

# Simultaneous diagonalization of pairwise commuting Hermitian matrices

Mnacho Echenim

February 6, 2026

## Abstract

A Hermitian matrix is a square complex matrix  $A$  that is equal to its conjugate transpose  $A^\dagger$ . The (finite-dimensional) spectral theorem states that for any such matrix  $A$ , we have the equality  $A = U \cdot B \cdot U^\dagger$ , where  $U$  is a unitary matrix and  $B$  is a diagonal matrix containing only real elements. We formalize the generalization of this result, which states that if  $\{A_1, \dots, A_n\}$  are Hermitian and pairwise commuting matrices, then there exists a unitary matrix  $U$  such that  $A_i = U \cdot B_i \cdot U^\dagger$ , for  $i = 1, \dots, n$ , and each  $B_i$  is diagonal and contains only real elements. Sets of pairwise commuting Hermitian matrices are called *Complete Sets of Commuting Observables* in Quantum Mechanics, where they represent physical quantities that can be simultaneously measured to uniquely distinguish quantum states.

## Contents

<b>1</b>	<b>Some preliminary results</b>	<b>1</b>
1.1	Roots of a polynomial . . . . .	1
1.2	Linear algebra preliminaries . . . . .	2
<b>2</b>	<b>Properties of the spectrum of a matrix</b>	<b>6</b>
2.1	Results on diagonal matrices . . . . .	6
2.2	Unitary equivalence . . . . .	13
2.3	On the spectrum of a matrix . . . . .	25
<b>3</b>	<b>Properties of the inner product</b>	<b>28</b>
3.1	Some analysis complements . . . . .	28
3.2	Inner product results . . . . .	32
<b>4</b>	<b>Matrix decomposition</b>	<b>38</b>
<b>5</b>	<b>Additional results on block decompositions of matrices</b>	<b>47</b>
5.1	Split block results . . . . .	47
5.2	Diagonal block matrices . . . . .	53

<b>6</b>	<b>Block matrix decomposition</b>	<b>70</b>
6.1	Subdiagonal extraction . . . . .	70
6.2	Predicates on diagonal block matrices . . . . .	77
6.3	Counting similar neighbours in a list . . . . .	82
<b>7</b>	<b>Sorted hermitian decomposition</b>	<b>96</b>
<b>8</b>	<b>Commuting Hermitian families</b>	<b>106</b>
8.1	Intermediate properties . . . . .	106
8.2	The main result . . . . .	118

**Acknowledgments** This work was partially supported by Agence Nationale de la Recherche, through *Plan France 2030 (ref. ANR-22-PETQ-0007)*.

**theory** *Spectral-Theory-Complements* **imports** *HOL-Combinatorics.Permutations*  
*Projective-Measurements.Linear-Algebra-Complements*  
*Projective-Measurements.Projective-Measurements*

**begin**

## 1 Some preliminary results

### 1.1 Roots of a polynomial

Results on polynomials, the main one being that the set of roots of a polynomial is uniquely defined.

**lemma** *root-poly-linear*:

**shows**  $\text{poly} (\prod a \leftarrow L. [- a, 1:]) (c :: 'a :: \text{field}) = 0 \implies c \in \text{set } L$

**proof** (*induct L*)

**case** *Nil*

**thus** *?case* **using** *Nil* **by** *simp*

**next**

**case** (*Cons a L*)

**show** *?case*

**proof** (*cases poly* ( $\prod a \leftarrow L. [- a, 1:]$ )  $c = 0$ )

**case** *True*

**then show** *?thesis* **using** *Cons* **by** *auto*

**next**

**case** *False*

**hence**  $\text{poly} [- a, 1:] c = 0$  **using** *Cons* **by** *auto*

**hence**  $a = c$  **by** *auto*

**thus** *?thesis* **by** *auto*

**qed**

**qed**

**lemma** *poly-root-set-subseteq*:

**assumes**  $(\prod (a::'a::field)\leftarrow L. [- a, 1:]) = (\prod a\leftarrow M. [- a, 1:])$

**shows**  $set L \subseteq set M$

**proof**

**fix**  $x$

**assume**  $x \in set L$

**hence**  $poly (\prod (a::'a::field)\leftarrow L. [- a, 1:]) x = 0$  **using** *linear-poly-root[of x]* **by** *simp*

**hence**  $poly (\prod (a::'a::field)\leftarrow M. [- a, 1:]) x = 0$  **using** *assms* **by** *simp*

**thus**  $x \in set M$  **using** *root-poly-linear[of M]* **by** *simp*

**qed**

**lemma** *poly-root-set-eq*:

**assumes**  $(\prod (a::'a::field)\leftarrow L. [- a, 1:]) = (\prod a\leftarrow M. [- a, 1:])$

**shows**  $set L = set M$  **using** *assms* *poly-root-set-subseteq*

**by** (*simp* *add: poly-root-set-subseteq equalityI*)

## 1.2 Linear algebra preliminaries

**lemma** *minus-zero-vec-eq*:

**fixes**  $v::'a::\{ab-group-add\}$  *Matrix.vec*

**assumes**  $dim-vec v = n$

**and**  $dim-vec w = n$

**and**  $v - w = 0_v n$

**shows**  $v = w$

**proof** –

**have**  $v = v - w + w$  **using** *assms*

**by** (*metis carrier-vec-dim-vec comm-add-vec left-zero-vec  
minus-add-minus-vec minus-cancel-vec uminus-eq-vec  
zero-minus-vec*)

**also have**  $\dots = 0_v n + w$  **using** *assms* **by** *simp*

**also have**  $\dots = w$  **using** *assms* *left-zero-vec[of w n]*

**by** (*metis carrier-vec-dim-vec*)

**finally show** *?thesis* .

**qed**

**lemma** *right-minus-zero-mat*:

**fixes**  $A::'a::\{group-add\}$  *Matrix.mat*

**shows**  $A - 0_m (dim-row A) (dim-col A) = A$

**by** (*intro eq-matI, auto*)

**lemma** *smult-zero*:

**shows**  $(0::'a::comm-ring) \cdot_m A = 0_m (dim-row A) (dim-col A)$  **by** *auto*

**lemma** *rank-1-proj-col-carrier*:

**assumes**  $i < dim-col A$

**shows**  $rank-1-proj (Matrix.col A i) \in carrier-mat (dim-row A) (dim-row A)$

**proof** –

**have**  $dim-vec (Matrix.col A i) = dim-row A$  **by** *simp*

**thus** *?thesis* **by** (*metis rank-1-proj-carrier*)  
**qed**

**lemma** *zero-adjoint*:

**shows** *Complex-Matrix.adjoint* ( $0_m$   $n$   $m$ ) = ( $(0_m$   $m$   $n$ ):: '*a*::*conjugatable-field* *Matrix.mat*)  
**by** (*rule eq-matI*, (*auto simp add: adjoint-eval*))

**lemma** *assoc-mat-mult-vec'*:

**assumes**  $A \in$  *carrier-mat*  $n$   $n$   
**and**  $B \in$  *carrier-mat*  $n$   $n$   
**and**  $C \in$  *carrier-mat*  $n$   $n$   
**and**  $v \in$  *carrier-vec*  $n$   
**shows**  $A * B * C *_v v = A *_v (B *_v (C *_v v))$  **using** *assms*  
**by** (*smt (verit) assoc-mult-mat-vec mult-carrier-mat mult-mat-vec-carrier*)

**lemma** *adjoint-dim'*:

$A \in$  *carrier-mat*  $n$   $m \implies$  *Complex-Matrix.adjoint*  $A \in$  *carrier-mat*  $m$   $n$   
**using** *adjoint-dim-col adjoint-dim-row* **by** *blast*

**definition** *mat-conj* **where**

*mat-conj*  $U$   $V = U * V * (Complex-Matrix.adjoint U)$

**lemma** *mat-conj-adjoint*:

**shows** *mat-conj* (*Complex-Matrix.adjoint*  $U$ )  $V =$   
*Complex-Matrix.adjoint*  $U * V * U$  **unfolding** *mat-conj-def*  
**by** (*simp add: Complex-Matrix.adjoint-adjoint*)

**lemma** *map2-mat-conj-exp*:

**assumes**  $length A = length B$   
**shows**  $map2 (*) (map2 (*) A B) (map Complex-Matrix.adjoint A) =$   
 $map2 mat-conj A B$  **using** *assms*  
**proof** (*induct A arbitrary: B*)  
**case** *Nil*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*Cons a A*)  
**hence**  $0 < length B$  **by** *auto*  
**hence**  $B = hd B \# (tl B)$  **by** *simp*  
**hence**  $length (tl B) = length A$  **using** *Cons* **by** *simp*  
**have**  $map2 (*) (map2 (*) (a \# A) B) (map Complex-Matrix.adjoint (a \# A))$   
 $=$   
 $a * hd B * Complex-Matrix.adjoint a \#$   
 $map2 (*) (map2 (*) A (tl B)) (map Complex-Matrix.adjoint A)$   
**by** (*metis (no-types, lifting) <B = hd B \# tl B> list.map(2)*)  
*split-conv zip-Cons-Cons*)  
**also have**  $\dots = mat-conj a (hd B) \# map2 mat-conj A (tl B)$   
**using** *Cons <length (tl B) = length A>*  
**unfolding** *mat-conj-def*

by *presburger*  
 also have ... = map2 mat-conj (a#A) B using ⟨B = hd B # (tl B)⟩  
 by (metis (no-types, opaque-lifting) list.map(2) prod.simps(2)  
 zip-Cons-Cons)  
 finally show ?case .  
 qed

lemma *mat-conj-unit-commute*:

assumes unitary U  
 and  $U * A = A * U$   
 and  $A \in \text{carrier-mat } n \ n$   
 and  $U \in \text{carrier-mat } n \ n$   
 shows mat-conj U A = A  
 proof –  
 have mat-conj U A =  $A * U * \text{Complex-Matrix.adjoint } U$  using *assms*  
 unfolding mat-conj-def by simp  
 also have ... =  $A * (U * \text{Complex-Matrix.adjoint } U)$   
 proof (rule *assoc-mult-mat*, auto simp add: *assms*)  
 show  $U \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } U)$   
 using *assms*(3) *assms*(4) by auto  
 qed  
 also have ... = A using *assms* by simp  
 finally show ?thesis .  
 qed

lemma *hermitian-mat-conj*:

assumes  $A \in \text{carrier-mat } n \ n$   
 and  $U \in \text{carrier-mat } n \ n$   
 and hermitian A  
 shows hermitian (mat-conj U A)  
 proof –  
 have  $\text{Complex-Matrix.adjoint } (U * A * \text{Complex-Matrix.adjoint } U) =$   
 $U * \text{Complex-Matrix.adjoint } (U * A)$   
 by (metis (no-types, lifting) *Complex-Matrix.adjoint-adjoint adjoint-dim'*  
*adjoint-mult assms*(1) *assms*(2) *mult-carrier-mat*)  
 also have ... =  $U * ((\text{Complex-Matrix.adjoint } A) * \text{Complex-Matrix.adjoint } U)$   
 by (metis *adjoint-mult assms*(1) *assms*(2))  
 also have ... =  $U * A * \text{Complex-Matrix.adjoint } U$   
 by (metis *adjoint-dim' assms assoc-mult-mat hermitian-def*)  
 finally show ?thesis unfolding *hermitian-def mat-conj-def* .  
 qed

lemma *hermitian-mat-conj'*:

assumes  $A \in \text{carrier-mat } n \ n$   
 and  $U \in \text{carrier-mat } n \ n$   
 and hermitian A  
 shows hermitian (mat-conj (Complex-Matrix.adjoint U) A)  
 by (metis *Complex-Matrix.adjoint-adjoint adjoint-dim-col assms*  
*carrier-matD*(1) *carrier-matD*(2) *carrier-mat-triv hermitian-mat-conj*)

**lemma** *mat-conj-uminus-eq*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A = \text{mat-conj } U \ B$   
**shows**  $-A = \text{mat-conj } U \ (-B)$  **using** *assms unfolding mat-conj-def* **by** *auto*

**lemma** *mat-conj-smult*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A = U * B * (\text{Complex-Matrix.adjoint } U)$   
**shows**  $x \cdot_m A = U * (x \cdot_m B) * (\text{Complex-Matrix.adjoint } U)$  **using** *assms mult-smult-distrib*  
**by** (*smt (verit) adjoint-dim' mult-carrier-mat mult-smult-assoc-mat*)

**lemma** *mult-adjoint-hermitian*:  
**fixes**  $A::'a::\text{conjugatable-field Matrix.mat}$   
**assumes**  $A \in \text{carrier-mat } n \ m$   
**shows** *hermitian*  $((\text{Complex-Matrix.adjoint } A) * A)$  **unfolding** *hermitian-def*  
**proof** –  
**define**  $C$  **where**  $C = (\text{Complex-Matrix.adjoint } A) * A$   
**have**  $\text{Complex-Matrix.adjoint } C =$   
 $\text{Complex-Matrix.adjoint } A * \text{Complex-Matrix.adjoint } (\text{Complex-Matrix.adjoint } A)$   
**using** *adjoint-mult assms C-def* **by** (*metis adjoint-dim' assms*)  
**also have**  $\dots = \text{Complex-Matrix.adjoint } A * A$  **using** *assms*  
**by** (*simp add: Complex-Matrix.adjoint-adjoint*)  
**finally show**  $\text{Complex-Matrix.adjoint } C = C$  **using** *C-def* **by** *simp*  
**qed**

**lemma** *hermitian-square-hermitian*:  
**fixes**  $A::'a::\text{conjugatable-field Matrix.mat}$   
**assumes** *hermitian*  $A$   
**shows** *hermitian*  $(A * A)$   
**proof** –  
**have**  $\text{Complex-Matrix.adjoint } (A * A) = \text{Complex-Matrix.adjoint } A * (\text{Complex-Matrix.adjoint } A)$   
**using** *adjoint-mult* **by** (*metis assms hermitian-square*)  
**also have**  $\dots = A * A$  **using** *assms unfolding hermitian-def* **by** *simp*  
**finally show** *?thesis* **unfolding** *hermitian-def* .  
**qed**

## 2 Properties of the spectrum of a matrix

### 2.1 Results on diagonal matrices

**lemma** *diagonal-mat-uminus*:

**fixes**  $A::'a::\{\text{ring}\}$  *Matrix.mat*  
**assumes** *diagonal-mat A*  
**shows** *diagonal-mat (-A)* **using** *assms unfolding diagonal-mat-def uminus-mat-def*  
**by** *auto*

**lemma** *diagonal-mat-smult:*  
**fixes**  $A::'a::\{\text{ring}\}$  *Matrix.mat*  
**assumes** *diagonal-mat A*  
**shows** *diagonal-mat (x ·<sub>m</sub>A)* **using** *assms unfolding diagonal-mat-def uminus-mat-def* **by** *auto*

**lemma** *diagonal-imp-upper-triangular:*  
**assumes** *diagonal-mat A*  
**and**  $A \in \text{carrier-mat } n \ n$   
**shows** *upper-triangular A* **unfolding** *upper-triangular-def*  
**proof** (*intro allI impI*)  
**fix**  $i \ j$   
**assume**  $i < \text{dim-row } A$  **and**  $j < i$   
**hence**  $j < \text{dim-col } A$   $j \neq i$  **using** *assms* **by** *auto*  
**thus**  $A \ \$\$ (i,j) = 0$  **using** *assms*  $\langle i < \text{dim-row } A \rangle$  **unfolding** *diagonal-mat-def*  
**by** *simp*  
**qed**

**lemma** *set-diag-mat-uminus:*  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**shows**  $\text{set } (\text{diag-mat } (-A)) = \{-a \mid a. a \in \text{set } (\text{diag-mat } A)\}$  (**is**  $?L = ?R$ )  
**proof**  
**show**  $?L \subseteq ?R$   
**proof**  
**fix**  $x$   
**assume**  $x \in \text{set } (\text{diag-mat } (-A))$   
**hence**  $\exists i < \text{length } (\text{diag-mat } (-A)). \text{nth } (\text{diag-mat } (-A)) \ i = x$   
**using** *in-set-conv-nth[of x]* **by** *simp*  
**from this obtain**  $i$  **where**  $i < \text{length } (\text{diag-mat } (-A))$  **and**  $\text{nth } (\text{diag-mat } (-A)) \ i = x$   
**by** *auto* **note**  $i\text{prop} = \text{this}$   
**hence**  $i < \text{dim-row } (-A)$  **unfolding** *diag-mat-def* **by** *simp*  
**hence**  $i < n$  **using** *assms* **by** *simp*  
**have**  $x = (-A)\ \$\$ (i,i)$  **using**  $i\text{prop}$  **unfolding** *diag-mat-def* **by** *simp*  
**also have**  $\dots = -A \ \$\$ (i,i)$  **using**  $\langle i < n \rangle$  *assms* **unfolding** *uminus-mat-def*  
**by** *auto*  
**also have**  $\dots \in ?R$  **using**  $i\text{prop}$  *assms*  $\langle i < n \rangle$   
*in-set-conv-nth[of A \$\$ (i,i)]* **by** (*metis (mono-tags, lifting) carrier-matD(1) diag-elems-mem diag-elems-set-diag-mat mem-Collect-eq*)  
**finally show**  $x \in ?R$  .  
**qed**  
**next**  
**show**  $?R \subseteq ?L$

**proof**  
**fix**  $x$   
**assume**  $x \in ?R$   
**hence**  $\exists i < \text{length}(\text{diag-mat } A). \neg(\text{nth}(\text{diag-mat } A) \ i) = x$   
**using**  $\text{in-set-conv-nth}[\text{of } x]$  **by**  $(\text{smt}(\text{verit}) \ \text{in-set-conv-nth} \ \text{mem-Collect-eq})$   
**from this obtain**  $i$  **where**  $i < \text{length}(\text{diag-mat } A)$  **and**  $\neg(\text{nth}(\text{diag-mat } A) \ i) = x$   
**by auto note**  $\text{iprop} = \text{this}$   
**hence**  $i < \text{dim-row}(-A)$  **unfolding**  $\text{diag-mat-def}$  **by**  $\text{simp}$   
**hence**  $i < n$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**have**  $x = -A\$(i,i)$  **using**  $\text{iprop}$  **unfolding**  $\text{diag-mat-def}$  **by**  $\text{simp}$   
**also have**  $\dots = (-A)\$(i,i)$  **using**  $\langle i < n \rangle \text{assms}$  **unfolding**  $\text{uminus-mat-def}$   
**by auto**  
**also have**  $\dots \in ?L$  **using**  $\text{iprop}$   $\text{assms}$   $\langle i < n \rangle$   
 $\text{in-set-conv-nth}[\text{of } A\$(i,i)]$   
**by**  $(\text{metis} \ \langle i < \text{dim-row}(-A) \rangle \ \text{diag-elems-mem} \ \text{diag-elems-set-diag-mat})$   
**finally show**  $x \in ?L$  .  
**qed**  
**qed**

**lemma**  $\text{set-diag-mat-smult}$ :

**assumes**  $A \in \text{carrier-mat } n \ n$   
**shows**  $\text{set}(\text{diag-mat}(x \cdot_m A)) = \{x * a \mid a. a \in \text{set}(\text{diag-mat } A)\}$  **(is**  $?L = ?R)$

**proof**

**show**  $?L \subseteq ?R$

**proof**

**fix**  $b$

**assume**  $b \in \text{set}(\text{diag-mat}(x \cdot_m A))$

**hence**  $\exists i < \text{length}(\text{diag-mat}(x \cdot_m A)). \text{nth}(\text{diag-mat}(x \cdot_m A)) \ i = b$

**using**  $\text{in-set-conv-nth}[\text{of } b]$  **by**  $\text{simp}$

**from this obtain**  $i$  **where**  $i < \text{length}(\text{diag-mat}(x \cdot_m A))$  **and**  $\text{nth}(\text{diag-mat}(x \cdot_m A)) \ i = b$

**by auto note**  $\text{iprop} = \text{this}$

**hence**  $i < \text{dim-row}(x \cdot_m A)$  **unfolding**  $\text{diag-mat-def}$  **by**  $\text{simp}$

**hence**  $i < n$  **using**  $\text{assms}$  **by**  $\text{simp}$

**have**  $b = (x \cdot_m A)\$(i,i)$  **using**  $\text{iprop}$  **unfolding**  $\text{diag-mat-def}$  **by**  $\text{simp}$

**also have**  $\dots = x * A\$(i,i)$  **using**  $\langle i < n \rangle \text{assms}$  **unfolding**  $\text{uminus-mat-def}$

**by auto**

**also have**  $\dots \in ?R$  **using**  $\text{iprop}$   $\text{assms}$   $\langle i < n \rangle$

$\text{in-set-conv-nth}[\text{of } A\$(i,i)]$

**by**  $(\text{metis}(\text{mono-tags}, \text{lifting}) \ \text{carrier-matD}(1) \ \text{diag-elems-mem} \ \text{diag-elems-set-diag-mat}$

$\text{mem-Collect-eq})$

**finally show**  $b \in ?R$  .

**qed**

**next**

**show**  $?R \subseteq ?L$

**proof**

**fix**  $b$

**assume**  $b \in ?R$   
**hence**  $\exists i < \text{length } (\text{diag-mat } A). x * (\text{nth } (\text{diag-mat } A) \ i) = b$   
**using** *in-set-conv-nth*[of  $x$ ] **by** (*smt* (*verit*) *in-set-conv-nth mem-Collect-eq*)  
**from this obtain**  $i$  **where**  $i < \text{length } (\text{diag-mat } A)$  **and**  $x * (\text{nth } (\text{diag-mat } A) \ i) = b$   
**by auto note** *iprop = this*  
**hence**  $i < \text{dim-row } (x \cdot_m A)$  **unfolding** *diag-mat-def* **by** *simp*  
**hence**  $i < n$  **using** *assms* **by** *simp*  
**have**  $b = x * A \$\$ (i, i)$  **using** *iprop* **unfolding** *diag-mat-def* **by** *simp*  
**also have**  $\dots = (x \cdot_m A) \$\$ (i, i)$  **using**  $\langle i < n \rangle$  *assms* **unfolding** *uminus-mat-def*  
**by auto**  
**also have**  $\dots \in ?L$  **using** *iprop* *assms*  $\langle i < n \rangle$   
*in-set-conv-nth*[of  $A \$\$ (i, i)$ ]  
**by** (*metis*  $\langle i < \text{dim-row } (x \cdot_m A) \rangle$  *diag-elems-mem diag-elems-set-diag-mat*)  
**finally show**  $b \in ?L$  .  
**qed**  
**qed**

