdot w$  using \*\* *assms(1)* *assms(3)* by *force*

also have  $\dots = (\sum j \in \{0..<n\}. (\text{row } A \ i)\$j * w\$j)$

by (*metis* (*no-types*, *lifting*) *dim-vec* *scalar-prod-def* *sum.cong* *w-def*)

also have  $\dots = (\sum j \in \{0..<k\}. (\text{row } A \ i)\$j * w\$j) + (\sum j \in \{k..<n\}. (\text{row } A \ i)\$j * w\$j)$

by (*simp* *add: assms(3)* *sum.atLeastLessThan-concat*)

also have  $\dots = (\sum j \in \{0..<k\}. (\text{row } A \ i)\$j * v\$j)$

using 1 2 by *auto*

also have  $\dots = (\sum j \in \{0..<k\}. (\text{row } B \ i)\$j * v\$j)$

using \*\*\* by *fastforce*

also have  $\dots = (\text{row } B \ i) \cdot v$

using *B-carrier* \*\*\*\* *unfolding* *scalar-prod-def* by *blast*

also have  $\dots = (B *_v v)\$i$  using \*\* *B-carrier* by *auto*

finally show  $(A *_v w)\$i = (B *_v v)\$i$ .

qed

have 5:  $v \in \text{carrier-vec } k$  by (*simp* *add: \*(1)*)

hence 6:  $B *_v v \in \text{carrier-vec } k \wedge \text{conjugate } v \in \text{carrier-vec } k$

by (*metis* *B-def* *Matrix.carrier-vec-conjugate* *assms(1)* *assms(3)* *leading-principal-submatrix-carrier* *mult-mat-vec-carrier*)

have  $QF A w = (A *_v w) \cdot c \ w$  by *simp*

also have  $\dots = (\sum i \in \{0..<n\}. (A *_v w)\$i * \text{conjugate } (w\$i))$

by (*smt* (*verit*, *ccfv-SIG*) *atLeastLessThan-iff* *dim-vec* *dim-vec-conjugate* *scalar-prod-def* *sum.cong* *vec-index-conjugate* *w-def*)

also have  $\dots = (\sum i \in \{0..<k\}. (A *_v w)\$i * \text{conjugate } (w\$i))$

+  $(\sum i \in \{k..<n\}. (A *_v w)\$i * \text{conjugate } (w\$i))$

by (*simp* *add: assms(3)* *sum.atLeastLessThan-concat*)

also have  $\dots = (\sum i \in \{0..<k\}. (A *_v w)\$i * \text{conjugate } (w\$i))$  using 1 by *simp*

also have  $\dots = (\sum i \in \{0..<k\}. (B *_v v)\$i * \text{conjugate } (v\$i))$  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 hermitian-ij-ji-iff cmplx-herm-mat.leading-principal-submatrix-hermitian
B-dims
  unfolding positive-definite-def B-def
  by (smt (verit, best) carrier-matD(1) leading-principal-submatrix-index-or-
der-less-le-trans square-mat.simps)
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 posi-
tive-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 Determinant.det A ≠ 0
using positive-definite-invertible[OF assms] invertible-det invertible-mat-def square-mat.simps
by blast

```

## 10 Matrix-Vector Multiplication

**lemma** *diagonal-unit-vec'*:

```

assumes B ∈ carrier-mat n n
assumes diagonal-mat (B::'a::{ring,zero-neq-one} Matrix.mat)
shows B *v (unit-vec n i) = B $$ (i,i) ·v (unit-vec n i)
proof -
  define v :: 'a Matrix.vec where v = unit-vec n i
  have B *v v = Matrix.vec n (λ i. Matrix.scalar-prod (Matrix.row B i) v)

```

**using** *assms* **unfolding** *mult-mat-vec-def* **by** *simp*  
**also have**  $\dots = \text{Matrix.vec } n \ (\lambda i. B \ \$\$ (i,i) * \text{Matrix.vec-index } v \ i)$   
**proof** –  
**have**  $\forall i < n. (\text{Matrix.scalar-prod } (\text{Matrix.row } B \ i) \ v = B \ \$\$ (i,i) * \text{Matrix.vec-index } v \ i)$   
**proof** (*intro allI impI*)  
**fix** *i*  
**assume**  $i < n$   
**have**  $(\text{Matrix.scalar-prod } (\text{Matrix.row } B \ i) \ v) =$   
 $(\sum j \in \{0 ..< n\}. \text{Matrix.vec-index } (\text{Matrix.row } B \ i) \ j * \text{Matrix.vec-index } v \ j)$  **using** *assms*  
**unfolding** *Matrix.scalar-prod-def v-def* **by** *simp*  
**also have**  $\dots = \text{Matrix.vec-index } (\text{Matrix.row } B \ i) \ i * \text{Matrix.vec-index } v \ i$   
**proof** (*rule sum-but-one*)  
**show**  $\forall j < n. j \neq i \longrightarrow \text{Matrix.vec-index } (\text{Matrix.row } B \ i) \ j = 0$   
**proof** (*intro allI impI*)  
**fix** *j*  
**assume**  $j < n$  **and**  $j \neq i$   
**hence**  $\text{Matrix.vec-index } (\text{Matrix.row } B \ i) \ j = B \ \$\$ (i,j)$  **using**  $\langle i < n \rangle \langle j < n \rangle$  *assms*  
**by** *auto*  
**also have**  $\dots = 0$  **using** *assms*  $\langle i < n \rangle \langle j < n \rangle \langle j \neq i \rangle$  **unfolding** *diagonal-mat-def* **by** *simp*  
**finally show**  $\text{Matrix.vec-index } (\text{Matrix.row } B \ i) \ j = 0$  .  
**qed**  
**show**  $i < n$  **using**  $\langle i < n \rangle$  .  
**qed**  
**also have**  $\dots = B \ \$\$ (i,i) * \text{Matrix.vec-index } v \ i$  **using** *assms*  $\langle i < n \rangle$  **by** *auto*  
**finally show**  $(\text{Matrix.scalar-prod } (\text{Matrix.row } B \ i) \ v) = B \ \$\$ (i,i) * \text{Matrix.vec-index } v \ i$  .  
**qed**  
**thus** *?thesis* **by** *auto*  
**qed**  
**also have**  $\dots = B \ \$\$ (i,i) \cdot_v v$  **unfolding** *v-def unit-vec-def* **by** *auto*  
**finally have**  $B *_v v = B \ \$\$ (i,i) \cdot_v v$  .  
**thus** *?thesis* **unfolding** *v-def* **by** *simp*  
**qed**

**lemma** *mat-vec-idx-disjunct-eq-zero*:

**assumes** *A*:  $A \in \text{carrier-mat } m \ n$

**assumes** *x*:  $x \in \text{carrier-vec } n$

**assumes** *some-zero*:  $\forall j < n. (\text{row } A \ i)\$j = 0 \vee x\$j = 0$

**assumes** *i*:  $i < m$

**shows**  $(A *_v x)\$i = 0$

**proof** –

**have** *dim-x*:  $\text{dim-vec } x = n$  **using** *x* **by** *simp*

**have** *dim-A*:  $\text{dim-row } A = m \ \text{dim-col } A = n$  **using** *A* **by** *blast+*

**have** *all-zero*:  $\forall j < n. (\text{row } A \ i)\$j * x\$j = 0$  **using** *some-zero* **by** *fastforce*

**have**  $(A *_v x)\$i = (\text{row } A \ i) \cdot x$  **by** (*rule index-mult-mat-vec, subst dim-A(1), rule i*)  
**also have**  $\dots = (\sum j < n. (\text{row } A \ i)\$j * x\$j)$   
**unfolding** *scalar-prod-def dim-x* **using** *atLeast0LessThan* **by** *presburger*  
**also have**  $\dots = 0$  **using** *all-zero* **by** *simp*  
**finally show** *?thesis* .  
**qed**

## 11 Module Span

**context** *module*  
**begin**

**lemma** *mk-coeffs-of-list:*

**assumes**  $\alpha \in (\text{set } A \rightarrow \text{carrier } R)$

**shows**  $\exists c \in \{0..<\text{length } A\} \rightarrow \text{carrier } R. \forall v \in \text{set } A. \text{mk-coeff } A \ c \ v = \alpha \ v$

**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**  $\alpha \in (\text{set } A' \rightarrow \text{carrier } R)$  **using** *Suc.prem*s **by** *simp*

**moreover from** *a* **have**  $\text{len-}A': n = \text{length } A'$  **using** *Suc(2)* **by** *auto*

**ultimately obtain** *c'* **where** *c'*:

$c' \in \{0..<\text{length } A'\} \rightarrow \text{carrier } R \wedge (\forall v \in \text{set } A'. \text{mk-coeff } A' \ c' \ v = \alpha \ v)$

**using** *Suc.hyps* **by** *blast*

**have**  $\text{len-}A'-A: \text{length } A' = \text{length } A - 1$  **using** *Suc(2)*  $\text{len-}A'$  **by** *presburger*

**moreover have**  $[0..<\text{length } A] = [0..<\text{length } A - 1] @ [\text{length } A - 1]$

**by** (*metis Suc(2) calculation len-A' upt-Suc-append zero-order(1)*)

**ultimately have**  $A-A'-\text{int}: [0..<\text{length } A] = [0..<\text{length } A'] @ [\text{length } A']$  **by** *presburger*

**show** *?case*

**proof**(*cases a \in set A'*)

**case** *True*

**hence**  $A-A'$ :  $\text{set } A = \text{set } A'$  **using** *a* **by** *auto*

**define** *c* **where**  $c \equiv (\lambda i. \text{if } i \in \{0..<\text{length } A'\} \text{ then } c' \ i \ \text{else } \mathbf{0})$

**hence**  $c$ -*carrier*:  $c \in \{0..<\text{length } A\} \rightarrow \text{carrier } R$  **using** *c'* **by** *force*

**moreover have**  $\forall v \in \text{set } A. \text{mk-coeff } A \ c \ v = \alpha \ v$

**proof**

**fix** *v* **assume**  $*$ :  $v \in \text{set } A$

**show**  $\text{mk-coeff } A \ c \ v = \alpha \ v$

**proof**(*cases v = a*)

**case** *True*

**hence**  $\text{find-indices } v \ A = (\text{find-indices } v \ A') @ [\text{length } A - 1]$

**proof**–

**from**  $A-A'$ -int **have** *find-indices*  $v A$   
 $= (\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A']) @ (\text{filter } (\lambda i. A ! i = v) [\text{length } A'])$

**unfolding** *find-indices-def* **by** (*metis Suc.hyps(2)*) *filter-append len-A'*  
**moreover then have**  $(\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A']) = (\text{find-indices } v A')$

**using** *a* **by** *auto*  
**moreover have**  $\text{filter } (\lambda i. A ! i = v) [\text{length } A'] = [\text{length } A']$  **by** (*simp add: True a*)

**ultimately show** *?thesis* **using**  $\text{len-}A'-A$  **by** *argo*  
**qed**  
**hence**  $\text{map } c (\text{find-indices } v A) = (\text{map } c (\text{find-indices } v A')) @ [c (\text{length } A - 1)]$

**by** *auto*  
**hence**  $\text{foldr } (\oplus) (\text{map } c (\text{find-indices } v A)) \mathbf{0}$   
 $= \text{foldr } (\oplus) (\text{map } c (\text{find-indices } v A')) (\mathbf{0} \oplus c (\text{length } A - 1))$   
**by** (*simp add: c-def len-A'-A*)  
**also have**  $\dots = (\text{foldr } (\oplus) (\text{map } c (\text{find-indices } v A')) \mathbf{0}) \oplus c (\text{length } A - 1)$

**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**  $\text{mk-coeff } A c v = \text{mk-coeff } A' c v \oplus c (\text{length } A - 1)$   
**unfolding** *mk-coeff-def R.sumlist-def* .  
**moreover have**  $c (\text{length } A - 1) = \mathbf{0}$  **unfolding** *c-def* **using**  $\text{len-}A'$  *a* **by** *force*

**moreover have**  $\text{mk-coeff } A' c v \in \text{carrier } R$   
**by** (*smt (verit) Pi-iff c' c-def mk-coeff-carrier*)  
**ultimately have**  $\text{mk-coeff } A c v = \text{mk-coeff } A' c v$  **by** *algebra*  
**moreover have**  $\text{mk-coeff } A' c v = \text{mk-coeff } A' c' v$   
**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**  $v\text{-neq-}a$ : *False*  
**hence**  $\text{find-indices } v A = \text{find-indices } v A'$   
**proof-**  
**from**  $A-A'$ -int **have** *find-indices*  $v A$   
 $= (\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A']) @ (\text{filter } (\lambda i. A ! i = v) [\text{length } A'])$

**unfolding** *find-indices-def* **by** (*metis Suc.hyps(2)*) *filter-append len-A'*  
**moreover have**  $(\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A']) = \text{find-indices } v A'$   
**unfolding** *find-indices-def*  
**by** (*smt (verit, cfv-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**  $(\text{filter } (\lambda i. A ! i = v) [\text{length } A']) = []$  **using**  $v\text{-neq-}a$

**by** *auto*  
**ultimately show** *?thesis* **by** *force*  
**qed**

hence  $mk\text{-coeff } A \ c \ v = mk\text{-coeff } A' \ c \ v$  **unfolding**  $mk\text{-coeff-def}$  **by**  $fastforce$   
 also have  $\dots = mk\text{-coeff } A' \ c' \ v$   
   **unfolding**  $mk\text{-coeff-def}$   $find\text{-indices-def}$   $c\text{-def}$   
   **by** ( $smt$  ( $verit$ ,  $best$ )  $atLeastLessThan\text{-upt}$   $list.map\text{-cong}$   $mem\text{-Collect-eq}$   $set\text{-filter}$ )  
   also have  $\dots = \alpha \ v$  **using**  $c' * A\text{-}A'$  **by**  $blast$   
   **finally show**  $?thesis$  .  
 qed  
 qed  
**ultimately show**  $?thesis$  **by**  $blast$   
 next  
 case  $False$   
 hence  $A\text{-}A'$ :  $set \ A' = set \ A - \{a\}$  **by** ( $simp \ add:$   $a$ )  
 define  $c$  **where**  $c \equiv (\lambda i. \text{if } i \in \{0..\text{length } A'\} \text{ then } c' \ i \ \text{else } \alpha \ a)$   
 hence  $c \in \{0..\text{length } A\} \rightarrow carrier \ R$  **using**  $Suc.prem\ a \ c'$  **by**  $fastforce$   
 moreover have  $\forall v \in set \ A. mk\text{-coeff } A \ c \ v = \alpha \ v$   
**proof**  
 fix  $v$  **assume**  $*$ :  $v \in set \ A$   
 show  $mk\text{-coeff } A \ c \ v = \alpha \ v$   
**proof**( $cases \ v = a$ )  
 case  $v\text{-eq-}a$ :  $True$   
 hence  $filter \ (\lambda i. \ A \ ! \ i = v) \ [0..\text{length } A] = [\text{length } A - 1]$   
**proof**–  
 from  $A\text{-}A'\text{-int}$  **have**  $filter \ (\lambda i. \ A \ ! \ i = v) \ [0..\text{length } A]$   
    $= (filter \ (\lambda i. \ A \ ! \ i = v) \ [0..\text{length } A']) \ @ \ (filter \ (\lambda i. \ A \ ! \ i = v) \ [\text{length}$   
 $A'])$   
 by ( $metis \ Suc.hyps(2) \ filter\text{-append} \ len\text{-}A'$ )  
 moreover **have**  $(filter \ (\lambda i. \ A \ ! \ i = v) \ [0..\text{length } A']) = []$   
**proof**–  
 have  $\bigwedge i. \ i < \text{length } A' \implies A \ ! \ i \neq v$  **by** ( $metis \ False \ a \ nth\text{-append}$   $nth\text{-mem} \ v\text{-eq-}a$ )  
 thus  $?thesis$  **by**  $fastforce$   
 qed  
 moreover **have**  $(filter \ (\lambda i. \ A \ ! \ i = v) \ [\text{length } A']) = [\text{length } A']$  **by** ( $simp$   $add:$   $a \ v\text{-eq-}a$ )  
**ultimately show**  $?thesis$  **by** ( $metis \ Suc.hyps(2) \ diff\text{-}Suc\text{-}1 \ len\text{-}A'$   $self\text{-append-conv}2$ )  
 qed  
 hence  $map \ c \ (filter \ (\lambda i. \ A \ ! \ i = v) \ [0..\text{length } A]) = [c \ (\text{length } A')]$   
 by ( $metis \ len\text{-}A' \ Suc.hyps(2) \ diff\text{-}Suc\text{-}1 \ list.map(1) \ list.map(2)$ )  
 moreover **have**  $c \ (\text{length } A') = \alpha \ v$  **by** ( $simp \ add:$   $v\text{-eq-}a \ c\text{-def}$ )  
**ultimately have**  $mk\text{-coeff } A \ c \ v = \alpha \ v \oplus \mathbf{0}$  **unfolding**  $mk\text{-coeff-def}$   $find\text{-indices-def}$  **by**  $force$   
 moreover **have**  $\alpha \ v \in carrier \ R$  **using**  $* \ Suc(3)$  **by**  $blast$   
**ultimately show**  $?thesis$  **by**  $auto$   
 next  
 case  $v\text{-neq-}a$ :  $False$   
 hence  $find\text{-indices } v \ A = find\text{-indices } v \ A'$   
**proof**–

**from**  $A-A'$ -int **have** *find-indices*  $v A$   
 $= (\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A]) @ (\text{filter } (\lambda i. A ! i = v) [\text{length}$   
 $A])$   
**unfolding** *find-indices-def* **by** (*metis Suc.hyps(2) filter-append len-A'*)  
**moreover have**  $(\text{filter } (\lambda i. A ! i = v) [0..<\text{length } A]) = \text{find-indices } v A'$   
**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**  $(\text{filter } (\lambda i. A ! i = v) [\text{length } A]) = []$  **using**  $a v\text{-neq-}a$   
**by auto**  
**ultimately show** *?thesis* **by force**  
**qed**  
**hence**  $\text{mk-coeff } A c v = \text{mk-coeff } A' c v$  **unfolding** *mk-coeff-def* **by fastforce**  
**also have**  $\dots = \text{mk-coeff } A' c' v$   
**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**  $\dots = \alpha v$  **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**  $\text{set } A \subseteq \text{carrier } M$   
**shows**  $\text{span-list } A = \text{span } (\text{set } A)$   
**proof** –  
**have**  $\text{span-list } A \subseteq \text{span } (\text{set } A)$   
**proof**(*rule subsetI*)  
**fix**  $x$  **assume**  $x \in \text{span-list } A$   
**then obtain**  $c$  **where**  $c: x = \text{lincomb-list } c A \wedge c \in \{0..<\text{length } A\} \rightarrow \text{carrier}$   
 $R$   
**unfolding** *span-list-def* **by blast**  
**hence**  $1: \text{lincomb-list } c A = \text{lincomb } (\text{mk-coeff } A c) (\text{set } A)$   
**using** *lincomb-list-as-lincomb[OF assms(1)]* **by presburger**  
**have**  $\text{mk-coeff } A c \in (\text{set } A) \rightarrow \text{carrier } R$  **by** (*simp add: c mk-coeff-carrier*)  
**hence**  $2: \text{lincomb } (\text{mk-coeff } A c) (\text{set } A) \in \text{span } (\text{set } A)$  **unfolding** *span-def*  
**by blast**  
**show**  $x \in \text{span } (\text{set } A)$  **using**  $1\ 2\ c$  **by argo**  
**qed**  
**moreover have**  $\text{span } (\text{set } A) \subseteq \text{span-list } A$   
**proof**(*rule subsetI*)  
**fix**  $x$  **assume**  $x \in \text{span } (\text{set } A)$   
**then obtain**  $\alpha$  **where**  $\alpha: x = \text{lincomb } \alpha (\text{set } A) \wedge \alpha \in (\text{set } A \rightarrow \text{carrier } R)$   
**using**  $*$  *finite-span assms* **by auto**  
**define**  $\alpha'$  **where**  $\alpha' = (\lambda v. \text{if } v \in \text{set } A \text{ then } \alpha v \text{ else } \mathbf{0})$   
**hence**  $\alpha - \alpha': \bigwedge v. v \in \text{set } A \implies \alpha v = \alpha' v$  **by presburger**

```

hence  $x-\alpha'$ :  $x = \text{lincomb } \alpha' (\text{set } A)$ 
using  $\alpha$ 
unfolding  $\text{lincomb-def}$ 
by (smt (verit)  $M.\text{add.finprod-cong}' \text{ Pi-iff assms basic-trans-rules}(31)$  carrier-is-submodule submoduleE(4))
have 1:  $\alpha' \in (\text{set } A \rightarrow \text{carrier } R)$  by (simp add: Pi-cong  $\alpha \alpha'$ -def)
then obtain  $c$  where  $c: c \in \{0..\text{length } A\} \rightarrow \text{carrier } R \wedge (\forall v \in \text{set } A.$ 
mk-coeff  $A \ c \ v = \alpha' \ v)$ 
using mk-coeffs-of-list by blast
have mk-coeff  $A \ c = \alpha'$  unfolding  $\alpha'$ -def by (metis mk-coeff-0  $c \ \alpha-\alpha'$ )
hence  $\text{lincomb } \alpha' (\text{set } A) = \text{lincomb-list } c \ A$ 
using lincomb-list-as-lincomb[OF assms(1), of c]  $c$  by argo
also have  $\dots \in \text{span-list } A$  using  $c$  in-span-listI by blast
finally show  $x \in \text{span-list } A$  using  $x-\alpha'$  by blast
qed
ultimately show ?thesis by blast
qed
end

```

## 12 Module Homomorphism, Linear Combination, and Span

```

context mod-hom
begin

```

```

lemma lincomb-list-distrib:
assumes  $\text{set } S \subseteq \text{carrier } M$ 
assumes  $\alpha \in \{..\text{length } S\} \rightarrow \text{carrier } R$ 
shows  $f (M.\text{lincomb-list } \alpha \ S) = N.\text{lincomb-list } \alpha \ (\text{map } f \ S)$ 
using assms
proof(induct length S arbitrary: S  $\alpha$ )
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:  $n = \text{length } S'$  using Suc(2)  $v$  by auto
have 2:  $\text{set } S' \subseteq \text{carrier } M$  using Suc(3)  $v$  by auto
have 3:  $(\alpha \circ \text{Suc}) \in \{..\text{length } S'\} \rightarrow \text{carrier } R$  using 1 Suc(4) Suc.hyps(2) by
fastforce

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

have *:  $M.\text{lincomb-list } \alpha \ (v \# \ S') = (\alpha \ 0 \odot_M v) \oplus_M (M.\text{lincomb-list } (\alpha \circ \text{Suc}) \ S')$ 
using M.lincomb-list-Cons .

```

**have**  $v \in \text{carrier } M$  **using**  $\text{Suc.prem}(1)$   $v$  **by force**  
**moreover have**  $\alpha \ 0 \in \text{carrier } R$  **using**  $\text{Suc}(4)$   $\text{Suc.hyps}(2)$  **by auto**  
**ultimately have**  $\alpha \ 0 \odot_M v \in \text{carrier } M$  **by blast**  
**moreover have**  $M.\text{lincomb-list } (\alpha \circ \text{Suc}) \ S' \in \text{carrier } M$   
**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**  $f (M.\text{lincomb-list } \alpha \ S) = (f (\alpha \ 0 \odot_M v)) \oplus_N (f (M.\text{lincomb-list } (\alpha \circ \text{Suc}) \ S'))$   
**using**  $f\text{-hom} \ * \ v$  **unfolding**  $\text{module-hom-def}$  **by force**  
**also have**  $\dots = (\alpha \ 0 \odot_N f \ v) \oplus_N (f (M.\text{lincomb-list } (\alpha \circ \text{Suc}) \ S'))$   
**by** (*simp add: <\alpha \ 0 \in \text{carrier } R> <v \in \text{carrier } M>*)  
**also have**  $\dots = (\alpha \ 0 \odot_N f \ v) \oplus_N (N.\text{lincomb-list } (\alpha \circ \text{Suc}) \ (\text{map } f \ S'))$  **using**  
*ih* **by argo**  
**also have**  $\dots = N.\text{lincomb-list } \alpha \ (\text{map } f \ S)$   
**using**  $N.\text{lincomb-list-Cons}$  **by** (*simp add: v*)  
**finally show** *?case* .  
**qed**

**lemma** *lincomb-distrib*:

**assumes** *inj-on f S*  
**assumes**  $S \subseteq \text{carrier } M$   
**assumes**  $\alpha \in S \rightarrow \text{carrier } R$   
**assumes**  $\forall v \in S. \alpha \ v = \beta (f \ v)$   
**assumes** *finite S*  
**shows**  $f (M.\text{lincomb } \alpha \ S) = N.\text{lincomb } \beta (f'S)$   
**proof** –  
**let**  $?a' = (\lambda v. (\alpha \ v) \odot_M v)$   
**let**  $?b' = (\lambda v. (\beta \ v) \odot_N v)$   
**have**  $*$ :  $?a' \in S \rightarrow \text{carrier } M$  **using** *assms(2,3)* **by auto**  
**have**  $f (M.\text{lincomb } \alpha \ S) = f (\text{finsum } M \ ?a' \ S)$  **using**  $M.\text{lincomb-def}$  **by presburger**  
**also have**  $\dots = (\bigoplus_{N a \in S. f ((\alpha \ a) \odot_M a)})$  **using**  $\text{hom-sum}[OF \ \text{assms}(2) \ *]$  .  
**also have**  $\dots = (\bigoplus_{N a \in S. (\alpha \ a) \odot_N (f \ a)})$   
**proof** –  
**have**  $\forall a \in S. a \in \text{carrier } M$  **using** *assms(2)* **by fastforce**  
**moreover have**  $\forall a \in S. \alpha \ a \in \text{carrier } R$  **using** *assms(3)* **by blast**  
**ultimately show** *?thesis*  
**using**  $f\text{-hom}$  **unfolding**  $\text{module-hom-def}$  **by** (*simp add: N.M.add.finprod-cong'*)  
**qed**  
**also have**  $\dots = (\bigoplus_{N a \in S. (\beta (f \ a)) \odot_N (f \ a)})$   
**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**  $\dots = (\bigoplus_{N a \in (f'S). (?b' \ a)})$   
**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**  $\dots = N.\text{lincomb } \beta (f'S)$  **using**  $N.\text{lincomb-def}$  **by presburger**  
**finally show** *?thesis* .  
**qed**

**lemma** *lincomb-distrib-obtain*:  
**assumes** *inj-on*  $f$   $S$   
**assumes**  $S \subseteq \text{carrier } M$   
**assumes**  $\alpha \in S \rightarrow \text{carrier } R$   
**assumes**  $\forall v \in S. \alpha v = \beta (f v)$   
**assumes** *finite*  $S$   
**obtains**  $\beta$  **where**  $(\forall v \in S. \alpha v = \beta (f v)) \wedge f (M.\text{lincomb } \alpha S) = N.\text{lincomb } \beta (f'S)$   
**proof** –  
**obtain**  $\beta$  **where**  $\beta: \forall v \in S. \alpha v = \beta (f v)$   
**proof** –  
**let**  $?\beta = \lambda y. \alpha (THE x. x \in S \wedge f x = y)$   
**have**  $\forall v \in S. \alpha v = ?\beta (f v)$   
**proof**  
**fix**  $v$  **assume**  $v \in S$   
**then have**  $v = (THE x. x \in S \wedge f x = f v)$   
**using** *assms(1)* **by** (*simp add: inj-on-def the-equality*)  
**thus**  $\alpha v = ?\beta (f v)$  **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*  $vs \subseteq \text{carrier } M$   
**shows**  $f'(M.\text{span-list } vs) = N.\text{span-list } (\text{map } f vs)$  (**is** *?lhs = ?rhs*)  
**proof** –  
**have**  $?lhs \subseteq ?rhs$   
**proof**(*rule subsetI*)  
**fix**  $w$  **assume**  $w \in ?lhs$   
**then obtain**  $v$  **where**  $v: v \in M.\text{span-list } vs \wedge f v = w$  **by** *blast*  
**then obtain**  $\alpha$  **where**  $\alpha: v = M.\text{lincomb-list } \alpha vs \wedge \alpha \in \{..<\text{length } vs\} \rightarrow \text{carrier } R$   
**unfolding** *M.span-list-def* **by** *fastforce*  
**hence**  $f v = N.\text{lincomb-list } \alpha (\text{map } f vs)$  **using** *lincomb-list-distrib[OF assms(1)]*  
 $v$  **by** *blast*  
**thus**  $w \in ?rhs$  **using**  $v$   $\alpha$  **unfolding** *N.span-list-def* **by** *force*  
**qed**  
**moreover have**  $?rhs \subseteq ?lhs$   
**proof**(*rule subsetI*)  
**fix**  $w$  **assume**  $w \in ?rhs$   
**then obtain**  $\alpha$  **where**  $\alpha: w = N.\text{lincomb-list } \alpha (\text{map } f vs) \wedge \alpha \in \{..<\text{length } vs\} \rightarrow \text{carrier } R$   
**unfolding** *N.span-list-def* **by** *fastforce*  
**hence**  $w = f (M.\text{lincomb-list } \alpha vs)$   
**using** *lincomb-list-distrib[OF assms]* **by** *presburger*  
**thus**  $w \in ?lhs$  **using**  $\alpha$  **unfolding** *M.span-list-def* **by** *fastforce*

qed  
ultimately show ?thesis by blast  
qed

lemma image-span:

assumes finite vs  
assumes  $vs \subseteq \text{carrier } M$   
shows  $f'(M.\text{span } vs) = N.\text{span } (f'vs)$   
proof –  
obtain vs-list where vs-list: set vs-list = vs using assms(1) finite-list by blast  
have  $f'vs = \text{set } (\text{map } f \text{ vs-list})$  using vs-list by simp  
hence  $N.\text{span } (f'vs) = N.\text{span-list } (\text{map } f \text{ vs-list})$   
by (metis N.span-list-span M.sum-simp assms(2) f-im image-subset-iff)  
moreover have  $M.\text{span } vs = M.\text{span-list } vs\text{-list}$   
using M.span-list-span vs-list assms(2) by presburger  
ultimately show ?thesis using image-span-list assms vs-list by presburger  
qed

lemma submodule-image:

assumes submodule R M' M  
shows submodule R (f'M') N  
using assms  
by (smt (verit, ccfv-threshold) submodule-def N.module-axioms f0-is-0 f-add f-im  
f-smult  
imageE rev-image-eqI subset-iff)

lemma submodule-restrict:

assumes submodule R M' M  
shows  $\text{mod-hom } R (M.\text{md } M') (N.\text{md } (f'M')) f$   
proof –  
interpret M': module R M.md M' by (simp add: M.submodule-is-module assms)  
interpret N': module R N.md (f'M') by (rule submodule-is-module, rule sub-  
module-image[OF assms])  
have  $f \in \text{module-hom } R (M.\text{md } M') (N.\text{md } (f'M'))$   
by (simp add: module-hom-def, meson M'.sum-simp assms f-add f-smult sub-  
module.subset)  
thus ?thesis  
apply (simp add: mod-hom-def mod-hom-axioms-def)  
using M'.module-axioms N'.module-axioms by blast  
qed

end

## 13 Vector Summation

lemma complex-vec-norm-sum:

fixes  $x :: \text{complex vec}$   
assumes  $x \in \text{carrier-vec } n$   
shows  $\text{vec-norm } x = \text{csqrt } ((\sum i \in \{..<n\}. (\text{cmod } (x\$i))^2))$

**proof**–  
**have** \*:  $\bigwedge i. i \in \{..<n\} \implies (\text{conjugate } x)\$i = \text{conjugate } (x\$i)$   
**using** *assms by auto*  
**have** \*\*:  $\bigwedge i. i \in \{..<n\} \implies (x\$i) * \text{conjugate } (x\$i) = (\text{cmod } (x\$i))^{\wedge 2}$   
**using** *mult-conj-cmod-square by blast*  
**have** *vec-norm*  $x = \text{csqrt } (x \cdot c \ x)$   
**by** (*simp add: vec-norm-def*)  
**also have**  $\dots = \text{csqrt } (\sum i \in \{..<n\}. (x\$i) * (\text{conjugate } x)\$i)$   
**by** (*metis assms atLeast0LessThan carrier-vecD dim-vec-conjugate scalar-prod-def*)  
**also have**  $\dots = \text{csqrt } (\sum i \in \{..<n\}. (x\$i) * \text{conjugate } (x\$i))$   
**by** (*simp add: \**)  
**also have**  $\dots = \text{csqrt } ((\sum i \in \{..<n\}. (\text{cmod } (x\$i))^{\wedge 2}))$  **using** \*\* **by** *fastforce*  
**finally show** *?thesis* .  
**qed**

**lemma** *inner-prod-vec-sum*:  
**assumes**  $v \in \text{carrier-vec } n$   
**assumes**  $w \in \text{carrier-vec } n$   
**assumes**  $B \subseteq \text{carrier-vec } n$   
**assumes** *finite B*  
**assumes**  $v = \text{finsum-vec } \text{TYPE}('a::\text{conjugatable-ring}) \ n \ (\lambda b. \text{cs } b \cdot_v \ b) \ B$   
**shows**  $\text{inner-prod } w \ v = (\sum b \in B. \text{cs } b * \text{inner-prod } w \ b)$

**proof**–  
**let** *?vs* =  $\lambda b. \text{cs } b \cdot_v \ b$   
**let** *?f* =  $\lambda i \ b. \text{if } i \in \{..<n\} \text{ then } (?vs \ b)\$i * (\text{conjugate } w)\$i \text{ else } 0$   
**have** *f*:  $\bigwedge y. \text{finite } \{x. ?f \ x \ y \neq 0\}$

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

**have**  $\text{inner-prod } w \ v = (\sum i \in \{..<n\}. v\$i * (\text{conjugate } w)\$i)$   
**unfolding** *scalar-prod-def* **using** *atLeast0LessThan assms(2) by force*  
**moreover have**  $\bigwedge i. i \in \{..<n\} \implies v\$i = (\sum b \in B. (?vs \ b)\$i)$   
**using** *index-finsum-vec[OF assms(4) - vs]* **unfolding** *assms(5) by blast*  
**ultimately have**  $\text{inner-prod } w \ v = (\sum i \in \{..<n\}. (\sum b \in B. (?vs \ b)\$i) * (\text{conjugate } w)\$i)$   
**by force**  
**also have**  $\dots = (\sum i \in \{..<n\}. \sum b \in B. (?vs \ b)\$i * (\text{conjugate } w)\$i)$   
**by** (*simp add: sum-distrib-right*)  
**also have**  $\dots = (\sum i \in \{..<n\}. \sum b \in B. ?f \ i \ b)$   
**by** *fastforce*  
**also have**  $\dots = \text{Sum-any } (\lambda i. \sum b \in B. ?f \ i \ b)$

**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**  $\dots = (\sum b \in B. (\text{Sum-any } (\lambda i. ?f i b)))$   
**using** *Sum-any-sum-swap*[*OF* *assms*(4) *f*, of  $\lambda x. x$ ].  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. (?vs b)\$i * (\text{conjugate } w)\$i))$   
**proof**–  
**have**  $\bigwedge b. b \in B \implies (\sum i \in \{..<n\}. (?vs b)\$i * (\text{conjugate } w)\$i) = \text{Sum-any}$   
 $(\lambda i. ?f i b)$   
**using** *Sum-any.conditionalize*[of  $\{..<n\}$ ] **by** *blast*  
**thus** *?thesis* **by** *fastforce*  
**qed**  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. (cs\ b * b\$i) * (\text{conjugate } w)\$i))$   
**using** *b-scale* **by** *simp*  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. cs\ b * (b\$i * (\text{conjugate } w)\$i)))$   
**using** *assoc* **by** *force*  
**also have**  $\dots = (\sum b \in B. cs\ b * (\sum i \in \{..<n\}. b\$i * (\text{conjugate } w)\$i))$   
**by** (*simp add: sum-distrib-left*)  
**also have**  $\dots = (\sum b \in B. cs\ b * \text{inner-prod } w\ b)$   
**unfolding** *scalar-prod-def* **using** *atLeast0LessThan* *assms*(2) **by** *force*  
**finally show** *?thesis* .  
**qed**

**lemma** *sprod-vec-sum*:

**assumes**  $v \in \text{carrier-vec } n$   
**assumes**  $w \in \text{carrier-vec } n$   
**assumes**  $B \subseteq \text{carrier-vec } n$   
**assumes** *finite*  $B$   
**assumes**  $v = \text{finsum-vec } \text{TYPE}('a::\{\text{comm-ring}\})\ n\ (\lambda b. cs\ b \cdot_v\ b)\ B$   
**shows**  $w \cdot v = (\sum b \in B. cs\ b * (w \cdot b))$

**proof**–

**let**  $?vs = \lambda b. cs\ b \cdot_v\ b$   
**let**  $?f = \lambda i\ b. \text{if } i \in \{..<n\} \text{ then } (?vs\ b)\$i * w\$i \text{ else } 0$   
**have**  $f: \bigwedge y. \text{finite } \{x. ?f\ x\ y \neq 0\}$   
**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**  $vs: ?vs \in B \rightarrow \text{carrier-vec } n$  **using** *assms*(3) **by** *force*  
**have** *b-scale*:  $\bigwedge i\ b. i \in \{..<n\} \implies b \in B \implies (?vs\ b)\$i = cs\ b * b\$i$   
**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*:  $\bigwedge b. b \in B \implies \text{dim-vec } b = n$   
**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**  $\bigwedge i. i \in \{..<n\} \implies v\$i = (\sum b \in B. (?vs\ b)\$i)$   
 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**  $\dots = (\sum i \in \{..<n\}. \sum b \in B. (?vs\ b)\$i * w\$i)$   
 by (*simp add: sum-distrib-right*)  
**also have**  $\dots = (\sum i \in \{..<n\}. \sum b \in B. ?f\ i\ b)$   
 by *fastforce*  
**also have**  $\dots = \text{Sum-any } (\lambda i. \sum b \in B. ?f\ i\ b)$   
 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**  $\dots = (\sum b \in B. (\text{Sum-any } (\lambda i. ?f\ i\ b)))$   
 using *Sum-any-sum-swap[OF assms(4) f, of \lambda x. x]* .  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. (?vs\ b)\$i * w\$i))$   
**proof** –  
 have  $\bigwedge b. b \in B \implies (\sum i \in \{..<n\}. (?vs\ b)\$i * w\$i) = \text{Sum-any } (\lambda i. ?f\ i\ b)$   
 using *Sum-any.conditionalize[of {..<n}]* **by** *blast*  
 thus *?thesis* **by** *fastforce*  
**qed**  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. (cs\ b * b\$i) * w\$i))$   
 using *b-scale* **by** *simp*  
**also have**  $\dots = (\sum b \in B. (\sum i \in \{..<n\}. cs\ b * (b\$i * w\$i)))$   
 using *assoc* **by** *force*  
**also have**  $\dots = (\sum b \in B. cs\ b * (\sum i \in \{..<n\}. b\$i * w\$i))$   
 by (*simp add: sum-distrib-left*)  
**also have**  $\dots = (\sum b \in B. cs\ b * (\sum i \in \{..<n\}. w\$i * b\$i))$   
 by (*metis (no-types, lifting) Groups.mult-ac(2) sum.cong*)  
**also have**  $\dots = (\sum b \in B. cs\ b * (w \cdot b))$   
**unfolding** *scalar-prod-def* **using** *atLeast0LessThan assms(3) B-dim* **by** *fast-*  
*force*  
**finally show** *?thesis* .  
**qed**

**lemma** *mat-vec-mult-sum:*

**assumes**  $v \in \text{carrier-vec } n$

**assumes**  $A \in \text{carrier-mat } n\ n$

**assumes**  $B \subseteq \text{carrier-vec } n$

**assumes** *finite B*

**assumes**  $v = \text{finsum-vec } \text{TYPE}('a::\text{comm-ring})\ n\ (\lambda b. cs\ b \cdot_v\ b)\ B$

**shows**  $A *_v\ v = \text{finsum-vec } \text{TYPE}('a::\text{comm-ring})\ n\ (\lambda b. cs\ b \cdot_v\ (A *_v\ b))\ B$

(**is** *?lhs = ?rhs*)

**proof** –

**have**  $\bigwedge i. i < n \implies ?lhs\$i = ?rhs\$i$

**proof** –

**fix**  $i$  **assume**  $*$ :  $i < n$

**let**  $?r = \text{row } A\ i$

**have**  $?lhs\$i = ?r \cdot v$  **using**  $*$  *assms(2)* **unfolding** *mult-mat-vec-def* **by** *simp*

**also have**  $\dots = (\sum b \in B. (cs\ b * (?r \cdot b)))$

```

    using sprod-vec-sum[OF assms(1) - assms(3) assms(4) assms(5)] assms(2)
  by fastforce
    also have ... = ( $\sum b \in B. (cs\ b * ((A *_{\nu} b)\$i))$ )
    using * assms(2) unfolding mult-mat-vec-def by simp
    also have ... = ( $\sum b \in B. ((cs\ b \cdot_{\nu} (A *_{\nu} b))\$i)$ )
    using assms(2) * by force
    also have ... = (finsum-vec TYPE('a::comm-ring) n ( $\lambda b. cs\ b \cdot_{\nu} (A *_{\nu} b)$ )
  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  $\in$  carrier-vec n using assms(1) assms(2) by force
  moreover have ?rhs  $\in$  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

```

## 14 Vector Space

```

context vectorspace
begin

```

```

lemma basis-subset-carrier: basis B  $\implies$  B  $\subseteq$  carrier V by (simp add: basis-def)

```

```

lemma dim-of-lin-indpt-span:
  assumes li: lin-indpt S
  assumes carrier: S  $\subseteq$  carrier V
  assumes fin: finite S
  shows vectorspace.dim K (vs (span S)) = card S
  using dim-span[unfolded maximal-def, OF carrier fin] li
  by fast

```

```

lemma subspace-fin-dim:
  assumes W: subspace K W V
  assumes fin: fin-dim
  shows vectorspace.fin-dim K (vs W)
proof–
  have ?thesis if { $\mathbf{0}_V$ } = W
    using W that
    by (metis empty-subsetI finite.emptyI local.carrier-vs-is-self local.span-empty
span-li-not-depend(1) subspace.submod
subspace-is-vs vectorspace.fin-dim-def)
  moreover have ?thesis if *: { $\mathbf{0}_V$ }  $\neq$  W
  proof–
  obtain b where b: b  $\in$  W  $\wedge$  b  $\neq$   $\mathbf{0}_V$  using W * submodule-def subspace-def
  by blast

```

```

let ?P = λB. B ⊆ W ∧ lin-indpt B
have ∧B. ?P B ⇒ finite B ∧ card B ≤ dim
using W fin by (metis submodule-def subspace-def dual-order.trans li-le-dim(1,2))
moreover have Pb: ?P {b}
proof-
  have b ∉ span {} by (simp add: b local.span-empty)
  moreover have {b} ⊆ carrier V using b W [unfolded subspace-def submodule-def] by fast
  ultimately show ?thesis using lindep-span[of {b}, simplified] b by blast
qed
ultimately obtain B where B: finite B ∧ maximal B ?P by (metis (no-types, lifting) maximal-exists)

have (λS. S ⊆ W ∧ ¬ LinearCombinations.module.lin-dep K (vs W) S) = ?P
using B [unfolded maximal-def] span-li-not-depend(2) W by auto
hence vectorspace.basis K (vs W) B
using B vectorspace.max-li-is-basis[OF subspace-is-vs[OF W], of B, simplified]
by argo
thus ?thesis
using B [unfolded maximal-def]
unfolding vectorspace.fin-dim-def[OF subspace-is-vs[OF W]] vectorspace.basis-def[OF subspace-is-vs[OF W]]
by auto
qed
ultimately show ?thesis by argo
qed

lemma nontriv-obtain:
  assumes dim ≠ 0 fin-dim
  obtains v where v ∈ carrier V v ≠ 0_V
proof-
  obtain B where B: basis B ∧ 0 < card B using assms dim-basis finite-basis-exists
  by force
  then obtain b where b ∈ B ∧ b ≠ 0_V
  by (metis all-not-in-conv basis-def bot-nat-0.not-eq-extremum card-eq-0-iff vs-zero-lin-dep)
  thus ?thesis using that[of b] B [unfolded basis-def] by blast
qed

lemma nontriv-subspace-obtain:
  assumes subspace K W V
  assumes vectorspace.dim K (vs W) ≠ 0
  assumes fin-dim
  obtains v where v ∈ W v ≠ 0_V
  using assms vectorspace.nontriv-obtain[of K vs W] subspace-fin-dim subspace-is-vs
  by auto

lemma nontriv-exists:
  assumes dim ≠ 0 fin-dim
  shows ∃ v ∈ carrier V. v ≠ 0_V

```

```

using nontriv-obtain[OF assms] by metis

lemma nontriv-subspace-exists:
assumes subspace K W V
assumes vectorspace.dim K (vs W) ≠ 0
assumes fin-dim
shows  $\exists v \in W. v \neq \mathbf{0}_V$ 
using nontriv-subspace-obtain[OF assms] by metis

lemma dim-sum-nontriv-int:
assumes W1: subspace K W1 V
assumes W2: subspace K W2 V
assumes dim: dim < vectorspace.dim K (vs W1) + vectorspace.dim K (vs W2)
assumes fin: fin-dim
shows  $\exists v \in W_1 \cap W_2. v \neq \mathbf{0}_V$ 
proof –
obtain B1 where B1: vectorspace.basis K (vs W1) B1 finite B1
using W1 fin subspace-fin-dim subspace-is-vs vectorspace.finite-basis-exists by
blast
obtain B2 where B2: vectorspace.basis K (vs W2) B2 finite B2
using W2 fin subspace-fin-dim subspace-is-vs vectorspace.finite-basis-exists by
blast

have basis-carriers: B1 ⊆ carrier V B2 ⊆ carrier V
using B1 W1 unfolding vectorspace.basis-def[OF subspace-is-vs[OF W1]] sub-
space-def submodule-def
using B2 W2 unfolding vectorspace.basis-def[OF subspace-is-vs[OF W2]] sub-
space-def submodule-def
by auto

show ?thesis
proof(cases B1 ∩ B2 ≠ {})
case True
then obtain b where b ∈ B1 ∩ B2 by blast
hence b ∈ W1 ∩ W2 ∧ b ≠ 0V
using B1 B2 W1 W2
using basic-trans-rules(31)[of B1 W1 b] basic-trans-rules(31)[of B2 W2 b]
one-zeroI
span-li-not-depend(2)[of B1 W1] subspace-is-vs[of W2] subspace-is-vs[of W1]
vectorspace.basis-def[of K vs W2 B2]
vectorspace.basis-def[of K vs W1 B1] zero-lin-dep[of B1]
by force
thus ?thesis by blast
next
case False
hence card (B1 ∪ B2) = card B1 + card B2 using B1(2) B2(2)
card-Un-disjoint by blast
hence not-li: ¬ lin-indpt (B1 ∪ B2)
using li-le-dim(2)[OF fin, of B1 ∪ B2] B1 B2 basis-carriers dim

```