**lemma** *diag-mat-diagonal-eq*:

**assumes**  $\text{diag-mat } A = \text{diag-mat } B$   
**and** *diagonal-mat*  $A$   
**and** *diagonal-mat*  $B$   
**and**  $\text{dim-col } A = \text{dim-col } B$   
**shows**  $A = B$   
**proof**  
**show**  $c: \text{dim-col } A = \text{dim-col } B$  **using** *assms* **by** *simp*  
**show**  $r: \text{dim-row } A = \text{dim-row } B$  **using** *assms* **unfolding** *diag-mat-def*  
**proof** –  
**assume**  $\text{map } (\lambda i. A \$\$ (i, i)) [0..<\text{dim-row } A] = \text{map } (\lambda i. B \$\$ (i, i)) [0..<\text{dim-row } B]$   
**then show** *?thesis*  
**by** (*metis* (*lifting*) *length-map length-upt verit-minus-simplify(2)*)  
**qed**  
**fix**  $i \ j$   
**assume**  $i < \text{dim-row } B$  **and**  $j < \text{dim-col } B$   
**show**  $A \$\$ (i, j) = B \$\$ (i, j)$   
**proof** (*cases*  $i = j$ )  
**case** *False*  
**thus** *?thesis* **using** *assms*  $c \ r$  **unfolding** *diagonal-mat-def*  
**by** (*simp* *add*:  $\langle \text{dim-row } A = \text{dim-row } B \rangle \langle i \neq j \rangle \langle i < \text{dim-row } B \rangle \langle j < \text{dim-col } B \rangle$ )  
**next**  
**case** *True*  
**hence**  $A \$\$ (i, j) = A \$\$ (i, i)$  **by** *simp*  
**also have**  $\dots = (\text{diag-mat } A)!i$  **using**  $c \ r \ \langle i < \text{dim-row } B \rangle$  **unfolding** *diag-mat-def*  
**by** *simp*  
**also have**  $\dots = (\text{diag-mat } B)!i$  **using** *assms* **by** *simp*  
**also have**  $\dots = B \$\$ (i, i)$  **using**  $c \ r \ \langle i < \text{dim-row } B \rangle$  **unfolding** *diag-mat-def*

by *simp*  
 also have ... =  $B \$(i,j)$  using *True* by *simp*  
 finally show  $A \$(i,j) = B \$(i,j)$  .  
 qed  
 qed

lemma *diag-elems-ne*:  
 assumes  $B \in \text{carrier-mat } n \ n$   
 and  $0 < n$   
 shows *diag-elems*  $B \neq \{\}$   
 proof -  
 have  $B \$(0,0) \in \text{diag-elems } B$  using *assms* by *simp*  
 thus ?thesis by *auto*  
 qed

lemma *diagonal-mat-mult-vec*:  
 fixes  $B::'a::\text{conjugatable-field } \text{Matrix.mat}$   
 assumes *diagonal-mat*  $B$   
 and  $B \in \text{carrier-mat } n \ n$   
 and  $v \in \text{carrier-vec } n$   
 and  $i < n$   
 shows *vec-index*  $(B *_v v) i = B \$(i,i) * (\text{vec-index } v i)$   
 proof -  
 have *vec-index*  $(B *_v v) i = \text{Matrix.scalar-prod } (\text{Matrix.row } B i) \ v$  using  
*mult-mat-vec-def* *assms*  
 by *simp*  
 also have ... =  $(\sum j \in \{0 ..< n\}. \text{vec-index } (\text{Matrix.row } B i) j * (\text{vec-index } v j))$   
 using *Matrix.scalar-prod-def* *assms(3)* *carrier-vecD* by *blast*  
 also have ... =  $(\sum j \in \{0 ..< n\}. B \$(i,j) * (\text{vec-index } v j))$   
 proof -  
 have  $\bigwedge j. j < n \implies \text{vec-index } (\text{Matrix.row } B i) j = B \$(i,j)$  using *assms* by  
*auto*  
 thus ?thesis by *auto*  
 qed  
 also have ... =  $B \$(i,i) * (\text{vec-index } v i)$   
 proof (rule *sum-but-one*, (auto *simp* add: *assms*))  
 show  $\bigwedge j. j < n \implies j \neq i \implies B \$(i,j) = 0$  using *assms* *unfolding*  
*diagonal-mat-def*  
 by *force*  
 qed  
 finally show ?thesis .  
 qed

lemma *diagonal-mat-mult-index*:  
 fixes  $B::'a::\{\text{ring}\} \text{Matrix.mat}$   
 assumes *diagonal-mat*  $A$   
 and  $A \in \text{carrier-mat } n \ n$

**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**and**  $j < n$   
**shows**  $(A * B) \ \$\$ \ (i,j) = A \ \$\$ \ (i,i) * B \ \$\$ \ (i,j)$  **unfolding** *diagonal-mat-def*  
**proof** –  
**have**  $\text{dim-row } (A * B) = n$  **using** *assms* **by** *simp*  
**have**  $\text{dim-col } (A * B) = n$  **using** *assms* **by** *simp*  
**have**  $\bigwedge j. j < n \implies \text{dim-vec } (\text{Matrix.col } B \ j) = n$  **using** *assms* **by** *simp*  
**have**  $(A * B) \ \$\$ \ (i,j) = \text{Matrix.scalar-prod } (\text{Matrix.row } A \ i) \ (\text{Matrix.col } B \ j)$   
**using** *assms* **by** (*metis carrier-matD(1) carrier-matD(2) index-mult-mat(1)*)  
**also have**  $\dots =$   
 $(\sum k \in \{0 \ ..< n\}. \text{vec-index } (\text{Matrix.row } A \ i) \ k * \text{vec-index } (\text{Matrix.col } B \ j) \ k)$   
**using** *assms* **unfolding** *Matrix.scalar-prod-def* **by** *simp*  
**also have**  $\dots = \text{vec-index } (\text{Matrix.row } A \ i) \ i * \text{vec-index } (\text{Matrix.col } B \ j) \ i$   
**proof** (*rule sum-but-one*)  
**show**  $i < n$  **using** *assms*  $\langle \text{dim-row } (A * B) = n \rangle$  **by** *simp*  
**show**  $\forall k < n. k \neq i \longrightarrow \text{vec-index } (\text{Matrix.row } A \ i) \ k = 0$  **using** *assms*  $\langle i < n \rangle$   
**unfolding** *diagonal-mat-def* **by** *auto*  
**qed**  
**also have**  $\dots = A \ \$\$ \ (i,i) * B \ \$\$ \ (i,j)$  **using** *assms*  
**by** (*metis carrier-matD(1) carrier-matD(2) index-col index-row(1)*)  
**finally show** *?thesis* .  
**qed**

**lemma** *diagonal-mat-mult-index'*:

**fixes**  $A :: 'a :: \text{comm-ring } \text{Matrix.mat}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and** *diagonal-mat*  $B$   
**and**  $j < n$   
**and**  $i < n$   
**shows**  $(A * B) \ \$\$ \ (i,j) = B \ \$\$ \ (j,j) * A \ \$\$ \ (i, j)$

**proof** –

**have**  $(A * B) \ \$\$ \ (i,j) = \text{Matrix.scalar-prod } (\text{Matrix.row } A \ i) \ (\text{Matrix.col } B \ j)$   
**using** *assms*  
*times-mat-def*[of  $A$ ] **by** *simp*  
**also have**  $\dots = \text{Matrix.scalar-prod } (\text{Matrix.col } B \ j) \ (\text{Matrix.row } A \ i)$   
**using** *comm-scalar-prod*[of *Matrix.row*  $A \ i \ n$ ] *assms* **by** *auto*  
**also have**  $\dots = (\text{Matrix.vec-index } (\text{Matrix.col } B \ j) \ j) * (\text{Matrix.vec-index } (\text{Matrix.row } A \ i) \ i)$   
**unfolding** *Matrix.scalar-prod-def*  
**proof** (*rule sum-but-one*)  
**show**  $j < \text{dim-vec } (\text{Matrix.row } A \ i)$  **using** *assms* **by** *simp*  
**show**  $\forall ia < \text{dim-vec } (\text{Matrix.row } A \ i). ia \neq j \longrightarrow \text{Matrix.vec-index } (\text{Matrix.col } B \ j) \ ia = 0$   
**using** *assms*

by (metis carrier-matD(1) carrier-matD(2) diagonal-mat-def index-col index-row(2))

qed

also have ... =  $B \$(j,j) * A \$(i,j)$  using *assms by auto*

finally show  $(A * B) \$(i, j) = B \$(j, j) * A \$(i, j)$ .

qed

lemma *diagonal-mat-times-diag*:

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

and  $B \in \text{carrier-mat } n \ n$

and *diagonal-mat*  $A$

and *diagonal-mat*  $B$

shows *diagonal-mat*  $(A*B)$  unfolding *diagonal-mat-def*

proof (intro allI impI)

fix  $i \ j$

assume  $i < \text{dim-row } (A * B)$  and  $j < \text{dim-col } (A * B)$  and  $i \neq j$

thus  $(A * B) \$(i, j) = 0$  using *assms diag-mat-mult-diag-mat[of A n B]*

by *simp*

qed

lemma *diagonal-mat-commute*:

fixes  $A::'a::\{\text{comm-ring}\} \text{Matrix.mat}$

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

and  $B \in \text{carrier-mat } n \ n$

and *diagonal-mat*  $A$

and *diagonal-mat*  $B$

shows  $A * B = B * A$

proof (rule eq-matI)

show  $\text{dim-row } (A * B) = \text{dim-row } (B * A)$  using *assms by simp*

show  $\text{dim-col } (A * B) = \text{dim-col } (B * A)$  using *assms by simp*

have  $\text{bac}: B*A \in \text{carrier-mat } n \ n$  using *assms by simp*

fix  $i \ j$

assume  $i < \text{dim-row } (B*A)$  and  $j < \text{dim-col } (B*A)$  note  $ij = \text{this}$

have  $(A * B) \$(i, j) = A \$(i, j) * B \$(i,j)$

using *ij diagonal-mat-mult-index assms bac*

by (metis carrier-matD(1) carrier-matD(2) diagonal-mat-def mult-zero-right)

also have ... =  $B \$(i,j) * A \$(i, j)$

by (*simp add: Groups.mult-ac(2)*)

also have ... =  $(B*A) \$(i,j)$  using *ij diagonal-mat-mult-index assms bac*

by (metis carrier-matD(1) carrier-matD(2) diagonal-mat-def mult-not-zero)

finally show  $(A * B) \$(i, j) = (B*A) \$(i,j)$ .

qed

lemma *diagonal-mat-sq-index*:

fixes  $B::'a::\{\text{ring}\} \text{Matrix.mat}$

assumes *diagonal-mat*  $B$

and  $B \in \text{carrier-mat } n \ n$

and  $i < n$

and  $j < n$

**shows**  $(B * B) \$\$ (i,j) = B\$\$(i,i) * B\$\$(j,i)$   
**proof** –  
**have**  $(B * B) \$\$ (i,j) = B\$\$(i,i) * B\$\$(i,j)$   
**using** *assms diagonal-mat-mult-index[of B]* **by** *simp*  
**also have**  $\dots = B\$\$(i,i) * B\$\$(j,i)$  **using** *assms unfolding diagonal-mat-def*  
**by** (*metis carrier-matD(1) carrier-matD(2)*)  
**finally show** *?thesis* .  
**qed**

**lemma** *diagonal-mat-sq-index'*:  
**fixes**  $B::'a::\{\text{ring}\} \text{Matrix.mat}$   
**assumes** *diagonal-mat B*  
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**and**  $j < n$   
**shows**  $(B * B) \$\$ (i,j) = B\$\$(i,j) * B\$\$(i,j)$   
**proof** –  
**have** *eq*:  $(B * B) \$\$ (i,j) = B\$\$(i,i) * B\$\$(j,i)$   
**using** *assms diagonal-mat-sq-index* **by** *metis*  
**show** *?thesis*  
**proof** (*cases i = j*)  
**case** *True*  
**then show** *?thesis* **using** *eq* **by** *simp*  
**next**  
**case** *False*  
**hence**  $B\$\$(i,j) = 0$  **using** *assms unfolding diagonal-mat-def* **by** *simp*  
**hence**  $(B * B) \$\$ (i,j) = 0$  **using** *eq*  
**by** (*metis assms diagonal-mat-mult-index mult-not-zero*)  
**thus** *?thesis* **using** *eq*  $\langle B\$\$(i,j) = 0 \rangle$  **by** *simp*  
**qed**  
**qed**

**lemma** *diagonal-mat-sq-diag*:  
**fixes**  $B::'a::\{\text{ring}\} \text{Matrix.mat}$   
**assumes** *diagonal-mat B*  
**and**  $B \in \text{carrier-mat } n \ n$   
**shows** *diagonal-mat (B \* B)* **unfolding** *diagonal-mat-def*  
**proof** (*intro allI impI*)  
**have**  $\text{dim-row } (B * B) = n$  **using** *assms* **by** *simp*  
**have**  $\text{dim-col } (B * B) = n$  **using** *assms* **by** *simp*  
**have** *jvec*:  $\bigwedge j. j < n \implies \text{dim-vec } (\text{Matrix.col } B \ j) = n$  **using** *assms* **by** *simp*  
**fix**  $i \ j$   
**assume**  $i < \text{dim-row } (B * B)$   
**and**  $j < \text{dim-col } (B * B)$   
**and**  $i \neq j$  **note** *ijprops = this*  
**thus**  $(B * B) \$\$ (i,j) = 0$  **using** *diagonal-mat-sq-index*  
**by** (*metis*  $\langle \text{dim-col } (B * B) = n \rangle \langle \text{dim-row } (B * B) = n \rangle$  *assms(1) assms(2)*  
*carrier-matD(1)*  
*carrier-matD(2) diagonal-mat-def mult-not-zero*)

qed

**lemma** *real-diagonal-hermitian*:

**fixes**  $B :: \text{complex Matrix.mat}$

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

**and** *diagonal-mat*  $B$

**and**  $\forall i < \text{dim-row } B. B \$(i, i) \in \text{Reals}$

**shows** *hermitian*  $B$  **unfolding** *hermitian-def*

**proof** (*rule eq-matI*)

**show**  $\text{dim-row } (\text{Complex-Matrix.adjoint } B) = \text{dim-row } B$  **using** *assms* **by** *auto*

**show**  $\text{dim-col } (\text{Complex-Matrix.adjoint } B) = \text{dim-col } B$  **using** *assms* **by** *auto*

**next**

**fix**  $i \ j$

**assume**  $i < \text{dim-row } B$  **and**  $j < \text{dim-col } B$  **note**  $ij = \text{this}$

**show**  $\text{Complex-Matrix.adjoint } B \$(i, j) = B \$(i, j)$

**proof** (*cases*  $i = j$ )

**case** *True*

**thus** *?thesis* **using** *assms*  $ij$  *Reals-cnj-iff*

**unfolding** *diagonal-mat-def* *Complex-Matrix.adjoint-def* **by** *simp*

**next**

**case** *False*

**then show** *?thesis* **using** *assms*  $ij$

**unfolding** *diagonal-mat-def* *Complex-Matrix.adjoint-def* **by** *simp*

qed

qed

## 2.2 Unitary equivalence

**definition** *unitarily-equiv* **where**

*unitarily-equiv*  $A \ B \ U \equiv (\text{unitary } U \wedge$

*similar-mat-wit*  $A \ B \ U \ (\text{Complex-Matrix.adjoint } U))$

**lemma** *unitarily-equivD*:

**assumes** *unitarily-equiv*  $A \ B \ U$

**shows** *unitary*  $U$

*similar-mat-wit*  $A \ B \ U \ (\text{Complex-Matrix.adjoint } U)$  **using** *assms*

**unfolding** *unitarily-equiv-def* **by** *auto*

**lemma** *unitarily-equivI*:

**assumes** *similar-mat-wit*  $A \ B \ U \ (\text{Complex-Matrix.adjoint } U)$

**and** *unitary*  $U$

**shows** *unitarily-equiv*  $A \ B \ U$  **using** *assms*

**unfolding** *unitarily-equiv-def* **by** *simp*

**lemma** *unitarily-equivI'*:

**assumes**  $A = \text{mat-conj } U \ B$

**and** *unitary*  $U$

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

**and**  $B \in \text{carrier-mat } n \ n$

**shows** *unitarily-equiv A B U using assms*  
**unfolding** *unitarily-equiv-def similar-mat-wit-def*  
**by** (*metis (mono-tags, opaque-lifting) Complex-Matrix.unitary-def*  
*carrier-matD(1) empty-subsetI index-mult-mat(2) index-one-mat(2)*  
*insert-commute insert-subset unitary-adjoint unitary-simps(1)*  
*unitary-simps(2) mat-conj-def*)

**lemma** *unitarily-equiv-carrier:*  
**assumes** *A ∈ carrier-mat n n*  
**and** *unitarily-equiv A B U*  
**shows** *B ∈ carrier-mat n n U ∈ carrier-mat n n*  
**proof** –  
**show** *B ∈ carrier-mat n n*  
**by** (*metis assms carrier-matD(1) similar-mat-witD(5) unitarily-equivD(2)*)  
**show** *U ∈ carrier-mat n n*  
**by** (*metis assms similar-mat-witD2(6) unitarily-equivD(2)*)  
**qed**

**lemma** *unitarily-equiv-carrier'*:  
**assumes** *unitarily-equiv A B U*  
**shows** *A ∈ carrier-mat (dim-row A) (dim-row A)*  
*B ∈ carrier-mat (dim-row A) (dim-row A)*  
*U ∈ carrier-mat (dim-row A) (dim-row A)*  
**proof** –  
**show** *A ∈ carrier-mat (dim-row A) (dim-row A)*  
**by** (*metis assms carrier-mat-triv similar-mat-witD2(4) unitarily-equivD(2)*)  
**thus** *U ∈ carrier-mat (dim-row A) (dim-row A)*  
**using** *assms unitarily-equiv-carrier(2) by blast*  
**show** *B ∈ carrier-mat (dim-row A) (dim-row A)*  
**by** (*metis assms similar-mat-witD(5) unitarily-equivD(2)*)  
**qed**

**lemma** *unitarily-equiv-eq:*  
**assumes** *unitarily-equiv A B U*  
**shows** *A = U \* B \* (Complex-Matrix.adjoint U) using assms*  
**unfolding** *unitarily-equiv-def similar-mat-wit-def by meson*

**lemma** *unitarily-equiv-smult:*  
**assumes** *A ∈ carrier-mat n n*  
**and** *unitarily-equiv A B U*  
**shows** *unitarily-equiv (x ·<sub>m</sub> A) (x ·<sub>m</sub> B) U*  
**proof** (*rule unitarily-equivI*)  
**show** *similar-mat-wit (x ·<sub>m</sub> A) (x ·<sub>m</sub> B) U (Complex-Matrix.adjoint U)*  
**using** *mat-conj-smult assms*  
**by** (*simp add: similar-mat-wit-smult unitarily-equivD(2)*)  
**show** *unitary U using assms unitarily-equivD(1)[of A] by simp*  
**qed**

**lemma** *unitarily-equiv-uminus:*

**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $\text{unitarily-equiv } A \ B \ U$   
**shows**  $\text{unitarily-equiv } (-A) \ (-B) \ U$   
**proof** (*rule unitarily-equivI*)  
**show**  $\text{similar-mat-wit } (-A) \ (-B) \ U \ (\text{Complex-Matrix.adjoint } U)$   
**using**  $\text{mat-conj-uminus-eq } \text{assms}$   
**by** (*smt (verit) adjoint-dim-col adjoint-dim-row carrier-matD(1)*  
*carrier-matD(2) carrier-mat-triv index-uminus-mat(2)*  
*index-uminus-mat(3) similar-mat-witI unitarily-equivD(1)*  
*unitarily-equiv-carrier(1) unitarily-equiv-carrier(2)*  
*unitarily-equiv-eq unitary-simps(1) unitary-simps(2) mat-conj-def*)  
**show**  $\text{unitary } U$  **using**  $\text{assms unitarily-equivD(1)[of } A]$  **by** *simp*  
**qed**

**lemma** *unitarily-equiv-adjoint*:  
**assumes**  $\text{unitarily-equiv } A \ B \ U$   
**shows**  $\text{unitarily-equiv } B \ A \ (\text{Complex-Matrix.adjoint } U)$   
**unfolding**  $\text{unitarily-equiv-def}$   
**proof**  
**show**  $\text{Complex-Matrix.unitary } (\text{Complex-Matrix.adjoint } U)$   
**using**  $\text{Complex-Matrix.unitary-def } \text{assms unitarily-equiv-def unitary-adjoint}$   
**by** *blast*  
**have**  $\text{similar-mat-wit } B \ A \ (\text{Complex-Matrix.adjoint } U) \ U$   
**unfolding**  $\text{similar-mat-wit-def Let-def}$   
**proof** (*intro conjI*)  
**show**  $\text{car: } \{B, A, \text{Complex-Matrix.adjoint } U, U\} \subseteq$   
 $\text{carrier-mat } (\text{dim-row } B) \ (\text{dim-row } B)$   
**by** (*metis assms insert-commute similar-mat-wit-def*  
*similar-mat-wit-dim-row unitarily-equivD(2)*)  
**show**  $\text{Complex-Matrix.adjoint } U * U = 1_m \ (\text{dim-row } B)$  **using** *car*  
**by** (*meson assms insert-subset unitarily-equivD(1) unitary-simps(1)*)  
**show**  $U * \text{Complex-Matrix.adjoint } U = 1_m \ (\text{dim-row } B)$   
**by** (*meson assms similar-mat-wit-def similar-mat-wit-sym*  
*unitarily-equivD(2)*)  
**have**  $\text{Complex-Matrix.adjoint } U * A * U =$   
 $\text{Complex-Matrix.adjoint } U * (U * B * \text{Complex-Matrix.adjoint } U) * U$   
**using**  $\text{assms unitarily-equiv-eq}$  **by** *auto*  
**also have**  $\dots = B$   
**by** (*metis assms similar-mat-wit-def similar-mat-wit-sym unitarily-equivD(2)*)  
**finally show**  $B = \text{Complex-Matrix.adjoint } U * A * U$  **by** *simp*  
**qed**  
**thus**  $\text{similar-mat-wit } B \ A \ (\text{Complex-Matrix.adjoint } U)$   
 $(\text{Complex-Matrix.adjoint } (\text{Complex-Matrix.adjoint } U))$   
**by** (*simp add: Complex-Matrix.adjoint-adjoint*)  
**qed**

**lemma** *unitary-mult-conjugate*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $V \in \text{carrier-mat } n \ n$

**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and** *unitary*  $V$   
**and**  $\text{mat-conj } (\text{Complex-Matrix.adjoint } V) \ A = \text{mat-conj } U \ B$   
**shows**  $A = V * U * B * \text{Complex-Matrix.adjoint } (V * U)$   
**proof** –  
**have**  $\text{Complex-Matrix.adjoint } V * A * V \in \text{carrier-mat } n \ n$  **using** *assms*  
**by** (*metis adjoint-dim-row carrier-matD(2) carrier-mat-triv*  
*index-mult-mat(2) index-mult-mat(3)*)  
**have**  $A * V = V * (\text{Complex-Matrix.adjoint } V) * (A * V)$  **using** *assms* **by**  
*simp*  
**also have**  $\dots = V * (\text{Complex-Matrix.adjoint } V * (A * V))$   
**proof** (*rule assoc-mult-mat, auto simp add: assms*)  
**show**  $A * V \in \text{carrier-mat } (\text{dim-row } V) \ (\text{dim-row } V)$  **using** *assms* **by** *auto*  
**qed**  
**also have**  $\dots = V * (\text{Complex-Matrix.adjoint } V * A * V)$   
**by** (*metis adjoint-dim' assms(1) assms(2) assoc-mult-mat*)  
**also have**  $\dots = V * (U * B * (\text{Complex-Matrix.adjoint } U))$  **using** *assms*  
**by** (*simp add: Complex-Matrix.adjoint-adjoint mat-conj-def*)  
**also have**  $\dots = V * (U * (B * (\text{Complex-Matrix.adjoint } U)))$   
**by** (*metis adjoint-dim' assms(3) assms(4) assoc-mult-mat*)  
**also have**  $\dots = V * U * (B * (\text{Complex-Matrix.adjoint } U))$   
**proof** (*rule assoc-mult-mat[symmetric], auto simp add: assms*)  
**show**  $U \in \text{carrier-mat } (\text{dim-col } V) \ (\text{dim-row } B)$  **using** *assms* **by** *auto*  
**qed**  
**also have**  $\dots = V * U * B * (\text{Complex-Matrix.adjoint } U)$   
**proof** (*rule assoc-mult-mat[symmetric], auto simp add: assms*)  
**show**  $B \in \text{carrier-mat } (\text{dim-col } U) \ (\text{dim-col } U)$  **using** *assms* **by** *auto*  
**qed**  
**finally have**  $\text{eq: } A * V = V * U * B * (\text{Complex-Matrix.adjoint } U)$  .  
**have**  $A = A * (V * \text{Complex-Matrix.adjoint } V)$  **using** *assms* **by** *simp*  
**also have**  $\dots = A * V * \text{Complex-Matrix.adjoint } V$   
**proof** (*rule assoc-mult-mat[symmetric], auto simp add: assms*)  
**show**  $V \in \text{carrier-mat } (\text{dim-col } A) \ (\text{dim-col } V)$  **using** *assms* **by** *auto*  
**qed**  
**also have**  $\dots = V * U * B * (\text{Complex-Matrix.adjoint } U) *$   
 $(\text{Complex-Matrix.adjoint } V)$  **using** *eq* **by** *simp*  
**also have**  $\dots = V * U * B * ((\text{Complex-Matrix.adjoint } U) *$   
 $(\text{Complex-Matrix.adjoint } V))$   
**proof** (*rule assoc-mult-mat, auto simp add: assms*)  
**show**  $\text{Complex-Matrix.adjoint } U \in \text{carrier-mat } (\text{dim-col } B) \ (\text{dim-col } V)$   
**using** *adjoint-dim' assms* **by** *auto*  
**qed**  
**also have**  $\dots = V * U * B * \text{Complex-Matrix.adjoint } (V * U)$   
**by** (*metis adjoint-mult assms(2) assms(3)*)  
**finally show** *?thesis* .  
**qed**

**lemma** *unitarily-equiv-conjugate*:

```

assumes  $A \in \text{carrier-mat } n \ n$ 
and  $V \in \text{carrier-mat } n \ n$ 
and  $U \in \text{carrier-mat } n \ n$ 
and  $B \in \text{carrier-mat } n \ n$ 
and  $\text{unitarily-equiv } (\text{mat-conj } (\text{Complex-Matrix.adjoint } V) \ A) \ B \ U$ 
and  $\text{unitary } V$ 
shows  $\text{unitarily-equiv } A \ B \ (V * U)$ 
unfolding  $\text{unitarily-equiv-def}$ 
proof
  show  $\text{Complex-Matrix.unitary } (V * U)$  using  $\text{assms}$ 
    by  $(\text{simp add: unitarily-equivD}(1) \ \text{unitary-times-unitary})$ 
  show  $\text{similar-mat-wit } A \ B \ (V * U) \ (\text{Complex-Matrix.adjoint } (V * U))$ 
    unfolding  $\text{similar-mat-wit-def Let-def}$ 
  proof  $(\text{intro conjI})$ 
    show  $\{A, B, V * U, \text{Complex-Matrix.adjoint } (V * U)\} \subseteq$ 
       $\text{carrier-mat } (\text{dim-row } A) \ (\text{dim-row } A)$  using  $\text{assms}$  by  $\text{auto}$ 
    show  $V * U * \text{Complex-Matrix.adjoint } (V * U) = 1_m \ (\text{dim-row } A)$ 
      by  $(\text{metis } \text{Complex-Matrix.unitary-def } \langle \text{Complex-Matrix.unitary } (V * U) \rangle$ 
         $\text{assms}(1) \ \text{assms}(2) \ \text{carrier-matD}(1) \ \text{index-mult-mat}(2) \ \text{inverts-mat-def})$ 
    show  $\text{Complex-Matrix.adjoint } (V * U) * (V * U) = 1_m \ (\text{dim-row } A)$ 
      by  $(\text{metis } \text{Complex-Matrix.unitary-def } \langle \text{Complex-Matrix.unitary } (V * U) \rangle$ 
         $\langle V * U * \text{Complex-Matrix.adjoint } (V * U) = 1_m \ (\text{dim-row } A) \rangle$ 
         $\text{index-mult-mat}(2) \ \text{index-one-mat}(2) \ \text{unitary-simps}(1))$ 
    show  $A = V * U * B * \text{Complex-Matrix.adjoint } (V * U)$ 
  proof  $(\text{rule unitary-mult-conjugate}[of - n], \ \text{auto simp add: assms})$ 
    show  $\text{mat-conj } (\text{Complex-Matrix.adjoint } V) \ A = \text{mat-conj } U \ B$  using  $\text{assms}$ 
      by  $(\text{simp add: mat-conj-def unitarily-equiv-eq})$ 
  qed
qed
qed

```

**lemma**  $\text{mat-conj-commute}$ :

```

assumes  $A \in \text{carrier-mat } n \ n$ 
and  $B \in \text{carrier-mat } n \ n$ 
and  $U \in \text{carrier-mat } n \ n$ 
and  $\text{unitary } U$ 
and  $A * B = B * A$ 
shows  $(\text{mat-conj } (\text{Complex-Matrix.adjoint } U) \ A) *$ 
   $(\text{mat-conj } (\text{Complex-Matrix.adjoint } U) \ B) =$ 
   $(\text{mat-conj } (\text{Complex-Matrix.adjoint } U) \ B) *$ 
   $(\text{mat-conj } (\text{Complex-Matrix.adjoint } U) \ A)$  (is  $?L * ?R = ?R * ?L)$ 
proof –
  have  $u: \text{Complex-Matrix.adjoint } U \in \text{carrier-mat } n \ n$  using  $\text{assms}$ 
    by  $(\text{simp add: adjoint-dim'})$ 
  have  $ca: \text{Complex-Matrix.adjoint } U * A * U \in \text{carrier-mat } n \ n$ 
    using  $\text{assms}$  by  $\text{auto}$ 
  have  $cb: \text{Complex-Matrix.adjoint } U * B * U \in \text{carrier-mat } n \ n$ 
    using  $\text{assms}$  by  $\text{auto}$ 
  have  $?L * ?R =$ 

```

$?L * (\text{Complex-Matrix.adjoint } U * (B * U))$   
**proof** –  
 have  $\text{Complex-Matrix.adjoint } U * B * U =$   
 $\text{Complex-Matrix.adjoint } U * (B * U)$   
 using  $\text{assoc-mult-mat}[\text{of } - \text{ } n \text{ } n \text{ } B \text{ } n \text{ } U] \text{ assms}$   
 by  $(\text{meson adjoint-dim}')$   
 thus  $?thesis$  using  $\text{mat-conj-adjoint}$  by  $\text{metis}$   
**qed**  
 also have  $\dots = ?L * \text{Complex-Matrix.adjoint } U * (B * U)$   
**proof** –  
 have  $\exists na \text{ } nb. \text{Complex-Matrix.adjoint } U \in \text{carrier-mat } n \text{ } na \wedge$   
 $B * U \in \text{carrier-mat } na \text{ } nb$   
 by  $(\text{metis (no-types) assms(2) carrier-matD(1) carrier-mat-triv index-mult-mat(2)})$   
 $u)$   
 then show  $?thesis$  using  $ca$   
 by  $(\text{metis assoc-mult-mat mat-conj-adjoint})$   
**qed**  
 also have  $\dots = \text{Complex-Matrix.adjoint } U * A * (U * (\text{Complex-Matrix.adjoint } U)) * (B * U)$   
**proof** –  
 have  $\text{Complex-Matrix.adjoint } U * A * U * \text{Complex-Matrix.adjoint } U =$   
 $\text{Complex-Matrix.adjoint } U * A * (U * \text{Complex-Matrix.adjoint } U)$   
 using  $\text{assoc-mult-mat}[\text{of } \text{Complex-Matrix.adjoint } U * A \text{ } n \text{ } n]$   
 by  $(\text{metis assms(1) assms(3) mult-carrier-mat } u)$   
 thus  $?thesis$  by  $(\text{simp add: mat-conj-adjoint})$   
**qed**  
 also have  $\dots = \text{Complex-Matrix.adjoint } U * A * (B * U)$   
 using  $\text{assms}$  by  $\text{auto}$   
 also have  $\dots = \text{Complex-Matrix.adjoint } U * A * B * U$   
**proof**  $(\text{rule assoc-mult-mat}[\text{symmetric}], \text{auto simp add: assms})$   
 show  $B \in \text{carrier-mat } (\text{dim-col } A) \text{ } (\text{dim-row } U)$  using  $\text{assms}$  by  $\text{simp}$   
**qed**  
 also have  $\dots = \text{Complex-Matrix.adjoint } U * (A * B) * U$   
 using  $\text{assms } u$  by  $\text{auto}$   
 also have  $\dots = \text{Complex-Matrix.adjoint } U * (B * A) * U$  using  $\text{assms}$  by  $\text{simp}$   
 also have  $\dots = \text{Complex-Matrix.adjoint } U * B * A * U$   
 using  $\text{assms } u$  by  $\text{auto}$   
 also have  $\dots = \text{Complex-Matrix.adjoint } U * B * (A * U)$   
**proof**  $(\text{rule assoc-mult-mat, auto simp add: assms})$   
 show  $A \in \text{carrier-mat } (\text{dim-col } B) \text{ } (\text{dim-row } U)$   
 using  $\text{assms}$  by  $\text{simp}$   
**qed**  
 also have  $\dots = \text{Complex-Matrix.adjoint } U * B * (U * (\text{Complex-Matrix.adjoint } U)) * (A * U)$   
 using  $\text{assms}$  by  $\text{auto}$   
 also have  $\dots = \text{Complex-Matrix.adjoint } U * B * U * (\text{Complex-Matrix.adjoint } U) * (A * U)$   
**proof** –  
 have  $\text{Complex-Matrix.adjoint } U * B * U * \text{Complex-Matrix.adjoint } U =$

$Complex-Matrix.adjoint\ U * B * (U * Complex-Matrix.adjoint\ U)$   
**proof** (rule assoc-mult-mat, auto simp add: assms)  
**show**  $U \in carrier-mat\ (dim-col\ B)\ (dim-col\ U)$  **using** assms **by** simp  
**qed**  
**thus** ?thesis **by** simp  
**qed**  
**also have**  $\dots = Complex-Matrix.adjoint\ U * B * U * ((Complex-Matrix.adjoint\ U) * (A * U))$   
**proof** (rule assoc-mult-mat, auto simp add: u cb)  
**show**  $A * U \in carrier-mat\ (dim-row\ U)\ n$  **using** assms **by** simp  
**qed**  
**also have**  $\dots = Complex-Matrix.adjoint\ U * B * U * ((Complex-Matrix.adjoint\ U) * A * U)$   
**proof** –  
**have**  $(Complex-Matrix.adjoint\ U) * (A * U) = (Complex-Matrix.adjoint\ U) * A * U$   
**proof** (rule assoc-mult-mat[symmetric], auto simp add: assms u)  
**show**  $A \in carrier-mat\ (dim-row\ U)\ (dim-row\ U)$  **using** assms **by** simp  
**qed**  
**thus** ?thesis **by** simp  
**qed**  
**finally show** ?thesis **by** (metis mat-conj-adjoint)  
**qed**

**lemma** unitarily-equiv-commute:  
**assumes** unitarily-equiv  $A\ B\ U$   
**and**  $A * C = C * A$   
**shows**  $B * (Complex-Matrix.adjoint\ U * C * U) = Complex-Matrix.adjoint\ U * C * U * B$   
**proof** –  
**note**  $car = unitarily-equiv-carrier'[OF\ assms(1)]$   
**have**  $cr: dim-row\ C = dim-col\ A$   
**by** (metis assms(2) car(1) carrier-matD(2) index-mult-mat(2))  
**have**  $cd: dim-col\ C = dim-row\ A$   
**by** (metis  $\langle dim-row\ C = dim-col\ A \rangle$  assms(2) index-mult-mat(2) index-mult-mat(3))  
**have**  $Complex-Matrix.adjoint\ U * A * U = B$   
**using** assms unitarily-equiv-adjoint  
**by** (metis Complex-Matrix.adjoint-adjoint unitarily-equiv-eq)  
**thus** ?thesis **using** mat-conj-commute assms car  
**by** (metis carrier-matD(2) carrier-matI cd cr mat-conj-adjoint unitarily-equivD(1))  
**qed**

**definition** unitary-diag **where**  
 $unitary-diag\ A\ B\ U \equiv unitarily-equiv\ A\ B\ U \wedge diagonal-mat\ B$

**lemma** unitary-diagI:  
**assumes** similar-mat-wit  $A\ B\ U\ (Complex-Matrix.adjoint\ U)$

**and** *diagonal-mat*  $B$   
**and** *unitary*  $U$   
**shows** *unitary-diag*  $A B U$  **using** *assms*  
**unfolding** *unitary-diag-def* *unitarily-equiv-def* **by** *simp*

**lemma** *unitary-diagI'*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and** *diagonal-mat*  $B$   
**and** *unitary*  $U$   
**and**  $A = \text{mat-conj } U \ B$   
**shows** *unitary-diag*  $A B U$  **unfolding** *unitary-diag-def*  
**proof**  
**show** *diagonal-mat*  $B$  **using** *assms* **by** *simp*  
**show** *unitarily-equiv*  $A B U$  **using** *assms* *unitarily-equivI'* **by** *metis*  
**qed**

**lemma** *unitary-diagD*:  
**assumes** *unitary-diag*  $A B U$   
**shows** *similar-mat-wit*  $A B U$  (*Complex-Matrix.adjoint*  $U$ )  
*diagonal-mat*  $B$  *unitary*  $U$  **using** *assms*  
**unfolding** *unitary-diag-def* *unitarily-equiv-def*  
**by** *simp+*

**lemma** *unitary-diag-imp-unitarily-equiv[simp]*:  
**assumes** *unitary-diag*  $A B U$   
**shows** *unitarily-equiv*  $A B U$  **using** *assms* **unfolding** *unitary-diag-def* **by** *simp*

**lemma** *unitary-diag-diagonal[simp]*:  
**assumes** *unitary-diag*  $A B U$   
**shows** *diagonal-mat*  $B$  **using** *assms* **unfolding** *unitary-diag-def* **by** *simp*

**lemma** *unitary-diag-carrier*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and** *unitary-diag*  $A B U$   
**shows**  $B \in \text{carrier-mat } n \ n$   $U \in \text{carrier-mat } n \ n$   
**proof** –  
**show**  $B \in \text{carrier-mat } n \ n$   
**using** *assms* *unitarily-equiv-carrier(1)*[*of*  $A \ n \ B \ U$ ] **by** *simp*  
**show**  $U \in \text{carrier-mat } n \ n$   
**using** *assms* *unitarily-equiv-carrier(2)*[*of*  $A \ n \ B \ U$ ] **by** *simp*  
**qed**

**lemma** *unitary-mult-square-eq*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A = \text{mat-conj } U \ B$   
**and** (*Complex-Matrix.adjoint*  $U$ ) \*  $U = 1_m \ n$

**shows**  $A * A = \text{mat-conj } U (B*B)$   
**proof** –  
**have**  $A * A = U * B * (\text{Complex-Matrix.adjoint } U) * (U * B * (\text{Complex-Matrix.adjoint } U))$   
**using** *assms unfolding mat-conj-def by simp*  
**also have**  $\dots = U * B * ((\text{Complex-Matrix.adjoint } U) * U) * (B * (\text{Complex-Matrix.adjoint } U))$   
**by** (*smt (verit, best) adjoint-dim' assms(2,3) assoc-mult-mat mult-carrier-mat*)  
**also have**  $\dots = U * B * (B * (\text{Complex-Matrix.adjoint } U))$  **using** *assms by simp*  
**also have**  $\dots = U * (B * B) * (\text{Complex-Matrix.adjoint } U)$   
**by** (*smt (verit) adjoint-dim-row assms(2) assms(3) assoc-mult-mat carrier-matD(2)*  
*carrier-mat-triv index-mult-mat(3)*)  
**finally show** *?thesis unfolding mat-conj-def* .  
**qed**

**lemma** *hermitian-square-similar-mat-wit*:  
**fixes**  $A::\text{complex Matrix.mat}$   
**assumes** *hermitian A*  
**and**  $A \in \text{carrier-mat } n \ n$   
**and** *unitary-diag A B U*  
**shows** *similar-mat-wit (A \* A) (B \* B) U (Complex-Matrix.adjoint U)*  
**proof** –  
**have**  $B \in \text{carrier-mat } n \ n$  **using** *unitary-diag-carrier[of A] assms by metis*  
**hence**  $B * B \in \text{carrier-mat } n \ n$  **by** *simp*  
**have** *unitary U* **using** *assms unitary-diagD[of A] by simp*  
**have**  $A * A = \text{mat-conj } U (B*B)$  **using** *assms unitary-mult-square-eq[of A n]*  
**by** (*metis ⟨B ∈ carrier-mat n n⟩ ⟨Complex-Matrix.unitary U⟩ mat-conj-def*  
*unitarily-equiv-carrier(2) unitarily-equiv-eq unitary-diag-def*  
*unitary-simps(1)*)  
**moreover have**  $\{A * A, B * B, U, \text{Complex-Matrix.adjoint } U\} \subseteq \text{carrier-mat } n \ n$   
**by** (*metis ⟨B \* B ∈ carrier-mat n n⟩ adjoint-dim' assms(2) assms(3) empty-subsetI*  
*insert-subsetI mult-carrier-mat unitary-diag-carrier(2)*)  
**moreover have**  $U * \text{Complex-Matrix.adjoint } U = 1_m \ n \wedge \text{Complex-Matrix.adjoint } U * U = 1_m \ n$   
**by** (*meson ⟨Complex-Matrix.unitary U⟩ calculation(2) insert-subset unitary-simps(1)*  
*unitary-simps(2)*)  
**ultimately show** *?thesis unfolding similar-mat-wit-def mat-conj-def by auto*  
**qed**

**lemma** *unitarily-equiv-square*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and** *unitarily-equiv A B U*  
**shows** *unitarily-equiv (A\*A) (B\*B) U*  
**proof** (*rule unitarily-equivI*)

**show** *unitary U* **using** *assms unitarily-equivD(1)[of A]* **by** *simp*  
**show** *similar-mat-wit (A \* A) (B \* B) U (Complex-Matrix.adjoint U)*  
**by** (*smt (verit) <Complex-Matrix.unitary U> assms carrier-matD(1)*  
*carrier-matD(2) carrier-mat-triv index-mult-mat(2)*  
*index-mult-mat(3) similar-mat-witI unitarily-equiv-carrier(1)*  
*unitarily-equiv-carrier(2) unitarily-equiv-eq unitary-mult-square-eq*  
*unitary-simps(1) unitary-simps(2) mat-conj-def*)

**qed**

**lemma** *conjugate-eq-unitarily-equiv:*

**assumes** *A ∈ carrier-mat n n*

**and** *V ∈ carrier-mat n n*

**and** *unitarily-equiv A B U*

**and** *unitary V*

**and** *V \* B \* (Complex-Matrix.adjoint V) = B*

**shows** *unitarily-equiv A B (U \* V)*

**unfolding** *unitarily-equiv-def similar-mat-wit-def Let-def*

**proof** (*intro conjI*)

**have** *B ∈ carrier-mat n n*

**using** *assms(1) assms(3) unitarily-equiv-carrier(1)* **by** *blast*

**have** *U ∈ carrier-mat n n*

**using** *assms(1) assms(3) unitarily-equiv-carrier(2)* **by** *auto*

**show** *u: unitary (U \* V)*

**by** (*metis Complex-Matrix.unitary-def adjoint-dim-col assms(1) assms(2)*  
*assms(3) assms(4) carrier-matD(2) index-mult-mat(3) unitarily-equivD(1)*  
*unitarily-equiv-eq unitary-times-unitary*)

**thus** *l: U \* V \* Complex-Matrix.adjoint (U \* V) = 1<sub>m</sub> (dim-row A)*

**by** (*metis Complex-Matrix.unitary-def assms(1) assms(2) carrier-matD(1)*  
*carrier-matD(2) index-mult-mat(3) inverts-mat-def*)

**thus** *r: Complex-Matrix.adjoint (U \* V) \* (U \* V) = 1<sub>m</sub> (dim-row A)* **using** *u*

**by** (*metis Complex-Matrix.unitary-def index-mult-mat(2) index-one-mat(2)*  
*unitary-simps(1)*)

**show** *{A, B, U \* V, Complex-Matrix.adjoint (U \* V)} ⊆*

*carrier-mat (dim-row A) (dim-row A)*

**using** *<B ∈ carrier-mat n n> <U ∈ carrier-mat n n> adjoint-dim' assms*

**by** *auto*

**have** *U \* V \* B \* Complex-Matrix.adjoint (U \* V) =*

*U \* V \* B \* (Complex-Matrix.adjoint V \* Complex-Matrix.adjoint U)*

**by** (*metis <U ∈ carrier-mat n n> adjoint-mult assms(2)*)

**also have** *... = U \* V \* B \* Complex-Matrix.adjoint V \**

*Complex-Matrix.adjoint U*

**proof** (*rule assoc-mult-mat[symmetric], auto simp add: assms*)

**show** *Complex-Matrix.adjoint V ∈ carrier-mat (dim-col B) (dim-col U)*

**using** *<B ∈ carrier-mat n n> <U ∈ carrier-mat n n> adjoint-dim assms(2)*

**by** *auto*

**qed**

**also have** *... = U \* V \* B \* (Complex-Matrix.adjoint V \**

*Complex-Matrix.adjoint U)*

**proof** (*rule assoc-mult-mat, auto simp add: assms*)

**show**  $\text{Complex-Matrix.adjoint } V \in \text{carrier-mat } (\text{dim-col } B) (\text{dim-col } U)$   
**by** (*metis*  $\langle B \in \text{carrier-mat } n \ n \rangle \langle U \in \text{carrier-mat } n \ n \rangle \text{adjoint-dim}'$   
*assms*(2) *carrier-matD*(2))

**qed**  
**also have**  $\dots = U * V * (B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U))$   
**proof** (*rule assoc-mult-mat*, *auto simp add: assms*)  
**show**  $B \in \text{carrier-mat } (\text{dim-col } V) (\text{dim-col } V)$   
**by** (*metis*  $\langle B \in \text{carrier-mat } n \ n \rangle$  *assms*(2) *carrier-matD*(2))

**qed**  
**also have**  $\dots = U * (V * (B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U)))$   
**proof** (*rule assoc-mult-mat*, *auto simp add: assms*)  
**show**  $V \in \text{carrier-mat } (\text{dim-col } U) (\text{dim-row } B)$   
**using**  $\langle B \in \text{carrier-mat } n \ n \rangle \langle U \in \text{carrier-mat } n \ n \rangle$  *assms*(2) **by** *auto*