```

unfolding vectorspace.dim-basis[OF subspace-is-vs[OF W1] B1(2) B1(1)]
unfolding vectorspace.dim-basis[OF subspace-is-vs[OF W2] B2(2) B2(1)]
by auto

have  $\{\} \subseteq B_1 \wedge \text{lin-indpt } (\{\} \cup B_2)$ 
by (metis B2(1) LinearCombinations.module.span-li-not-depend(2) LinearCombinations.submodule-def Un-empty-left
VectorSpace.subspace-def W2 empty-subsetI local.module.carrier-vs-is-self
subspace-is-vs vectorspace.basis-def)
then obtain B1' where B1': maximal B1' ( $\lambda B_1'. B_1' \subseteq B_1 \wedge \text{lin-indpt } (B_1' \cup B_2)$ )
using B1(2) by (metis (no-types, lifting) maximal-exists-superset)

have B1'  $\neq B_1$ 
using B1' not-li unfolding maximal-def by fastforce
then obtain b1 where b1: b1  $\in B_1 - B_1'$ 
using B1' by (metis (lifting) maximal-def psubsetI psubset-imp-ex-mem)
let ?S = B1'  $\cup B_2$ 

have lin-dep (?S  $\cup \{b_1\}$ ) using b1 B1' unfolding maximal-def by auto
hence b1-span: b1  $\in \text{span } ?S$ 
using B1' [unfolded maximal-def]
using lin-dep-iff-in-span[of ?S b1] basis-carriers False b1
by blast
also have ... = subspace-sum (span B1') (span B2)
using span-union-is-sum[of B1' B2] basis-carriers B1' [unfolded maximal-def]
by force
finally obtain v1' v2 where vs: b1 = v1'  $\oplus_V v_2 \wedge v_1' \in \text{span } B_1' \wedge v_2 \in \text{span } B_2$ 
unfolding submodule-sum-def by fast

have v2  $\neq \mathbf{0}_V$ 
proof(rule ccontr, clarify)
assume v2 =  $\mathbf{0}_V$ 
hence b1  $\in \text{span } B_1'$ 
using vs B1' basis-carriers(1)
by (metis (no-types, lifting) M.add.r-one maximal-def span-closed subset-trans)
moreover have lin-indpt B1'
using B1' by (metis (lifting) Un-subset-iff maximal-def order.refl supset-ld-is-ld)
moreover have  $\neg \text{lin-dep } (B_1' \cup \{b_1\})$ 
proof-
have B1'  $\cup \{b_1\} \subseteq B_1$ 
using B1' b1 by (metis (lifting) DiffE insert-subset insert-union maximal-def)
thus ?thesis
using B1' [unfolded vectorspace.basis-def[OF subspace-is-vs[OF W1]]] W1
by (metis VectorSpace.subspace-def local.module.carrier-vs-is-self span-li-not-depend(2) subset-li-is-li)

```

```

qed
ultimately show False
  using lin-dep-iff-in-span[of B1' b1] B1' b1 basis-carriers(1)
  by (metis (no-types, lifting) Diff-iff Set.basic-monos(7) basic-trans-rules(23))
maximal-def)
qed
moreover have  $b_1 \ominus_V v_1' = v_2$ 
  using B1' b1-span basis-carriers vs
  by (smt (verit, ccfv-SIG) M.add.m-comm M.r-neg1 Un-least a-minus-def
basic-trans-rules(23)
local.span-neg maximal-def span-closed)
moreover have  $b_1 \ominus_V v_1' \in \text{span } B_1$ 
proof-
  have  $v_1' \in \text{span } B_1$ 
  using vs B1' by (metis (no-types, lifting) maximal-def span-is-monotone
subsetD)
  thus ?thesis
  using b1 basis-carriers calculation(2) vs by (metis Diff-iff local.span-add
span-closed span-mem)
qed
moreover have  $v_2 \in \text{span } B_2$  using vs by fast
ultimately have  $v_2 \neq \mathbf{0}_V \wedge v_2 \in \text{span } B_1 \cap \text{span } B_2$ 
  by fastforce
thus ?thesis
  using B1 B2 W1 W2
  by (metis is-module subspace-is-vs vectorspace.basis-def local.module.carrier-vs-is-self
span-li-not-depend(1))
qed
qed
end

```

## 15 Linear Map

```

context linear-map
begin

```

```

lemma inj-image-lin-indpt:

```

```

  assumes inj-on T (carrier V)

```

```

  assumes  $S \subseteq \text{carrier } V$ 

```

```

  assumes V.module.lin-indpt S

```

```

  assumes finite S

```

```

  shows W.module.lin-indpt (T'S)

```

```

proof(rule ccontr)

```

```

  assume  $\neg W.\text{module.lin-indpt } (T'S)$ 

```

```

  then obtain  $B \beta b$  where B: finite B

```

```

     $\wedge B \subseteq T'S$ 

```

```

     $\wedge (\beta \in (B \rightarrow \text{carrier } K))$ 

```

```

     $\wedge (\text{lincomb } \beta B = \mathbf{0}_W)$ 

```

$\wedge (b \in B)$   
 $\wedge (\beta b \neq \mathbf{0}_K)$   
**using** *W.module.lin-dep-def* **by** *auto*  
**define** *A* **where**  $A \equiv \{a \in S. T a \in B\}$   
**define**  $\alpha$  **where**  $\alpha \equiv (\lambda v. \beta (T v))$   
**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*:  $A \subseteq \text{carrier } V$  **using** *A-def assms(2)* **by** *blast*  
**have** *3*:  $\alpha \in A \rightarrow \text{carrier } K$   
**proof**  
**fix** *x* **assume**  $x \in A$   
**moreover then have**  $T x \in \text{carrier } W$  **using** *2* **by** *blast*  
**ultimately show**  $\alpha x \in \text{carrier } K$  **unfolding**  $\alpha$ -*def* **using** *B A-def* **by** *blast*  
**qed**  
**have** *4*:  $\forall v \in A. \alpha v = \beta (T v)$  **using**  $\alpha$ -*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**  $\text{lincomb } \beta B = T (V.\text{module.lincomb } \alpha A)$  **using** *lincomb-distrib[OF 1 2 3 4 5]* **by** *argo*  
**hence**  $T (V.\text{module.lincomb } \alpha A) = \mathbf{0}_W$  **using** *B* **by** *argo*  
**moreover have**  $T (\mathbf{0}_V) = \mathbf{0}_W$  **by** *auto*  
**ultimately have**  $*$ :  $(V.\text{module.lincomb } \alpha A) = \mathbf{0}_V$  **using** *assms(1)* **by** (*simp add: 2 3 inj-onD*)  
**moreover obtain** *a* **where**  $T a = b \wedge a \in A$  **using** *B*  $\langle B = T'A \rangle$  **by** *blast*  
**moreover then have**  $\alpha a \neq \mathbf{0}_K$  **by** (*simp add: B*  $\alpha$ -*def*)  
**moreover have**  $A \subseteq S$  **using** *A-def* **by** *blast*  
**ultimately show** *False* **using** *assms(3) 5 3 V.module.lin-dep-def* **by** *blast*  
**qed**

**lemma** *subspace-restrict*:

**assumes** *subspace K V' V*  
**shows** *linear-map K (V.vs V') (W.vs (T'V')) T*  
**using** *assms submodule-restrict[of V']*  
**by** (*simp add: field-axioms linear-map-def mod-hom.axioms(1,2) vectorspace-def*)

**lemma** *subspace-image*:

**assumes** *subspace K V' V*  
**shows** *subspace K (T'V') W*  
**using** *assms submodule-image[of V']* **by** (*meson subspace-def W.vectorspace-axioms*)

**lemma** *inj-subspace-image*:

**assumes** *inj: inj-on T (carrier V)*  
**assumes** *subspace: subspace K V' V*  
**assumes** *fin: V.fin-dim*  
**shows**  $\text{vectorspace.dim } K (W.\text{vs } (T'V')) = \text{vectorspace.dim } K (V.\text{vs } V')$

**proof** –

**interpret** *restrict: linear-map K V.vs V' W.vs (T'V') T* **using** *subspace-restrict[OF*

```

subspace] .
  have inj': inj-on T (carrier (V.vs V'))
  using inj subspace by (simp add: submodule-def VectorSpace.subspace-def inj-on-subset)
  have carrier-im: T'carrier (V.vs V') = carrier (W.vs (T'V'))
  using V.carrier-vs-is-self W.carrier-vs-is-self by presburger
  show ?thesis
  by (rule restrict.dim-eq[OF V.subspace-fin-dim[OF subspace fin] inj' carrier-im,
symmetric])
qed

end

lemma linear-map-mat:
  assumes A ∈ 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 ∀ v1 ∈ carrier ?V. ∀ v2 ∈ carrier ?V. ?T (v1 + v2) = ?T
v1 + ?T v2
      by (metis module-vec-def assms monoid-record-simps(1) mult-add-distrib-mat-vec)
      moreover have ∀ α ∈ carrier ?K. ∀ v ∈ 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

```

## 16 Matrix as Linear Map

```

locale mat-as-linear-map =
  fixes A :: 'a::field mat
  fixes m n
  assumes carrier: A ∈ carrier-mat m n
begin

```

**abbreviation**  $V \equiv \text{module-vec } \text{TYPE}(a) \ n$

**abbreviation**  $W \equiv \text{module-vec } \text{TYPE}(a) \ m$

**sublocale**  $\text{linear-map class-ring } V \ W \ (*_v) \ A$   
**by** (*rule linear-map-mat[OF carrier]*)

**abbreviation**  $f \equiv (*_v) \ A$

**end**

## 17 Matrix Isometry

**locale**  $\text{isometry-mat} = \text{mat-as-linear-map } A \ m \ n$  **for**  $A :: \text{complex mat}$  **and**  $m \ n$   
+

**assumes**  $\text{isom}: A^H * A = 1_m \ n$

**begin**

**lemma** *preserves-norm*:  $v \in \text{carrier-vec } n \implies \text{vec-norm } v = \text{vec-norm } (f \ v)$

**by** (*metis adjoint-is-conjugate-transpose carrier inner-prod-mult-mat-vec-right isom one-mult-mat-vec vec-norm-def*)

**lemma** *is-inj*:  $\text{inj-on } f \ (\text{carrier } V)$

**by** (*smt (verit, ccfv-threshold) adjoint-dim' adjoint-is-conjugate-transpose as-soc-mult-mat-vec carrier inj-on-def isom one-mult-mat-vec vec-space.cV*)

**end**

## 18 Compression

**lemma** *compression-is-hermitian*:

**assumes**  $B: B \in \text{carrier-mat } n \ m$

**assumes**  $A: A \in \text{carrier-mat } n \ n$  *hermitian*  $A$

**shows** *hermitian*  $(B^H * A * B) \ B^H * A * B \in \text{carrier-mat } m \ m$

**proof** –

**have**  $((B^H * A) * B)^H = B^H * (B^H * A)^H$

**using**  $A(1) \ B$

**by** (*metis adjoint-dim' adjoint-is-conjugate-transpose adjoint-mult mult-carrier-mat*)

**also have**  $\dots = B^H * (A^H * B^{HH})$

**using**  $A(1) \ \text{assms}(1)$

**by** (*metis adjoint-dim' adjoint-is-conjugate-transpose adjoint-mult*)

**also have**  $\dots = B^H * A * B$

**using**  $A \ B$

**by** (*smt (verit, best) Matrix.transpose-transpose adjoint-dim' adjoint-is-conjugate-transpose assoc-mult-mat conjugate-id hermitian-def transpose-conjugate*)

**finally show** *hermitian*  $(B^H * A * B)$   
**unfolding** *hermitian-def adjoint-is-conjugate-transpose*[of  $B^H * A * B$ ].  
**show**  $B^H * A * B \in \text{carrier-mat } m \ m$   
**using**  $A(1) \ B$  **by** (*meson More-Matrix.carrier-vec-conjugate mult-carrier-mat transpose-carrier-mat*)  
**qed**

## 19 Submatrix

### 19.1 Submatrix of Injective Function on Indices, as Compression

Note, with this definition of submatrix, reordering the indices is possible.

**definition** *submatrix-of-inj* :: 'a mat  $\Rightarrow$  (nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat)  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  'a mat **where**

*submatrix-of-inj*  $A \ f \ g \ m \ n \equiv \text{mat } m \ n \ (\lambda(i,j). \ A\$\$(f \ i, \ g \ j))$

**lemma** *submatrix-of-inj-as-compression*:

**fixes**  $A :: \text{complex mat}$

**fixes**  $f :: \text{nat} \Rightarrow \text{nat}$

**assumes**  $A: A \in \text{carrier-mat } n \ n$

**assumes**  $f: f : \{..<m\} \rightarrow \{..<n\}$

**assumes** *inj*: *inj-on*  $f \ \{..<m\}$

**defines**  $B \equiv \text{submatrix-of-inj } A \ f \ f \ m \ m$

**defines**  $(Q::\text{complex mat}) \equiv \text{mat } n \ m \ (\lambda(i,j). \ \text{if } i = f \ j \ \text{then } 1 \ \text{else } 0)$

**shows**  $B = Q^H * A * Q$  *isometry-mat*  $Q \ n \ m \ Q \in \text{carrier-mat } n \ m$

**proof** –

**show** *dim*:  $Q \in \text{carrier-mat } n \ m$  **unfolding** *Q-def* **by** *simp*

**note** *m-leq-n* = *card-inj*[*OF f inj, simplified*]

**let**  $? \varphi = \lambda j \ k. \ \text{if } k = f \ j \ \text{then } 1 \ \text{else } 0$

**have** \*:  $\bigwedge j. \ j < m \implies (\sum k < n. \ \text{cnj } (? \varphi \ j \ k) * ? \varphi \ j \ k) = 1$

**proof** –

**fix**  $j$  **assume**  $j: j < m$

**have** *ff*:  $f \ j \in \{..<n\}$  **using** *ff* **by** *blast*

**have**  $(\sum k < n. \ \text{cnj } (? \varphi \ j \ k) * ? \varphi \ j \ k) = (\sum k \in \{..<n\} - \{f \ j\}. \ \text{cnj } (? \varphi \ j \ k) * ? \varphi \ j \ k) + (\text{cnj } (? \varphi \ j \ (f \ j)) * ? \varphi \ j \ (f \ j))$

**using** *ff* **by** (*subst sum-diff1, simp-all*)

**thus**  $(\sum k < n. \ \text{cnj } (? \varphi \ j \ k) * ? \varphi \ j \ k) = 1$  **by** *simp*

**qed**

**have**  $\bigwedge i \ j \ k. \ i < m \implies j < m \implies i \neq j \implies k < n \implies \text{cnj } (? \varphi \ i \ k) * ? \varphi \ j \ k = 0$

**using** *inj* **by** (*simp add: inj-on-eq-iff*)

**hence** \*\*:  $\bigwedge i \ j. \ i < m \implies j < m \implies i \neq j \implies (\sum k < n. \ \text{cnj } (? \varphi \ i \ k) * ? \varphi \ j \ k) = 0$

**by** (*meson lessThan-iff sum.not-neutral-contains-not-neutral*)

```

have  $Q^H * Q = 1_m$ 
  by (rule eq-matI, auto simp: Q-def m-leq-n **)
thus isometry-mat Q n m
  using dim isometry-mat-axioms-def[of Q m] mat-as-linear-map.intro[of Q n m]
  unfolding isometry-mat-def
  by blast

have  $\bigwedge i j. i < m \implies j < m \implies \text{row } (Q^H * A) i \$ (f j) = A \$ \$ (f i, f j)$ 
proof -
  fix i j assume i:  $i < m$  and j:  $j < m$ 
  have fj:  $f j < n$  using f j m-leq-n by blast
  have row  $(Q^H * A) i \$ (f j) = \text{row } A (f i) \$ (f j)$ 
    using i f j A f sum-diff1[of  $\{..<n\}$   $\lambda k. \text{cnj } (if k = f i \text{ then } 1 \text{ else } 0) * A \$ \$$ 
    ( $k, f j$ )  $f i$ ]
    by (subst adjoint-is-conjugate-transpose) (force simp: Q-def row-def ad-
joint-def)
  thus row  $(Q^H * A) i \$ (f j) = A \$ \$ (f i, f j)$ 
    using A f j by (metis carrier-matD(2) index-vec row-def)
qed
hence  $\bigwedge i j k. i < m \implies j < m \implies k < n \implies \text{row } (Q^H * A) i \$ k * \text{col } Q j \$$ 
 $k = (if k = f j \text{ then } A \$ \$ (f i, f j) \text{ else } 0)$ 
  unfolding Q-def row-def col-def times-mat-def scalar-prod-def by force
hence *:  $\bigwedge i j. i < \text{dim-row } Q^H \implies j < \text{dim-col } Q \implies B \$ \$ (i, j) = \text{row } (Q^H$ 
 $* A) i \cdot \text{col } Q j$ 
  using dim m-leq-n f by (auto simp: scalar-prod-def B-def submatrix-of-inj-def)

show  $B = Q^H * A * Q$ 
  apply (rule eq-matI, simp add: *)
  using dim A by (simp add: B-def submatrix-of-inj-def)+
qed

```

**lemma** *submatrix-of-inj-as-compression-obt*:

```

fixes A :: complex mat
fixes f :: nat  $\Rightarrow$  nat
assumes A:  $A \in \text{carrier-mat } n n$ 
assumes f:  $f : \{..<m\} \rightarrow \{..<n\}$ 
assumes inj: inj-on f  $\{..<m\}$ 
defines B  $\equiv$  submatrix-of-inj A f f m m
obtains Q where  $B = Q^H * A * Q$  isometry-mat Q n m  $Q \in \text{carrier-mat } n m$ 
using submatrix-of-inj-as-compression[OF assms(1-3), folded B-def] by blast

```

## 19.2 Submatrix of pick Function, as Compression

**lemma** *submatrix-inj*:

```

assumes A:  $A \in \text{carrier-mat } m n$ 
assumes I:  $I \subseteq \{..<m\}$ 
assumes J:  $J \subseteq \{..<n\}$ 
defines  $m' \equiv \text{card } I$ 
defines  $n' \equiv \text{card } J$ 

```

**defines**  $B \equiv \text{submatrix } A \ I \ J$   
**defines**  $f \equiv \lambda i. \text{pick } I \ i$   
**defines**  $g \equiv \lambda j. \text{pick } J \ j$   
**shows**  $f : \{..<m'\} \rightarrow \{..<m\}$   $g : \{..<n'\} \rightarrow \{..<n\}$   
 $\text{inj-on } f \ \{..<m'\} \ \text{inj-on } g \ \{..<n'\}$   
 $B = \text{submatrix-of-inj } A \ f \ g \ m' \ n'$   
**proof** –  
**show**  $f : \{..<m'\} \rightarrow \{..<m\}$  **unfolding**  $f\text{-def } m'\text{-def}$  **using**  $I(1)$   $\text{pick-in-set-le}$   
**by**  $\text{fastforce}$   
**show**  $g : \{..<n'\} \rightarrow \{..<n\}$  **unfolding**  $g\text{-def } n'\text{-def}$  **using**  $J(1)$   $\text{pick-in-set-le}$  **by**  
 $\text{fastforce}$   
**show**  $\text{inj-on } f \ \{..<m'\}$   
**unfolding**  $\text{inj-on-def } f\text{-def } m'\text{-def}$  **by**  $(\text{metis lessThan-iff nat-neq-iff pick-mono-le})$   
**show**  $\text{inj-on } g \ \{..<n'\}$   
**unfolding**  $\text{inj-on-def } g\text{-def } n'\text{-def}$  **by**  $(\text{metis lessThan-iff nat-neq-iff pick-mono-le})$   
  
**have**  $\text{card } \{i. i < \text{dim-row } A \wedge i \in I\} = \text{card } I$   
**using**  $A \ I(1)$   
**by**  $(\text{smt } (\text{verit}, \text{del-insts}) \text{basic-trans-rules}(31) \text{carrier-matD}(1) \text{equalityI lessThan-iff}$   
 $\text{mem-Collect-eq subsetI})$   
**moreover have**  $\text{card } \{j. j < \text{dim-col } A \wedge j \in J\} = \text{card } J$   
**using**  $A \ J(1)$   
**by**  $(\text{smt } (\text{verit}, \text{del-insts}) \text{basic-trans-rules}(31) \text{carrier-matD}(2) \text{equalityI lessThan-iff}$   
 $\text{mem-Collect-eq subsetI})$   
**ultimately show**  $B = \text{submatrix-of-inj } A \ f \ g \ m' \ n'$   
**unfolding**  $\text{submatrix-of-inj-def } B\text{-def } \text{submatrix-def } f\text{-def } g\text{-def } m'\text{-def } n'\text{-def}$  **by**  
 $\text{argo}$   
**qed**

**lemma**  $\text{submatrix-inj-square}$ :  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**assumes**  $I: I \subseteq \{..<n\}$   
**defines**  $m \equiv \text{card } I$   
**defines**  $B \equiv \text{submatrix } A \ I \ I$   
**defines**  $f \equiv \lambda i. \text{pick } I \ i$   
**shows**  $f : \{..<m\} \rightarrow \{..<n\}$   $\text{inj-on } f \ \{..<m\}$   $B = \text{submatrix-of-inj } A \ f \ f \ m \ m$   
**by**  $(\text{rule } \text{submatrix-inj}[\text{OF } A \ I \ I, \text{folded } m\text{-def } B\text{-def } f\text{-def}])+$

**lemma**  $\text{submatrix-inj-obt}$ :  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**assumes**  $I: I \subseteq \{..<m\}$   
**assumes**  $J: J \subseteq \{..<n\}$   
**defines**  $m' \equiv \text{card } I$   
**defines**  $n' \equiv \text{card } J$   
**defines**  $B \equiv \text{submatrix } A \ I \ J$   
**obtains**  $f \ g$  **where**  $f : \{..<m'\} \rightarrow \{..<m\}$   $g : \{..<n'\} \rightarrow \{..<n\}$   
 $\text{inj-on } f \ \{..<m'\} \ \text{inj-on } g \ \{..<n'\}$   
 $B = \text{submatrix-of-inj } A \ f \ g \ m' \ n'$   
**using**  $\text{submatrix-inj}[\text{OF } A \ I \ J] \ m'\text{-def } n'\text{-def } B\text{-def}$  **by**  $\text{blast}$

```

lemma submatrix-inj-square-obt:
  assumes  $A: A \in \text{carrier-mat } n \ n$ 
  assumes  $I: I \subseteq \{..<n\}$ 
  defines  $m \equiv \text{card } I$ 
  defines  $B \equiv \text{submatrix } A \ I \ I$ 
  obtains  $f$  where  $f: \{..<m\} \rightarrow \{..<n\}$  inj-on  $f \ \{..<m\} \ B = \text{submatrix-of-inj } A$ 
   $f \ f \ m \ m$ 
  using submatrix-inj-square[OF  $A \ I$ ] m-def  $B$ -def by blast

```

```

lemma submatrix-as-compression:
  fixes  $A :: \text{complex mat}$ 
  fixes  $f :: \text{nat} \Rightarrow \text{nat}$ 
  assumes  $A: A \in \text{carrier-mat } n \ n$ 
  assumes  $I: I \subseteq \{..<n\}$ 
  defines  $m \equiv \text{card } I$ 
  defines  $B \equiv \text{submatrix } A \ I \ I$ 
  defines  $(Q :: \text{complex mat}) \equiv \text{mat } n \ m \ (\lambda(i,j). \text{if } i = \text{pick } I \ j \ \text{then } 1 \ \text{else } 0)$ 
  shows  $B = Q^H * A * Q$  isometry-mat  $Q \ n \ m \ Q \in \text{carrier-mat } n \ m$ 
  using submatrix-inj-square[OF  $A \ I$ ] submatrix-of-inj-as-compression[OF  $A$ ]
  unfolding m-def  $B$ -def  $Q$ -def
  by presburger+

```

```

lemma submatrix-as-compression-obt:
  fixes  $A :: \text{complex mat}$ 
  assumes  $A \in \text{carrier-mat } n \ n$ 
  assumes  $I \subseteq \{..<n\}$ 
  defines  $m \equiv \text{card } I$ 
  defines  $B \equiv \text{submatrix } A \ I \ I$ 
  obtains  $Q$  where  $B = Q^H * A * Q$  isometry-mat  $Q \ n \ m$ 
  using submatrix-as-compression[OF assms(1,2)] m-def  $B$ -def by blast

```

## 20 Schur Decomposition

```

context
  fixes  $A \ \Lambda \ U :: 'a :: \{\text{conjugatable-field, real-algebra-1, ord}\} \ \text{mat}$ 
  fixes  $n$ 
  assumes  $*$ : real-diag-decomp  $A \ \Lambda \ U$ 
  defines  $n \equiv \text{dim-row } A$ 
begin

```

```

lemma diag-decomp-inverse: inverts-mat  $U$  (adjoint  $U$ )
  using  $*$ 
  unfolding real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def
  using Complex-Matrix.unitary-def by blast

```

```

lemma diag-decomp-invertible: invertible-mat  $U$ 
  using diag-decomp-inverse  $*$ 
  by (meson Complex-Matrix.unitary-def carrier-matD(2) invertible-mat-def real-diag-decompD(1))

```

*square-mat.simps unitaryD2*  
*unitary-diagD(3)*)

**lemma** *diag-decomp-cols-distinct: distinct (cols U)*  
**using** *diag-decomp-invertible \**  
**by** (*metis carrier-mat-triv invertible-det invertible-mat-def order-less-irrefl square-mat.simps*  
*vec-space.low-rank-det-zero*  
*vec-space.non-distinct-low-rank*)

**lemma** *diag-decomp-ortho: corthogonal-mat U*  
**using** *unitary-is-corthogonal \**  
**unfolding** *real-diag-decomp-def unitary-diag-def unitarily-equiv-def unitary-def*  
**by** *blast*

**private lemma** *carriers: A ∈ carrier-mat n n U ∈ carrier-mat n n U<sup>H</sup> ∈ carrier-mat n n Λ ∈ carrier-mat n n*  
**using** *\*[unfolded real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def,*  
*folded adjoint-is-conjugate-transpose]*  
**using** *insert-subset n-def*  
**by** *metis+*

**lemma** *diag-decomp-cols-lin-indpt: module.lin-indpt class-ring (module-vec TYPE('a) n) (set (cols U))*  
**using** *\* carriers unfolding real-diag-decomp-def unitary-diag-def unitary-def n-def*  
**by** (*metis diag-decomp-invertible idom-vec.lin-dep-cols-imp-det-0 invertible-det*)

**lemma** *real-diag-decomp-eq: A = U \* Λ \* U<sup>H</sup>*  
**using** *\*[unfolded real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def]*  
**by** (*metis adjoint-is-conjugate-transpose*)

**private lemma** *U-unitary: U<sup>H</sup> \* U = 1<sub>m n</sub> ∧ U \* U<sup>H</sup> = 1<sub>m n</sub>*  
**using** *\*[unfolded real-diag-decomp-def unitary-diag-def unitarily-equiv-def unitary-def]*  
**using** *n-def carriers*  
**by** (*metis adjoint-is-conjugate-transpose carrier-matD(2) inverts-mat-def similar-mat-wit-def*)

**private lemma** *Λ-diag: diagonal-mat Λ*  
**using** *\* real-diag-decompD(1) unitary-diagD(2) by blast*

**lemma** *diag-decomp-cols-are-eigenvectors:*  
**assumes** *i < n*  
**shows** *eigenvector A (col U i) ((diag-mat Λ)!i) (is eigenvector A ?v ?μ)*  
**proof** –  
**have** *A \*<sub>v</sub> ?v = (U \* Λ) \*<sub>v</sub> (U<sup>H</sup> \*<sub>v</sub> ?v)*  
**using** *\* real-diag-decomp-eq carriers assoc-mult-mat-vec[of U \* Λ n n U<sup>H</sup> n*

?v] **by force**  
**also have** ... =  $(U * \Lambda) *_v (\text{col } (1_m \ n) \ i)$   
**using** *U-unitary n-def assms carriers*  
**by** (*metis (mono-tags, lifting) adjoint-is-conjugate-transpose col-mult2*)  
**also have** ... =  $U *_v (\Lambda *_v (\text{unit-vec } n \ i))$   
**using** \* *real-diag-decomp-eq carriers col-one assms* **by fastforce**  
**also have** ... =  $U *_v ((\text{diag-mat } \Lambda)!i \cdot_v (\text{unit-vec } n \ i))$   
**using** \*[*unfolded real-diag-decomp-def unitary-diag-def*]  
**using** *diagonal-unit-vec'[OF carriers(4) \Lambda-diag, of i] assms carriers(4)*  
**unfolding** *diag-mat-def*  
**by force**  
**also have** ... =  $(\text{diag-mat } \Lambda)!i \cdot_v (U *_v (\text{unit-vec } n \ i))$   
**using** *carriers(2)* **by auto**  
**also have** ... =  $(\text{diag-mat } \Lambda)!i \cdot_v (\text{col } U \ i)$   
**by** (*simp add: assms carriers(2) mat-unit-vec-col*)  
**finally show** ?thesis  
**unfolding** *eigenvector-def*  
**using** *assms carriers(2) n-def*  
**by** (*metis carrier-matD(1,2) col-dim conjugate-zero-vec corthogonal-matD diag-decomp-ortho scalar-prod-left-zero*)  
**qed**  
**end**

## 21 Rayleigh Quotient

**definition** *rayleigh-quotient-complex* :: *complex mat*  $\Rightarrow$  *complex vec*  $\Rightarrow$  *complex*  
 $(\varrho_c)$  **where**  
 $\varrho_c \ M \ x = (QF \ M \ x) / (x \cdot_c \ x)$

**definition** *rayleigh-quotient* :: *complex mat*  $\Rightarrow$  *complex vec*  $\Rightarrow$  *real*  $(\varrho)$  **where**  
 $\varrho \ M \ x = \text{Re } (\varrho_c \ M \ x)$

**lemma** *rayleigh-quotient-negative*:  $A \in \text{carrier-mat } n \ n \Longrightarrow x \in \text{carrier-vec } n \Longrightarrow$   
 $\varrho \ A \ x = - \varrho \ (- \ A) \ x$   
**by** (*simp add: rayleigh-quotient-def rayleigh-quotient-complex-def*)

**lemma** *rayleigh-quotient-complex-scale*:

**fixes**  $k :: \text{real}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $v \in \text{carrier-vec } n$   
**assumes**  $k \neq 0$   
**shows**  $\varrho_c \ A \ v = \varrho_c \ A \ (k \cdot_v \ v)$

**proof**–

**have**  $\varrho_c \ A \ v = (k^{\wedge}2 * ((A *_v \ v) \cdot_c \ v)) / (k^{\wedge}2 * (v \cdot_c \ v))$   
**using** *assms(3)* **by** (*simp add: rayleigh-quotient-complex-def*)  
**also have** ... =  $((k \cdot_v (A *_v \ v)) \cdot_c (k \cdot_v \ v)) / (((k \cdot_v \ v) \cdot_c (k \cdot_v \ v)))$   
**using** *assms(1,2) inner-prod-smult-right* **by force**  
**also have** ... =  $((A *_v (k \cdot_v \ v)) \cdot_c (k \cdot_v \ v)) / (((k \cdot_v \ v) \cdot_c (k \cdot_v \ v)))$

**using** *assms* **by** (*metis mult-mat-vec*)  
**also have**  $\dots = \varrho_c A (k \cdot_v v)$  **by** (*simp add: rayleigh-quotient-complex-def*)  
**finally show** *?thesis* .  
**qed**

**lemma** *rayleigh-quotient-scale-complex*:

**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $v \in \text{carrier-vec } n$   
**assumes**  $k \neq 0$   
**shows**  $\varrho A (k \cdot_v v) = \varrho A v$   
**proof** –  
**let**  $?v' = (k \cdot_v v)$   
**have**  $(A *_v ?v') \cdot_c ?v' = (cmod\ k)^{\wedge} 2 * (QF\ A\ v)$   
**using** *assms(1,2)*  
**by** (*smt (verit, ccfv-SIG) cross3-simps(11) inner-prod-smult-left-right mult-conj-cmod-square mult-mat-vec*  
*mult-mat-vec-carrier quadratic-form-def*)  
**moreover have**  $?v' \cdot_c ?v' = (cmod\ k)^{\wedge} 2 * (v \cdot_c v)$   
**using** *assms(2) complex-norm-square inner-prod-smult-left-right* **by force**  
**ultimately show**  $\varrho A ?v' = \varrho A v$   
**by** (*simp add: rayleigh-quotient-def rayleigh-quotient-complex-def assms(3)*)  
**qed**

**lemma** *rayleigh-quotient-scale-real*:

**fixes**  $k :: \text{real}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**assumes**  $v \in \text{carrier-vec } n$   
**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*)

**context** *cmplx-herm-mat*

**begin**

**lemma** *hermitian-rayleigh-quotient-real*:

**assumes**  $v \in \text{carrier-vec } n$   
**assumes**  $v \neq 0_v \ n$   
**shows**  $\varrho_c A v \in \text{Reals}$   
**proof** –  
**have**  $QF\ A\ v \in \text{Reals}$   
**using** *hermitian-quadratic-form-real assms* **by blast**  
**moreover have**  $\text{inner-prod } v \ v \in \text{Reals}$  **by** (*simp add: self-inner-prod-real*)  
**moreover have**  $\text{inner-prod } v \ v \neq 0$  **using** *assms* **by fastforce**  
**ultimately show** *?thesis* **unfolding** *rayleigh-quotient-complex-def* **using** *Reals-divide* **by blast**  
**qed**

```

lemma rayleigh-quotient-QF:
  assumes  $x: x \neq 0_v \ n \ x \in \text{carrier-vec } n$ 
  shows  $\varrho A x = (QF A x) / (\text{vec-norm } x)^2$ 
  using hermitian-rayleigh-quotient-real[OF x(2,1)]
  unfolding rayleigh-quotient-def rayleigh-quotient-complex-def vec-norm-def
  using of-real-Re power2-csqrt
  by presburger

end

lemma rayleigh-eigenvector:
  assumes eigenvector A v e
  shows  $\varrho A v = e$ 
  unfolding rayleigh-quotient-def rayleigh-quotient-complex-def quadratic-form-def
  using assms[unfolded eigenvector-def]
  by force

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

begin

```

## 22 Courant-Fischer Minimax Theorem

We took inspiration from 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>. In particular, the approach presented there explicitly obtains the eigenvector basis via diagonal decomposition, which is conducive to the theorems we have available.

```

definition sup-defined :: 'a::preorder set  $\Rightarrow$  bool where
  sup-defined  $S \longleftrightarrow S \neq \{\} \wedge \text{bdd-above } S$ 

```

```

definition inf-defined :: 'a::preorder set  $\Rightarrow$  bool where
  inf-defined  $S \longleftrightarrow S \neq \{\} \wedge \text{bdd-below } S$ 

```

```

locale cf-setup = cmplx-herm-mat n for  $n$ 
begin

```

Although the *cf-setup* locale adds no more assumptions beyond those from the *cmplx-herm-mat* locale, we separate the Courant Fischer proof into a separate locale so that the terms involved in the proof (such as  $\Lambda$  and  $U$ ) don't pollute the namespace.

```

definition dimensional :: complex vec set  $\Rightarrow$  nat  $\Rightarrow$  bool where
  dimensional  $\mathcal{V} k \longleftrightarrow \text{subspace class-ring } \mathcal{V} V \wedge \text{vector-space.dim class-ring } (vs \mathcal{V}) = k$ 

```

**lemma** *dimensional-n*:  $\text{dimensional } \mathcal{V} k \implies \mathcal{V} \subseteq \text{carrier-vec } n$   
**unfolding** *dimensional-def subspace-def submodule-def* **by** *auto*

**lemma** *dimensional-n-vec*:  $\bigwedge v. v \in \mathcal{V} \implies \text{dimensional } \mathcal{V} k \implies v \in \text{carrier-vec } n$   
**using** *dimensional-n* **by** *fast*

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

**definition** *rayleigh-min* **where**  
*rayleigh-min*  $\mathcal{V} = \text{Inf } \{\rho A v \mid v. v \neq 0_v n \wedge v \in \mathcal{V}\}$

**definition** *rayleigh-max* **where**  
*rayleigh-max*  $\mathcal{V} = \text{Sup } \{\rho A v \mid v. v \neq 0_v n \wedge v \in \mathcal{V}\}$

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

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

**definition** *maximin-defined* **where**  
*maximin-defined*  $k \longleftrightarrow \text{sup-defined } \{\text{rayleigh-min } \mathcal{V} \mid \mathcal{V}. \text{dimensional } \mathcal{V} k\}$

**definition** *minimax-defined* **where**  
*minimax-defined*  $k \longleftrightarrow \text{inf-defined } \{\text{rayleigh-max } \mathcal{V} \mid \mathcal{V}. \text{dimensional } \mathcal{V} (n - k + 1)\}$

**abbreviation** *es* :: *complex list* **where** *es*  $\equiv$  *eigenvalues*

**definition**  $\Lambda$ -*U* :: *complex mat*  $\times$  *complex mat* **where**  
 $\Lambda$ -*U*  $\equiv$  *SOME* ( $\Lambda, U$ ). *real-diag-decomp*  $A \Lambda U \wedge \text{es} = \text{diag-mat } \Lambda$

**definition**  $\Lambda$  :: *complex mat* **where**  
 $\Lambda \equiv \text{fst } \Lambda$ -*U*

**definition** *U* :: *complex mat* **where**  
 $U \equiv \text{snd } \Lambda$ -*U*

**lemma** *A-decomp*: *real-diag-decomp*  $A \Lambda U$  **and** *es*: *es* = *diag-mat*  $\Lambda$

**proof** –

**obtain**  $\Lambda' U'$  **where** *decomp*: *real-diag-decomp*  $A \Lambda' U'$  *diag-mat*  $\Lambda' = \text{es}$   
**by** (*meson* *A-dim* *A-herm* *eigenvalues* *hermitian-real-diag-decomp-eigvals*)

**thus** *real-diag-decomp*  $A \Lambda U \text{es} = \text{diag-mat } \Lambda$

**unfolding**  $\Lambda$ -*def* *U-def*  $\Lambda$ -*U-def*

**by** (*metis* (*mono-tags*, *lifting*) *someI-ex* *case-prod-unfold* *old.prod.case*)+

**qed**

**lemma** *U-carrier*:  $U \in \text{carrier-mat } n n$  **and**  $\Lambda$ -*carrier*:  $\Lambda \in \text{carrier-mat } n n$   
**using** *A-decomp* *A-dim*

**unfolding** *real-diag-decomp-def unitary-diag-def unitarily-equiv-def similar-mat-wit-def*  
**by** *auto+*

**lemma** *UH-carrier*:  $U^H \in \text{carrier-mat } n \ n$  **by** (*simp add: U-carrier*)

**lemma** *UH-unitary*: *unitary*  $U^H$   
**by** (*metis A-decomp U-carrier adjoint-is-conjugate-transpose real-diag-decompD(1)*  
*unitary-adjoint unitary-diagD(3)*)

**lemma** *len-U-cols*:  $\text{length } (\text{cols } U) = n$  **using** *U-carrier* **by** *simp*

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

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

**lemma** *U-cols-orthogonal-span*:  
**assumes**  $I: I \subseteq \{0..<n\}$   
**assumes**  $v: v \in \text{span } \{\text{col } U \ i \mid i. i \in I\}$   
**assumes**  $j: j \in \{0..<n\} - I$   
**shows**  $(U^H *_{v} v)\$j = 0$

**proof** –

**let**  $?B = \{\text{col } U \ i \mid i. i \in I\}$

**have** *v-carrier*:  $v \in \text{carrier-vec } n$

**proof** –

**have**  $\text{col } U \ i \in \text{carrier-vec } n$  **if**  $i < n$  **for**  $i$

**using** *U-carrier col-carrier-vec that by blast*

**thus** *?thesis*

**using**  $v \ I$  **by** (*smt (verit, best) atLeastLessThan-iff mem-Collect-eq span-closed*  
*subset-iff*)

**qed**

**have** *B-carrier*:  $?B \subseteq \text{carrier-vec } n$  **using** *U-carrier col-dim* **by** *blast*

**obtain**  $cs \ B$  **where**  $cs: \text{finite } B \ B \subseteq ?B \ v = \text{lincomb } cs \ B$

**using**  $v[\text{unfolded span-def}]$  **by** *blast*

**have**  $(U^H *_{v} v)\$j = (\text{conjugate } (\text{col } U \ j)) \cdot v$  **using**  $j \ \text{len-U-cols}$  **by** *auto*

**also have**  $\dots = (\sum b \in B. cs \ b * ((\text{conjugate } (\text{col } U \ j)) \cdot b))$

**apply** (*rule sprod-vec-sum[of v n conjugate (col U j) B cs]*)

**apply** (*rule v-carrier*)

**using** *UH-carrier*

**apply** *fastforce*

**using** *B-carrier cs(2)*

**apply** *order*

**apply** (*simp add: cs(1)*)

**by** (*simp add: cs(3) lincomb-def*)

**also have**  $\dots = 0$

**proof** –

**have**  $b \in B \implies b \cdot c ((\text{col } U \ j)) = 0$  **for**  $b$

**using**  $I \ j \ cs(2) \ \text{len-U-cols} \ A\text{-decomp} \ \text{diag-decomp-ortho}$

```

    using corthogonal-matD[OF diag-decomp-ortho[OF A-decomp]]
    by (smt (verit, best) Diff-iff atLeastLessThan-iff cols-length mem-Collect-eq
subsetD)
    hence  $b \in B \implies (\text{conjugate } (\text{col } U j)) \cdot b = 0$  for  $b$ 
    using U-carrier B-carrier cs(2)
    by (metis (lifting) carrier-matD(1) col-dim conjugate-vec-sprod-comm subsetD)
    thus ?thesis by force
qed
finally show  $((U^H *_v v)\$j) = 0$  by force
qed

```

lemma *U-cols-basis*:

```

assumes I:  $I \subseteq \{0..<n\}$ 
defines [simp]:  $\mathcal{B} \equiv \{\text{col } U i \mid i. i \in I\}$ 
shows  $\mathcal{B} \subseteq \text{carrier-vec } n$ 
    lin-indpt  $\mathcal{B}$ 
    card  $\mathcal{B} = \text{card } I$ 
    vectorspace.dim class-ring (vs (span  $\mathcal{B}$ )) = card I
proof -
  show carrier:  $\mathcal{B} \subseteq \text{carrier-vec } n$  using U-carrier B-def col-dim by blast

  have lin-indpt (set (cols U)) using A-dim diag-decomp-cols-lin-indpt[OF A-decomp]
  by blast
  moreover have  $\mathcal{B} \subseteq \text{set } (\text{cols } U)$ 
    unfolding B-def
    using I U-carrier len-U-cols
    by (smt (verit, best) cols-def cols-length image-iff list.set-map mem-Collect-eq
set-upt subset-iff)
  ultimately show li: lin-indpt  $\mathcal{B}$  using supset-ld-is-ld by blast

  have  $i \in I \implies j \in I \implies i \neq j \implies \text{col } U i \neq \text{col } U j$  for  $i j$ 
    using I A-decomp U-carrier
    by (metis det-identical-cols diag-decomp-invertible invertible-det lessThan-atLeast0
lessThan-iff subset-eq)
  hence inj-on (col U) I using I inj-on-subset unfolding inj-on-def by blast
  moreover have  $\mathcal{B} = (\text{col } U)^{\smallfrown} I$  unfolding B-def by blast
  ultimately show card  $\mathcal{B} = \text{card } I$  using card-image[of col U I] by argo
  thus vectorspace.dim class-ring (vs (span  $\mathcal{B}$ )) = card I
    apply (subst dim-of-lin-indpt-span[OF li carrier])
    using I by (simp add: finite-subset)
qed

```

end