**qed**  
**finally have**  $\text{eq: } U * V * B * \text{Complex-Matrix.adjoint } (U * V) =$   
 $U * (V * (B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U)))$  .  
**have**  $V * (B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U)) =$   
 $V * B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U)$   
**proof** (*rule assoc-mult-mat[symmetric]*, *auto simp add: assms*)  
**show**  $B \in \text{carrier-mat } (\text{dim-col } V) (\text{dim-col } V)$   
**using**  $\langle B \in \text{carrier-mat } n \ n \rangle$  *assms*(2) **by** *auto*

**qed**  
**also have**  $\dots = V * B * \text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U$   
**proof** (*rule assoc-mult-mat[symmetric]*, *auto simp add: assms*)  
**show**  $\text{Complex-Matrix.adjoint } V \in \text{carrier-mat } (\text{dim-col } B) (\text{dim-col } U)$   
**by** (*metis*  $\langle B \in \text{carrier-mat } n \ n \rangle \langle U \in \text{carrier-mat } n \ n \rangle$   
*adjoint-dim-row assms*(2) *assms*(5) *carrier-matD*(2) *carrier-mat-triv*  
*index-mult-mat*(3))

**qed**  
**also have**  $\dots = B * \text{Complex-Matrix.adjoint } U$  **using** *assms* **by** *simp*  
**finally have**  $V * (B * (\text{Complex-Matrix.adjoint } V * \text{Complex-Matrix.adjoint } U))$   
 $=$   
 $B * \text{Complex-Matrix.adjoint } U$  .  
**hence**  $U * V * B * \text{Complex-Matrix.adjoint } (U * V) = U * B * \text{Complex-Matrix.adjoint } U$  **using** *eq*  
**by** (*metis*  $\langle B \in \text{carrier-mat } n \ n \rangle \langle U \in \text{carrier-mat } n \ n \rangle$  *adjoint-dim'* *assoc-mult-mat*)  
**also have**  $\dots = A$  **using** *assms unitarily-equiv-eq[of A B U]* **by** *simp*  
**finally show**  $A = U * V * B * \text{Complex-Matrix.adjoint } (U * V)$  **by** *simp*

**qed**

**definition** *real-diag-decomp* **where**  
*real-diag-decomp*  $A B U \equiv \text{unitary-diag } A B U \wedge$   
 $(\forall i < \text{dim-row } B. B\$\$(i, i) \in \text{Reals})$

**lemma** *real-diag-decompD[simp]*:  
**assumes** *real-diag-decomp*  $A B U$

**shows** *unitary-diag*  $A B U$   
 $(\forall i < \dim\text{-row } B. B \$(i, i) \in \text{Reals})$  **using** *assms*  
**unfolding** *real-diag-decomp-def unitary-diag-def* **by** *auto*

**lemma** *hermitian-decomp-decomp'*:  
**fixes**  $A::\text{complex Matrix.mat}$   
**assumes** *hermitian-decomp*  $A B U$   
**shows** *real-diag-decomp*  $A B U$   
**using** *assms* **unfolding** *hermitian-decomp-def*  
**by** (*metis real-diag-decomp-def unitarily-equiv-def unitary-diag-def*)

**lemma** *real-diag-decomp-hermitian*:  
**fixes**  $A::\text{complex Matrix.mat}$   
**assumes** *real-diag-decomp*  $A B U$   
**shows** *hermitian*  $A$

**proof** –  
**have**  $ud: \text{unitary-diag } A B U$  **using** *assms real-diag-decompD* **by** *simp*  
**hence**  $A = U * B * (\text{Complex-Matrix.adjoint } U)$   
**by** (*simp add: unitarily-equiv-eq*)  
**have**  $\text{Complex-Matrix.adjoint } A =$   
 $\text{Complex-Matrix.adjoint } (U * B * (\text{Complex-Matrix.adjoint } U))$   
**using**  $ud$  *assms unitarily-equiv-eq unitary-diag-imp-unitarily-equiv* **by** *blast*  
**also have**  $\dots = \text{Complex-Matrix.adjoint } (\text{Complex-Matrix.adjoint } U) *$   
 $\text{Complex-Matrix.adjoint } B * \text{Complex-Matrix.adjoint } U$   
**by** (*smt (verit) ud Complex-Matrix.adjoint-adjoint Complex-Matrix.unitary-def*)

*adjoint-dim-col adjoint-mult assms assoc-mult-mat calculation*  
*carrier-matD(2) carrier-mat-triv index-mult-mat(2) index-mult-mat(3)*  
*similar-mat-witD2(5) similar-mat-wit-dim-row unitary-diagD(1)*  
*unitary-diagD(3))*

**also have**  $\dots = U * \text{Complex-Matrix.adjoint } B * \text{Complex-Matrix.adjoint } U$   
**by** (*simp add: Complex-Matrix.adjoint-adjoint*)  
**also have**  $\dots = U * B * \text{Complex-Matrix.adjoint } U$   
**using** *real-diagonal-hermitian*  
**by** (*metis assms hermitian-def real-diag-decomp-def similar-mat-witD(5)*  
*unitary-diagD(1) unitary-diagD(2)*)  
**also have**  $\dots = A$  **using**  $\langle A = U * B * (\text{Complex-Matrix.adjoint } U) \rangle$  **by** *simp*  
**finally show** *?thesis* **unfolding** *hermitian-def* **by** *simp*

**qed**

**lemma** *unitary-conjugate-real-diag-decomp*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $Us \in \text{carrier-mat } n \ n$   
**and** *unitary*  $Us$   
**and** *real-diag-decomp* (*mat-conj* ( $\text{Complex-Matrix.adjoint } Us$ )  $A$ )  $B U$   
**shows** *real-diag-decomp*  $A B (Us * U)$  **unfolding** *real-diag-decomp-def*  
**proof** (*intro conjI allI impI*)  
**show**  $\bigwedge i. i < \dim\text{-row } B \implies B \$(i, i) \in \mathbf{R}$  **using** *assms*

```

  unfolding real-diag-decomp-def by simp
show unitary-diag A B (Us * U) unfolding unitary-diag-def
proof (rule conjI)
  show diagonal-mat B using assms real-diag-decompD(1) unitary-diagD(2)
  by metis
show unitarily-equiv A B (Us * U)
proof (rule unitarily-equiv-conjugate)
  show A ∈ carrier-mat n n using assms by simp
  show unitary Us using assms by simp
  show Us ∈ carrier-mat n n using assms by simp
  show unitarily-equiv (mat-conj (Complex-Matrix.adjoint Us) A) B U
  using assms real-diag-decompD(1) unfolding unitary-diag-def by metis
thus U ∈ carrier-mat n n
  by (metis (mono-tags) adjoint-dim' assms(2) carrier-matD(1)
    index-mult-mat(2) mat-conj-def unitarily-equiv-carrier'(3))
show B ∈ carrier-mat n n
  using ⟨unitarily-equiv (mat-conj (Complex-Matrix.adjoint Us) A) B U⟩
  assms(2) unitarily-equiv-carrier'(2)
  by (metis ⟨U ∈ carrier-mat n n⟩ carrier-matD(2)
    unitarily-equiv-carrier'(3))
qed
qed
qed

```

### 2.3 On the spectrum of a matrix

```

lemma similar-spectrum-eq:
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and similar-mat A B
  and upper-triangular B
  shows spectrum A = set (diag-mat B)
proof -
  have (∏ a ← (eigvals A). [:- a, 1:]) = char-poly A
  using eigvals-poly-length assms by simp
  also have ... = char-poly B
  proof (rule char-poly-similar)
    show similar-mat A B using assms real-diag-decompD(1)
    using similar-mat-def by blast
  qed
  also have ... = (∏ a ← diag-mat B. [:- a, 1:])
  proof (rule char-poly-upper-triangular)
    show B ∈ carrier-mat n n using assms similar-matD by auto
    thus upper-triangular B using assms by simp
  qed
  finally have (∏ a ← (eigvals A). [:- a, 1:]) = (∏ a ← diag-mat B. [:- a, 1:]) .
  thus ?thesis using poly-root-set-eq unfolding spectrum-def by metis
qed

```

```

lemma unitary-diag-spectrum-eq:
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and unitary-diag A B U
shows spectrum A = set (diag-mat B)
proof (rule similar-spectrum-eq)
  show A ∈ carrier-mat n n using assms by simp
  show similar-mat A B using assms unitary-diagD(1)
    by (metis similar-mat-def)
  show upper-triangular B using assms
    unitary-diagD(2) unitary-diagD(1) diagonal-imp-upper-triangular
    by (metis similar-mat-witD2(5))
qed

```

```

lemma unitary-diag-spectrum-eq':
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and unitary-diag A B U
shows spectrum A = diag-elems B
proof –
  have spectrum A = set (diag-mat B) using assms unitary-diag-spectrum-eq
    by simp
  also have ... = diag-elems B using diag-elems-set-diag-mat[of B] by simp
  finally show spectrum A = diag-elems B .
qed

```

```

lemma hermitian-real-diag-decomp:
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and  $0 < n$ 
  and hermitian A
obtains B U where real-diag-decomp A B U
proof –
  {
    have es: char-poly A = (∏ (e :: complex) ← (eigvals A). [:- e, 1:])
      using assms eigvals-poly-length by auto
    obtain B U Q where us: unitary-schur-decomposition A (eigvals A) = (B, U, Q)

    by (cases unitary-schur-decomposition A (eigvals A))
    hence pr: similar-mat-wit A B U (Complex-Matrix.adjoint U) ∧ diagonal-mat
      B ∧
      diag-mat B = (eigvals A) ∧ unitary U ∧ (∀ i < n. B$$(i, i) ∈ Reals)
    using hermitian-eigenvalue-real assms es by auto
    moreover have dim-row B = n using assms similar-mat-wit-dim-row[of A]
      pr by auto
    ultimately have real-diag-decomp A B U using unitary-diagI[of A]
    unfolding real-diag-decomp-def by simp
    hence  $\exists B U. \text{real-diag-decomp } A B U$  by auto
  }

```

thus ?thesis using that by auto  
qed

**lemma** *spectrum-smult*:

fixes  $A::\text{complex Matrix.mat}$

assumes *hermitian*  $A$

and  $A \in \text{carrier-mat } n \ n$

and  $0 < n$

shows  $\text{spectrum } (x \cdot_m A) = \{x * a \mid a. a \in \text{spectrum } A\}$

**proof** –

**obtain**  $B \ U$  **where**  $bu: \text{real-diag-decomp } A \ B \ U$

**using** *assms hermitian-real-diag-decomp*[of  $A$ ] **by** *auto*

**hence**  $\text{spectrum } (x \cdot_m A) = \text{set } (\text{diag-mat } (x \cdot_m B))$

**using** *assms unitary-diag-spectrum-eq*[of  $x \cdot_m A$ ]

*unitarily-equiv-smult*[of  $A$ ]

**by** (*meson diagonal-mat-smult real-diag-decompD(1) real-diag-decompD(2)*  
*smult-carrier-mat unitary-diag-def*)

**also have**  $\dots = \{x * a \mid a. a \in \text{set } (\text{diag-mat } B)\}$

**using** *assms set-diag-mat-smult*[of  $B \ n \ x$ ]

**by** (*meson bu real-diag-decompD(1) unitary-diag-carrier(1)*)

**also have**  $\dots = \{x * a \mid a. a \in \text{spectrum } A\}$

**using** *assms unitary-diag-spectrum-eq*[of  $A$ ] *bu real-diag-decompD(1)*

**by** *metis*

**finally show** ?thesis .

qed

**lemma** *spectrum-uminus*:

fixes  $A::\text{complex Matrix.mat}$

assumes *hermitian*  $A$

and  $A \in \text{carrier-mat } n \ n$

and  $0 < n$

shows  $\text{spectrum } (-A) = \{-a \mid a. a \in \text{spectrum } A\}$

**proof** –

**obtain**  $B \ U$  **where**  $bu: \text{real-diag-decomp } A \ B \ U$

**using** *assms hermitian-real-diag-decomp*[of  $A$ ] **by** *auto*

**hence**  $\text{spectrum } (-A) = \text{set } (\text{diag-mat } (-B))$

**using** *assms unitary-diag-spectrum-eq*[of  $-A$ ]

*unitarily-equiv-uminus*[of  $A$ ]

**by** (*meson diagonal-mat-uminus real-diag-decompD uminus-carrier-iff-mat*  
*unitary-diag-def*)

**also have**  $\dots = \{-a \mid a. a \in \text{set } (\text{diag-mat } B)\}$

**using** *assms set-diag-mat-uminus*[of  $B \ n$ ]

**by** (*meson bu real-diag-decompD(1) unitary-diag-carrier(1)*)

**also have**  $\dots = \{-a \mid a. a \in \text{spectrum } A\}$

**using** *assms unitary-diag-spectrum-eq*[of  $A$ ] *bu real-diag-decompD(1)*

**by** *metis*

**finally show** ?thesis .

qed

### 3 Properties of the inner product

#### 3.1 Some analysis complements

**lemma** *add-conj-le*:

**shows**  $z + \text{cnj } z \leq 2 * \text{cmod } z$

**proof** –

**have**  $z: z + \text{cnj } z = 2 * \text{Re } z$  **by** (*simp add: complex-add-cnj*)

**have**  $\text{Re } z \leq \text{cmod } z$  **by** (*simp add: complex-Re-le-cmod*)

**hence**  $2 * \text{complex-of-real } (\text{Re } z) \leq 2 * \text{complex-of-real } (\text{cmod } z)$

**using** *less-eq-complex-def* **by** *auto*

**thus** *?thesis* **using**  $z$  **by** *simp*

**qed**

**lemma** *abs-real*:

**fixes**  $x::\text{complex}$

**assumes**  $x \in \text{Reals}$

**shows**  $\text{abs } x \in \text{Reals}$  **unfolding** *abs-complex-def* **by** *auto*

**lemma** *csqrt-cmod-square*:

**shows**  $\text{csqrt } ((\text{cmod } z)^2) = \text{cmod } z$

**proof** –

**have**  $\text{csqrt } ((\text{cmod } z)^2) = \text{sqrt } (\text{Re } ((\text{cmod } z)^2))$  **by** *force*

**also have**  $\dots = \text{cmod } z$  **by** *simp*

**finally show** *?thesis* .

**qed**

**lemma** *cpx-real-le*:

**fixes**  $z::\text{complex}$

**assumes**  $0 \leq z$

**and**  $0 \leq u$

**and**  $z^2 \leq u^2$

**shows**  $z \leq u$

**proof** –

**have**  $z^{\wedge}2 = \text{Re } (z^{\wedge}2) u^{\wedge}2 = \text{Re } (u^{\wedge}2)$  **using** *assms*

**by** (*metis Im-complex-of-real Im-power-real Re-complex-of-real*  
*complex-eq-iff less-eq-complex-def zero-complex.sel(2)*)**+**

**hence** *rl*:  $\text{Re } (z^{\wedge}2) \leq \text{Re } (u^{\wedge}2)$  **using** *assms less-eq-complex-def* **by** *simp*

**have**  $\text{sqrt } (\text{Re } (z^{\wedge}2)) = z \text{sqrt } (\text{Re } (u^{\wedge}2)) = u$  **using** *assms complex-eqI*  
*less-eq-complex-def* **by** *auto*

**have**  $z = \text{sqrt } (\text{Re } (z^{\wedge}2))$  **using** *assms complex-eqI less-eq-complex-def*  
**by** *auto*

**also have**  $\dots \leq \text{sqrt } (\text{Re } (u^{\wedge}2))$  **using** *rl less-eq-complex-def* **by** *simp*

**finally show**  $z \leq u$  **using** *assms complex-eqI less-eq-complex-def* **by** *auto*

**qed**

**lemma** *mult-conj-real*:

**fixes**  $v::\text{complex}$

**shows**  $v * (\text{conjugate } v) \in \text{Reals}$

**proof** –

**have**  $0 \leq v * (\text{conjugate } v)$  **using** *less-eq-complex-def* **by** *simp*  
**thus** *?thesis* **by** (*simp add: complex-is-Real-iff*)  
**qed**

**lemma** *real-sum-real*:  
**assumes**  $\bigwedge i. i < n \implies ((f\ i)::\text{complex}) \in \text{Reals}$   
**shows**  $(\sum i \in \{0 \dots n\}. f\ i) \in \text{Reals}$   
**using** *assms atLeastLessThan-iff* **by** *blast*

**lemma** *real-mult-re*:  
**assumes**  $a \in \text{Reals}$  **and**  $b \in \text{Reals}$   
**shows**  $\text{Re } (a * b) = \text{Re } a * \text{Re } b$  **using** *assms*  
**by** (*metis Re-complex-of-real Reals-cases of-real-mult*)

**lemma** *complex-positive-Im*:  
**fixes**  $b::\text{complex}$   
**assumes**  $0 \leq b$   
**shows**  $\text{Im } b = 0$   
**by** (*metis assms less-eq-complex-def zero-complex.simps(2)*)

**lemma** *cmod-pos*:  
**fixes**  $z::\text{complex}$   
**assumes**  $0 \leq z$   
**shows**  $\text{cmod } z = z$   
**proof** –  
**have**  $\text{Im } z = 0$  **using** *assms complex-positive-Im* **by** *simp*  
**thus** *?thesis* **using** *complex-norm*  
**by** (*metis assms complex.exhaust-sel complex-of-real-def less-eq-complex-def*  
*norm-of-real*  
*real-sqrt-abs real-sqrt-pow2 real-sqrt-power zero-complex.simps(1)*)  
**qed**

**lemma** *cpx-pos-square-pos*:  
**fixes**  $z::\text{complex}$   
**assumes**  $0 \leq z$   
**shows**  $0 \leq z^2$   
**proof** –  
**have**  $\text{Im } z = 0$  **using** *assms* **by** (*simp add: complex-positive-Im*)  
**hence**  $\text{Re } (z^2) = (\text{Re } z)^2$  **by** *simp*  
**moreover** **have**  $\text{Im } (z^2) = 0$  **by** (*simp add: <Im z = 0>*)  
**ultimately show** *?thesis* **by** (*simp add: less-eq-complex-def*)  
**qed**

**lemma** *cmod-mult-pos*:  
**fixes**  $b::\text{complex}$   
**fixes**  $z::\text{complex}$   
**assumes**  $0 \leq b$   
**shows**  $\text{cmod } (b * z) = \text{Re } b * \text{cmod } z$  **using** *complex-positive-Im*

*Im-complex-of-real Re-complex-of-real abs-of-nonneg assms cmod-Im-le-iff  
less-eq-complex-def norm-mult of-real-0  
by (metis (full-types) cmod-eq-Re)*

**lemma** *cmod-conjugate-square-eq:*  
**fixes** *z::complex*  
**shows**  $cmod (z * (conjugate z)) = z * (conjugate z)$   
**proof** –  
**have**  $0 \leq z * (conjugate z)$   
**using** *conjugate-square-positive less-eq-complex-def* **by** *auto*  
**thus** *?thesis* **using** *cmod-pos* **by** *simp*  
**qed**

**lemma** *pos-sum-gt-comp:*  
**assumes** *finite I*  
**and**  $\bigwedge i. i \in I \implies (0::real) \leq f i$   
**and**  $j \in I$   
**and**  $c < f j$   
**shows**  $c < sum f I$   
**proof** –  
**have**  $c < f j$  **using** *assms* **by** *simp*  
**also have**  $\dots \leq f j + sum f (I - \{j\})$   
**by** (*smt (verit) DiffD1 assms(2) sum-nonneg*)  
**also have**  $\dots = sum f I$  **using** *assms*  
**by** (*simp add: sum-diff1*)  
**finally show** *?thesis* .  
**qed**

**lemma** *pos-sum-le-comp:*  
**assumes** *finite I*  
**and**  $\bigwedge i. i \in I \implies (0::real) \leq f i$   
**and**  $sum f I \leq c$   
**shows**  $\forall i \in I. f i \leq c$   
**proof** (*rule ccontr*)  
**assume**  $\neg (\forall i \in I. f i \leq c)$   
**hence**  $\exists j \in I. c < f j$  **by** *fastforce*  
**from this obtain** *j* **where**  $j \in I$  **and**  $c < f j$  **by** *auto*  
**hence**  $c < sum f I$  **using** *assms pos-sum-gt-comp[of I]* **by** *simp*  
**thus** *False* **using** *assms* **by** *simp*  
**qed**

**lemma** *square-pos-mult-le:*  
**assumes** *finite I*  
**and**  $\bigwedge i. i \in I \implies ((0::real) \leq f i \wedge f i \leq 1)$   
**shows**  $sum (\lambda x. f x * f x) I \leq sum f I$  **using** *assms*

```

proof (induct rule:finite-induct)
case empty
  then show ?case by simp
next
  case (insert x F)
  have sum (λx. f x * f x) (insert x F) = f x * f x + sum (λx. f x * f x) F
    by (simp add: insert)
  also have ... ≤ f x * f x + sum f F using insert by simp
  also have ... ≤ f x + sum f F using insert mult-left-le[of f x f x]
    by simp
  also have ... = sum f (insert x F) using insert by simp
  finally show ?case .
qed

```

```

lemma square-pos-mult-lt:
  assumes finite I
  and  $\bigwedge i. i \in I \implies ((0::real) \leq f i \wedge f i \leq 1)$ 
  and  $j \in I$ 
  and  $f j < 1$ 
  and  $0 < f j$ 
shows sum (λx. f x * f x) I < sum f I using assms
proof -
  have sum (λx. f x * f x) I =
    sum (λx. f x * f x) {j} + sum (λx. f x * f x) (I - {j})
    using assms sum.remove by fastforce
  also have ... = f j * f j + sum (λx. f x * f x) (I - {j}) by simp
  also have ... < f j + sum (λx. f x * f x) (I - {j}) using assms by simp
  also have ... ≤ f j + sum f (I - {j})
    using assms square-pos-mult-le[of I - {j}] by simp
  also have ... = sum f I
    by (metis assms(1) assms(3) sum.remove)
  finally show ?thesis .
qed

```

### 3.2 Inner product results

In particular we prove the triangle inequality, i.e. that for vectors  $u$  and  $v$  we have  $\|u + v\| \leq \|u\| + \|v\|$ .

```

lemma inner-prod-vec-norm-pow2:
  shows (vec-norm v)2 = v · c v using vec-norm-def
  by (metis power2-csqr)

```

```

lemma inner-prod-mult-mat-vec-left:
  assumes v ∈ carrier-vec n
  and w ∈ carrier-vec n'
  and A ∈ carrier-mat m n
  and B ∈ carrier-mat m n'