```

context cmplx-herm-mat
begin

```

```

interpretation A?: cf-setup A n by unfold-locales

```

In the local context, we interpret the *cf-setup* locale, giving us access

to the abbreviation  $es$  as well as other constants like  $U$  and  $\Lambda$  which form the decomposition of  $A$ . Once the local context is closed, this interpretation is no longer accessible, so these names do not pollute the *cmplx-herm-mat* namespace.

**lemma** *len-eigenvalues*:  $length\ es = n$

**by** (*metis*  $\Lambda$ -carrier  $es$  carrier-matD(1) diag-mat-def length-map length-upt verit-minus-simplify(2))

**lemma** *unit-vec-rayleigh-formula*:

**assumes** *unit-v*:  $vec\text{-norm}\ v = 1$

**assumes** *v-dim*:  $v \in carrier\text{-vec}\ n$

**shows**  $\varrho\ A\ v = (\sum j \in \{..<n\}. es!j * (cmod\ ((U^H *_{*v}\ v)\$j))^2)$

**proof**–

**have**  $U\text{-}\Lambda$ : *unitary*  $U \wedge$  *unitary*  $U^H\ A = U * \Lambda * U^H\ U^H \in carrier\text{-mat}\ n\ n$

**apply** (*metis*  $U$ -carrier  $A$ -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*:  $U$ -carrier)

**have** *diagonal-mat*  $\Lambda$  **using**  $A$ -decomp **unfolding** *real-diag-decomp-def* **by** *fast-force*

**hence**  $\Lambda$ -diag-mult:  $\bigwedge x\ i.\ x \in carrier\text{-vec}\ n \implies i \in \{..<n\} \implies (\Lambda *_{*v}\ x)\$i = (\Lambda)\$i * x\$i$

**using**  $\Lambda$ -carrier *diagonal-mat-mult-vec* **by** *blast*

**have**  $\Lambda$ -diag-eigenvals:  $\bigwedge i.\ i \in \{..<n\} \implies \Lambda)\$i = es!i$

**using**  $A$ -decomp  $\Lambda$ -carrier **by** (*simp* *add*: *diag-mat-def*  $es$ )

**define**  $x$  **where**  $x \equiv U^H *_{*v}\ v$

**hence**  $x$ -dim:  $x \in carrier\text{-vec}\ n$  **using**  $U$ - $\Lambda$ (3) *mult-mat-vec-carrier*  $v$ -dim **by** *blast*

**hence**  $x$ -conj-dim: *conjugate*  $x \in carrier\text{-vec}\ n$  **by** *simp*

**have**  $x$ -norm:  $vec\text{-norm}\ x = 1$  **using**  $U$ - $\Lambda$  *unit-v* *unitary-vec-norm*  $v$ -dim  $x$ -def **by** *presburger*

**have**  $*$ :  $\bigwedge i.\ i \in \{..<n\} \implies (conjugate\ x)\$i = conjugate\ (x\$i)$

**unfolding** *conjugate-vec-def* **using**  $x$ -dim **by** *auto*

**have**  $v \cdot c\ v = 1$  **using** *unit-v* *csqrt-eq-1* **unfolding** *vec-norm-def* **by** *blast*

**hence**  $QF\ A\ v / Complex\text{-Matrix}.inner\text{-prod}\ v\ v = QF\ A\ v$  **by** *simp*

**hence**  $\varrho\ A\ v = complex\text{-of-real}\ (Re\ (QF\ A\ v))$

**unfolding** *rayleigh-quotient-def* *rayleigh-quotient-complex-def* **by** *simp*

**also** **have**  $\dots = QF\ A\ v$

**using** *hermitian-quadratic-form-real*[ $OF\ v$ -dim] **by** *simp*

**also** **have**  $\dots = inner\text{-prod}\ v\ ((U * \Lambda * U^H) *_{*v}\ v)$

**unfolding** *quadratic-form-def* **using**  $U$ - $\Lambda$  **by** *argo*

**also** **have**  $\dots = inner\text{-prod}\ v\ (U *_{*v}\ ((\Lambda * U^H) *_{*v}\ v))$

**by** (*smt* (*verit*, *best*)  $\Lambda$ -carrier  $U$ -carrier *More-Matrix.carrier-vec-conjugate* *assoc-mat-mult-vec'* *carrier-dim-vec* *mat-vec-mult-assoc* *transpose-carrier-mat*  $v$ -dim)

**also** **have**  $\dots = inner\text{-prod}\ (U^H *_{*v}\ v)\ ((\Lambda * U^H) *_{*v}\ v)$

**by** (*metis*  $\Lambda$ -carrier  $U$ -carrier *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**  $\dots = (\Lambda *_v x) \cdot c x$   
**by** (*metis*  $\Lambda$ -carrier U-carrier More-Matrix.carrier-vec-conjugate carrier-vecD mat-vec-mult-assoc transpose-carrier-mat v-dim x-def)  
**also have**  $\dots = (\sum j \in \{..<n\}. (es!j * x\$j) * conjugate (x\$j))$   
**using**  $\Lambda$ -diag-mult x-dim  $\Lambda$ -diag-eigenvals atLeast0LessThan x-dim **by** (*simp add: inner-prod-def scalar-prod-def*)  
**also have**  $\dots = (\sum j \in \{..<n\}. es!j * (cmod (x\$j))^2)$   
**by** (*smt (verit) cring-simprules(11) mult-conj-cmod-square of-real-mult of-real-sum sum.cong*)  
**finally show**  $\varrho A v = (\sum j \in \{..<n\}. es!j * (cmod (x\$j))^2)$   
**using** *of-real-eq-iff* **by** *blast*  
**qed**

**lemma** *rayleigh-bdd-below'*:  $\forall v \in \text{carrier-vec } n. v \neq 0_v n \longrightarrow \varrho A v \geq \text{Min } (Re \text{ ' set } es)$

**proof** –

**define**  $m$  **where**  $m \equiv \text{Min } (Re \text{ ' set } es)$   
**have**  $\varrho A v \geq m$  **if**  $*$ :  $v \in \text{carrier-vec } n v \neq 0_v n$  **for**  $v$   
**proof** –  
**define**  $v'$  **where**  $v' \equiv \text{vec-normalize } v$   
**have**  $v'$ : *vec-norm*  $v' = 1 \wedge v' \in \text{carrier-vec } n$   
**using** *normalized-vec-norm*[of  $v n$ ] **unfolding** *vec-norm-def v'-def*  
**using**  $*$  *csqrt-1 normalized-vec-dim* **by** *presburger*  
**have**  $\varrho A v = \varrho A v'$   
**using** *A-dim \*(1) v' v'-def*  
**by** (*metis normalize-zero rayleigh-quotient-scale-complex vec-eq-norm-smult-normalized vec-norm-zero*)  
**also have**  $\dots = (\sum i \in \{..<n\}. es!i * (cmod ((U^H *_v v')\$i))^2)$   
**using** *unit-vec-rayleigh-formula \* v'* **by** *blast*  
**also have**  $\dots \geq m$   
**proof** –  
**have** *vec-norm*  $(U^H *_v v') = 1$   
**by** (*metis v' U-carrier 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*  $(U^H *_v v') = \text{csqrt } (\sum i \in \{..<n\}. (cmod ((U^H *_v v')\$i))^2)$   
**by** (*metis complex-vec-norm-sum U-carrier carrier-dim-vec carrier-matD(2) dim-mult-mat-vec dim-row-conjugate index-transpose-mat(2)*)  
**ultimately have** *norm*:  $(\sum i \in \{..<n\}. (cmod ((U^H *_v v')\$i))^2) = 1$   
**by** (*metis Re-complex-of-real one-complex.sel(1) one-power2 power2-csqrt*)  
  
**have** *finite*  $(Re \text{ ' set } es)$  **by** *simp*  
**hence**  $\forall x \in Re \text{ ' set } es. m \leq x$  **using** *Min-le m-def* **by** *blast*  
**moreover have**  $\forall i \in \{..<n\}. \exists x \in Re \text{ ' set } es. x = es!i$   
**using** *es eigenvalues-real len-eigenvalues* **by** (*simp add: subsetD*)  
**ultimately have**  $\bigwedge i. i \in \{..<n\} \implies m \leq es!i$

**by** (*metis Im-complex-of-real Re-complex-of-real less-eq-complex-def*)  
**hence**  $\text{ineq: } \bigwedge i. i \in \{..<n\} \implies m * (\text{cmod } ((U^H *_v v')\$i))^{\wedge 2} \leq \text{es!}i * (\text{cmod } ((U^H *_v v')\$i))^{\wedge 2}$   
**by** (*metis conjugate-square-positive mult-conj-cmod-square mult-right-mono of-real-hom.hom-mult*)  
**have**  $m \leq m * (\sum i \in \{..<n\}. (\text{cmod } ((U^H *_v v')\$i))^{\wedge 2})$  **using** *norm* **by** *auto*  
**also have**  $\dots = (\sum i \in \{..<n\}. m * (\text{cmod } ((U^H *_v v')\$i))^{\wedge 2})$   
**by** (*simp add: mult-hom.hom-sum*)  
**also have**  $\dots \leq (\sum i \in \{..<n\}. \text{es!}i * (\text{cmod } ((U^H *_v v')\$i))^{\wedge 2})$   
**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* **unfolding** *m-def* **by** *blast*  
**qed**

**lemma** *rayleigh-bdd-below*:  
**assumes** *dimensional*  $\mathcal{V} k$   
**shows**  $\exists m. \forall v \in \mathcal{V}. v \neq 0_v n \longrightarrow \varrho A v \geq m$   
**by** (*meson assms dimensional-n rayleigh-bdd-below' subsetD*)

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

**lemma** *courant-fischer-unit-rayleigh-helper1*:  
**assumes** *dimensional*  $\mathcal{V} (k + 1)$   
**shows**  $\exists v. \text{vec-norm } v = 1 \wedge v \in \mathcal{V} \wedge v \neq 0_v n \wedge \varrho A v \leq \text{es!}k$   
**proof**–

**define**  $\mathcal{B}'$  **where**  $\mathcal{B}' = \{\text{col } U i \mid i. i \in \{k..<n\}\}$   
**have**  $\mathcal{V}$ : *subspace class-ring*  $\mathcal{V} V$   
**using** *assms dimensional-def* **by** *blast*  
**have**  $k: k < n$   
**using** *assms* **unfolding** *dimensional-def*  
**using** *subspace-dim[of \mathcal{V}] subspace-fin-dim[of \mathcal{V}] fin-dim dim*  
**by** *linarith*  
**have**  $\mathcal{B}'\text{-dim: vectorspace.dim class-ring } (vs (\text{span } \mathcal{B}')) = n - k$  **and** *carrier-\mathcal{B}'*:  
 $\mathcal{B}' \subseteq \text{carrier-vec } n$   
**using** *U-cols-basis[of \{k..<n\}, folded \mathcal{B}'-def]* **by** *auto*

**obtain**  $v$  **where**  $v: v \in \mathcal{V} \cap \text{span } \mathcal{B}'$  **and**  $\text{nz: } v \neq 0_v n$   
**using** *dim-sum-nontriv-int[OF \mathcal{V} span-is-subspace[OF carrier-\mathcal{B}']] - fin-dim*,

```

unfolded  $\mathcal{B}'$ -dim]
  using assms[unfolded dimensional-def] dim k
  by auto
define v' where v'  $\equiv$  vec-normalize v
have v': vec-norm v' = 1 v'  $\in$   $\mathcal{V} \cap \text{span } \mathcal{B}'$  v'  $\in$  carrier-vec n
proof -
  have subspace class-ring (span  $\mathcal{B}'$ ) V using carrier- $\mathcal{B}'$  span-is-subspace by
presburger
  thus v'  $\in$   $\mathcal{V} \cap \text{span } \mathcal{B}'$ 
  using v carrier- $\mathcal{B}'$  assms  $\mathcal{V}$ 
  unfolding v'-def vec-normalize-def dimensional-def
  using assms dimensional-n module-incl-imp-submodule submoduleE(4) sub-
module-is-module
  by auto
  show vec-norm v' = 1 v'  $\in$  carrier-vec n
  using v nz assms unfolding v'-def
  apply (metis IntE basic-trans-rules(31) csqrt-1 dimensional-n normal-
ized-vec-norm vec-norm-def)
  using v nz assms unfolding v'-def
  by (meson IntE basic-trans-rules(31) dimensional-n normalized-vec-dim)
qed

have *: i < k  $\implies$  cmod ((UH *v v')$i) = 0 for i
  using U-cols-orthogonal-span[of {k.. $n$ } v' i] k v'(2)[unfolded  $\mathcal{B}'$ -def] by force
have  $\varrho$  A v' = ( $\sum_{i=0..<n}$  es!i * complex-of-real ((cmod ((UH *v v')$i))2))
  by (rule unit-vec-rayleigh-formula[OF v'(1,3), folded atLeast0LessThan])
also have ... = ( $\sum_{i=0..<k}$  es!i * complex-of-real ((cmod ((UH *v v')$i))2))
  + ( $\sum_{i=k..<n}$  es!i * complex-of-real ((cmod ((UH *v v')$i))2))
  using k by (metis (lifting) le-eq-less-or-eq sum.atLeastLessThan-concat zero-le)
also have ... = ( $\sum_{i=k..<n}$  es!i * complex-of-real ((cmod ((UH *v v')$i))2))
  using * by fastforce
also have ...  $\leq$  ( $\sum_{i=k..<n}$  es!k * complex-of-real ((cmod ((UH *v v')$i))2))
proof -
  have  $\forall i < n. k \leq i \implies \text{es!}i \leq \text{es!}k$ 
  using eigenvalues-sorted by (simp add: le-eq-less-or-eq len-eigenvalues sorted-wrt-iff-nth-less)
  hence  $\forall i \in \{k..<n\}. \text{es!}i * \text{complex-of-real} ((\text{cmod} ((U^H *_{v'} v')\$i))^2)$ 
   $\leq \text{es!}k * \text{complex-of-real} ((\text{cmod} ((U^H *_{v'} v')\$i))^2)$ 
  apply (clarify, simp add: atLeastLessThan-def lessThan-def atLeast-def)
  by (metis conjugate-square-positive mult-conj-cmod-square mult-right-mono
of-real-power)
  thus ?thesis by (meson sum-mono)
qed
also have ... = es!k * ( $\sum_{i=k..<n}$  ((cmod ((UH *v v')$i))2))
  by (simp add: mult-hom.hom-sum)
also have ... = es!k * (vec-norm (UH *v v'))2
proof -
  have 1: UH *v v'  $\in$  carrier-vec n using UH-carrier v'(3) by force
  have (vec-norm (UH *v v'))2 = ( $\sum_{i<n}$  ((cmod ((UH *v v')$i))2))
  unfolding complex-vec-norm-sum[OF 1] by force

```

**also have**  $\dots = (\sum_{i=0..<k}. ((\text{cmod } ((U^H *_v v')\$i)^2))) + (\sum_{i=k..<n}. ((\text{cmod } ((U^H *_v v')\$i)^2)))$   
**using**  $k$  **by** (*metis*  $k$  *atLeast0LessThan* *le-eq-less-or-eq* *sum.atLeastLessThan-concat* *zero-order(1)*)  
**also have**  $\dots = (\sum_{i=k..<n}. ((\text{cmod } ((U^H *_v v')\$i)^2)))$  **using**  $*$  **by** *simp*  
**finally show** *?thesis* **by** *simp*  
**qed**  
**also have**  $\dots = \text{es!}k$   
**using** *unitary-vec-norm*[*OF*  $v'(3)$  *UH-carrier* *UH-unitary*, *unfolded*  $v'(1)$ ] **by** *simp*  
**finally show** *?thesis* **using**  $v'$  **by** (*metis* *IntE* *field.one-not-zero* *vec-norm-zero*)  
**qed**

**lemma** *courant-fischer-unit-rayleigh-helper2*:

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

**have**  $\text{es-R}: \text{map } \text{complex-of-real } \text{es-R} = \text{es}$

**proof** –

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

**let**  $?B = \{\text{col } U i \mid i. i \in \{..<k + 1\}\}$

**define**  $\mathcal{V}$  **where**  $\mathcal{V} \equiv \text{span } ?B$

**have**  $\mathcal{B}\text{-card}: \text{card } ?B = k + 1$  **and**  $\mathcal{B}\text{-dim}: ?B \subseteq \text{carrier-vec } n$  **and**  $1: ?Q \mathcal{V}$

**unfolding** *dimensional-def*  $\mathcal{V}\text{-def}$

**using** *assms* *U-carrier* *U-cols-basis*[*of*  $\{..<k + 1\}$ ]

**by** (*simp-all* *add: span-is-subspace* *lessThan-atLeast0*)

**moreover have**  $2: \forall v. ?P \mathcal{V} v \longrightarrow ?geq v$

**proof** *clarify*

**fix**  $v :: \text{complex vec}$

**assume**  $*$ :  $v \neq 0_v n$  **and**  $**$ :  $v \in \mathcal{V}$  **and**  $***$ :  $\text{vec-norm } v = 1$

**have**  $v\text{-dim}: v \in \text{carrier-vec } n$  **using**  $**$   $\mathcal{B}\text{-dim}$   $\mathcal{V}\text{-def}$  *span-closed* **by** *blast*

**define**  $x$  **where**  $x \equiv U^H *_v v$

**have**  $x\text{-dim}: x \in \text{carrier-vec } n$

**by** (*metis* *U-carrier* *More-Matrix.carrier-vec-conjugate* *mult-mat-vec-carrier*)

*transpose-carrier-mat v-dim x-def*  
**have** *x-norm: vec-norm x = 1*  
**using** \*\*\*  
**using** *A-decomp[unfolding real-diag-decomp-def unitary-diag-def unitarily-equiv-def]*  
**using** *x-def x-dim v-dim*  
**by** (*metis Complex-Matrix.unitary-def adjoint-is-conjugate-transpose carrier-vecD dim-mult-mat-vec unitary-adjoint unitary-vec-norm*)

**have** *ineq:  $\bigwedge j. j \in \{..<n\} \implies es!k * (cmod (x\$j))^2 \leq es!j * (cmod (x\$j))^2$*   
**proof** –  
**fix** *j*  
**assume** *\*: j ∈ {..<n}*  
**show** *es!k \* (cmod (x\\$j))^2 ≤ es!j \* (cmod (x\\$j))^2*  
**proof**(*cases j ≤ k*)  
**case** *True*  
**hence** *es!k ≤ es!j*  
**using** *k by (metis len-eigenvalues antisym-conv1 eigenvalues-sorted nless-le sorted-wrt-nth-less)*  
**thus** *?thesis by (simp add: less-eq-complex-def mult-right-mono)*  
**next**  
**case** *False*  
**hence** *j > k by simp*  
**hence** *cmod (x\\$j) = 0*  
**using** *\* \*\*[unfolding V-def] assms U-cols-orthogonal-span[of {..<k + 1} v j, folded x-def]*  
**by** (*metis Diff-iff Suc-eq-plus1 atLeastLessThan-iff lessThan-atLeast0 lessThan-subset-iff less-iff-succ-less-eq norm-zero not-less-eq*)  
**thus** *?thesis by fastforce*  
**qed**  
**qed**

**have** *unit:  $(\sum j \in \{..<n\}. (cmod (x\$j))^2) = 1$*   
**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** *es-R!k = es!k using k eigenvalues-real es-R len-eigenvalues by (metis length-map nth-map)*  
**also have** *... = es!k \*  $(\sum j \in \{..<n\}. (cmod (x\$j))^2)$  by (simp add: unit)*  
**also have** *... =  $(\sum j \in \{..<n\}. es!k * (cmod (x\$j))^2)$  by (simp add: mult-hom.hom-sum)*  
**also have** *... ≤  $(\sum j \in \{..<n\}. es!j * (cmod (x\$j))^2)$  by (meson ineq sum-mono)*  
**also have** *... =  $\rho A v$  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 have** *\*: dimensional V (k + 1) ∧ (∀ v. ?P V v → es-R ! k ≤  $\rho A v$ ) by blast*  
**moreover have** *∀ v. v ≠ 0\_v n → v ∈ V →  $\rho A v = \rho A (vec-normalize v) ∧$*

```

?P V (vec-normalize v)
proof (clarify, rule conjI)
  fix v assume v: v ≠ 0_v n v ∈ V
  show ?P V (vec-normalize v)
    using 1 * v vec-normalize-norm[of v n] submodule-is-module[of V]
    using vec-normalize-def[of v] dimensional-def[of V k + 1] subspace-def[of
class-ring V V]
    by (metis dimensional-n dimensional-n-vec module-incl-imp-submodule rmult-0
submoduleE(4)
vec-eq-norm-smult-normalized)
  show ρ A v = ρ A (vec-normalize v)
    unfolding vec-normalize-def
    using rayleigh-quotient-scale-complex[OF A-dim, of v 1 / vec-norm v] v *
dimensional-n-vec vec-norm-zero
    by auto
  qed
ultimately show ?thesis by metis
qed

```

## 22.1 Max-Min Statement

**proposition** *courant-fischer-maximin*:

**assumes** k: k < n

**shows** es!k = maximin (k + 1)

maximin-defined (k + 1)

**proof** –

**define** es-R **where** es-R ≡ map Re es

**have** es-R: map complex-of-real es-R = es

**proof** –

{ **fix** i **assume** \*: i < length es

**hence** es!i ∈ Reals **using** eigenvalues-real **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** A-dim eigenvalues eigvals-poly-length es-R-def **by** simp

**hence** es-R-k: es-R!k = Re (es!k) **by** (simp add: k)

**let** ?leq = λv. ρ A v ≤ es-R!k

**let** ?geq = λv. ρ A v ≥ es-R!k

**let** ?P = λV v. v ≠ 0\_v n ∧ v ∈ V

**let** ?Q = λV. dimensional V (k + 1)

**let** ?S\_ρ = λV. {ρ A v | v. ?P V v}

**have** 1: ∧V. ?Q V ⇒ (∃v. ?P V v ∧ ?leq v)

**using** es-R-k **by** (metis Re-complex-of-real courant-fischer-unit-rayleigh-helper1 less-eq-complex-def)

**have**  $\varrho$ :  $\exists \mathcal{V}. ?Q \mathcal{V} \wedge (\forall v. ?P \mathcal{V} v \longrightarrow ?geq v)$   
**using** *courant-fischer-unit-rayleigh-helper2*[*OF k*] **unfolding** *es-R-def* **by** *blast*

**from**  $\varrho$  **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**  $\varrho A v' = es-R!k$  **by** *fastforce*  
**hence**  $es-R!k \in ?S_\varrho \mathcal{V}'$  **using**  $v'$  **by** *force*  
**moreover** **have**  $\forall x \in ?S_\varrho \mathcal{V}'. x \geq es-R!k$  **using** *all-v-geq* **by** *blast*  
**ultimately** **have**  $es-R!k = \text{Inf} (?S_\varrho \mathcal{V}')$  **by** (*smt (verit) rayleigh-bdd-below*  
*cInf-eq-minimum*)  
**moreover** **have**  $\bigwedge \mathcal{V}. ?Q \mathcal{V} \implies \text{Inf} (?S_\varrho \mathcal{V}) \leq es-R!k$   
**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**  $\varrho A v \in ?S_\varrho \mathcal{V}$  **by** *blast*  
**then** **have**  $\text{Inf} (?S_\varrho \mathcal{V}) \leq \varrho A v$   
**using** *rayleigh-min-exists*[*OF \**] *k rayleigh-min-def cf-setup-axioms* **by** *auto*  
**thus**  $\text{Inf} (?S_\varrho \mathcal{V}) \leq es-R!k$  **using**  $v$  **by** *linarith*  
**qed**  
**ultimately** **have**  $*$ :  $es-R!k \in \{\text{Inf} (?S_\varrho \mathcal{V}) \mid \mathcal{V}. ?Q \mathcal{V}\} \wedge (\forall x \in \{\text{Inf} (?S_\varrho \mathcal{V}) \mid \mathcal{V}. ?Q \mathcal{V}\}. x \leq es-R!k)$   
**using**  $\mathcal{V}'$  **by** *blast*  
**hence**  $\text{Sup} \{\text{Inf} (?S_\varrho \mathcal{V}) \mid \mathcal{V}. ?Q \mathcal{V}\} = es-R!k$  **by** (*meson cSup-eq-maximum*)  
**moreover** **have**  $\text{Sup} \{\text{Inf} (?S_\varrho \mathcal{V}) \mid \mathcal{V}. ?Q \mathcal{V}\} = \text{maximin} (k + 1)$   
**unfolding** *maximin-def rayleigh-min-def dimensional-def* **by** *blast*  
**ultimately** **show**  $es!k = \text{maximin} (k + 1)$   
**using**  $k$  *es-R* **by** (*metis len-eigenvalues length-map nth-map*)  
**show** *maximin-defined*  $(k + 1)$   
**using**  $*$  **unfolding** *maximin-defined-def sup-defined-def rayleigh-min-def bdd-above-def*  
**by** *blast*  
**qed**

## 22.2 Min-Max Statement

**interpretation** *neg*: *cf-setup* –  $A$   $n$

**by** (*unfold-locales, simp add: A-dim, simp add: negative-hermitian*)

**lemma** *neg-es*:  $neg.es = rev (\text{map} (\lambda x. -x) es)$

**proof** –

**let**  $?l = (\text{map} (\lambda x. -x) es)$

**{ fix**  $i j$  **assume**  $i < j < \text{length } ?l$

**moreover** **then** **have**  $es!i \geq es!j$

**using** *eigenvalues-sorted sorted-wrt-iff-nth-less*[*of*  $(\geq)$   $es$ ] **by** *force*

**ultimately** **have**  $?!i \leq ?!j$  **by** *simp*

**}**

**hence** *sorted-wrt*  $(\leq)$   $?l$  **by** (*metis sorted-wrt-iff-nth-less*)

**thus** *?thesis*

**using** *sorted-wrt-rev*  
**by** (*metis A-herm hermitian-is-square neg.eigenvalues-unique neg-mat-eigvals*  
*rev-rev-ident*  
*eigenvalues(1)*)  
**qed**

**lemma** *maximin-minimax*:  
**assumes**  $k: k < n$   
**shows**  $neg.maximin (n - k) = - minimax (k + 1)$   
 $neg.maximin-defined (n - k) \implies minimax-defined (k + 1)$   
**proof** –  
**define**  $P$  **where**  $P \equiv \lambda(v::complex\ vec)\ \mathcal{V}. v \neq 0_v \wedge v \in \mathcal{V}$   
**define**  $Q$  **where**  $Q \equiv \lambda\mathcal{V}. neg.\text{dimensional}\ \mathcal{V}\ (n - k)$

**have**  $Inf\ \{Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\} = -\ Sup\ (uminus\ \{Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\})$   
**using** *Inf-real-def* .  
**moreover have**  $*$ :  $uminus\ \{Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\} = \{-\ Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\}$   
**by** *blast*  
**moreover have**  $**$ :  $\{-\ Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\} = \{Inf\ \{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}\ |\mathcal{V}. Q\ \mathcal{V}\}$   
**proof** –  
**have**  $\bigwedge\mathcal{V}. Q\ \mathcal{V} \implies -\ Sup\ \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\} = Inf\ \{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}$   
**proof** –  
**fix**  $\mathcal{V}$  **assume**  $*$ :  $Q\ \mathcal{V}$   
**have**  $Inf\ \{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\} = -\ Sup\ (uminus\ \{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\})$   
**using** *Inf-real-def* **by** *fast*  
**moreover have**  $uminus\ \{\varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\} = \{-\ \varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\}$  **by**  
*blast*  
**moreover have**  $\{-\ \varrho\ (-A)\ v\ |v.\ P\ v\ \mathcal{V}\} = \{\varrho\ A\ v\ |v.\ P\ v\ \mathcal{V}\}$   
**proof** –  
**have**  $\bigwedge v.\ P\ v\ \mathcal{V} \implies -\ \varrho\ (-A)\ v = \varrho\ A\ v$   
**using**  $*$  *A-dim*  
**unfolding** *P-def Q-def*  
**by** (*metis 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**  $k$  **by** *fastforce*  
**ultimately show**  $cf-setup.maximin\ (-A)\ n\ (n - k) = -\ cf-setup.minimax\ A\ n\ (k + 1)$

**by** (*simp add: minimax-def neg.maximin-def neg.rayleigh-min-def rayleigh-max-def P-def Q-def*)

**show** *cf-setup.minimax-defined*  $A$   $n$   $(k + 1)$  **if** *cf-setup.maximin-defined*  $(-A)$   $n$   $(n - k)$

**proof**–

**have**  $\{Sup \{ \varrho A v \mid v. P v \mathcal{V} \} \mid \mathcal{V}. Q \mathcal{V} \} \neq \{ \}$

**using** *that*

**unfolding** *neg.maximin-defined-def sup-defined-def neg.rayleigh-min-def P-def Q-def*

**by** *fast*

**moreover have** *bdd-below*  $\{Sup \{ \varrho A v \mid v. P v \mathcal{V} \} \mid \mathcal{V}. Q \mathcal{V} \}$

**proof**–

**have** *bdd-above*  $\{Inf \{ \varrho (-A) v \mid v. P v \mathcal{V} \} \mid \mathcal{V}. Q \mathcal{V} \}$

**using** *that*

**unfolding** *neg.maximin-defined-def sup-defined-def neg.rayleigh-min-def P-def Q-def*

**by** *argo*

**hence** *bdd-above*  $\{ - Sup \{ \varrho A v \mid v. P v \mathcal{V} \} \mid \mathcal{V}. Q \mathcal{V} \}$  **using** **\*\* by** *argo*

**hence** *bdd-below*  $\{ Sup \{ \varrho A v \mid v. P v \mathcal{V} \} \mid \mathcal{V}. Q \mathcal{V} \}$  **by** (*smt (verit, best) \* bdd-above-uminus*)

**thus** *?thesis* **by** *blast*

**qed**

**moreover have**  $n - k = n - (k + 1) + 1$  **using**  $k$  **by** *simp*

**ultimately show** *?thesis*

**unfolding** *minimax-defined-def inf-defined-def rayleigh-max-def P-def Q-def*

**by** *algebra*

**qed**

**qed**

**lemma** *courant-fischer-minimax:*

**assumes**  $k: k < n$

**shows**  $es!k = \text{minimax } (k + 1) \text{ minimax-defined } (k + 1)$

**proof**–

**define**  $es'$  **where**  $es' = \text{rev } (\text{map } (\lambda x. -x) \text{ es})$

**have**  $*$ :  $es!(n - k - 1) = \text{neg.maximin } (n - k)$

$\wedge \text{neg.maximin-defined } (n - k)$

**using**  $k$  *neg.courant-fischer-maximin[of n - k - 1]*

**using** *neg.eigenvalues-unique neg-es es'-def*

**by** (*metis One-nat-def Suc-diff-Suc Suc-less-eq diff-less-Suc minus-nat.simps(1) semiring-norm(174)*)

*zero-less-diff*)

**moreover have**  $es!k = - es!(n - k - 1)$

**proof**–

**have**  $n - k - 1 < \text{length } es$  **using**  $A$ -*dim eigenvalues equals-poly-length* **by** *force*

**moreover have**  $\text{length } es = n$  **using**  $A$ -*dim eigenvalues* **by** *auto*

**ultimately have**  $es!k = (\text{rev } es)!(n - k - 1)$

**using** *rev-nth[of n - k - 1 es]* **by** (*simp add: Suc-diff-Suc k le-simps(1)*)

**also have**  $\dots = \text{rev } (\text{map } (\lambda x. -x) (\text{map } (\lambda x. -x) \text{ es}))!(n - k - 1)$   
**by** *simp*  
**also have**  $\dots = - \text{es}!(n - k - 1)$   
**by** (*metis*  $\langle n - k - 1 < \text{length es} \rangle$  *es'-def length-map length-rev nth-map rev-map*)  
**finally show** *?thesis* .  
**qed**  
**ultimately show**  $\text{es!}k = \text{cf-setup.minimax } A \ n \ (k + 1)$  *cf-setup.minimax-defined*  
 $A \ n \ (k + 1)$   
**using** *assms maximin-minimax* **by** *force+*  
**qed**

## 22.3 Theorem Statement

**theorem** *courant-fischer*:  
**assumes**  $k < n$   
**shows**  $\text{es!}k = \text{minimax } (k + 1)$   
 $\text{es!}k = \text{maximin } (k + 1)$   
 $\text{minimax-defined } (k + 1)$   
 $\text{maximin-defined } (k + 1)$   
**using** *courant-fischer-minimax courant-fischer-maximin maximin-minimax assms*  
**by** *algebra+*

## 23 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>

### 23.1 Theorem Statement and Proof

**lemma** *cauchy-eigval-interlacing-aux*:  
**fixes**  $W \ B :: \text{complex mat}$   
**fixes**  $\alpha \ \beta :: \text{complex list}$   
**fixes**  $j \ m :: \text{nat}$   
  
**defines**  $B \equiv W^H * A * W$   
**defines**  $\alpha \equiv \text{eigenvalues}$   
**defines**  $\beta \equiv \text{cmplx-herm-mat.eigenvalues } B$   
  
**assumes**  $j: j < n \ j < m$   
**assumes**  $m: 0 < m \ m \leq n$   
  
**assumes**  $W\text{-dim}: W \in \text{carrier-mat } n \ m$   
**assumes**  $W\text{-isom}: \text{isometry-mat } W \ n \ m$   
  
**shows**  $\beta!j \leq \alpha!j$  *set*  $\alpha \subseteq \mathbb{R}$  *set*  $\beta \subseteq \mathbb{R}$  *length*  $\alpha = n$  *length*  $\beta = m$   
**proof**–  
**interpret**  $B: \text{cf-setup } B \ m$

```

using A-dim A-herm W-dim B-def compression-is-hermitian(1)
by (metis carrier-matD(2) carrier-mat-triv cmplx-herm-mat.intro
    cf-setup.intro dim-row-conjugate index-mult-mat(2,3) index-transpose-mat(2))

show  $\alpha$ -real: set  $\alpha \subseteq \mathbb{R}$ 
  using hermitian-real-diag-decomp-eigvals[OF A-dim A-herm]  $\alpha$ -def eigenvalues(1) by metis
show  $\beta$ -real: set  $\beta \subseteq \mathbb{R}$ 
  using hermitian-real-diag-decomp-eigvals[of B n  $\beta$ ] B.eigenvalues(1)  $\beta$ -def
  by (metis B.A-dim B.A-herm hermitian-real-diag-decomp-eigvals)
show  $\alpha$ -length: length  $\alpha = n$  using  $\alpha$ -def eigenvalues(1) A-dim  $\alpha$ -def by auto
have  $B \in$  carrier-mat  $m\ m$  by (simp add: B.A-dim)
show  $\beta$ -length: length  $\beta = m$  using B.A-dim B.eigenvalues(1)  $\beta$ -def by fastforce

let  $?S_1 = \{B.\text{rayleigh-min } \mathcal{V} \mid \mathcal{V}. B.\text{dimensional } \mathcal{V} (j + 1)\}$ 
let  $?S_2 = \{A.\text{rayleigh-min } ((*_v) W \text{ ' } \mathcal{V}) \mid \mathcal{V}. B.\text{dimensional } \mathcal{V} (j + 1)\}$ 
let  $?S_3 = \{A.\text{rayleigh-min } \mathcal{V} \mid \mathcal{V}. A.\text{dimensional } \mathcal{V} (\text{Suc } j)\}$ 

interpret isom: isometry-mat  $W\ n\ m$  by (rule W-isom)
have  $*$ :  $\bigwedge \mathcal{V}. B.\text{dimensional } \mathcal{V} (\text{Suc } j) \implies A.\text{dimensional } ((*_v) W \text{ ' } \mathcal{V}) (\text{Suc } j)$ 
proof –
  fix  $\mathcal{V}$  assume  $*$ :  $B.\text{dimensional } \mathcal{V} (\text{Suc } j)$ 
  note  $Bdim =$ 
    conjunct1[OF *][unfolded B.dimensionality-def]
    conjunct2[OF *][unfolded B.dimensionality-def]
  show  $A.\text{dimensional } ((*_v) W \text{ ' } \mathcal{V}) (\text{Suc } j)$ 
  unfolding  $A.\text{dimensionality-def isom.inj-subspace-image}$ [OF isom.is-inj Bdim(1)
B.fn-dim]  $Bdim$ (2)
    using isom.subspace-image[OF Bdim(1)]
    by blast
qed
hence  $S_2$ -sub- $S_3$ :  $?S_2 \subseteq ?S_3$  by auto
have  $bdd$ -above- $S_3$ :  $bdd$ -above  $?S_3$ 
  using  $j$ (1) courant-fischer  $\alpha$ -def
  by (metis (lifting) ext A.maximin-defined-def Suc-eq-plus1 sup-defined-def)
hence  $bdd$ -above- $S_2$ :  $bdd$ -above  $?S_2$ 
  using  $S_2$ -sub- $S_3$  by (meson basic-trans-rules(31) bdd-above-def)
have  $S_2$ -ne:  $?S_2 \neq \{\}$ 
  using  $j$ (2)
by (metis (mono-tags, lifting) B.courant-fischer-unit-rayleigh-helper2 empty-Collect-eq)

have  $Re (\beta!j) = B.\text{maximin } (j + 1)$ 
  using B.courant-fischer(2)[OF j(2)]  $\beta$ -def by simp
also have  $\dots \leq \text{Sup } \{A.\text{rayleigh-min } ((\lambda v. W *_v v) \text{ ' } \mathcal{V}) \mid \mathcal{V}. B.\text{dimensional } \mathcal{V} (j + 1)\}$ 
proof –
  note  $1 = bdd$ -above- $S_2$ 
  have  $2$ :  $\forall y \in ?S_1. \exists x \in ?S_2. y \leq x$ 
  proof clarify

```

```

fix  $\mathcal{V}$  assume  $dim: B.dimension\ \mathcal{V}\ (j + 1)$ 
define  $\mathcal{V}'$  where  $\mathcal{V}' \equiv (*_v)\ W \ ' \ \mathcal{V}$ 
  have  $dim\text{-}\mathcal{V}'$ :  $A.dimension\ \mathcal{V}'\ (j + 1)$  using  $*[OF\ dim[simplified],\ folded\ \mathcal{V}'\text{-def}]$  by fastforce

define  $x :: real$  where  $x \equiv A.rayleigh\text{-}min\ \mathcal{V}'$ 
have  $x \in ?S_2$  unfolding  $x\text{-def}\ \mathcal{V}'\text{-def}$  using  $dim$  by blast
moreover have  $B.rayleigh\text{-}min\ \mathcal{V} \leq x$ 
proof–
  have  $le$ :  $\bigwedge x. x \in \{v. v \neq 0_v\ n \wedge v \in \mathcal{V}'\} \implies B.rayleigh\text{-}min\ \mathcal{V} \leq \rho\ A\ x$ 
  proof–
    fix  $x$  assume  $x: x \in \{v. v \neq 0_v\ n \wedge v \in \mathcal{V}'\}$ 
    then obtain  $v$  where  $xv: x = W *_v\ v$  and  $v: v \in \mathcal{V}$  and  $v\text{-nz}: v \neq 0_v\ m$ 
      unfolding  $\mathcal{V}'\text{-def}$  using that isom.T0-is-0 by fastforce

      have  $1$ :  $inner\text{-}prod\ (W *_v\ v)\ (A *_v\ (W *_v\ v)) = inner\text{-}prod\ v\ ((W^H * A *_v\ W *_v\ v))$ 
      proof–
        have  $inner\text{-}prod\ (W *_v\ v)\ (A *_v\ (W *_v\ v)) = inner\text{-}prod\ v\ (W^H *_v\ (A *_v\ (W *_v\ v)))$ 
        using  $W\text{-dim}\ v\ dim$ 
        by (meson A-dim B.cmplx-herm-mat-axioms cscalar-prod-conjugate-transpose B.dimension-n-vec mult-mat-vec-carrier)
        also have  $\dots = inner\text{-}prod\ v\ ((W^H * A * W) *_v\ v)$ 
        using  $W\text{-dim}\ A\text{-dim}\ v\ dim$ 
        by (smt (verit, ccfv-SIG) B.dimension-n-vec More-Matrix.carrier-vec-conjugate assoc-mult-mat-vec mult-carrier-mat mult-mat-vec-carrier transpose-carrier-mat)
        finally show  $?thesis$  .
      qed
      have  $2$ :  $inner\text{-}prod\ (W *_v\ v)\ (W *_v\ v) = inner\text{-}prod\ v\ v$ 
      using  $isom.preserves\text{-}norm\ v\ dim$  by (metis B.dimension-n-vec inner-prod-vec-norm-pow2)
      have  $bdd$ :  $bdd\text{-}below\ \{\rho\ B\ v \mid v. v \neq 0_v\ m \wedge v \in \mathcal{V}'\}$ 
      using  $B.rayleigh\text{-}bdd\text{-}below'$   $dim[unfolded\ B.dimension\text{-}def\ subspace\text{-}def\ submodule\text{-}def]$ 
      unfolding  $bdd\text{-}below\text{-}def$ 
      by auto
      have  $\rho\ A\ x = \rho\ A\ (W *_v\ v)$  unfolding  $xv\ ..$ 
      also have  $\dots = \rho\ B\ v$ 
      unfolding  $rayleigh\text{-}quotient\text{-}def\ rayleigh\text{-}quotient\text{-}complex\text{-}def\ B\text{-def}\ quadratic\text{-}form\text{-}def\ 1\ 2\ ..$ 
      finally show  $B.rayleigh\text{-}min\ \mathcal{V} \leq \rho\ A\ x$ 
      unfolding  $B.rayleigh\text{-}min\text{-}def$ 
      using  $v\ v\text{-nz}\ bdd$ 
      by (metis (mono-tags, lifting) cInf-lower mem-Collect-eq)
    qed
    have  $ne$ :  $\{v. v \neq 0_v\ n \wedge v \in \mathcal{V}'\} \neq \{\}$ 
    using  $dim\text{-}\mathcal{V}'[unfolded\ A.dimension\text{-}def]$   $nontriv\text{-}subspace\text{-}exists$  by

```

*fastforce*  
**have** *set-rw*:  $\{\varrho A v \mid v. v \neq 0_v \wedge v \in \mathcal{V}'\} = \varrho A \{v. v \neq 0_v \wedge v \in \mathcal{V}'\}$  **by** *blast*  
**show** *?thesis*  
**unfolding** *x-def A.rayleigh-min-def set-rw*  
**by** (*rule cINF-greatest[OF ne, of B.rayleigh-min  $\mathcal{V}$   $\lambda v. \varrho A v$ ], rule le*)  
**qed**  
**ultimately show**  $\exists x \in ?S_2. B.rayleigh-min \mathcal{V} \leq x$  **by** *blast*  
**qed**  
**have**  $?S_1 \neq \{\}$  **using** *S<sub>2</sub>-ne* **by** *blast*  
**show** *?thesis*  
**apply** (*simp add: B.maximin-def*)  
**using** *1 2 3* **by** (*metis (lifting) ext Suc-eq-plus1 cSup-mono*)  
**qed**  
**also have**  $\dots \leq Sup \{A.rayleigh-min \mathcal{V} \mid \mathcal{V}. A.dimension \mathcal{V} (j + 1)\}$   
**by** (*metis (no-types, lifting) ext S<sub>2</sub>-ne S<sub>2</sub>-sub-S<sub>3</sub> Suc-eq-plus1 bdd-above-S<sub>3</sub> cSup-subset-mono*)  
**also have**  $\dots = Re (\alpha!j)$   
**using** *bdd-above-S<sub>3</sub>  $\alpha$ -def courant-fischer(2)[OF j(1), unfolded A.maximin-def]*  
**by** *force*  
**finally have**  $Re (\beta!j) \leq Re (\alpha!j)$  .  
**moreover have**  $\alpha!j \in \mathbb{R}$  **using**  *$\alpha$ -real  $\alpha$ -length j(1)* **by** *fastforce*  
**moreover have**  $\beta!j \in \mathbb{R}$  **using**  *$\beta$ -real  $\beta$ -length j(2)* **by** *fastforce*  
**ultimately show**  $\beta!j \leq \alpha!j$  **by** (*simp add: complex-is-Real-iff less-eq-complex-def*)  
**qed**

**theorem** *cauchy-eigval-interlacing*:

**fixes** *W B* :: *complex mat*

**fixes**  $\alpha \beta$  :: *complex list*

**fixes** *j m* :: *nat*

**defines**  $B \equiv W^H * A * W$

**defines**  $\alpha \equiv$  *eigenvalues*

**defines**  $\beta \equiv$  *cmplx-herm-mat.eigenvalues B*

**assumes** *j: j < n j < m*

**assumes** *m: 0 < m m ≤ n*

**assumes** *W-dim: W ∈ carrier-mat n m*

**assumes** *W-isom: isometry-mat W n m*

**shows**  $\alpha!(n - m + j) \leq \beta!j \beta!j \leq \alpha!j$  *set  $\alpha \subseteq \mathbb{R}$  set  $\beta \subseteq \mathbb{R}$  length  $\alpha = n$  length  $\beta = m$*

**proof** –

**interpret** *B: cf-setup B m*

**apply** *unfold-locales*

**using** *A-dim A-herm W-dim B-def compression-is-hermitian*

**by** *auto*

**show**  $\beta!j \leq \alpha!j$  *set  $\alpha \subseteq \mathbb{R}$  set  $\beta \subseteq \mathbb{R}$  and lengths: length  $\alpha = n$  length  $\beta = m$*

**by** (*rule cauchy-eigval-interlacing-aux[OF j m W-dim W-isom, folded B-def*

$\alpha$ -def  $\beta$ -def))+

```

define  $A'$  where  $A' \equiv - A$ 
interpret  $A'$ : cf-setup  $A' n$ 
  using  $A$ -dim  $A$ -herm negative-hermitian by (unfold-locales, auto simp add:
 $A'$ -def)
define  $B'$  where  $B' \equiv W^H * A' * W$ 
have  $neg$ - $B$ :  $B' = - B$  unfolding  $B'$ -def  $A'$ -def  $B$ -def using  $W$ -dim  $A$ -dim by
force
then interpret  $B'$ : cf-setup  $B' m$ 
  by (simp add: B.A-dim B.negative-hermitian cmplx-herm-mat.intro cf-setup.intro)
define  $\alpha'$  where  $\alpha' = A'.es$ 
define  $\beta'$  where  $\beta' \equiv B'.es$ 

let  $?j' = m - j - 1$ 
have  $j'$ :  $?j' < n$   $?j' < m$  using  $m$  by simp-all
have  $\alpha!(n - m + j) = - \alpha'!(m - j - 1)$ 
  using lengths(1) m j j' neg-es[folded A'-def] by (simp add:  $\alpha'$ -def  $\alpha$ -def rev-nth)
moreover have  $\beta!j = - \beta'!(m - j - 1)$ 
  using lengths(2) m j j' neg-es
  by (simp add:  $\beta'$ -def  $\beta$ -def B.neg-es Suc-diff-Suc neg-B rev-nth)
ultimately show  $\alpha!(n - m + j) \leq \beta!j$ 
  using cmplx-herm-mat.cauchy-eigval-interlacing-aux(1)
  [OF A'.cmplx-herm-mat-axioms j' m W-dim W-isom, folded B'-def]
  unfolding  $\beta'$ -def  $\alpha'$ -def  $A'$ -def  $B'$ -def
  by force
qed

```

## 23.2 Principal Submatrix Corollaries (Using *pick* Function)

**corollary** *submatrix-eigval-interlacing*:

**fixes**  $I :: \text{nat set}$

**defines**  $B \equiv \text{submatrix } A \ I \ I$

**defines**  $m \equiv \text{card } I$

**defines**  $\alpha \equiv \text{eigenvalues}$

**defines**  $\beta \equiv \text{cmplx-herm-mat.eigenvalues } B$

**assumes**  $j$ :  $j < m$

**assumes**  $I$ :  $I \subseteq \{..<n\}$   $I \neq \{\}$

**shows**  $\alpha!(n - m + j) \leq \beta!j$   $\beta!j \leq \alpha!j$  *set  $\alpha \subseteq \mathbf{R}$  set  $\beta \subseteq \mathbf{R}$  length  $\alpha = n$  length  $\beta = m$*

**proof**–

**obtain**  $Q$  **where**  $Q$ :  $B = Q^H * A * Q$  *isometry-mat*  $Q \ n \ m$

**using**  $A$ -dim  $I$   $B$ -def  $m$ -def *submatrix-as-compression-obt* **by** *blast*

**have**  $Q$ -carrier:  $Q \in \text{carrier-mat } n \ m$

**using**  $Q(2)$ [*unfolded isometry-mat-def mat-as-linear-map-def*] ..