```

**shows**  $\text{inner-prod } (A *_v v) (B *_v w) =$   
 $\text{inner-prod } (((\text{Complex-Matrix.adjoint } B) * A) *_v v) w$   
**proof** –  
**have**  $\text{inner-prod } (A *_v v) (B *_v w) =$   
 $\text{inner-prod } (\text{Complex-Matrix.adjoint } B *_v (A *_v v)) w$   
**using** *adjoint-def-alter* **by** (*metis* *assms* *mult-mat-vec-carrier*)  
**also have**  $\dots = \text{inner-prod } (((\text{Complex-Matrix.adjoint } B) * A) *_v v) w$   
**proof** –  
**have**  $\text{Complex-Matrix.adjoint } B *_v (A *_v v) =$   
 $((\text{Complex-Matrix.adjoint } B) * A) *_v v$   
**proof** (*rule* *assoc-mult-mat-vec[symmetric]*, (*auto simp add: assms*))  
**show**  $v \in \text{carrier-vec } n$  **using** *assms* **by** *simp*  
**show**  $A \in \text{carrier-mat } (\text{dim-row } B) n$  **using** *assms* **by** *auto*  
**qed**  
**thus** *?thesis* **by** *simp*  
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *rank-1-proj-trace-inner*:  
**fixes**  $A :: 'a::\text{conjugatable-field } \text{Matrix.mat}$  **and**  $v :: 'a \text{ Matrix.vec}$   
**assumes**  $A \in \text{carrier-mat } n n$   
**and**  $v \in \text{carrier-vec } n$   
**shows**  $\text{Complex-Matrix.trace } (A * (\text{rank-1-proj } v)) = \text{Complex-Matrix.inner-prod}$   
 $v (A *_v v)$   
**using** *assms* *trace-outer-prod-right[of A]* **unfolding** *rank-1-proj-def* **by** *simp*

**lemma** *unitary-inner-prod*:  
**assumes**  $v \in \text{carrier-vec } n$   
**and**  $w \in \text{carrier-vec } n$   
**and**  $U \in \text{carrier-mat } n n$   
**and**  $\text{Complex-Matrix.unitary } U$   
**shows**  $\text{inner-prod } (U *_v v) (U *_v w) = \text{inner-prod } v w$   
**proof** –  
**have**  $\text{inner-prod } (U *_v v) (U *_v w) =$   
 $\text{inner-prod } (((\text{Complex-Matrix.adjoint } U) * U) *_v v) w$   
**using** *assms* **by** (*simp add: inner-prod-mult-mat-vec-left*)  
**also have**  $\dots = \text{inner-prod } (1_m n *_v v) w$  **using** *assms* **by** *simp*  
**also have**  $\dots = \text{inner-prod } v w$  **using** *assms* **by** *simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *unitary-vec-norm*:  
**assumes**  $v \in \text{carrier-vec } n$   
**and**  $U \in \text{carrier-mat } n n$   
**and**  $\text{Complex-Matrix.unitary } U$   
**shows**  $\text{vec-norm } (U *_v v) = \text{vec-norm } v$  **using** *unitary-inner-prod* *assms* **unfolding**  
*vec-norm-def*  
**by** *metis*

**lemma** *unitary-col-norm-square*:  
**assumes** *unitary*  $U$   
**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**shows**  $\| \text{Matrix.col } U \ i \|^2 = 1$   
**proof** –  
**define**  $vn::\text{complex Matrix.vec}$  **where**  $vn = \text{unit-vec } n \ i$   
**have**  $\| \text{Matrix.col } U \ i \|^2 = (\text{vec-norm } (\text{Matrix.col } U \ i))^2$  **using** *vec-norm-sq-cpx-vec-length-sq*  
**by** *simp*  
**also have**  $\dots = (\text{vec-norm } vn)^2$  **using** *assms unitary-vec-norm*  
**by** (*metis mat-unit-vec-col unit-vec-carrier vn-def*)  
**also have**  $\dots = \|vn\|^2$  **using** *vec-norm-sq-cpx-vec-length-sq* **by** *simp*  
**also have**  $\dots = 1$  **using** *assms unfolding vn-def* **by** *simp*  
**finally show** *?thesis* **by** *blast*  
**qed**

**lemma** *unitary-col-norm*:  
**assumes** *unitary*  $U$   
**and**  $U \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**shows**  $\| \text{Matrix.col } U \ i \| = 1$  **using** *assms unitary-col-norm-square cpx-vec-length-inner-prod*

*inner-prod-csqrt* **by** (*metis csqrt-1 of-real-eq-1-iff*)

**lemma** *inner-mult-diag-expand*:  
**fixes**  $B::\text{complex Matrix.mat}$   
**assumes** *diagonal-mat*  $B$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $v \in \text{carrier-vec } n$   
**shows**  $\text{inner-prod } (B *_v v) \ v =$   
 $(\sum i \in \{0 \ ..< n\}. (\text{conjugate } (B \ \$\$ (i,i))) * (\text{vec-index } v \ i * (\text{conjugate } (\text{vec-index } v \ i))))$   
**proof** –  
**have**  $idx: \bigwedge i. i < n \implies \text{vec-index } (B *_v v) \ i = B \ \$\$ (i,i) * (\text{vec-index } v \ i)$   
**using** *assms diagonal-mat-mult-vec* **by** *blast*  
**have**  $\text{inner-prod } (B *_v v) \ v =$   
 $(\sum i \in \{0 \ ..< n\}. \text{vec-index } v \ i * \text{vec-index } (\text{conjugate } (B *_v v)) \ i)$   
**unfolding** *Matrix.scalar-prod-def* **using** *assms* **by** *fastforce*  
**also have**  $\dots = (\sum i \in \{0 \ ..< n\}. \text{vec-index } v \ i * \text{conjugate } (\text{vec-index } (B *_v v) \ i))$   
**using** *assms* **by** *force*  
**also have**  $\dots = (\sum i \in \{0 \ ..< n\}. (\text{conjugate } (B \ \$\$ (i,i))) * (\text{vec-index } v \ i * (\text{conjugate } (\text{vec-index } v \ i))))$   
**proof** (*rule sum.cong, simp*)  
**show**  $\bigwedge i. i \in \{0 \ ..< n\} \implies \text{vec-index } v \ i * \text{conjugate } (\text{vec-index } (B *_v v) \ i)$   
 $=$   
 $(\text{conjugate } (B \ \$\$ (i,i))) * (\text{vec-index } v \ i * (\text{conjugate } (\text{vec-index } v \ i)))$   
**by** (*simp add: idx*)

qed  
 finally show ?thesis .  
 qed

lemma inner-mult-diag-expand':

fixes  $B::\text{complex Matrix.mat}$   
 assumes diagonal-mat  $B$   
 and  $B \in \text{carrier-mat } n \ n$   
 and  $v \in \text{carrier-vec } n$   
 shows inner-prod  $v (B *_v v) =$   
 $(\sum i \in \{0 ..< n\}. B \$\$ (i,i) * (\text{vec-index } v \ i * \text{conjugate } (\text{vec-index } v \ i))))$   
 proof -  
 have idx:  $\bigwedge i. i < n \implies \text{vec-index } (B *_v v) \ i = B \$\$ (i,i) * (\text{vec-index } v \ i)$   
 using assms diagonal-mat-mult-vec by blast  
 have inner-prod  $v (B *_v v) =$   
 $(\sum i \in \{0 ..< n\}. \text{vec-index } (B *_v v) \ i * \text{vec-index } (\text{conjugate } v) \ i)$   
 unfolding Matrix.scalar-prod-def using assms by fastforce  
 also have ... =  
 $(\sum i \in \{0 ..< n\}. \text{vec-index } (B *_v v) \ i * \text{conjugate } (\text{vec-index } v \ i))$   
 using assms by force  
 also have ... =  $(\sum i \in \{0 ..< n\}. (B \$\$ (i,i) * (\text{vec-index } v \ i * \text{conjugate } (\text{vec-index } v \ i))))$   
 proof (rule sum.cong, simp)  
 show  $\bigwedge i. i \in \{0 ..< n\} \implies$   
 $\text{vec-index } (B *_v v) \ i * \text{conjugate } (\text{vec-index } v \ i) =$   
 $(B \$\$ (i,i) * (\text{vec-index } v \ i * (\text{conjugate } (\text{vec-index } v \ i))))$   
 by (simp add: idx)  
 qed  
 finally show ?thesis .  
 qed

lemma self-inner-prod-real:

fixes  $v::\text{complex Matrix.vec}$   
 shows Complex-Matrix.inner-prod  $v \ v \in \text{Reals}$   
 proof -  
 have  $\text{Im } (\text{Complex-Matrix.inner-prod } v \ v) = 0$   
 using self-cscalar-prod-geq-0 by simp  
 thus ?thesis using complex-is-Real-iff by auto  
 qed

lemma inner-mult-diag-real:

fixes  $B::\text{complex Matrix.mat}$   
 assumes diagonal-mat  $B$   
 and  $B \in \text{carrier-mat } n \ n$   
 and  $\forall i < n. B \$\$ (i, i) \in \text{Reals}$   
 and  $v \in \text{carrier-vec } n$   
 shows inner-prod  $(B *_v v) \ v \in \text{Reals}$   
 proof -

**have**  $\text{inner-prod } (B *_v v) v =$   
 $(\sum i \in \{0 ..< n\}. (\text{conjugate } (B \$\$ (i,i))) * (\text{vec-index } v i * (\text{conjugate } (\text{vec-index } v i))))$  **using** *inner-mult-diag-expand* *assms*  
**by** *simp*  
**also have**  $\dots \in \text{Reals}$   
**proof** (*rule real-sum-real*)  
**show**  $\bigwedge i. i < n \implies$   
 $\text{conjugate } (B \$\$ (i, i)) * ((\text{vec-index } v i) * \text{conjugate } (\text{vec-index } v i)) \in \mathbb{R}$   
**using** *assms mult-conj-real* **by** *auto*  
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *inner-mult-diag-real'*:  
**fixes**  $B :: \text{complex Matrix.mat}$   
**assumes** *diagonal-mat*  $B$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $\forall i < n. B \$\$ (i, i) \in \text{Reals}$   
**and**  $v \in \text{carrier-vec } n$   
**shows**  $\text{inner-prod } v (B *_v v) \in \text{Reals}$   
**proof** –  
**have**  $\text{inner-prod } v (B *_v v) =$   
 $(\sum i \in \{0 ..< n\}. B \$\$ (i,i) * (\text{vec-index } v i * (\text{conjugate } (\text{vec-index } v i))))$   
**using** *inner-mult-diag-expand'* *assms* **by** *simp*  
**also have**  $\dots \in \text{Reals}$   
**proof** (*rule real-sum-real*)  
**show**  $\bigwedge i. i < n \implies$   
 $B \$\$ (i, i) * ((\text{vec-index } v i) * \text{conjugate } (\text{vec-index } v i)) \in \mathbb{R}$   
**using** *assms mult-conj-real* **by** *auto*  
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *inner-prod-mult-mat-vec-right*:  
**assumes**  $v \in \text{carrier-vec } n$   
**and**  $w \in \text{carrier-vec } n'$   
**and**  $A \in \text{carrier-mat } m \ n$   
**and**  $B \in \text{carrier-mat } m \ n'$   
**shows**  $\text{inner-prod } (A *_v v) (B *_v w) =$   
 $\text{inner-prod } v (((\text{Complex-Matrix.adjoint } A) * B) *_v w)$   
**proof** –  
**have**  $\text{inner-prod } (A *_v v) (B *_v w) =$   
 $\text{inner-prod } ((\text{Complex-Matrix.adjoint } (\text{Complex-Matrix.adjoint } A)) *_v v)$   
 $(B *_v w)$   
**by** (*simp add: Complex-Matrix.adjoint-adjoint*)  
**also have**  $\dots = \text{inner-prod } v ((\text{Complex-Matrix.adjoint } A) *_v (B *_v w))$   
**proof** (*rule adjoint-def-alter[symmetric]*)

**show**  $v \in \text{carrier-vec } n$  **using** *assms* **by** *simp*  
**show**  $B *_v w \in \text{carrier-vec } m$  **using** *assms* **by** *simp*  
**show**  $\text{Complex-Matrix.adjoint } A \in \text{carrier-mat } n \ m$   
**using** *assms*  $\text{adjoint-dim'}$ [of  $A$ ] **by** *simp*  
**qed**  
**also have**  $\dots = \text{inner-prod } v \ (((\text{Complex-Matrix.adjoint } A) * B) *_v w)$   
**using** *assms*  
**proof** –  
**have**  $(\text{Complex-Matrix.adjoint } A) *_v (B *_v w) =$   
 $((\text{Complex-Matrix.adjoint } A) * B) *_v w$   
**proof** (*rule* *assoc-mult-mat-vec*[*symmetric*], (*auto simp add: assms*))  
**show**  $w \in \text{carrier-vec } n'$  **using** *assms* **by** *simp*  
**show**  $B \in \text{carrier-mat } (\text{dim-row } A) \ n'$  **using** *assms* **by** *auto*  
**qed**  
**thus** *?thesis* **by** *simp*  
**qed**  
**finally show** *?thesis* .  
**qed**

**lemma** *Cauchy-Schwarz-complex-vec-norm*:

**assumes**  $\text{dim-vec } x = \text{dim-vec } y$

**shows**  $\text{cmod } (\text{inner-prod } x \ y) \leq \text{vec-norm } x * \text{vec-norm } y$

**proof** –

**have**  $x: x \in \text{carrier-vec } (\text{dim-vec } x)$  **by** *simp*

**moreover have**  $y: y \in \text{carrier-vec } (\text{dim-vec } x)$  **using** *assms* **by** *simp*

**ultimately have**  $(\text{cmod } (\text{inner-prod } x \ y))^2 = \text{inner-prod } x \ y * \text{inner-prod } y \ x$

**using** *complex-norm-square* **by** (*metis* *inner-prod-swap mult-conj-cmod-square*)

**also have**  $\dots \leq \text{inner-prod } x \ x * \text{inner-prod } y \ y$

**using** *Cauchy-Schwarz-complex-vec*  $x \ y$  **by** *blast*

**finally have**  $(\text{cmod } (\text{inner-prod } x \ y))^2 \leq \text{inner-prod } x \ x * \text{inner-prod } y \ y$  .

**hence**  $(\text{cmod } (\text{inner-prod } x \ y))^2 \leq \text{Re } (\text{inner-prod } x \ x) * \text{Re } (\text{inner-prod } y \ y)$

**using** *less-eq-complex-def* **by** *simp*

**hence**  $\text{sqrt } ((\text{cmod } (\text{inner-prod } x \ y))^2) \leq$

$\text{sqrt } (\text{Re } (\text{inner-prod } x \ x) * \text{Re } (\text{inner-prod } y \ y))$

**using** *real-sqrt-le-iff* **by** *blast*

**also have**  $\dots = \text{sqrt } (\text{Re } (\text{inner-prod } x \ x)) * \text{sqrt } (\text{Re } (\text{inner-prod } y \ y))$

**by** (*simp add: real-sqrt-mult*)

**finally have**  $\text{sqrt } ((\text{cmod } (\text{inner-prod } x \ y))^2) \leq$

$\text{sqrt } (\text{Re } (\text{inner-prod } x \ x)) * \text{sqrt } (\text{Re } (\text{inner-prod } y \ y))$  .

**thus** *?thesis* **using** *less-eq-complex-def* **by** (*simp add: vec-norm-def*)

**qed**

**lemma** *vec-norm-triangle-sq*:

**fixes**  $u::\text{complex Matrix.vec}$

**assumes**  $\text{dim-vec } u = \text{dim-vec } v$

**shows**  $(\text{vec-norm } (u+v))^2 \leq (\text{vec-norm } u + \text{vec-norm } v)^2$

**proof** –

**have**  $(\text{vec-norm } (u+v))^2 = \text{inner-prod } (u+v) \ (u+v)$

**by** (*simp add: inner-prod-vec-norm-pow2*)

**also have** ... =  $\text{inner-prod } u \ u + \text{inner-prod } u \ v + \text{inner-prod } v \ u + \text{inner-prod } v \ v$   
**using** *assms add-scalar-prod-distrib conjugate-add-vec*  
**by** (*smt (verit) ab-semigroup-add-class.add-ac(1) carrier-vec-dim-vec dim-vec-conjugate index-add-vec(2) scalar-prod-add-distrib*)  
**also have** ... =  $(\text{vec-norm } u)^2 + \text{inner-prod } u \ v + \text{inner-prod } v \ u + (\text{vec-norm } v)^2$   
**by** (*simp add: inner-prod-vec-norm-pow2*)  
**also have** ...  $\leq (\text{vec-norm } u)^2 + 2 * \text{cmod } (\text{inner-prod } u \ v) + (\text{vec-norm } v)^2$   
**by** (*metis add-conj-le add-left-mono add-right-mono assms carrier-vec-dim-vec conjugate-complex-def inner-prod-swap is-num-normalize(1)*)  
**also have** ...  $\leq (\text{vec-norm } u)^2 + 2 * ((\text{vec-norm } u) * (\text{vec-norm } v)) + (\text{vec-norm } v)^2$   
**using** *Cauchy-Schwarz-complex-vec-norm[of u v] assms less-eq-complex-def*  
**by** *auto*  
**also have** ... =  $(\text{vec-norm } u + \text{vec-norm } v)^2$  **by** (*simp add: power2-sum*)  
**finally show** ?thesis .  
**qed**

**lemma** *vec-norm-triangle*:  
**fixes** *u::complex Matrix.vec*  
**assumes**  $\text{dim-vec } u = \text{dim-vec } v$   
**shows**  $\text{vec-norm } (u + v) \leq \text{vec-norm } u + \text{vec-norm } v$   
**proof** (*rule cpx-real-le*)  
**show**  $(\text{vec-norm } (u + v))^2 \leq (\text{vec-norm } u + \text{vec-norm } v)^2$   
**using** *assms vec-norm-triangle-sq* **by** *simp*  
**show**  $0 \leq \text{vec-norm } (u + v)$  **using** *vec-norm-geq-0* **by** *simp*  
**show**  $0 \leq \text{vec-norm } u + \text{vec-norm } v$  **using** *vec-norm-geq-0* **by** *simp*  
**qed**

## 4 Matrix decomposition

**lemma** (*in cpx-sq-mat*) *sum-decomp-cols*:  
**fixes** *A::complex Matrix.mat*  
**assumes** *hermitian A*  
**and**  $A \in \text{fc-mats}$   
**and** *unitary-diag A B U*  
**shows**  $\text{sum-mat } (\lambda i. (\text{diag-mat } B \ ! \ i) \cdot_m \text{rank-1-proj } (\text{Matrix.col } U \ i)) \{.. < \text{dimR}\} = A$   
**proof** –  
**have** *similar-mat-wit A B U (Complex-Matrix.adjoint U)  $\wedge$  diagonal-mat B  $\wedge$  unitary U*  
**by** (*metis assms(3) unitary-diagD(1) unitary-diagD(2) unitary-diagD(3)*)  
**hence** *A: A = U \* B \* (Complex-Matrix.adjoint U)* **and** *dB: diagonal-mat B*  
**and** *dimB: B  $\in$  carrier-mat dimR dimR* **and** *dimP: U  $\in$  carrier-mat dimR dimR*  
**and** *unit: unitary U*  
**unfolding** *similar-mat-wit-def Let-def* **using** *assms fc-mats-carrier* **by** *auto*

**have**  $pz: \bigwedge i. i < \dim R \implies (\text{Matrix.col } U \ i) = \text{zero-col } U \ i$   
**unfolding** *zero-col-def* **by** *simp*  
**have**  $\text{sum-mat } (\lambda i. (\text{diag-mat } B \ ! \ i) \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } U \ i)) \ \{.. < \dim R\} =$   
 $U * B * \text{Complex-Matrix.adjoint } U$   
**proof** (*rule weighted-sum-rank-1-proj-unitary*)  
**show** *diagonal-mat B* **using** *dB* .  
**show** *Complex-Matrix.unitary U* **using** *unit* .  
**show**  $U \in \text{fc-mats}$  **using** *fc-mats-carrier dim-eq dimP* **by** *simp*  
**show**  $B \in \text{fc-mats}$  **using** *fc-mats-carrier dim-eq dimB* **by** *simp*  
**qed**  
**thus** *?thesis* **using** *A* **by** *simp*  
**qed**

**lemma** *unitary-col-inner-prod*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $0 < n$   
**and** *Complex-Matrix.unitary A*  
**and**  $j < n$   
**and**  $k < n$   
**shows**  $\text{Complex-Matrix.inner-prod } (\text{Matrix.col } A \ j) \ (\text{Matrix.col } A \ k) =$   
 $(1_m \ n) \ \$\$ \ (j,k)$   
**proof** –  
**have**  $\text{Complex-Matrix.inner-prod } (\text{Matrix.col } A \ j) \ (\text{Matrix.col } A \ k) =$   
 $(\text{Complex-Matrix.adjoint } A * A) \ \$\$ \ (j, k)$   
**using** *inner-prod-adjoint-comp[of A n A]* *assms*  
**by** *simp*  
**also have**  $\dots = (1_m \ n) \ \$\$ \ (j,k)$  **using** *assms*  
**unfolding** *Complex-Matrix.unitary-def*  
**by** (*simp add: assms(3)*)  
**finally show** *?thesis* .  
**qed**

**lemma** (*in cpx-sq-mat*) *sum-mat-ortho-proj*:  
**assumes** *finite I*  
**and**  $j \in I$   
**and**  $A \ j * A \ j = A \ j$   
**and**  $\bigwedge i. i \in I \implies A \ i \in \text{fc-mats}$   
**and**  $\bigwedge i . i \in I \implies i \neq j \implies A \ i * (A \ j) = (0_m \ \dim R \ \dim R)$   
**shows**  $(\text{sum-mat } A \ I) * (A \ j) = (A \ j)$  **using** *assms*  
**proof** (*induct rule:finite-induct*)  
**case** *empty*  
**then show** *?case* **using** *dim-eq* **by** *auto*  
**next**  
**case** (*insert x F*)  
**have**  $(\text{sum-mat } A \ (\text{insert } x \ F)) * (A \ j) =$   
 $(A \ x + \text{sum-mat } A \ F) * (A \ j)$  **using** *insert sum-mat-insert[of A]*  
**by** (*simp add: image-subset-iff*)  
**also have**  $\dots = A \ x * (A \ j) + \text{sum-mat } A \ F * (A \ j)$

```

proof (rule add-mult-distrib-mat)
  show  $A x \in \text{carrier-mat } \dim R \ \dim C$  using insert fc-mats-carrier by simp
  show  $\text{sum-mat } A \ F \in \text{carrier-mat } \dim R \ \dim C$  using insert
    by (metis insert-iff local.fc-mats-carrier sum-mat-carrier)
  show  $A \ j \in \text{carrier-mat } \dim C \ \dim C$  using insert dim-eq fc-mats-carrier by
force
qed
also have  $\dots = A \ j$ 
proof (cases  $x = j$ )
  case True
    hence  $j \notin F$  using insert by auto
    hence  $\text{sum-mat } A \ F * A \ j = 0_m \ \dim R \ \dim R$  using insert sum-mat-left-ortho-zero[of
F A A j]
      using True ball-insert dim-eq by auto
    thus ?thesis using insert True dim-eq fc-mats-carrier by auto
  next
    case False
      hence  $j \in F$  using insert by auto
      moreover have  $\bigwedge i. i \in F \implies A \ i \in \text{fc-mats}$  using insert by simp
      moreover have  $\bigwedge i. i \in F \implies i \neq j \implies A \ i * A \ j = 0_m \ \dim R \ \dim R$  using
insert by simp
      ultimately have  $\text{sum-mat } A \ F * A \ j = A \ j$  using insert by simp
      thus ?thesis using False dim-eq fc-mats-carrier insert by auto
    qed
  finally show ?case .
qed

lemma (in cpx-sq-mat) sum-mat-ortho-one:
  assumes finite I
  and  $j \in I$ 
  and  $B \in \text{fc-mats}$ 
  and  $\bigwedge i. i \in I \implies A \ i \in \text{fc-mats}$ 
  and  $\bigwedge i. i \in I \implies i \neq j \implies A \ i * B = (0_m \ \dim R \ \dim R)$ 
  shows  $(\text{sum-mat } A \ I) * B = A \ j * B$  using assms
proof (induct rule:finite-induct)
  case empty
  then show ?case using dim-eq by auto
next
  case (insert x F)
  have  $(\text{sum-mat } A \ (\text{insert } x \ F)) * B =$ 
 $(A \ x + \text{sum-mat } A \ F) * B$  using insert sum-mat-insert[of A]
    by (simp add: image-subset-iff)
  also have  $\dots = A \ x * B + \text{sum-mat } A \ F * B$ 
proof (rule add-mult-distrib-mat)
  show  $A \ x \in \text{carrier-mat } \dim R \ \dim C$  using insert fc-mats-carrier by simp
  show  $\text{sum-mat } A \ F \in \text{carrier-mat } \dim R \ \dim C$  using insert
    by (metis insert-iff local.fc-mats-carrier sum-mat-carrier)
  show  $B \in \text{carrier-mat } \dim C \ \dim C$  using insert dim-eq fc-mats-carrier by force
qed

```

**also have**  $\dots = A j * B$   
**proof** (*cases*  $x = j$ )  
  **case** *True*  
    **hence**  $j \notin F$  **using** *insert by auto*  
    **hence**  $\text{sum-mat } A F * B = 0_m \text{ dimR dimR}$  **using** *insert sum-mat-left-ortho-zero[of*  
*F A B]*  
    **using** *True ball-insert dim-eq by auto*  
    **thus** *?thesis using insert True dim-eq fc-mats-carrier*  
    **by** (*metis Complex-Matrix.right-add-zero-mat cpx-sq-mat-mult*)  
  **next**  
  **case** *False*  
    **hence**  $j \in F$  **using** *insert by auto*  
    **moreover have**  $\bigwedge i. i \in F \implies A i \in \text{fc-mats}$  **using** *insert by simp*  
    **moreover have**  $\bigwedge i. i \in F \implies i \neq j \implies A i * B = 0_m \text{ dimR dimR}$  **using**  
*insert by simp*  
    **ultimately have**  $\text{sum-mat } A F * B = A j * B$  **using** *insert by simp*  
    **thus** *?thesis using False dim-eq fc-mats-carrier insert*  
    **by** (*metis add-zero cpx-sq-mat-mult insertI1*)  
  **qed**  
  **finally show** *?case .*  
**qed**

**lemma** *unitarily-equiv-rank-1-proj-col-carrier:*  
  **assumes**  $A \in \text{carrier-mat } n \ n$   
  **and** *unitarily-equiv*  $A \ B \ U$   
  **and**  $i < n$   
  **shows**  $\text{rank-1-proj } (\text{Matrix.col } U \ i) \in \text{carrier-mat } n \ n$   
  **using** *rank-1-proj-col-carrier assms*  
  **by** (*metis carrier-matD(1) carrier-matD(2) unitarily-equiv-carrier(2)*)

**lemma** *decomp-eigenvector:*  
  **fixes**  $A::\text{complex Matrix.mat}$   
  **assumes**  $A \in \text{carrier-mat } n \ n$   
  **and**  $0 < n$   
  **and** *hermitian*  $A$   
  **and** *unitary-diag*  $A \ B \ U$   
  **and**  $j < n$   
  **shows**  $\text{Complex-Matrix.trace } (A * (\text{rank-1-proj } (\text{Matrix.col } U \ j))) = B \ \$(j,j)$   
  **proof** –  
  **define**  $\text{fc}::\text{complex Matrix.mat set}$  **where**  $\text{fc} = \text{carrier-mat } n \ n$   
  **interpret** *cpx-sq-mat*  $n \ n \ \text{fc}$   
  **proof**  
  **show**  $0 < n$  **using** *assms by simp*  
  **qed** (*auto simp add: fc-def*)  
  **have**  $\text{rf}: \bigwedge i. i < n \implies \text{rank-1-proj } (\text{Matrix.col } U \ i) \in \text{fc}$  **using**  
  *assms unitarily-equiv-rank-1-proj-col-carrier*  
  **by** (*metis fc-def unitary-diag-imp-unitarily-equiv*)  
  **hence**  $\text{sm}: \bigwedge i. i < n \implies \text{diag-mat } B \ ! \ i \cdot_m \text{rank-1-proj } (\text{Matrix.col } U \ i) \in \text{fc}$   
  **using** *fc-mats-carrier dim-eq by simp*

```

have A * (rank-1-proj (Matrix.col U j)) =
  (sum-mat (λi. (diag-mat B ! i) ·m (rank-1-proj (Matrix.col U i))) {..n) *
  (rank-1-proj (Matrix.col U j)) using assms sum-decomp-cols
unfolding fc-def by simp
also have ... = diag-mat B ! j ·m rank-1-proj (Matrix.col U j) *
  rank-1-proj (Matrix.col U j)
proof (rule sum-mat-ortho-one, (auto simp add: assms))
show rank-1-proj (Matrix.col U j) ∈ fc by (simp add: assms rf)
show ∧i. i < n ⇒ diag-mat B ! i ·m rank-1-proj (Matrix.col U i) ∈ fc
  by (simp add: sm)
show ∧i. i < n ⇒ i ≠ j ⇒
  diag-mat B ! i ·m rank-1-proj (Matrix.col U i) *
  rank-1-proj (Matrix.col U j) = 0m n n
proof -
  fix i
  assume i < n and i ≠ j
  define OP where OP = outer-prod (Matrix.col U i) (Matrix.col U j)
  have cm: OP ∈ carrier-mat n n unfolding OP-def
  proof (rule outer-prod-dim)
    have dim-row U = n
      using assms unitary-diag-carrier(2) fc-mats-carrier
      by (metis carrier-matD(1))
    thus Matrix.col U i ∈ carrier-vec n using ⟨i < n⟩
      by (simp add: carrier-vecI)
    show Matrix.col U j ∈ carrier-vec n using assms ⟨dim-row U = n⟩
      by (simp add: carrier-vecI)
  qed
  have rank-1-proj (Matrix.col U i) * rank-1-proj (Matrix.col U j) =
    1m n $$ (i, j) ·m outer-prod (Matrix.col U i) (Matrix.col U j)
  proof (rule rank-1-proj-unitary, (auto simp add: ⟨i < n⟩ assms))
    show U ∈ fc using assms unitary-diag-carrier(2)
      fc-mats-carrier by simp
    show Complex-Matrix.unitary U using assms unitary-diag-carrier
      unitary-diagD(3) by blast
  qed
  also have ... = 0 ·m outer-prod (Matrix.col U i) (Matrix.col U j)
    using ⟨i ≠ j⟩
    by (metis ⟨i < n⟩ assms(5) index-one-mat(1))
  also have ... = 0m n n using cm smult-zero unfolding OP-def by auto
  finally show diag-mat B ! i ·m rank-1-proj (Matrix.col U i) *
    rank-1-proj (Matrix.col U j) = 0m n n
    by (metis ⟨i < n⟩ ⟨rank-1-proj (Matrix.col U j) ∈ fc⟩
      fc-mats-carrier mult-smult-assoc-mat rf smult-zero-mat)
  qed
qed
also have ... = diag-mat B ! j ·m rank-1-proj (Matrix.col U j)
proof -
  have diag-mat B ! j ·m rank-1-proj (Matrix.col U j) *
    rank-1-proj (Matrix.col U j) =

```

```

      diag-mat B ! j ·m (rank-1-proj (Matrix.col U j) *
        rank-1-proj (Matrix.col U j))
    proof (rule mult-smult-assoc-mat)
      show rank-1-proj (Matrix.col U j) ∈ carrier-mat n n using ⟨j < n⟩ rf
        fc-mats-carrier by simp
      show rank-1-proj (Matrix.col U j) ∈ carrier-mat n n
        using assms rf fc-mats-carrier by simp
    qed
    moreover have rank-1-proj (Matrix.col U j) * rank-1-proj (Matrix.col U j) =
      rank-1-proj (Matrix.col U j)
    proof (rule rank-1-proj-unitary-eq, (auto simp add: assms))
      show U ∈ fc using assms unitary-diag-carrier(2)
        using fc-mats-carrier by simp
      show Complex-Matrix.unitary U using assms unitary-diagD by blast
    qed
    ultimately show ?thesis by simp
  qed
  finally have A * (rank-1-proj (Matrix.col U j)) =
    diag-mat B ! j ·m rank-1-proj (Matrix.col U j) .
  hence Complex-Matrix.trace (A * (rank-1-proj (Matrix.col U j))) =
    diag-mat B ! j * Complex-Matrix.trace (rank-1-proj (Matrix.col U j))
    using ⟨j < n⟩ rf fc-mats-carrier trace-smult dim-eq by auto
  also have ... = diag-mat B ! j
  proof -
    have Complex-Matrix.trace (rank-1-proj (Matrix.col U j)) = 1
    proof (rule rank-1-proj-trace)
      show ‖Matrix.col U j‖ = 1 using unitary-col-norm[of U n j] assms
        unitary-diag-carrier(2) fc-mats-carrier
        by (metis unitary-diagD(3))
    qed
    thus ?thesis by simp
  qed
  also have ... = B $$ (j,j)
  proof -
    have dim-row B = n using unitary-diag-carrier(1) assms fc-mats-carrier
      by (metis carrier-matD(1))
    thus ?thesis using assms unfolding diag-mat-def by simp
  qed
  finally show ?thesis .
qed

lemma positive-unitary-diag-pos:
  fixes A::complex Matrix.mat
  assumes A ∈ carrier-mat n n
  and Complex-Matrix.positive A
  and unitary-diag A B U
  and j < n
  shows 0 ≤ B $$ (j, j)
  proof -

```

```

define fc::complex Matrix.mat set where fc = carrier-mat n n
interpret cpx-sq-mat n n fc
proof
  show  $0 < n$  using assms by simp
qed (auto simp add: fc-def)
define Uj where Uj = Matrix.col U j
have dim-row U = n using assms unitary-diag-carrier(2) by blast
hence uj: Matrix.col U j ∈ carrier-vec n by (simp add: carrier-vecI)
have hermitian A using assms positive-is-hermitian by simp
have  $0 \leq \text{Complex-Matrix.inner-prod } Uj (A *_v Uj)$  using assms Complex-Matrix.positive-def
  by (metis Uj-def ‹dim-row U = n› carrier-matD(2) dim-col)
also have  $\dots = \text{Complex-Matrix.trace } (A * (\text{rank-1-proj } Uj))$  using rank-1-proj-trace-inner
uj
  assms unfolding Uj-def by metis
also have  $\dots = B \text{ $$$ } (j,j)$  using decomp-eigenvector assms
  ‹hermitian A› unfolding Uj-def fc-def by simp
finally show ?thesis .
qed

```

**lemma** *unitary-diag-trace-mult-sum:*

```

fixes A::complex Matrix.mat
assumes A ∈ carrier-mat n n
and C ∈ carrier-mat n n
and hermitian A
and unitary-diag A B U
and  $0 < n$ 
shows  $\text{Complex-Matrix.trace } (C * A) =$ 
   $(\sum i = 0 ..< n. B \text{ $$$ } (i,i) * \text{Complex-Matrix.trace } (C * \text{rank-1-proj } (\text{Matrix.col } U i)))$ 

```

**proof** –

```

define fc::complex Matrix.mat set where fc = carrier-mat n n
interpret cpx-sq-mat n n fc
proof
  show  $0 < n$  using assms by simp
qed (auto simp add: fc-def)
have rf: ‹∧i. i < n ⇒ rank-1-proj (Matrix.col U i) ∈ carrier-mat n n
  using assms unitary-diag-imp-unitarily-equiv
  unitarily-equiv-rank-1-proj-col-carrier
  unfolding fc-def by blast
have  $C * A =$ 
   $C * (\text{sum-mat } (\lambda i. (\text{diag-mat } B ! i) \cdot_m$ 
     $\text{rank-1-proj } (\text{Matrix.col } U i)) \{..< n\})$ 
  using sum-decomp-cols assms ‹hermitian A›
  unfolding fc-def by simp
also have  $\dots = \text{sum-mat } (\lambda i. C * ((\text{diag-mat } B ! i) \cdot_m$ 
   $\text{rank-1-proj } (\text{Matrix.col } U i))) \{..< n\}$ 
  by (rule sum-mat-distrib-left[symmetric],
  (auto simp add: assms rf smult-mem fc-def))
also have  $\dots = \text{sum-mat } (\lambda i. (\text{diag-mat } B ! i) \cdot_m$ 

```

```

    (C * rank-1-proj (Matrix.col U i)) {.. $n$ }
proof (rule sum-mat-cong,
    (auto simp add: rf smult-mem assms unitarily-equiv-rank-1-proj-col-carrier
    fc-def))
show  $\bigwedge i. i < n \implies \text{diag-mat } B ! i \cdot_m (C * \text{rank-1-proj } (Matrix.col U i)) \in$ 
    carrier-mat  $n n$ 
using assms unitarily-equiv-rank-1-proj-col-carrier cpx-sq-mat-mult smult-mem
by (simp add: rf)
show  $\bigwedge i. i < n \implies C * (\text{diag-mat } B ! i \cdot_m \text{rank-1-proj } (Matrix.col U i)) \in$ 
    carrier-mat  $n n$ 
using assms unitarily-equiv-rank-1-proj-col-carrier cpx-sq-mat-mult
    smult-mem by (simp add: rf)
show  $\bigwedge i. i < n \implies C * (\text{diag-mat } B ! i \cdot_m \text{rank-1-proj } (Matrix.col U i)) =$ 
    diag-mat  $B ! i \cdot_m (C * \text{rank-1-proj } (Matrix.col U i))$ 
by (metis assms(2) fc-mats-carrier mult-smult-distrib rf)
qed
finally have  $ceq: C * A = \text{sum-mat } (\lambda i. (\text{diag-mat } B ! i) \cdot_m$ 
    (C * rank-1-proj (Matrix.col U i))) {.. $n$ } .
have Complex-Matrix.trace (sum-mat ( $\lambda i. (\text{diag-mat } B ! i) \cdot_m$ 
    (C * rank-1-proj (Matrix.col U i))) {.. $n$ }) = ( $\sum i = 0 ..< n.$ 
    Complex-Matrix.trace ((diag-mat B ! i)  $\cdot_m$ 
    (C * rank-1-proj (Matrix.col U i))))
by (smt (verit) assms(2) atLeast0LessThan cpx-sq-mat-mult cpx-sq-mat-smult
    finite-lessThan
    lessThan-iff rf sum.cong trace-sum-mat fc-def)
also have ... = ( $\sum i = 0 ..< n. (\text{diag-mat } B ! i) *$ 
    Complex-Matrix.trace (C * rank-1-proj (Matrix.col U i)))
proof (rule sum.cong, simp)
show  $\bigwedge x. x \in \{0..<n\} \implies$ 
    Complex-Matrix.trace (diag-mat B ! x  $\cdot_m$  (C * rank-1-proj (Matrix.col U
    x))) =
    diag-mat B ! x * Complex-Matrix.trace (C * rank-1-proj (Matrix.col U x))
using trace-smult
by (metis assms(2) atLeastLessThan-iff cpx-sq-mat-mult fc-mats-carrier rf)
qed
also have ... = ( $\sum i = 0 ..< n. B \text{ $$ } (i,i) *$ 
    Complex-Matrix.trace (C * rank-1-proj (Matrix.col U i)))
proof (rule sum.cong, simp)
have B  $\in$  carrier-mat  $n n$  using unitary-diag-carrier(1) assms
    fc-mats-carrier dim-eq by simp
hence  $\bigwedge x. x \in \{0..<n\} \implies \text{diag-mat } B ! x = B \text{ $$ } (x,x)$  unfolding diag-mat-def
by simp
thus  $\bigwedge x. x \in \{0..<n\} \implies$ 
    diag-mat B ! x * Complex-Matrix.trace (C * rank-1-proj (Matrix.col U x))
    =
    B $$ (x, x) * Complex-Matrix.trace (C * rank-1-proj (Matrix.col U x)) by
    simp
qed
finally have Complex-Matrix.trace

```

$(\text{sum-mat } (\lambda i. (\text{diag-mat } B ! i) \cdot_m (C * \text{rank-1-proj } (\text{Matrix.col } U i))) \{..< n\}) =$   
 $(\sum i = 0 ..< n. B \$\$ (i,i) * \text{Complex-Matrix.trace } (C * \text{rank-1-proj } (\text{Matrix.col } U i))) .$   
**thus ?thesis using ceq by simp**  
**qed**

**lemma** *unitarily-equiv-trace:*

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

**and** *unitarily-equiv*  $A \ B \ U$

**shows**  $\text{Complex-Matrix.trace } A = \text{Complex-Matrix.trace } B$

**proof** –

**have**  $\text{Complex-Matrix.trace } A = \text{Complex-Matrix.trace } (U * B * (\text{Complex-Matrix.adjoint } U))$

**using** *assms unitarily-equiv-eq[of A] unitary-diag-imp-unitarily-equiv[of A]* **by simp**

**also have**  $... = \text{Complex-Matrix.trace } (\text{Complex-Matrix.adjoint } U * (U * B))$

**using** *trace-comm assms*

**by** (*metis adjoint-dim' carrier-matD(2) carrier-matI index-mult-mat(2) index-mult-mat(3) unitarily-equiv-carrier(1) unitarily-equiv-carrier(2)*)

**also have**  $... = \text{Complex-Matrix.trace } B$  **using** *assms*

**by** (*smt (verit, ccfv-threshold) assoc-mult-mat carrier-matD(1) left-mult-one-mat' unitarily-equivD(1) unitarily-equiv-adjoint*

*unitarily-equiv-carrier(1,2) unitary-simps(1)*)

**finally show** *?thesis .*

**qed**

**lemma** *unitarily-equiv-trace':*

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

**and** *unitarily-equiv*  $A \ B \ U$

**shows**  $\text{Complex-Matrix.trace } A = (\sum i = 0 ..< \text{dim-row } A. B \$\$ (i,i))$

**proof** –

**have**  $\text{Complex-Matrix.trace } A = \text{Complex-Matrix.trace } B$  **using** *assms unitarily-equiv-trace[of A]*

**by** (*meson unitary-diag-imp-unitarily-equiv*)

**also have**  $... = (\sum i = 0 ..< \text{dim-row } A. B \$\$ (i,i))$  **using** *assms*

**by** (*metis Complex-Matrix.trace-def carrier-matD(1) unitarily-equiv-carrier(1)*)

**finally show** *?thesis .*

**qed**

**lemma** *positive-decomp-cmod-le:*

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

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

**and**  $C \in \text{carrier-mat } n \ n$

**and**  $0 < n$

**and** *Complex-Matrix.positive*  $A$

**and** *unitary-diag*  $A \ B \ U$

**and**  $\bigwedge i. i < n \implies \text{cmod } (\text{Complex-Matrix.trace } (C * \text{rank-1-proj } (\text{Matrix.col } U i))) \leq M$

**shows**  $cmod (Complex-Matrix.trace (C * A)) \leq Re (Complex-Matrix.trace A) * M$   
**proof** –  
**have**  $dim-row B = n$  **using** *assms unitary-diag-carrier(1)*  
**by** (*metis carrier-matD(1)*)  
**have** *hermitian A* **using** *assms positive-is-hermitian* **by** *simp*  
**hence**  $cmod (Complex-Matrix.trace (C * A)) =$   
 $cmod (\sum i = 0 ..< n. B\$\$(i,i) * Complex-Matrix.trace (C * rank-1-proj$   
 $(Matrix.col U i)))$   
**using** *assms unitary-diag-trace-mult-sum* **by** *simp*  
**also have**  $... \leq (\sum i = 0 ..< n.$   
 $cmod (B\$\$(i,i) * Complex-Matrix.trace (C * rank-1-proj (Matrix.col U i))))$   
**by** (*simp add: sum-norm-le*)  
**also have**  $... = (\sum i = 0 ..< n.$   
 $Re (B\$\$(i,i) * cmod (Complex-Matrix.trace (C * rank-1-proj (Matrix.col U$   
 $i))))$   
**proof** (*rule sum.cong, simp*)  
**show**  $\bigwedge x. x \in \{0..<n\} \implies$   
 $cmod (B \ \$\$ (x, x) * Complex-Matrix.trace (C * rank-1-proj (Matrix.col U$   
 $x))) =$   
 $Re (B \ \$\$ (x, x) * cmod (Complex-Matrix.trace (C * rank-1-proj (Matrix.col$   
 $U x)))$   
**using** *cmod-mult-pos positive-unitary-diag-pos assms* **by** (*metis atLeast-LessThan-iff*)  
**qed**  
**also have**  $... \leq (\sum i = 0 ..< n. Re (B\$\$(i,i) * M)$   
**proof** –  
**have**  $\bigwedge i. i < n \implies 0 \leq Re (B\$\$(i,i))$  **using** *assms positive-unitary-diag-pos*  
*less-eq-complex-def* **by** *simp*  
**thus** *?thesis* **using** *assms* **by** (*meson atLeastLessThan-iff mult-left-mono sum-mono*)  
**qed**  
**also have**  $... = (\sum i = 0 ..< n. Re (B\$\$(i,i))) * M$  **by** (*simp add: sum-distrib-right*)  
**also have**  $... = Re (\sum i = 0 ..< n. B\$\$(i,i) * M)$  **by** (*metis Re-sum*)  
**also have**  $... = Re (Complex-Matrix.trace B) * M$  **unfolding** *Complex-Matrix.trace-def*  
**using**  $\langle dim-row B = n \rangle$  **by** *simp*  
**finally show** *?thesis* **using** *assms unitarily-equiv-trace[of A]*  
**by** (*metis unitary-diag-imp-unitarily-equiv*)  
**qed**  
**end**

**theory** *Commuting-Hermitian* **imports** *Spectral-Theory-Complements* *Commuting-Hermitian-Misc*  
*Projective-Measurements.Linear-Algebra-Complements*  
*Projective-Measurements.Projective-Measurements* **begin**

## 5 Additional results on block decompositions of matrices

### 5.1 Split block results

**lemma** *split-block-diag-carrier*:

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

**and**  $a \leq n$

**and** *split-block*  $D \ a \ a = (D1, D2, D3, D4)$

**shows**  $D1 \in \text{carrier-mat } a \ a \ D4 \in \text{carrier-mat } (n-a) \ (n-a)$

**proof** –

**show**  $D1 \in \text{carrier-mat } a \ a$  **using** *assms unfolding split-block-def*

**by** (*metis Pair-inject mat-carrier*)

**show**  $D4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms unfolding split-block-def*

**by** (*metis Pair-inject carrier-matD(1) carrier-matD(2) mat-carrier*)

**qed**

**lemma** *split-block-diagonal*:

**assumes** *diagonal-mat*  $D$

**and**  $D \in \text{carrier-mat } n \ n$

**and**  $a \leq n$

**and** *split-block*  $D \ a \ a = (D1, D2, D3, D4)$

**shows** *diagonal-mat*  $D1 \wedge \text{diagonal-mat } D4$  **unfolding** *diagonal-mat-def*

**proof** (*intro allI conjI impI*)

**have**  $D1 \in \text{carrier-mat } a \ a$  **using** *assms unfolding split-block-def Let-def*

**by** *fastforce*

**fix**  $i \ j$

**assume**  $i < \text{dim-row } D1$

**and**  $j < \text{dim-col } D1$

**and**  $i \neq j$

**have**  $D1 \ \$\$ (i,j) = D \ \$\$ (i,j)$  **using** *assms unfolding split-block-def Let-def*

**using**  $\langle i < \text{dim-row } D1 \rangle \langle j < \text{dim-col } D1 \rangle$  **by** *fastforce*

**also have**  $\dots = 0$  **using** *assms*  $\langle i \neq j \rangle \langle D1 \in \text{carrier-mat } a \ a \rangle$