```

interpret B: cf-setup B m
  apply unfold-locales
  using A-dim A-herm B-def Q compression-is-hermitian(2)
  apply (metis Q-carrier)
  by (simp add: I(1) B-def principal-submatrix-hermitian)

have 1: j < n
  using j m-def I(1)
  by (metis atLeast0LessThan dual-order.strict-trans le-eq-less-or-eq subset-eq-atLeast0-lessThan-card)
have 2: 0 < m using j by linarith
have 3: m ≤ n
  using I(1) m-def atLeast0LessThan subset-eq-atLeast0-lessThan-card by pres-
  burger

  show α!(n - m + j) ≤ β!j β!j ≤ α!j set α ⊆ ℝ set β ⊆ ℝ length α = n length
  β = m
  using cauchy-eigval-interlacing[OF 1 j 2 3 Q-carrier Q(2)]
  unfolding Q(1)[symmetric]
  by (simp-all add: α-def β-def)
qed

```

### 23.3 Principal Submatrix Corollary (as Injective Function on Indices)

```

corollary submatrix-eigval-interlacing':
  fixes B :: complex mat
  fixes m :: nat
  fixes f :: nat ⇒ nat

  defines B ≡ submatrix-of-inj A f f m m
  defines α ≡ eigenvalues
  defines β ≡ cmplx-herm-mat.eigenvalues B

  assumes f: f : {..<m} → {..<n}
  assumes inj: inj-on f {..<m}
  assumes j: j < m

  shows α!(n - m + j) ≤ β!j β!j ≤ α!j set α ⊆ ℝ set β ⊆ ℝ length α = n length
  β = m
proof -
  obtain Q where Q: B = QH * A * Q isometry-mat Q n m
  using f inj α-def B-def submatrix-of-inj-as-compression-obt by (metis A-dim)
  have Q-carrier: Q ∈ carrier-mat n m
  using Q(2)[unfolded isometry-mat-def mat-as-linear-map-def] ..

  interpret B: cf-setup B m
  apply unfold-locales
  using A-dim A-herm B-def Q compression-is-hermitian Q-carrier
  by meson+

```

**note**  $\beta = \text{card-inj}[OF f \text{ inj, simplified}]$   
**hence**  $1: j < n$  **using**  $j$  **by** *auto*  
**have**  $2: 0 < m$  **using**  $j$  **by** *linarith*  
**show**  $\alpha!(n - m + j) \leq \beta!j$   $\beta!j \leq \alpha!j$  *set*  $\alpha \subseteq \mathbb{R}$  *set*  $\beta \subseteq \mathbb{R}$  *length*  $\alpha = n$  *length*  
 $\beta = m$   
**using**  $\alpha$ -*def*  $\beta$ -*def*  $Q(1)$  *cauchy-eigval-interlacing*[*OF 1 j 2 3 Q-carrier Q(2)*]  
**by** *fastforce+*  
**qed**  
**end**

## 24 Theorem Statements (Outside Locale)

**theorem** *courant-fischer*:

**fixes**  $A :: \text{complex mat}$

**defines**  $es \equiv \text{cmplx-herm-mat.eigenvalues } A$

**assumes** *carrier*:  $A \in \text{carrier-mat } n \ n$

**assumes** *herm*: *hermitian*  $A$

**assumes**  $k: k < n$

**shows**  $es!k = \text{cf-setup.minimax } A \ n \ (k + 1)$

$es!k = \text{cf-setup.maximin } A \ n \ (k + 1)$

$\text{cf-setup.minimax-defined } A \ n \ (k + 1)$

$\text{cf-setup.maximin-defined } A \ n \ (k + 1)$

**by** (*rule cmplx-herm-mat.courant-fischer*

[*OF cmplx-herm-mat.intro, OF carrier herm k, folded es-def*])+

**theorem** *cauchy-eigval-interlacing*:

**fixes**  $A \ W \ B :: \text{complex mat}$

**fixes**  $\alpha \ \beta :: \text{complex list}$

**fixes**  $j \ m \ n :: \text{nat}$

**defines**  $B \equiv W^H * A * W$

**defines**  $\alpha \equiv \text{cmplx-herm-mat.eigenvalues } A$

**defines**  $\beta \equiv \text{cmplx-herm-mat.eigenvalues } B$

**assumes**  $A: A \in \text{carrier-mat } n \ n$  *hermitian*  $A$

**assumes**  $j: j < n$   $j < m$

**assumes**  $m: 0 < m$   $m \leq n$

**assumes**  $W$ -*dim*:  $W \in \text{carrier-mat } n \ m$

**assumes**  $W$ -*isom*: *isometry-mat*  $W \ n \ m$

**shows**  $\alpha!(n - m + j) \leq \beta!j$   $\beta!j \leq \alpha!j$  *set*  $\alpha \subseteq \mathbb{R}$  *set*  $\beta \subseteq \mathbb{R}$  *length*  $\alpha = n$  *length*  
 $\beta = m$

**by** (*rule cmplx-herm-mat.cauchy-eigval-interlacing*

[*OF cmplx-herm-mat.intro, OF A j m W-dim W-isom, folded B-def alpha-def*]

$\beta$ -def])+

**corollary** *submatrix-eigval-interlacing*:

**fixes**  $A :: \text{complex mat}$

**fixes**  $I :: \text{nat set}$

**defines**  $B \equiv \text{submatrix } A \ I \ I$

**defines**  $m \equiv \text{card } I$

**defines**  $\alpha \equiv \text{cmplx-herm-mat.eigenvalues } A$

**defines**  $\beta \equiv \text{cmplx-herm-mat.eigenvalues } B$

**assumes**  $A: A \in \text{carrier-mat } n \ n \ \text{hermitian } A$

**assumes**  $j: j < m$

**assumes**  $I: I \subseteq \{..<n\} \ I \neq \{\}$

**shows**  $\alpha!(n - m + j) \leq \beta!j \ \beta!j \leq \alpha!j \ \text{set } \alpha \subseteq \mathbf{R} \ \text{set } \beta \subseteq \mathbf{R} \ \text{length } \alpha = n \ \text{length}$   
 $\beta = m$

**by** (*rule cmplx-herm-mat.submatrix-eigval-interlacing*  
[*OF cmplx-herm-mat.intro, of A n j I, folded B-def m-def  $\alpha$ -def  $\beta$ -def, OF A j I*])+

**corollary** *submatrix-eigval-interlacing'*:

**fixes**  $A \ B :: \text{complex mat}$

**fixes**  $m :: \text{nat}$

**fixes**  $f :: \text{nat} \Rightarrow \text{nat}$

**defines**  $B \equiv \text{submatrix-of-inj } A \ f \ f \ m \ m$

**defines**  $\alpha \equiv \text{cmplx-herm-mat.eigenvalues } A$

**defines**  $\beta \equiv \text{cmplx-herm-mat.eigenvalues } B$

**assumes**  $A: A \in \text{carrier-mat } n \ n \ \text{hermitian } A$

**assumes**  $f: f : \{..<m\} \rightarrow \{..<n\}$

**assumes**  $\text{inj}: \text{inj-on } f \ \{..<m\}$

**assumes**  $j: j < m$

**shows**  $\alpha!(n - m + j) \leq \beta!j \ \beta!j \leq \alpha!j \ \text{set } \alpha \subseteq \mathbf{R} \ \text{set } \beta \subseteq \mathbf{R} \ \text{length } \alpha = n \ \text{length}$   
 $\beta = m$

**by** (*rule cmplx-herm-mat.submatrix-eigval-interlacing'*  
[*OF cmplx-herm-mat.intro, of A n f m, folded B-def  $\alpha$ -def  $\beta$ -def, OF A f inj j*])+

**end**

**theory** *Sylvester-Criterion*

**imports**

*Cauchy-Eigenvalue-Interlacing*

*Jordan-Normal-Form.Determinant-Impl*

**begin**

## 25 Sylvester's Criterion Setup

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

**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** *assms nested-leading-principle-submatrices*  
**by** (*smt* (*verit*, *del-insts*) *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** *Determinant.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**

## 26 Sylvester's Criterion

### 26.1 Forward Implication

**lemma** (*in cmplx-herm-mat*) *sylvester-criterion-forward*:

**assumes** x  $\in$  carrier-vec n  
**assumes** *sylvester-criterion* A  
**assumes** x  $\neq 0_v$  n  
**shows** *Re* (QF A x)  $> 0$   
**using** *assms A-dim A-herm*  
**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** \*:  $\bigwedge k. k \leq \text{dim-row } A \implies \text{Determinant.det } (\text{leading-principal-submatrix } A \ k) > 0$   
**using** *Suc(3) unfolding sylvester-criterion-def by blast*

**define** A<sub>n</sub> **where** A<sub>n</sub>  $\equiv$  (*leading-principal-submatrix* A n)  
**define** v<sub>n</sub> **where** v<sub>n</sub>  $\equiv$  *vec-first* (col A n) n  
**define** v<sub>n</sub>c **where** v<sub>n</sub>c  $\equiv$  *conjugate* v<sub>n</sub>

**define**  $w_n$  **where**  $w_n \equiv \text{vec-first } (\text{row } A \ n) \ n$   
**define**  $a$  **where**  $a \equiv A \ \$\$ \ (n, n)$   
**define**  $x_n$  **where**  $x_n \equiv \text{vec-first } x \ n$   
**define**  $x_n c$  **where**  $x_n c = \text{conjugate } x_n$   
**define**  $b$  **where**  $b \equiv x \$ n$   
**define**  $b\text{-conj}$  **where**  $b\text{-conj} \equiv \text{conjugate } b$

**have**  $\text{carrier-}A_n$ :  $A_n \in \text{carrier-mat } n \ n$   
**by** (*metis*  $A_n\text{-def}$   $\text{Suc}(5)$   $\text{le-add2}$   $\text{leading-principal-submatrix-carrier-plus-1-eq-Suc}$ )  
**have**  $\text{herm-}A_n$ :  $\text{hermitian } A_n$   
**using**  $\text{principal-submatrix-hermitian}$ [of  $\{..n\}$ ]  $A_n\text{-def}$   $\text{Suc}$   $A_n\text{-def}$   
**using**  $\text{cmplx-herm-mat.leading-principal-submatrix-hermitian}$   $\text{cmplx-herm-mat-def}$   
 $\text{lessI}$   $\text{less-or-eq-imp-le}$   
**by** *presburger*

**have**  $(\text{col } A \ n) = \text{conjugate } (\text{row } A \ n)$   
**using**  $\text{Suc}(5,6)$  **by** (*metis*  $\text{adjoint-col}$   $\text{carrier-matD}(1)$   $\text{hermitian-def}$   $\text{lessI}$ )  
**hence**  $w_n\text{-vn-conj}$ :  $w_n = v_n c$   
**using**  $\text{Suc}(5)$   $v_n\text{-def}$   $v_n c\text{-def}$   $w_n\text{-def}$   
**by** (*metis*  $\text{conjugate-vec-first}$   $\text{col-carrier-vec}$   $\text{conjugate-id}$   $\text{le-add2}$   $\text{lessI}$   $\text{plus-1-eq-Suc}$ )

**have**  $\text{invertible-mat } A_n$   
**using**  $\text{Suc}(5) * A_n\text{-def}$   $\text{carrier-}A_n$   
**by** (*metis*  $\text{carrier-matD}(1)$   $\text{invertible-det}$   $\text{le-add2}$   $\text{less-irrefl}$   $\text{plus-1-eq-Suc}$ )  
**then obtain**  $A_n'$  **where**  $A_n'$ :  $\text{inverts-mat } A_n' \ A_n \wedge A_n' \in \text{carrier-mat } n \ n$   
**using**  $A_n\text{-def}$   $\text{Suc}(5)$   
**by** (*metis* (*no-types*, *lifting*)  $\text{invertible-mat-def}$   $\text{carrier-matD}(1)$   $\text{carrier-matI}$   $\text{index-mult-mat}(3)$   $\text{index-one-mat}(3)$   $\text{inverts-mat-def}$   $\text{le-add2}$   $\text{leading-principal-submatrix-carrier-plus-1-eq-Suc}$   $\text{square-mat.simps}$ )

**have**  $x_n$ :  $x_n \in \text{carrier-vec } n$  **by** (*simp*  $\text{add}$ :  $x_n\text{-def}$ )  
**moreover have**  $A_n$ :  $A_n \in \text{carrier-mat } n \ n$   
**using**  $\text{leading-principal-submatrix-carrier}$   $\text{Suc}(5)$   $A_n\text{-def}$  **by** (*metis*  $\text{Suc-n-not-le-n}$   $\text{linorder-linear}$ )  
**ultimately have**  $A_n x_n$ :  $A_n *_v x_n \in \text{carrier-vec } n$  **by** *fastforce*  
**have**  $v_n$ :  $v_n \in \text{carrier-vec } n$  **by** (*simp*  $\text{add}$ :  $v_n\text{-def}$ )  
**hence**  $b \cdot_v v_n$ :  $b \cdot_v v_n \in \text{carrier-vec } n$  **by** *simp*

**have**  $(A_n *_v x_n + b \cdot_v v_n) \in \text{carrier-vec } n$  **using**  $A_n x_n$   $b \cdot_v v_n$  **by** *auto*

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

**have** \*\*:  $(A_n *_v (x_n + b \cdot_v (A_n' *_v v_n))) \cdot (x_n c + b\text{-conj} \cdot_v (A_n'^T *_v v_n c))$   
 $= (\text{QF } A_n \ x_n) + b * (x_n c \cdot v_n) + b\text{-conj} * (x_n \cdot v_n c) + (\text{cmod } b)^{\wedge} 2 * ((A_n' *_v v_n) \cdot v_n c)$

(is ?lhs = ?rhs)

**proof**–

**define**  $E$  **where**  $E \equiv ((A_n *_v x_n) \cdot (x_n c + b \cdot_{\text{conj}} \cdot_v (A_n^{tT} *_v v_n c)))$

**define**  $F$  **where**  $F \equiv ((b \cdot_v v_n) \cdot (x_n c + b \cdot_{\text{conj}} \cdot_v (A_n^{tT} *_v v_n c)))$

**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-An 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-An mult-mat-vec mult-mat-vec-carrier vn*)

**also have** ... =  $(A_n *_v x_n) + b \cdot_v v_n$

**by** (*metis An' carrier-An carrier-matD(1) inverts-mat-def inverts-mat-symm one-mult-mat-vec vn*)

**finally show** ?thesis .

**qed**

**hence** ?lhs =  $((A_n *_v x_n) + b \cdot_v v_n) \cdot (x_n c + b \cdot_{\text{conj}} \cdot_v (A_n^{tT} *_v v_n c))$  **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**  $E = QF A_n x_n + b \cdot_{\text{conj}} * (x_n \cdot v_n c)$

**proof**–

**have**  $E = ((A_n *_v x_n) \cdot x_n c) + ((A_n *_v x_n) \cdot (b \cdot_{\text{conj}} \cdot_v (A_n^{tT} *_v v_n c)))$

**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**  $((A_n *_v x_n) \cdot (b \cdot_{\text{conj}} \cdot_v (A_n^{tT} *_v v_n c))) = b \cdot_{\text{conj}} * (x_n \cdot v_n c)$

(is ?lhs = -)

**proof**–

**have** ?lhs =  $b \cdot_{\text{conj}} * ((A_n *_v x_n) \cdot (A_n^{tT} *_v v_n c))$  **using** *An An'* **by** *auto*

**also have** ... =  $b \cdot_{\text{conj}} * ((A_n *_v x_n) \cdot (\text{conjugate } A_n' *_v v_n c))$

**using** *A\_n-inv-conj* **by** *presburger*

**also have** ... =  $b \cdot_{\text{conj}} * (((A_n' * A_n) *_v x_n) \cdot v_n c)$

**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** ... =  $b \cdot_{\text{conj}} * (x_n \cdot v_n c)$

**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**  $F = b * (x_n c \cdot v_n) + (c \text{ mod } b)^{\wedge 2} * ((A_n' *_v v_n) \cdot v_n c)$   
**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**  $F\text{-def}$   
**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 x\_n*)  
**moreover have**  $(b \cdot_v v_n) \cdot (x_n c) = b * (v_n \cdot x_n c)$  **using** *vn x\_n c-def x\_n* **by**  
*auto*  
**moreover have**  $(b \cdot_v v_n) \cdot (b\text{-conj} \cdot_v (A_n'^T *_v v_n c)) = (c \text{ mod } b)^{\wedge 2} * ((A_n' *_v v_n) \cdot v_n c)$   
**(is ?lhs = -)**  
**proof**–  
**have**  $?lhs = (c \text{ mod } b)^{\wedge 2} * (v_n \cdot (A_n'^T *_v v_n c))$   
**using** *An' vn b-conj-def complex-norm-square* **by force**  
**also have**  $\dots = (c \text{ mod } b)^{\wedge 2} * ((A_n' *_v v_n) \cdot v_n c)$   
**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 x\_n*)  
**qed**  
**ultimately show** *?thesis* **by** *fastforce*  
**qed**

**let**  $?c_n = b \cdot_v (A_n' *_v v_n)$   
**have** *cn: ?c\_n ∈ carrier-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**  $A \in \text{carrier-mat } (Suc\ n) (Suc\ n)$   
**by** (*simp add: Suc(5)*)  
**moreover have**  $x \in \text{carrier-vec } (Suc\ n)$   
**by** (*simp add: Suc(2)*)  
**ultimately have**  $Ax: A *_v x = (A_n *_v x_n + b \cdot_v v_n) @_v (\text{vec } 1 (\lambda i. (w_n \cdot x_n) + a * b))$   
**(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**  $QF\ A\ x = \dots \cdot c\ x$  **by force**  
**also have**  $\dots = ((A_n *_v x_n + b \cdot_v v_n) \cdot c\ x_n) + ((w_n \cdot x_n) + a * b) * b\text{-conj}$   
**proof**–  
**have**  $x \in \text{carrier-vec } (\text{dim-vec } ((A_n *_v x_n + b \cdot_v v_n) @_v ?Ax\text{-last}))$   
**using** *Suc(2) vn* **by force**  
**moreover have**  $(A_n *_v x_n + b \cdot_v v_n) \cdot c (\text{vec-first } x (\text{dim-vec } (A_n *_v x_n + b \cdot_v v_n)))$   
 $= (A_n *_v x_n + b \cdot_v v_n) \cdot c\ x_n$

by (simp add:  $v_n$ -def  $x_n$ -def)  
 moreover have  $\dim\text{-vec } ?Ax\text{-last} = 1$  by simp  
 moreover have  $?Ax\text{-last} \cdot c (\text{vec-last } x \ 1) = (w_n \cdot x_n + a * b) * b\text{-conj}$   
 proof –  
   have  $\dim\text{-vec } ?Ax\text{-last} = 1$  by simp  
   moreover have  $(\text{vec-last } x \ 1)\$0 = b$   
   using Suc(2)  $b$ -def  
   by (smt (verit) add.commute add.right-neutral add-diff-cancel-right' carrier-vecD index-vec plus-1-eq-Suc vec-last-def zero-less-one-class.zero-less-one)  
   moreover have  $?Ax\text{-last}\$0 = (w_n \cdot x_n + a * b)$  by simp  
   moreover have  $?Ax\text{-last} \cdot c (\text{vec-last } x \ 1) = ?Ax\text{-last}\$0 * \text{conjugate } ((\text{vec-last } x \ 1)\$0)$   
   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  $\dots = QF \ A_n \ x_n + ((b \cdot_v v_n) \cdot c \ x_n) + ((w_n \cdot x_n) * b\text{-conj}) + (a * b * b\text{-conj})$   
   using inner-prod-distrib-right[of  $x_n \ n \ A_n * _v \ x_n \ b \cdot_v \ v_n$ ]  $b$ -vn  $A_n$ -xn  
   by (simp add: ring-class.ring-distrib(2)  $x_n$ )  
 also have  $\dots = QF \ A_n \ x_n + ((b \cdot_v v_n) \cdot c \ x_n) + ((w_n \cdot x_n) * b\text{-conj}) + (a * (\text{cmod } b)^{\wedge}2)$   
   using  $b$ -conj-def complex-norm-square by auto  
 also have  $\dots = QF \ A_n \ x_n + b * (x_n c \cdot v_n) + b\text{-conj} * (x_n \cdot v_n c) + (a * (\text{cmod } b)^{\wedge}2)$   
   by (metis conjugate-vec-sprod-comm inner-prod-smult-left mult.commute  $v_n c$ -def vn vn-vn-conj  $x_n c$ -def  $x_n$ )  
 also have  $\dots = (A_n * _v (x_n + b \cdot_v (A_n' * _v v_n))) \cdot (x_n c + b\text{-conj} \cdot_v (A_n'^T * _v v_n c))$   
   –  $(\text{cmod } b)^{\wedge}2 * ((A_n' * _v v_n) \cdot v_n c) + (a * (\text{cmod } b)^{\wedge}2)$   
   using \*\* by fastforce  
 also have  $\dots = (A_n * _v (x_n + b \cdot_v (A_n' * _v v_n))) \cdot (x_n c + b\text{-conj} \cdot_v (A_n'^T * _v v_n c))$   
   +  $(\text{cmod } b)^{\wedge}2 * (a - QF \ A_n' \ v_n)$   
   by (simp add: right-diff-distrib  $v_n c$ -def)  
 also have  $\dots = QF \ A_n (x_n + ?c_n) + (\text{cmod } b)^{\wedge}2 * (a - QF \ A_n' \ v_n)$   
 proof –  
   have  $\text{conjugate } (A_n' * _v v_n) = (\text{conjugate } A_n' * _v v_n c)$   
   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  $x_n$  quadratic-form-def)  
 qed  
 finally have eq:  $QF \ A \ x = QF \ A_n (x_n + ?c_n) + (\text{cmod } b)^{\wedge}2 * (a - QF \ A_n' \ v_n)$ .  
 have  $x_n\text{-}c_n: (x_n + ?c_n) \in \text{carrier-vec } n$  using add-carrier-vec cn  $x_n$  by blast

```

have sylvester-criterion  $A_n$ 
  using leading-principle-submatrix-sylvester  $A_n$ -def  $Suc(3,5)$ 
  by (metis  $Suc$ - $n$ -not-le- $n$  linorder-linear)
have 1:  $Re(QF A_n (x_n + ?c_n)) > 0$  if  $x_n + ?c_n \neq 0_v n$ 
  using  $Suc.IH[OF x_n$ - $c_n$  - that carrier- $A_n$  herm- $A_n]$  by blast
have 2:  $x_n + ?c_n \neq 0_v n$  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  $\neg x_n + ?c_n \neq 0_v n$ 
    hence  $x_n = 0_v n$  using * by argo
    hence  $\forall i < n. x_n \$i = 0$  by fastforce
    moreover have  $\forall i < n. x_n \$i = x \$i$  by (simp add: vec-first-def  $x_n$ -def)
    moreover have  $x \$n = 0$  using that unfolding  $b$ -def .
    ultimately have  $\forall i < Suc n. x \$i = 0$  using less- $Suc$ -eq by presburger
    thus False using  $Suc$  by auto
  qed
qed
have 3:  $a - QF A_n' v_n > 0$ 
proof-
  have  $Determinant.det A = Determinant.det A_n * (a - QF A_n' v_n)$ 
  proof-
    let  $?B = mat$ -of-cols  $n [v_n]$ 
    let  $?C = mat$ -of-rows  $n [conjugate v_n]$ 
    let  $?D = mat 1 1 (\lambda. a)$ 

    have  $(A_n, ?B, ?C, ?D) = split$ -block  $A n n$ 
    proof-
      have  $A_n = mat n n ((\$ \$) A)$ 
      using  $A_n$ -def  $An An$ - $xn Suc(5)$ 
      by (metis carrier-mat $D(2)$  carrier-vec $D dim$ -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  $A - n) (\lambda(i, j). A \$ \$ (i, j + n))$ 
      (is ?lhs = ?rhs)
    proof
      show  $dim$ -row ?lhs =  $dim$ -row ?rhs by simp
      show  $dim$ -col ?lhs =  $dim$ -col ?rhs using  $Suc(5)$  by force
      show  $\bigwedge i j. i < dim$ -row ?rhs  $\implies j < dim$ -col ?rhs  $\implies ?lhs \$ \$ (i, j) =$ 
?rhs  $\$ \$ (i, j)$ 
    proof-
      fix  $i j$  assume *:  $i < dim$ -row ?rhs  $j < dim$ -col ?rhs
      hence  $j = 0$  using  $Suc(5)$  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(5)
    by auto
  qed
  qed
  moreover have ?C = mat (dim-row A - n) n (λ(i, j). A $$ (i + n, j))
    (is ?lhs = ?rhs)
  proof
    show dim-row ?lhs = dim-row ?rhs using Suc(5) by force
    show dim-col ?lhs = dim-col ?rhs by simp
    show ∧i j. i < dim-row ?rhs ⇒ j < dim-col ?rhs ⇒ ?lhs$$ (i, j) =
?rhs$$ (i, j)
    proof -
      fix i j assume *: i < dim-row ?rhs j < dim-col ?rhs
      hence i = 0 using Suc(5) by auto
      moreover have conjugate (vec n (($) (col A n))) = (vec n (($) (row A
n)))
        using Suc(4)
        unfolding hermitian-def adjoint-def
        by (metis vn-def vnc-def vec-first-def wn-def wn-vn-conj)
      ultimately show ?lhs$$ (i, j) = ?rhs$$ (i, j)
        apply (simp add: vn-def vec-first-def mat-of-cols-def)
        using *(2) Suc(5)
        by (simp add: mat-of-rows-def)
    qed
  qed
  moreover have ?D = mat (dim-row A - n) (dim-col A - n) (λ(i, j). A
$$ (i + n, j + n))
    (is ?lhs = ?rhs)
  proof
    show row: dim-row ?lhs = dim-row ?rhs and col: dim-col ?lhs = dim-col
?rhs
      using Suc(5) 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(5) 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 Determinant.det A = Determinant.det An * Determinant.det (?D -
?C * An' * ?B)
    using schur-formula[of An ?B ?C ?D A n n An'] An' Suc(5,6) herm-An
hermitian-is-square

```

by (metis carrier-matD(1) carrier-matD(2) lessI)  
 moreover have  $\text{Determinant.det } (?D - ?C * A_n' * ?B) = (a - QF A_n' v_n)$   
 proof-  
 have  $?C * A_n' * ?B = \text{mat } 1 \ 1 \ (\lambda-. QF A_n' v_n)$  (is ?lhs = ?rhs)  
 proof  
 have dim:  $?C * A_n' * ?B \in \text{carrier-mat } 1 \ 1$  by (simp add: carrier-matI)  
 thus dim-row ?lhs = dim-row ?rhs dim-col ?lhs = dim-col ?rhs by auto  
 have col  $(A_n' * ?B) \ 0 = A_n' * v_n$   
 using  $A_n' \ v_n$   
 by (metis mat-vec-as-mat-mat-mult)  
 moreover have row  $?C \ 0 = \text{conjugate } v_n$  using  $v_n \ c\text{-def } w_n\text{-def } w_n\text{-}v_n\text{-conj}$   
 by auto  
 moreover have  $(?C * (A_n' * ?B))\$(0,0) = \text{row } ?C \ 0 \cdot \text{col } (A_n' * ?B)$   
 0 by simp  
 moreover have  $?C * (A_n' * ?B) = ?C * A_n' * ?B$   
 using  $A_n'$   
 by (metis assoc-mult-mat carrier-matI mat-of-cols-carrier(2) mat-of-rows-carrier(3))  
 ultimately have  $(?C * A_n' * ?B)\$(0,0) = (\text{conjugate } v_n) \cdot (A_n' * v_n)$   
 by argo  
 also have  $\dots = (A_n' * v_n) \cdot c \ v_n$   
 using  $A_n' \ v_n$  by (metis conjugate-vec-sprod-comm mult-mat-vec-carrier)  
 also have  $\dots = QF A_n' v_n$  by simp  
 finally show  $\bigwedge i \ j. i < \text{dim-row } ?rhs \implies j < \text{dim-col } ?rhs \implies ?lhs\$(i,j)$   
 =  $?rhs\$(i,j)$   
 by fastforce  
 qed  
 hence  $?D - ?C * A_n' * ?B = \text{mat } 1 \ 1 \ (\lambda-. a - QF A_n' v_n)$  by fastforce  
 thus ?thesis by (simp add: det-single)  
 qed  
 ultimately show ?thesis by argo  
 qed  
 moreover have  $\text{Determinant.det } A > 0$  using  $\text{Suc.premis sylvester-criterion-positive-det}$   
 by blast  
 moreover have  $\text{Determinant.det } A_n > 0$  using  $\text{Suc unfolding } A_n\text{-def sylvester-criterion-def}$   
 by simp  
 ultimately show ?thesis by (simp add: less-complex-def zero-less-mult-iff)  
 qed  
 have  $4: (c \bmod b)^2 > 0$  if  $b \neq 0$  using that by force  
  
 have ?case if  $b = 0$   
 proof-  
 have  $\text{Re } (QF A_n (x_n + ?c_n)) > 0$  using that 1 2 by blast  
 thus ?thesis unfolding eq by (simp add: that)  
 qed  
 moreover have ?case if  $b \neq 0$   
 proof-  
 have  $(c \bmod b)^2 * (a - QF A_n' v_n) > 0$   
 using 3 4[OF that] by (simp add: less-le square-nneg-complex)  
 moreover have  $\text{Re } (QF A_n (x_n + ?c_n)) \geq 0$  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

```

## 26.2 Reverse Implication

```

lemma prod-list-gz:
  fixes l :: real list
  assumes  $\forall x \in \text{set } l. x > 0$ 
  shows prod-list l > 0
  using assms by (induct l, auto+)

```

```

context cmplx-herm-mat
begin

```

```

lemma positive-definite-imp-positive-eigenvalues:
  assumes pd: positive-definite A
  shows  $\forall \mu \in \text{spectrum } A. \mu > 0$ 
proof
  have square: square-mat A using pd[unfolded positive-definite-def] hermitian-is-square
  ..