$\langle i < \text{dim-row } D1 \rangle \langle j < \text{dim-col } D1 \rangle$  **unfolding** *diagonal-mat-def* **by** *fastforce*

**finally show**  $D1 \ \$\$ (i,j) = 0$  .

**next**

**have**  $D4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms*

**unfolding** *split-block-def Let-def* **by** *fastforce*

**fix**  $i \ j$

**assume**  $i < \text{dim-row } D4$

**and**  $j < \text{dim-col } D4$

**and**  $i \neq j$

**have**  $D4 \ \$\$ (i,j) = D \ \$\$ (i + a, j + a)$  **using** *assms unfolding split-block-def*

*Let-def*

**using**  $\langle i < \text{dim-row } D4 \rangle \langle j < \text{dim-col } D4 \rangle$  **by** *fastforce*

**also have**  $\dots = 0$  **using** *assms*  $\langle i \neq j \rangle \langle D4 \in \text{carrier-mat } (n-a) \ (n-a) \rangle$

$\langle i < \text{dim-row } D4 \rangle \langle j < \text{dim-col } D4 \rangle$  **unfolding** *diagonal-mat-def* **by** *fastforce*

**finally show**  $D4 \ \$\$ (i,j) = 0$  .

**qed**

**lemma** *split-block-times-diag-index*:  
**fixes**  $B::'a::\text{comm-ring Matrix.mat}$   
**assumes** *diagonal-mat*  $D$   
**and**  $D \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $a \leq n$   
**and** *split-block*  $B \ a \ a = (B1, B2, B3, B4)$   
**and** *split-block*  $D \ a \ a = (D1, D2, D3, D4)$   
**and**  $i < \text{dim-row } (D4 * B4)$   
**and**  $j < \text{dim-col } (D4 * B4)$   
**shows**  $(B4 * D4) \ \#\# \ (i, j) = (B*D) \ \#\# \ (i+a, j+a)$   
 $(D4 * B4) \ \#\# \ (i, j) = (D*B) \ \#\# \ (i+a, j+a)$   
**proof** –  
**have**  $d4: D4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms*  
*split-block(4)[of D]* **by** *simp*  
**have**  $b4: B4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms*  
*split-block(4)[of B]* **by** *simp*  
**have** *diagonal-mat*  $D4$  **using** *assms* *split-block-diagonal[of D]* **by** *blast*  
**have**  $i < n-a$  **using**  $\langle i < \text{dim-row } (D4 * B4) \rangle$   $b4 \ d4$  **by** *simp*  
**have**  $j < n-a$  **using**  $\langle j < \text{dim-col } (D4 * B4) \rangle$   $b4 \ d4$  **by** *simp*  
**have**  $(B4 * D4) \ \#\# \ (i, j) = D4 \ \#\# \ (j, j) * B4 \ \#\# \ (i, j)$   
**proof** (*rule diagonal-mat-mult-index'*)  
**show** *diagonal-mat*  $D4$  **using**  $\langle \text{diagonal-mat } D4 \rangle$  .  
**show**  $B4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using**  $b4$  .  
**show**  $D4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using**  $d4$  .  
**show**  $i < n-a$  **using**  $\langle i < n-a \rangle$  .  
**show**  $j < n-a$  **using**  $\langle j < n-a \rangle$  .  
**qed**  
**also have**  $\dots = D \ \#\# \ (j+a, j+a) * B \ \#\# \ (i+a, j+a)$   
**using** *assms*  $\langle i < n-a \rangle \langle j < n-a \rangle$   
**unfolding** *split-block-def* *Let-def* **by** *fastforce*  
**also have**  $\dots = (B*D) \ \#\# \ (i+a, j+a)$  **using** *diagonal-mat-mult-index'* *assms*  
**by** (*metis*  $\langle i < n-a \rangle \langle j < n-a \rangle$  *less-diff-conv*)  
**finally show**  $(B4 * D4) \ \#\# \ (i, j) = (B*D) \ \#\# \ (i+a, j+a)$  .  
**have**  $(D4 * B4) \ \#\# \ (i, j) = D4 \ \#\# \ (i, i) * B4 \ \#\# \ (i, j)$   
**using** *diagonal-mat-mult-index*  $\langle \text{diagonal-mat } D4 \rangle \langle i < n-a \rangle \langle j < n-a \rangle$   $b4$   
 $d4$   
**by** *blast*  
**also have**  $\dots = D \ \#\# \ (i+a, i+a) * B \ \#\# \ (i+a, j+a)$   
**using** *assms*  $\langle i < n-a \rangle \langle j < n-a \rangle$   
**unfolding** *split-block-def* *Let-def* **by** *fastforce*  
**also have**  $\dots = (D*B) \ \#\# \ (i+a, j+a)$  **using** *diagonal-mat-mult-index* *assms*  
**by** (*metis*  $\langle i < n-a \rangle \langle j < n-a \rangle$  *less-diff-conv*)  
**finally show**  $(D4 * B4) \ \#\# \ (i, j) = (D*B) \ \#\# \ (i+a, j+a)$  .  
**qed**

**lemma** *split-block-commute-subblock*:  
**fixes**  $B::'a::\text{comm-ring Matrix.mat}$

**assumes** *diagonal-mat*  $D$   
**and**  $D \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $a \leq n$   
**and** *split-block*  $B \ a \ a = (B_1, B_2, B_3, B_4)$   
**and** *split-block*  $D \ a \ a = (D_1, D_2, D_3, D_4)$   
**and**  $B * D = D * B$   
**shows**  $B_4 * D_4 = D_4 * B_4$   
**proof**  
**have**  $d_4: D_4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms*  
*split-block*(4)[of  $D$ ] **by** *simp*  
**have**  $b_4: B_4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms*  
*split-block*(4)[of  $B$ ] **by** *simp*  
**have** *diagonal-mat*  $D_4$  **using** *assms* *split-block-diagonal*[of  $D$ ] **by** *blast*  
**show**  $\text{dim-row } (B_4 * D_4) = \text{dim-row } (D_4 * B_4)$  **using**  $d_4 \ b_4$  **by** *simp*  
**show**  $\text{dim-col } (B_4 * D_4) = \text{dim-col } (D_4 * B_4)$  **using**  $d_4 \ b_4$  **by** *simp*  
**fix**  $i \ j$   
**assume**  $i < \text{dim-row } (D_4 * B_4)$   
**and**  $j < \text{dim-col } (D_4 * B_4)$   
**have**  $(B_4 * D_4) \ \$(i,j) = (B * D) \ \$(i+a, j+a)$   
**using** *split-block-times-diag-index*[of  $D \ n \ B \ a$ ] *assms*  
 $\langle i < \text{dim-row } (D_4 * B_4) \rangle \langle j < \text{dim-col } (D_4 * B_4) \rangle$  **by** *blast*  
**also have**  $\dots = (D * B) \ \$(i+a, j+a)$  **using** *assms* **by** *simp*  
**also have**  $\dots = (D_4 * B_4) \ \$(i, j)$   
**using** *split-block-times-diag-index*[of  $D \ n \ B \ a$ ] *assms*  
**by** (*metis*  $\langle i < \text{dim-row } (D_4 * B_4) \rangle \langle j < \text{dim-col } (D_4 * B_4) \rangle$ )  
**finally show**  $(B_4 * D_4) \ \$(i,j) = (D_4 * B_4) \ \$(i, j)$  .  
**qed**

**lemma** *commute-diag-mat-zero-comp*:

**fixes**  $D::'a::\{\text{field}\} \text{Matrix.mat}$   
**assumes** *diagonal-mat*  $D$   
**and**  $D \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and**  $B * D = D * B$   
**and**  $i < n$   
**and**  $j < n$   
**and**  $D \ \$(i,i) \neq D \ \$(j,j)$   
**shows**  $B \ \$(i,j) = 0$   
**proof** –  
**have**  $B \ \$(i,j) * D \ \$(j,j) = (B * D) \ \$(i,j)$   
**using** *diagonal-mat-mult-index'*[of  $B \ n \ D$ ] *assms* **by** *simp*  
**also have**  $\dots = (D * B) \ \$(i,j)$  **using** *assms* **by** *simp*  
**also have**  $\dots = B \ \$(i,j) * D \ \$(i,i)$   
**using** *diagonal-mat-mult-index* *assms*  
**by** (*metis* *Groups.mult-ac*(2))  
**finally have**  $B \ \$(i,j) * D \ \$(j,j) = B \ \$(i,j) * D \ \$(i,i)$  .  
**hence**  $B \ \$(i,j) * (D \ \$(j,j) - D \ \$(i,i)) = 0$  **by** *auto*  
**thus**  $B \ \$(i,j) = 0$  **using** *assms* **by** *simp*

qed

**lemma** *commute-diag-mat-split-block*:

```
fixes D::'a::{field} Matrix.mat
assumes diagonal-mat D
and D ∈ carrier-mat n n
and B ∈ carrier-mat n n
and B * D = D * B
and k ≤ n
and ∀ i j. (i < k ∧ k ≤ j ∧ j < n) ⟶ D $$ (i,i) ≠ D $$ (j,j)
and (B1, B2, B3, B4) = split-block B k k
shows B2 = 0_m k (n-k) B3 = 0_m (n-k) k
proof (intro eq-matI)
  show dim-row B2 = dim-row (0_m k (n - k))
    using assms unfolding split-block-def Let-def by simp
  show dim-col B2 = dim-col (0_m k (n - k))
    using assms unfolding split-block-def Let-def by simp
  fix i j
  assume i < dim-row (0_m k (n - k))
  and j < dim-col (0_m k (n - k)) note ijprop = this
  have B2 $$ (i, j) = B $$ (i, j+k) using assms ijprop
    unfolding split-block-def Let-def by simp
  also have ... = 0
  proof (rule commute-diag-mat-zero-comp[of D n], (auto simp add: assms))
    show i < n using ijprop assms by simp
    show j + k < n using ijprop assms by simp
    show D $$ (i, i) = D $$ (j + k, j + k) ⟹ False using ijprop assms
      by (metis ⟨j + k < n⟩ index-zero-mat(2) le-add2)
  qed
  finally show B2 $$ (i, j) = 0_m k (n - k) $$ (i, j) using ijprop by simp
next
show B3 = 0_m (n-k) k
proof (intro eq-matI)
  show dim-row B3 = dim-row (0_m (n - k) k)
    using assms unfolding split-block-def Let-def by simp
  show dim-col B3 = dim-col (0_m (n - k) k)
    using assms unfolding split-block-def Let-def by simp
  fix i j
  assume i < dim-row (0_m (n - k) k)
  and j < dim-col (0_m (n - k) k) note ijprop = this
  have B3 $$ (i, j) = B $$ (i+k, j) using assms ijprop
    unfolding split-block-def Let-def by simp
  also have ... = 0
  proof (rule commute-diag-mat-zero-comp[of D n], (auto simp add: assms))
    show i + k < n using ijprop assms by simp
    show j < n using ijprop assms by simp
    show D $$ (i+k, i+k) = D $$ (j, j) ⟹ False using ijprop assms
      by (metis ⟨i + k < n⟩ index-zero-mat(3) le-add2)
  qed
qed
```

finally show  $B3 \ \$(i, j) = 0_m \ (n - k) \ k \ \$(i, j)$  using *ijprop* by *simp*  
 qed  
 qed

lemma *split-block-hermitian-1*:

assumes *hermitian* *A*  
 and  $n \leq \text{dim-row } A$   
 and  $(A1, A2, A3, A4) = \text{split-block } A \ n \ n$   
 shows *hermitian* *A1* **unfolding** *hermitian-def*  
 proof (rule *eq-matI*, *auto*)  
 have  $\text{dim-row } A = \text{dim-col } A$  using *assms*  
 by (*metis carrier-matD(2)* *hermitian-square*)  
 show  $\text{dim-col } A1 = \text{dim-row } A1$  using *assms* **unfolding** *split-block-def* *Let-def*  
 by *simp*  
 thus  $\text{dim-row } A1 = \text{dim-col } A1$  by *simp*  
 show  $\bigwedge i \ j. \ i < \text{dim-row } A1 \implies j < \text{dim-col } A1 \implies$   
 $\text{Complex-Matrix.adjoint } A1 \ \$(i, j) = A1 \ \$(i, j)$   
 proof -  
 fix *i j*  
 assume  $i < \text{dim-row } A1$  and  $j < \text{dim-col } A1$  note *ij = this*  
 have  $r: \text{dim-row } A1 = n$  using *assms* **unfolding** *split-block-def* *Let-def*  
 by *simp*  
 have  $c: \text{dim-col } A1 = n$  using *assms* **unfolding** *split-block-def* *Let-def*  
 by *simp*  
 have  $\text{Complex-Matrix.adjoint } A1 \ \$(i, j) = \text{conjugate } (A1 \ \$(j, i))$   
 using *ij r c* **unfolding** *Complex-Matrix.adjoint-def* by *simp*  
 also have  $\dots = \text{conjugate } (A \ \$(j, i))$  using *assms ij r c*  
**unfolding** *split-block-def* *Let-def* by *simp*  
 also have  $\dots = A \ \$(i, j)$  using *assms ij r c*  $\langle \text{dim-row } A = \text{dim-col } A \rangle$   
**unfolding** *hermitian-def* *Complex-Matrix.adjoint-def*  
 by (*metis adjoint-eval* *assms(1)* *hermitian-def* *order-less-le-trans*)  
 also have  $\dots = A1 \ \$(i, j)$  using *assms ij r c*  
**unfolding** *split-block-def* *Let-def* by *simp*  
 finally show  $\text{Complex-Matrix.adjoint } A1 \ \$(i, j) = A1 \ \$(i, j)$  .  
 qed  
 qed

lemma *split-block-hermitian-4*:

assumes *hermitian* *A*  
 and  $n \leq \text{dim-row } A$   
 and  $(A1, A2, A3, A4) = \text{split-block } A \ n \ n$   
 shows *hermitian* *A4* **unfolding** *hermitian-def*  
 proof (rule *eq-matI*, *auto*)  
 have  $\text{arc}: \text{dim-row } A = \text{dim-col } A$  using *assms*  
 by (*metis carrier-matD(2)* *hermitian-square*)  
 thus  $\text{dim-col } A4 = \text{dim-row } A4$  using *assms* **unfolding** *split-block-def* *Let-def*  
 by *simp*  
 thus  $\text{dim-row } A4 = \text{dim-col } A4$  by *simp*  
 show  $\bigwedge i \ j. \ i < \text{dim-row } A4 \implies j < \text{dim-col } A4 \implies$

$\text{Complex-Matrix.adjoint } A_4 \ \$(i, j) = A_4 \ \$(i, j)$   
**proof** –  
**fix**  $i\ j$   
**assume**  $i < \text{dim-row } A_4$  **and**  $j < \text{dim-col } A_4$  **note**  $ij = \text{this}$   
**have**  $r: \text{dim-row } A_4 = \text{dim-row } A - n$  **using**  $\text{assms}$   
**unfolding**  $\text{split-block-def Let-def}$  **by**  $\text{simp}$   
**have**  $c: \text{dim-col } A_4 = \text{dim-col } A - n$  **using**  $\text{assms}$   
**unfolding**  $\text{split-block-def Let-def}$  **by**  $\text{simp}$   
**have**  $\text{Complex-Matrix.adjoint } A_4 \ \$(i, j) = \text{conjugate } (A_4 \ \$(j, i))$   
**using**  $ij\ r\ c\ \text{arc}$  **unfolding**  $\text{Complex-Matrix.adjoint-def}$  **by**  $\text{simp}$   
**also have**  $\dots = \text{conjugate } (A \ \$(j + n, i + n))$  **using**  $\text{assms } ij\ r\ c\ \text{arc}$   
**unfolding**  $\text{split-block-def Let-def}$  **by**  $\text{simp}$   
**also have**  $\dots = A \ \$(i + n, j + n)$  **using**  $\text{assms } ij\ r\ c\ \text{arc}$   
**unfolding**  $\text{hermitian-def Complex-Matrix.adjoint-def}$   
**by**  $(\text{metis index-mat}(1)\ \text{less-diff-conv split-conv})$   
**also have**  $\dots = A_4 \ \$(i, j)$  **using**  $\text{assms } ij\ r\ c$   
**unfolding**  $\text{split-block-def Let-def}$  **by**  $\text{simp}$   
**finally show**  $\text{Complex-Matrix.adjoint } A_4 \ \$(i, j) = A_4 \ \$(i, j)$  .  
**qed**  
**qed**

**lemma**  $\text{diag-block-split-block}$ :

**assumes**  $B \in \text{carrier-mat } n\ n$   
**and**  $k < n$   
**and**  $(B_1, B_2, B_3, B_4) = \text{split-block } B\ k\ k$   
**and**  $B_2 = 0_m\ k\ (n - k)$   
**and**  $B_3 = 0_m\ (n - k)\ k$   
**shows**  $B = \text{diag-block-mat } [B_1, B_4]$   
**proof** –  
**have**  $dr: \text{dim-row } B = k + (n - k)$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**have**  $dc: \text{dim-col } B = k + (n - k)$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**have**  $c1: B_1 \in \text{carrier-mat } k\ k$  **using**  $\text{assms}$   
 $\text{split-block}(1)[\text{of } B, OF - dr\ dc]$  **by**  $\text{metis}$   
**have**  $c4: B_4 \in \text{carrier-mat } (n - k)\ (n - k)$  **using**  $\text{assms}$   
 $\text{split-block}(4)[\text{of } B, OF - dr\ dc]$  **by**  $\text{metis}$   
**have**  $d4: \text{diag-block-mat } [B_4] = B_4$  **using**  $\text{diag-block-mat-singleton}[\text{of } B_4]$   
**by**  $\text{simp}$   
**have**  $B = \text{four-block-mat } B_1\ B_2\ B_3\ B_4$  **using**  $\text{assms}$   $\text{split-block}(3)[\text{of } B\ k]$   
**by**  $(\text{metis carrier-matD}(1)\ \text{carrier-matD}(2)\ \text{diff-is-0-eq}$   
 $\text{le-add-diff-inverse nat-le-linear semiring-norm}(137)$   
 $\text{split-block}(5)\ \text{zero-less-diff})$   
**also have**  $\dots = \text{four-block-mat } B_1\ (0_m\ k\ (n - k))\ (0_m\ (n - k)\ k)\ B_4$   
**using**  $\text{assms}$  **by**  $\text{simp}$   
**also have**  $\dots = \text{four-block-mat } B_1\ (0_m\ k\ (n - k))\ (0_m\ (n - k)\ k)$   
 $(\text{diag-block-mat } [B_4])$  **using**  $\text{diag-block-mat-singleton}[\text{of } B_4]$  **by**  $\text{simp}$   
**also have**  $\dots = \text{diag-block-mat } [B_1, B_4]$   
**using**  $\text{diag-block-mat.simps}(2)[\text{of } B_1\ [B_4]]\ c1\ c4$   
**unfolding**  $\text{Let-def}$  **by**  $\text{auto}$   
**finally show**  $?thesis$  .

qed

## 5.2 Diagonal block matrices

**abbreviation** *four-block-diag* **where**

*four-block-diag*  $B1\ B2 \equiv$   
(*four-block-mat*  $B1\ (0_m\ (dim\text{-}row\ B1)\ (dim\text{-}col\ B2))$ )  
( $0_m\ (dim\text{-}row\ B2)\ (dim\text{-}col\ B1)$ )  $B2$ )

**lemma** *four-block-diag-cong-comp*:

**assumes**  $dim\text{-}row\ A1 = dim\text{-}row\ B1$

**and**  $dim\text{-}col\ A1 = dim\text{-}col\ B1$

**and** *four-block-diag*  $A1\ A2 = \textit{four-block-diag}\ B1\ B2$

**shows**  $A1 = B1$

**proof** (*rule eq-matI*, *auto simp:assms*)

**define**  $A$  **where**  $A = \textit{four-block-diag}\ A1\ A2$

**define**  $B$  **where**  $B = \textit{four-block-diag}\ B1\ B2$

**fix**  $i\ j$

**assume**  $i < dim\text{-}row\ B1$  **and**  $j < dim\text{-}col\ B1$  **note**  $ij=this$

**hence**  $i < dim\text{-}row\ A1$   $j < dim\text{-}col\ A1$  **using** *assms* **by** *auto*

**hence**  $A1\ \$(i,j) = A\ \$(i,j)$

**unfolding** *A-def four-block-mat-def Let-def* **by** *force*

**also have**  $\dots = B\ \$(i,j)$  **using** *assms* **unfolding** *A-def B-def* **by** *simp*

**also have**  $\dots = B1\ \$(i,j)$

**using**  $ij$  **unfolding** *B-def four-block-mat-def Let-def* **by** *force*

**finally show**  $A1\ \$(i,j) = B1\ \$(i,j)$  .

qed

**lemma** *four-block-diag-cong-comp'*:

**assumes**  $dim\text{-}row\ A1 = dim\text{-}row\ B1$

**and**  $dim\text{-}col\ A1 = dim\text{-}col\ B1$

**and** *four-block-diag*  $A1\ A2 = \textit{four-block-diag}\ B1\ B2$

**shows**  $A2 = B2$

**proof** (*rule eq-matI*)

**define**  $n$  **where**  $n = dim\text{-}row\ A1$

**define**  $m$  **where**  $m = dim\text{-}col\ A1$

**define**  $A$  **where**  $A = \textit{four-block-diag}\ A1\ A2$

**define**  $B$  **where**  $B = \textit{four-block-diag}\ B1\ B2$

**show**  $dim\text{-}row\ A2 = dim\text{-}row\ B2$

**using** *assms* **unfolding** *four-block-mat-def Let-def*

**by** (*metis assms(3) diff-add-inverse index-mat-four-block(2)*)

**show**  $dim\text{-}col\ A2 = dim\text{-}col\ B2$

**using** *assms* **unfolding** *four-block-mat-def Let-def*

**by** (*metis assms(3) diff-add-inverse index-mat-four-block(3)*)

**fix**  $i\ j$

**assume**  $i < dim\text{-}row\ B2$  **and**  $j < dim\text{-}col\ B2$  **note**  $ij=this$

**hence**  $i+n < dim\text{-}row\ A$

**unfolding** *A-def n-def m-def four-block-mat-def Let-def*

**by** (*simp add: <dim-row A2 = dim-row B2>*)

```

have  $j+m < \dim\text{-col } A$ 
  unfolding  $A\text{-def } n\text{-def } m\text{-def } \text{four-block-mat-def } \text{Let-def}$ 
  by ( $\text{simp add: } \langle \dim\text{-col } A2 = \dim\text{-col } B2 \rangle ij$ )
{
  have  $n \leq i+n$  by  $\text{simp}$ 
  have  $m \leq j+m$  by  $\text{simp}$ 
  have  $i + n - n = i$  by  $\text{simp}$ 
  have  $j + m - m = j$  by  $\text{simp}$ 
} note  $ijeq = \text{this}$ 
have  $A2\$(i,j) = A\$(i+n, j+m)$  using  $ijeq$ 
  using  $A\text{-def } \langle i + n < \dim\text{-row } A \rangle \langle j + m < \dim\text{-col } A \rangle m\text{-def } n\text{-def}$  by  $\text{force}$ 
also have  $\dots = B\$(i+n, j+m)$  using  $\text{assms unfolding } A\text{-def } B\text{-def}$  by  $\text{simp}$ 
also have  $\dots = B2\$(i,j)$  using  $ijeq$ 
  by ( $\text{metis } A\text{-def } B\text{-def } \langle i + n < \dim\text{-row } A \rangle \langle j + m < \dim\text{-col } A \rangle$ 
     $\text{add-implies-diff } \text{assms}(1) \text{ assms}(2) \text{ assms}(3) \text{ index-mat-four-block}(1)$ 
     $\text{index-mat-four-block}(2) \text{ index-mat-four-block}(3) m\text{-def } n\text{-def}$ 
     $\text{not-add-less2}$ )
finally show  $A2\$(i,j) = B2\$(i,j)$  .
qed