```

```

    fix  $\mu$  assume  $\mu: \mu \in \text{spectrum } A$ 
    moreover have  $\mu > 0$ 
    proof -
      obtain x where x:  $x \in \text{carrier-vec } (\text{dim-row } A) \wedge x \neq 0_v (\text{dim-row } A) \wedge A * _v$ 
       $x = \mu \cdot_v x$ 
      using  $\mu$  pd
      by (metis eigenvalue-def eigenvector-def hermitian-square positive-definite-def
      spectrum-eigenvalues)
      hence  $\mu * (x \cdot_c x) > 0$ 
      using pd x  $\mu$  square
      unfolding positive-definite-def
      by (metis dim-vec-conjugate quadratic-form-def scalar-prod-smult-left square-mat.simps)
      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  $\mu > 0$  by (simp add: less-complex-def)
  qed

```

Technically, the following direction is not needed for the final result, but is useful later.

```

lemma positive-eigenvalues-imp-positive-definite:
  assumes pos-spectrum:  $\forall \mu \in \text{spectrum } A. \mu > 0$ 
  shows positive-definite A
proof -
  have ?thesis if n = 0
    using A-dim that unfolding positive-definite-def hermitian-def adjoint-def by
  auto

```

**moreover have** *?thesis* **if**  $n\text{-nz}: n \neq 0$   
**proof**–  
**have**  $(\forall x \in \text{carrier-vec } n. x \neq 0_v \ n \longrightarrow 0 < QF\ A\ x)$   
**proof clarify**  
**fix**  $x :: \text{complex vec}$  **assume**  $x: x \in \text{carrier-vec } n\ x \neq 0_v\ n$   
**have**  $Min\ (Re\ \text{' set eigenvalues}) > 0$   
**proof**–  
**have**  $\forall \mu \in \text{set eigenvalues}. \mu > 0$   
**using**  $A\text{-dim cmplx-herm-mat.eigenvalues pos-spectrum}$   
**by**  $(metis\ eigenvalues\ eigvals-of-spectrum\ spectrum-connect)$   
**moreover have**  $\text{set eigenvalues} \subseteq \mathbb{R}$   
**using**  $A\text{-herm eigenvalues-real}$  **by force**  
**ultimately have**  $\forall \mu \in Re\ \text{' set eigenvalues}. \mu > 0$  **using**  $\text{less-complex-def}$   
**by fastforce**  
**moreover have**  $\text{set eigenvalues} \neq \{\}$   $\wedge$   $\text{finite (set eigenvalues)}$   
**using**  $\text{cmplx-herm-mat.eigenvalues[unfolded eigvals-of] } A\text{-dim } n\text{-nz len-eigenvalues}$   
**by fastforce**  
**ultimately show** *?thesis* **by auto**  
**qed**  
**hence**  $0 < \varrho\ A\ x$  **using**  $\text{rayleigh-bdd-below' } x$  **by fastforce**  
**moreover have**  $(\text{vec-norm } x)^2 > 0$  **using**  $x\ \text{vec-norm-zero}$  **by**  $(simp\ add:$   
 $\text{inner-prod-vec-norm-pow2})$   
**ultimately have**  $\varrho\ A\ x * (\text{vec-norm } x)^2 > 0$  **by**  $(simp\ add: \text{less-eq-complex-def}$   
 $\text{less-le})$   
**thus**  $0 < QF\ A\ x$  **using**  $\text{rayleigh-quotient-QF[OF } x(2,1)]\ \text{vec-norm-zero } x$   
**by auto**  
**qed**  
**thus** *?thesis* **unfolding**  $\text{positive-definite-def}$  **using**  $A\text{-herm } A\text{-dim}$  **by blast**  
**qed**  
**ultimately show** *?thesis* **by blast**  
**qed**

**lemma**  $\text{positive-definite-imp-positive-eigenvalues}_0\text{-Re}$ :  
**assumes**  $pd: \text{positive-definite } A$   
**shows**  $\forall \mu \in \text{set (map } Re\ \text{eigenvalues)}. \mu > 0$   
**proof**  
**fix**  $\mu$  **assume**  $\mu: \mu \in \text{set (map } Re\ \text{eigenvalues)}$   
**then obtain**  $\mu'$  **where**  $\mu': \mu' \in \text{set eigenvalues} \wedge \mu = Re\ \mu'$  **by auto**  
**hence**  $\mu' > 0$   
**using**  $\text{positive-definite-imp-positive-eigenvalues[OF } pd]$   
**using**  $\text{eigenvalues}_0\text{-def eigenvalues-set-eq spectrum-def}$   
**by auto**  
**moreover hence**  $\mu' \in \mathbb{R}$  **using**  $\text{complex-is-real-iff-compare0}$  **by fastforce**  
**ultimately show**  $\mu > 0$  **by**  $(simp\ add: \mu'\ \text{less-complex-def})$   
**qed**

**lemma**  $\text{sylvester-criterion-reverse}$ :  
**assumes**  $pd: \text{positive-definite } A$   
**shows**  $\text{sylvester-criterion } A$

```

unfolding sylvester-criterion-def
proof clarify
  fix k assume k:  $k \leq \text{dim-row } A$ 
  let  $?A' = \text{lps } A \ k$ 
  have pd: positive-definite ?A'
    using A-dim pd leading-principal-submatrix-positive-definite k by auto
  hence det-nz: Determinant.det ?A'  $\neq 0$  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:  $?A' \in \text{carrier-mat } k \ k$  using A-dim k leading-principal-submatrix-carrier
by auto

  have prod-list (map Re (eigvals ?A'))  $> 0$ 
    using cmplx-herm-mat.positive-definite-imp-positive-eigenvalues0-Re[of - ?A']
pd prod-list-gz
    using A'-dim cmplx-herm-mat.intro positive-definite-def
    by (smt (verit, best) cmplx-herm-mat.positive-definite-imp-positive-eigenvalues
imageE less-complex-def
      list.set-map spectrum-def zero-complex.sel(1))
  moreover have prod-list (eigvals ?A') = prod-list (map Re (eigvals ?A'))
proof–
  have  $\forall i < (\text{length } (\text{eigvals } ?A')). (\text{eigvals } ?A')!i = (\text{map } \text{Re } (\text{eigvals } ?A'))!i$ 
proof safe
  fix i assume  $*: i < \text{length } (\text{eigvals } ?A')$ 
  hence (eigvals ?A')!i  $\in \text{Reals}$ 
    using pd A'-dim
    by (metis cmplx-herm-mat.hermitian-eigenvalues-real cmplx-herm-mat-def
nth-mem
      positive-definite-def spectrum-def spectrum-eigenvalues)
  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  $0 < \text{Determinant.det } ?A'$ 
    using det-is-prod-of-eigenvalues[OF square] by (simp add: less-complex-def)
qed

```

## 26.3 Theorem Statement

**theorem** *sylvester-criterion*:

**shows** *sylvester-criterion*  $A \longleftrightarrow \text{positive-definite } A$

**proof**

**show** *1*: *sylvester-criterion*  $A \implies \text{positive-definite } A$

**using** *sylvester-criterion-forward*

**unfolding** *positive-definite-def*

**using** *A-herm A-dim sylvester-criterion-forward*

**using** *complex-is-Real-iff hermitian-quadratic-form-real less-complex-def*

**by** *auto*

```

  show 2: positive-definite A  $\implies$  sylvester-criterion A
    by (rule sylvester-criterion-reverse)
qed
end

```

## 26.4 Theorem Statement (Outside Locale)

```

theorem sylvester-criterion:
  fixes A :: complex mat
  assumes hermitian A
  shows sylvester-criterion A  $\longleftrightarrow$  positive-definite A
  using assms cmplx-herm-mat.sylvester-criterion cmplx-herm-mat.intro hermi-
  tian-square by blast

```

## 27 Code

```

definition rat-of-nat-mat :: nat mat  $\Rightarrow$  rat mat where
  rat-of-nat-mat A = mat (dim-row A) (dim-col A) ( $\lambda(i,j).$  rat-of-nat (A$$$ $(i,j)$ ))

```

```

definition rat-of-int-mat :: int mat  $\Rightarrow$  rat mat where
  rat-of-int-mat A = mat (dim-row A) (dim-col A) ( $\lambda(i,j).$  rat-of-int (A$$$ $(i,j)$ ))

```

```

context comm-ring-hom
begin

```

```

lemma mat-hom-submatrix:
  assumes I: I  $\subseteq$  {.. $dim$ -row A}
  assumes J: J  $\subseteq$  {.. $dim$ -col A}
  shows submatrix (math A) I J = math (submatrix A I J)
  unfolding submatrix-inj[of A  $dim$ -row A  $dim$ -col A, simplified, OF I J]
  unfolding submatrix-inj[of (math A)  $dim$ -row A  $dim$ -col A, simplified, OF I J]
  apply (simp add: submatrix-of-inj-def map-mat-def)
  apply (rule cong-mat[of card I card I card J card J, simplified])
  using I J
  by (simp add: subset-iff pick-in-set-le)

```

```

lemma mat-hom-carrier: A  $\in$  carrier-mat m n  $\implies$  math A  $\in$  carrier-mat m n
  by simp

```

```

end

```

```

lemma of-rat-less-complex: x < y  $\longleftrightarrow$  complex-of-rat x < complex-of-rat y
proof -
  have complex-of-rat x = (of-real  $\circ$  of-rat) x complex-of-rat y = (of-real  $\circ$  of-rat)
  y
  by (simp-all add: of-rat.rep-eq)
  moreover have real-of-rat x < real-of-rat y  $\longleftrightarrow$  x < y using of-rat-less by blast

```

**ultimately show** *?thesis* **by** (*simp add: less-complex-def*)  
**qed**

**lemma** *sylveste-r-criter-ion-rat*:

**assumes** *square*:  $A \in \text{carrier-mat } n \ n$

**defines** [*simp*]:  $(\text{hom}_H :: \text{rat mat} \Rightarrow \text{complex mat}) \equiv \text{of-rat-hom.mat-hom}$

**shows** *sylveste-r-criter-ion*  $A \longleftrightarrow \text{sylveste-r-criter-ion } (\text{hom}_H A)$

**unfolding** *sylveste-r-criter-ion-def*

**proof** –

**have**  $\forall k \leq \text{dim-row } A. 0 < \text{Determinant.det } (\text{lps } A \ k) \longleftrightarrow 0 < \text{Determinant.det } (\text{lps } (\text{hom}_H A) \ k)$

**proof** *clarify*

**fix**  $k$  **assume**  $k: k \leq \text{dim-row } A$

**have**  $\text{Determinant.det } (\text{lps } (\text{hom}_H A) \ k) = \text{complex-of-rat } (\text{Determinant.det } (\text{lps } A \ k))$

**using** *of-rat-hom.hom-det*[*of lps A k, where 'a = complex*]

**unfolding** *hom\_H-def leading-principal-submatrix-def*

**apply** (*subst of-rat-hom.mat-hom-submatrix*[*of {..<k} A {..<k}*]])

**using** *k square*

**by** *simp-all*

**thus**  $0 < \text{Determinant.det } (\text{lps } A \ k) \longleftrightarrow 0 < \text{Determinant.det } (\text{lps } (\text{hom}_H A) \ k)$

**using** *of-rat-less-complex k hom\_H-def* **by** (*metis of-rat-hom.hom-zero*)

**qed**

**thus**  $(\forall k \leq \text{dim-row } A. 0 < \text{Determinant.det } (\text{lps } A \ k)) = (\forall k \leq \text{dim-row } (\text{hom}_H A). 0 < \text{Determinant.det } (\text{lps } (\text{hom}_H A) \ k))$

**by** *simp*

**qed**

**lemma** *rat-subset-real*:  $\mathbb{Q} \subseteq (\mathbb{R} :: \text{complex set})$

**by** (*metis Im-complex-of-real Rats-cases' complex-is-Real-iff of-real-divide of-real-of-int-eq subset-eq*)

**lemma** *conjugate-of-rat*:  $\text{conjugate } (\text{of-rat } x) = ((\text{of-rat } x) :: \text{complex})$

**proof** –

**have**  $((\text{of-rat } x) :: \text{complex}) \in \mathbb{R}$  **using** *rat-subset-real* **by** *fastforce*

**moreover** **have**  $\forall x \in \mathbb{R} :: \text{complex set}. \text{conjugate } x = x$  **using** *Reals-cnj-iff* **by** *force*

**ultimately show** *?thesis* **by** *fast*

**qed**

**lemma** *hermitian-rat*:

**fixes**  $A :: \text{rat mat}$

**shows** *hermitian*  $A \longleftrightarrow \text{hermitian } ((\text{of-rat-hom.mat-hom } A) :: \text{complex mat})$

**apply** *standard*

**unfolding** *hermitian-ij-ji-iff*

**apply** (*simp add: hermitian-ij-ji-iff conjugate-of-rat*)

**subgoal** **using** *conjugate-of-rat* **by** (*metis conjugate-complex-def*)

**subgoal** **by** (*metis conjugate-of-rat conjugate-rat-def index-map-mat(1,2,3) of-rat-eq-iff*)

*square-mat.simps*)  
**done**

**definition** *compute-lps* :: 'a mat  $\Rightarrow$  nat  $\Rightarrow$  'a mat **where**  
[*code*]: *compute-lps* A k = mat k k ( $\lambda(i,j). A\$\$(i,j)$ )

**definition** *compute-sylvester-criterion* :: rat mat  $\Rightarrow$  bool **where**  
[*code*]: *compute-sylvester-criterion* A  $\longleftrightarrow$  ( $\forall k \in \{0..dim\text{-row } A\}. det\text{-field} (compute\text{-lps } A \ k) > 0$ )

**lemma** *compute-lps-correct*:

**assumes** A: A  $\in$  carrier-mat n n

**assumes** k: k  $\leq$  n

**shows** *compute-lps* A k = *lps* A k

**apply** (*rule eq-matI*)

**unfolding** *compute-lps-def*

**subgoal**

**using** k A

**apply** (*simp add: submatrix-def*)

**by** (*metis Collect-conj-eq card-lessThan index-mat(1) inf.absorb-iff2 lessThan-def*

*lessThan-subset-iff*

*pick-n-le*)

**subgoal using** k A **by** (*metis carrier-matD(1) dim-row-mat(1) leading-principal-submatrix-carrier*)

**subgoal using** k A **by** (*metis carrier-matD(2) dim-col-mat(1) leading-principal-submatrix-carrier*)

**done**

**lemma** *compute-sylvester-criterion-correct*:

**fixes** A :: rat mat

**assumes** A: *hermitian* A

**shows** *compute-sylvester-criterion* A  $\longleftrightarrow$  *sylvester-criterion* A

**unfolding** *compute-sylvester-criterion-def sylvester-criterion-def*

**using** *compute-lps-correct[OF hermitian-square[OF A]]*

**by** *auto*

**lemma** *compute-sylvester-criterion-correct'*:

**fixes** A :: rat mat

**assumes** A: *hermitian* A

**assumes** square: A  $\in$  carrier-mat n n

**shows** *compute-sylvester-criterion* A  $\longleftrightarrow$  *positive-definite* ((*of-rat-hom.mat-hom* A)::*complex mat*)

**apply** (*subst compute-sylvester-criterion-correct[OF A]*)

**apply** (*subst sylvester-criterion-rat[OF square]*)

**by** (*rule sylvester-criterion*

[ *of of-rat-hom.mat-hom A,*

*unfolded hermitian-rat[of A, symmetric],*

*OF A]*)

**definition** *smallest-eigval-gr* :: rat mat  $\Rightarrow$  rat  $\Rightarrow$  bool **where**

[*code*]: *smallest-eigval-gr* A r = (*compute-sylvester-criterion* (A - (r  $\cdot$  *m* 1<sub>m</sub>

(*dim-row A*)))

The function *smallest-eigval-gr* checks if the smallest eigenvalue of  $A$  is bigger than  $r$ .

**lemma** *hermitian-minus-I*:  
**fixes**  $A :: \text{rat mat}$   
**assumes**  $A: \text{hermitian } A$   
**shows**  $\text{hermitian } (A - r \cdot_m 1_m (\text{dim-row } A))$   
**unfolding** *hermitian-def adjoint-def*  
**apply** (*rule eq-matI*)  
**apply** *simp-all*  
**using**  $A[\text{unfolded } \text{hermitian-def adjoint-def}, \text{simplified}]$   
**by** (*metis index-transpose-mat(1,3) transpose-mat-def*)

**context** *comm-ring-hom*  
**begin**

**lemma** *mat-hom-add*:  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**assumes**  $B \in \text{carrier-mat } m \ n$   
**shows**  $\text{mat}_h (A + B) = \text{mat}_h A + \text{mat}_h B$   
**using** *assms hom-add by auto*

**lemma** *mat-hom-uminus*:  
**shows**  $\text{mat}_h (-A) = - \text{mat}_h A$   
**using** *hom-uminus by auto*

**lemma** *mat-hom-sub*:  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**assumes**  $B \in \text{carrier-mat } m \ n$   
**shows**  $\text{mat}_h (A - B) = \text{mat}_h A - \text{mat}_h B$   
**using** *assms mat-hom-add mat-hom-uminus*  
**by** (*smt (verit, cfv-threshold) add-uminus-minus-mat comm-ring-hom.mat-hom-add comm-ring-hom-axioms map-carrier-mat uminus-carrier-mat*)

**end**

**lemma** *smallest-eigval-gr-correct-aux*:  
**fixes**  $A :: \text{rat mat}$   
**assumes**  $A: \text{hermitian } A$   
**shows**  $\text{smallest-eigval-gr } A \ r \longleftrightarrow \text{positive-definite } ((\text{of-rat-hom.mat-hom } (A - r \cdot_m 1_m (\text{dim-row } A)))::\text{complex mat})$   
**unfolding** *smallest-eigval-gr-def*  
**by** (*rule compute-sylvester-criterion-correct'[OF hermitian-minus-I[OF A]], fast-force*)

**lemma** *smallest-eigval-gr-correct*:  
**fixes**  $A :: \text{rat mat}$   
**assumes**  $A: \text{hermitian } A$

```

shows smallest-eigval-gr  $A$   $r \iff (\forall \mu :: \text{complex} \in \text{spectrum } (\text{of-rat-hom.mat-hom } A)). r < \mu$ 
proof –
  let  $?A' = A - r \cdot_m 1_m (\text{dim-row } A)$ 
  let  $?cA' = (\text{of-rat-hom.mat-hom } ?A') :: \text{complex mat}$ 
  let  $?cA = (\text{of-rat-hom.mat-hom } A) :: \text{complex mat}$ 
  have  $A'$ -square: square-mat  $?A'$  using hermitian-is-square[OF  $A$ ] hermitian-minus-I
by auto
  have smallest-eigval-gr  $A$   $r \iff \text{positive-definite } ?cA'$ 
    by (rule smallest-eigval-gr-correct-aux[OF  $A$ , of  $r$ ])
  also have  $\dots \iff (\forall \mu \in \text{spectrum } ?cA'. \mu > 0)$ 
    using cmplx-herm-mat.positive-definite-imp-positive-eigenvalues[OF cmplx-herm-mat.intro,
of  $?cA'$ ]
    using cmplx-herm-mat.positive-eigenvalues-imp-positive-definite[OF cmplx-herm-mat.intro,
of  $?cA'$ ]
    using  $A$ 
    by (meson hermitian-minus-I hermitian-rat hermitian-square)
  also have  $\dots \iff (\forall \mu \in \text{spectrum } ?cA. r < \mu)$ 
proof –
  obtain  $n$  where  $n: ?cA \in \text{carrier-mat } n$ 
    using  $A$  hermitian-rat hermitian-square by blast
  have  $?cA' = \text{of-rat-hom.mat-hom } A - \text{of-rat-hom.mat-hom } (r \cdot_m 1_m (\text{dim-row } A))$ 
  using  $A$  of-rat-hom.mat-hom-sub[of  $A$  dim-row  $A$  dim-row  $A$   $r \cdot_m 1_m (\text{dim-row } A)$ ]
  by (metis hermitian-square one-carrier-mat smult-carrier-mat)
  also have  $\dots = \text{of-rat-hom.mat-hom } A - r \cdot_m \text{of-rat-hom.mat-hom } (1_m (\text{dim-row } A))$ 
  by fastforce
  finally have  $\mu \in \text{spectrum } ?cA \iff \mu - r \in \text{spectrum } ?cA'$  for  $\mu$ 
    using spectrum-shift[OF  $n$ , of  $r$ ]  $n$  by (simp add: of-rat-hom.mat-hom-one)
  thus  $?thesis$  by (metis diff-gt-0-iff-gt add-diff-cancel-right' diff-gt-0-iff-gt)
qed
finally show  $?thesis$  .
qed

```

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.