```

```

lemma  $\text{four-block-mat-real-diag}$ :
  assumes  $\forall i < \dim\text{-row } B1. B1\$(i,i) \in \text{Reals}$ 
  and  $\forall i < \dim\text{-row } B2. B2\$(i,i) \in \text{Reals}$ 
  and  $\dim\text{-row } B1 = \dim\text{-col } B1$ 
  and  $\dim\text{-row } B2 = \dim\text{-col } B2$ 
  and  $i < \dim\text{-row } (\text{four-block-diag } B1 B2)$ 
shows  $(\text{four-block-diag } B1 B2) \$(i,i) \in \text{Reals}$ 
proof ( $\text{cases } i < \dim\text{-row } B1$ )
  case  $\text{True}$ 
  then show  $?thesis$  using  $\text{assms}$  by  $\text{simp}$ 
next
  case  $\text{False}$ 
  then show  $?thesis$  using  $\text{assms}$  by  $\text{force}$ 
qed

```

```

lemma  $\text{four-block-diagonal}$ :
  assumes  $\dim\text{-row } B1 = \dim\text{-col } B1$ 
  and  $\dim\text{-row } B2 = \dim\text{-col } B2$ 
  and  $\text{diagonal-mat } B1$ 
  and  $\text{diagonal-mat } B2$ 
shows  $\text{diagonal-mat } (\text{four-block-diag } B1 B2)$  unfolding  $\text{diagonal-mat-def}$ 
proof ( $\text{intro allI impI}$ )
  fix  $i j$ 
  assume  $i < \dim\text{-row } (\text{four-block-diag } B1 B2)$ 
  and  $j < \dim\text{-col } (\text{four-block-diag } B1 B2)$ 
  and  $i \neq j$  note  $ijprops = \text{this}$ 
  show  $(\text{four-block-diag } B1 B2) \$(i,j) = 0$ 
  proof ( $\text{cases } i < \dim\text{-row } B1$ )
    case  $\text{True}$ 

```

```

then show ?thesis
  using assms(3) diagonal-mat-def ijprops(2) ijprops(3)
  by (metis add-less-imp-less-left
    ijprops(1) index-mat-four-block(1) index-mat-four-block(2)
    index-mat-four-block(3) index-zero-mat(1)
    linordered-semidom-class.add-diff-inverse)
next
  case False
  then show ?thesis using ijprops
    by (metis (no-types, lifting) add-less-cancel-left assms(1)
      assms(4) diagonal-mat-def index-mat-four-block(1)
      index-mat-four-block(2) index-mat-four-block(3)
      index-zero-mat(1) linordered-semidom-class.add-diff-inverse)
qed
qed

```

```

lemma four-block-diag-zero:
  assumes B ∈ carrier-mat 0 0
  shows four-block-diag A B = A
proof (rule eq-matI, auto)
  show dim-row B = 0 using assms by simp
  show dim-col B = 0 using assms by simp
qed

```

```

lemma four-block-diag-zero':
  assumes B ∈ carrier-mat 0 0
  shows four-block-diag B A = A
proof (rule eq-matI)
  show dim-row (four-block-diag B A) = dim-row A using assms by simp
  show dim-col (four-block-diag B A) = dim-col A using assms by simp
  fix i j
  assume i < dim-row A and j < dim-col A
  thus four-block-diag B A $$ (i, j) = A $$ (i, j)
    using  $\langle \text{dim-col (four-block-diag B A) = dim-col A} \rangle$ 
     $\langle \text{dim-row (four-block-diag B A) = dim-row A} \rangle$ 
  by auto
qed

```

```

lemma mult-four-block-diag:
  assumes A1 ∈ carrier-mat nr1 n1 D1 ∈ carrier-mat nr2 n2
  and A2 ∈ carrier-mat n1 nc1 D2 ∈ carrier-mat n2 nc2
shows four-block-diag A1 D1 *
  four-block-diag A2 D2
  = four-block-diag (A1 * A2) (D1 * D2)
proof –
  define fb1 where fb1 = four-block-mat A1 (0m nr1 n2) (0m nr2 n1) D1
  define fb2 where fb2 = four-block-mat A2 (0m n1 nc2) (0m n2 nc1) D2
  have fb1 * fb2 = four-block-mat (A1 * A2 + 0m nr1 n2 * 0m n2 nc1)
  (A1 * 0m n1 nc2 + 0m nr1 n2 * D2) (0m nr2 n1 * A2 + D1 * 0m n2 nc1)

```

```

    ( $0_m$   $nr2$   $n1$  *  $0_m$   $n1$   $nc2$  +  $D1$  *  $D2$ ) unfolding fb1-def fb2-def
proof (rule mult-four-block-mat)
  show  $A1 \in \text{carrier-mat } nr1 \ n1$  using assms by simp
  show  $D1 \in \text{carrier-mat } nr2 \ n2$  using assms by simp
  show  $A2 \in \text{carrier-mat } n1 \ nc1$   $D2 \in \text{carrier-mat } n2 \ nc2$  using assms by auto
qed auto
also have ... = four-block-mat ( $A1$  *  $A2$ ) ( $0_m$   $nr1$   $nc2$ ) ( $0_m$   $nr2$   $nc1$ ) ( $D1$  *
 $D2$ )
  using assms by simp
finally show ?thesis unfolding fb1-def fb2-def
  using assms by simp
qed

```

```

lemma four-block-diag-adjoint:
shows (Complex-Matrix.adjoint (four-block-diag  $A1$   $A2$ )) =
  (four-block-diag (Complex-Matrix.adjoint  $A1$ )
  (Complex-Matrix.adjoint  $A2$ ))
by (rule eq-matI,
  auto simp: four-block-mat-adjoint zero-adjoint adjoint-eval)

```

```

lemma four-block-diag-unitary:
assumes unitary  $U1$ 
and unitary  $U2$ 
shows unitary
  (four-block-diag  $U1$   $U2$ )
(is unitary ?fU)
  unfolding unitary-def
proof
show  $?fU \in \text{carrier-mat } (\text{dim-row } ?fU) (\text{dim-row } ?fU)$ 
  by (metis Complex-Matrix.unitary-def assms(1) assms(2)
  four-block-carrier-mat index-mat-four-block(2))
define  $n$  where  $n = \text{dim-row } ?fU$ 
show inverts-mat  $?fU$  (Complex-Matrix.adjoint  $?fU$ )
proof –
  have (Complex-Matrix.adjoint  $?fU$ ) =
    (four-block-mat (Complex-Matrix.adjoint  $U1$ )
    ( $0_m$  (dim-col  $U1$ ) (dim-row  $U2$ ))
    ( $0_m$  (dim-col  $U2$ ) (dim-row  $U1$ ))
    (Complex-Matrix.adjoint  $U2$ ))
  by (rule eq-matI,
  auto simp: four-block-mat-adjoint zero-adjoint adjoint-eval)
hence  $?fU * (\text{Complex-Matrix.adjoint } ?fU) =$ 
   $?fU * (\text{four-block-diag } (\text{Complex-Matrix.adjoint } U1)$ 
  (Complex-Matrix.adjoint  $U2$ )) by simp
also have ... = four-block-diag
  ( $U1 * (\text{Complex-Matrix.adjoint } U1)$ )
  ( $U2 * (\text{Complex-Matrix.adjoint } U2)$ )
  by (rule mult-four-block-diag, (auto simp add: assms))
also have ... = four-block-mat

```

```

    (1m (dim-row U1))
    (0m (dim-row U1) (dim-row U2))
    (0m (dim-row U2) (dim-row U1))
    (1m (dim-row U2)) using assms
    unfolding unitary-def inverts-mat-def
    by simp
    also have ... = 1m (dim-row U1 + dim-row U2) by simp
    finally show ?thesis unfolding inverts-mat-def by simp
qed
qed

```

**lemma** *four-block-diag-similar*:

```

assumes unitarily-equiv A1 B1 U1
and unitarily-equiv A2 B2 U2
and dim-row A1 = dim-col A1
and dim-row A2 = dim-col A2
shows similar-mat-wit
    (four-block-diag A1 A2)
    (four-block-diag B1 B2)
    (four-block-diag U1 U2)
    (Complex-Matrix.adjoint (four-block-diag U1 U2))
unfolding similar-mat-wit-def
proof (simp add: Let-def, intro conjI)
define n where n = dim-row A1 + dim-row A2
show four-block-diag A1 A2 ∈ carrier-mat n n unfolding n-def using assms
by auto
show four-block-diag B1 B2 ∈ carrier-mat n n unfolding n-def using assms
by (metis carrier-matI four-block-carrier-mat unitarily-equiv-carrier(1))
show u: four-block-diag U1 U2 ∈ carrier-mat n n unfolding n-def using assms
by (metis carrier-matI four-block-carrier-mat unitarily-equiv-carrier(2))
thus cu: Complex-Matrix.adjoint (four-block-diag U1 U2) ∈ carrier-mat n n
unfolding n-def using adjoint-dim' by blast
show four-block-diag U1 U2 * Complex-Matrix.adjoint (four-block-diag U1 U2) =
    1m n unfolding n-def
using u assms four-block-diag-unitary n-def
    unitarily-equiv-def unitary-simps(2) by blast
thus Complex-Matrix.adjoint (four-block-diag U1 U2) * four-block-diag U1 U2 =
    1m n
using cu mat-mult-left-right-inverse u by blast
have four-block-diag A1 A2 =
    four-block-diag (U1 * B1 * (Complex-Matrix.adjoint U1))
    (U2 * B2 * (Complex-Matrix.adjoint U2))
using assms unitarily-equiv-eq by blast
also have ... = (four-block-diag (U1*B1) (U2*B2)) *
    (four-block-diag (Complex-Matrix.adjoint U1)
    (Complex-Matrix.adjoint U2))
proof (rule mult-four-block-diag[symmetric])
show U1 * B1 ∈ carrier-mat (dim-row A1) (dim-row A1)
by (metis assms(1) assms(3) carrier-mat-triv mult-carrier-mat

```

$unitarily-equiv-carrier(1) \ unitarily-equiv-carrier(2)$   
**show**  $U2 * B2 \in carrier-mat \ (dim-row \ A2) \ (dim-row \ A2)$   
**by**  $(metis \ assms(2) \ assms(4) \ carrier-mat-triv \ mult-carrier-mat \ unitarily-equiv-carrier(1) \ unitarily-equiv-carrier(2))$   
**show**  $Complex-Matrix.adjoint \ U1 \in carrier-mat \ (dim-row \ A1) \ (dim-row \ A1)$   
**by**  $(metis \ Complex-Matrix.unitary-def \ adjoint-dim \ assms(1) \ index-mult-mat(2) \ unitarily-equivD(1) \ unitarily-equiv-eq)$   
**show**  $Complex-Matrix.adjoint \ U2 \in carrier-mat \ (dim-row \ A2) \ (dim-row \ A2)$   
**by**  $(meson \ assms(2) \ carrier-mat-triv \ similar-mat-witD2(7) \ unitarily-equiv-def)$   
**qed**  
**also have**  $\dots = four-block-diag \ U1 \ U2 * four-block-diag \ B1 \ B2 * Complex-Matrix.adjoint \ (four-block-diag \ U1 \ U2)$   
**proof** –  
**have**  $four-block-diag \ (U1*B1) \ (U2*B2) = four-block-diag \ U1 \ U2 * four-block-diag \ B1 \ B2$   
**proof**  $(rule \ mult-four-block-diag[symmetric])$   
**show**  $U1 \in carrier-mat \ (dim-row \ A1) \ (dim-row \ A1)$   
**by**  $(metis \ assms(1) \ assms(3) \ carrier-mat-triv \ unitarily-equiv-carrier(2))$   
**show**  $B1 \in carrier-mat \ (dim-row \ A1) \ (dim-row \ A1)$   
**by**  $(metis \ assms(1) \ assms(3) \ carrier-mat-triv \ unitarily-equiv-carrier(1))$   
**show**  $U2 \in carrier-mat \ (dim-row \ A2) \ (dim-row \ A2)$   
**by**  $(metis \ assms(2) \ assms(4) \ carrier-mat-triv \ unitarily-equiv-carrier(2))$   
**show**  $B2 \in carrier-mat \ (dim-row \ A2) \ (dim-row \ A2)$   
**by**  $(metis \ assms(2) \ assms(4) \ carrier-mat-triv \ unitarily-equiv-carrier(1))$   
**qed**  
**moreover have**  $four-block-diag \ (Complex-Matrix.adjoint \ U1) \ (Complex-Matrix.adjoint \ U2) = Complex-Matrix.adjoint \ (four-block-diag \ U1 \ U2)$   
**by**  $(rule \ four-block-diag-adjoint[symmetric])$   
**ultimately show**  $?thesis \ by \ simp$   
**qed**  
**finally show**  $four-block-diag \ A1 \ A2 = four-block-diag \ U1 \ U2 * four-block-diag \ B1 \ B2 * Complex-Matrix.adjoint \ (four-block-diag \ U1 \ U2) .$   
**qed**

**lemma**  $four-block-unitarily-equiv:$   
**assumes**  $unitarily-equiv \ A1 \ B1 \ U1$   
**and**  $unitarily-equiv \ A2 \ B2 \ U2$   
**and**  $dim-row \ A1 = dim-col \ A1$   
**and**  $dim-row \ A2 = dim-col \ A2$   
**shows**  $unitarily-equiv \ (four-block-diag \ A1 \ A2) \ (four-block-diag \ B1 \ B2)$

```

    (four-block-diag U1 U2)
  (is unitarily-equiv ?fA ?fB ?fU)
  unfolding unitarily-equiv-def
proof
  show unitary ?fU using four-block-diag-unitary assms unitarily-equivD(1)
    by blast
  show similar-mat-wit ?fA ?fB ?fU (Complex-Matrix.adjoint ?fU)
    using assms four-block-diag-similar[of A1] by simp
qed

```

```

lemma four-block-unitary-diag:
  assumes unitary-diag A1 B1 U1
  and unitary-diag A2 B2 U2
  and dim-row A1 = dim-col A1
  and dim-row A2 = dim-col A2
shows unitary-diag
  (four-block-diag A1 A2)
  (four-block-diag B1 B2)
  (four-block-diag U1 U2)
(is unitary-diag ?fA ?fB ?fU)
  unfolding unitary-diag-def
proof
  show unitarily-equiv ?fA ?fB ?fU
    using four-block-unitarily-equiv[of A1] assms by simp
  have dim-row B1 = dim-col B1 unfolding unitary-diag-def
    by (metis assms(1) assms(3) carrier-matD(1) carrier-matD(2)
      carrier-mat-triv unitary-diag-carrier(1))
  moreover have dim-row B2 = dim-col B2 unfolding unitary-diag-def
    by (metis assms(2) assms(4) carrier-matD(1) carrier-matD(2)
      carrier-mat-triv unitary-diag-carrier(1))
  ultimately show diagonal-mat ?fB using four-block-diagonal assms
    unfolding unitary-diag-def by blast
qed

```

```

lemma four-block-real-diag-decomp:
  assumes real-diag-decomp A1 B1 U1
  and real-diag-decomp A2 B2 U2
  and dim-row A1 = dim-col A1
  and dim-row A2 = dim-col A2
shows real-diag-decomp
  (four-block-diag A1 A2)
  (four-block-diag B1 B2)
  (four-block-diag U1 U2)
(is real-diag-decomp ?fA ?fB ?fU)
  unfolding real-diag-decomp-def
proof (intro conjI allI impI)
  show unitary-diag ?fA ?fB ?fU using four-block-unitary-diag assms
    unfolding real-diag-decomp-def by blast
  fix i

```

```

assume  $i < \text{dim-row } ?fB$ 
show  $?fB \ \$$ (i,i) \in \text{Reals}$ 
proof (rule four-block-mat-real-diag)
  show  $i < \text{dim-row } ?fB$  using  $\langle i < \text{dim-row } ?fB \rangle$  .
  show  $\forall i < \text{dim-row } B1. B1 \ \$$ (i, i) \in \mathbf{R}$  using assms
    unfolding real-diag-decomp-def by simp
  show  $\forall i < \text{dim-row } B2. B2 \ \$$ (i, i) \in \mathbf{R}$  using assms
    unfolding real-diag-decomp-def by simp
  show  $\text{dim-row } B1 = \text{dim-col } B1$  unfolding unitary-diag-def
    by (metis assms(1) assms(3) carrier-matD(1) carrier-matD(2)
      carrier-mat-triv real-diag-decompD(1) unitary-diag-carrier(1))
  show  $\text{dim-row } B2 = \text{dim-col } B2$  unfolding unitary-diag-def
    by (metis assms(2) assms(4) carrier-matD(1) carrier-matD(2)
      carrier-mat-triv real-diag-decompD(1) unitary-diag-carrier(1))
qed
qed

lemma diag-block-mat-mult:
  assumes  $\text{length } A1 = \text{length } B1$ 
  and  $\forall i < \text{length } A1. \text{dim-col } (A1!i) = \text{dim-row } (B1!i)$ 
shows  $\text{diag-block-mat } A1 * (\text{diag-block-mat } B1) =$ 
   $(\text{diag-block-mat } (\text{map2 } (*) \ A1 \ B1))$  using assms
proof (induct A1 arbitrary: B1)
  case Nil
  then show ?case by simp
next
  case (Cons a A1)
  define A where  $A = \text{diag-block-mat } A1$ 
  define B where  $B = \text{diag-block-mat } (\text{tl } B1)$ 
  have  $0 < \text{length } B1$  using Cons by auto
  hence  $B1 = \text{hd } B1 \ \# (\text{tl } B1)$  by simp
  have  $\text{length } (\text{tl } B1) = \text{length } A1$  using Cons by simp
  have dim:  $\forall i < \text{length } A1. \text{dim-col } (A1!i) = \text{dim-row } (\text{tl } B1!i)$ 
  proof (intro allI impI)
    fix i
    assume  $i < \text{length } A1$ 
    hence  $\text{dim-col } (A1!i) = \text{dim-col } ((a\#A1)!(\text{Suc } i))$  by simp
    also have  $\dots = \text{dim-row } (B1!(\text{Suc } i))$  using Cons
      by (metis Suc-lessI  $\langle i < \text{length } A1 \rangle$  length-Cons less-Suc-eq)
    also have  $\dots = \text{dim-row } (\text{tl } B1!i)$ 
      by (metis  $\langle B1 = \text{hd } B1 \ \# \ \text{tl } B1 \rangle$  nth-Cons-Suc)
    finally show  $\text{dim-col } (A1!i) = \text{dim-row } (\text{tl } B1!i)$  .
  qed
define C where  $C = \text{map2 } (*) \ (a \ \# \ A1) \ B1$ 
have  $\text{hd } C = a * \text{hd } B1$  using  $\langle B1 = \text{hd } B1 \ \# \ \text{tl } B1 \rangle$  unfolding C-def
  by (metis list.map(2) list.sel(1) prod.simps(2) zip-Cons-Cons)
have  $\text{tl } C = \text{map2 } (*) \ A1 \ (\text{tl } B1)$ 
  by (metis (no-types, lifting) C-def  $\langle B1 = \text{hd } B1 \ \# \ \text{tl } B1 \rangle$  list.sel(3)
    map-tl zip-Cons-Cons)

```

```

have C = hd C # (tl C) unfolding C-def
  by (metis Nil-eq-zip-iff Nil-is-map-conv ⟨Bl = hd Bl # tl Bl⟩
      list.exhaust-sel list.simps(3))
have dim-row B = sum-list (map dim-row (tl Bl)) unfolding B-def
  by (simp add: dim-diag-block-mat(1))
also have ... = sum-list (map dim-col Al)
proof (rule sum-list-cong)
  show length (map dim-row (tl Bl)) = length (map dim-col Al)
    using ⟨length (tl Bl) = length Al⟩ by simp
  show ∀ i < length (map dim-row (tl Bl)).
    map dim-row (tl Bl) ! i = map dim-col Al ! i
    by (metis ⟨length (tl Bl) = length Al⟩ dim length-map nth-map)
qed
also have ... = dim-col A unfolding A-def
  by (simp add: dim-diag-block-mat(2))
finally have ba: dim-row B = dim-col A .
have diag-block-mat (a # Al) * (diag-block-mat Bl) =
  four-block-diag a A * (four-block-diag (hd Bl) B)
  using diag-block-mat.simps(2) ⟨Bl = hd Bl # (tl Bl)⟩
  unfolding Let-def A-def B-def by metis
also have ... = four-block-diag (a * hd Bl) (A * B)
proof (rule mult-four-block-diag)
  show a ∈ carrier-mat (dim-row a) (dim-col a) by simp
  show hd Bl ∈ carrier-mat (dim-col a) (dim-col (hd Bl))
    using Cons
  by (metis ⟨0 < length Bl⟩ ⟨Bl = hd Bl # tl Bl⟩ carrier-mat-triv nth-Cons-0)
  show A ∈ carrier-mat (dim-row A) (dim-col A) by simp
  show B ∈ carrier-mat (dim-col A) (dim-col B) using ba by auto
qed
also have ... = four-block-diag (hd C) (diag-block-mat (tl C))
  unfolding A-def B-def
  using C-def ⟨hd C = a * hd Bl⟩ ⟨length (tl Bl) = length Al⟩
  ⟨tl C = map2 (*) Al (tl Bl)⟩ dim local.Cons(1)
  by presburger
also have ... = diag-block-mat C
  using ⟨C = hd C # (tl C)⟩ diag-block-mat.simps(2) unfolding Let-def by metis
finally show ?case unfolding C-def .
qed

lemma real-diag-decomp-block:
  fixes Al::complex Matrix.mat list
  assumes Al ≠ []
  and list-all (λA. 0 < dim-row A ∧ hermitian A) Al
shows ∃ Bl Ul. length Ul = length Al ∧
  (∀ i < length Al.
    Ul!i ∈ carrier-mat (dim-row (Al!i)) (dim-col (Al!i)) ∧ unitary (Ul!i) ∧
    Bl!i ∈ carrier-mat (dim-row (Al!i)) (dim-col (Al!i))) ∧
  real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl) (diag-block-mat Ul)
  using assms

```

```

proof (induct Al)
  case Nil
  then show ?case by simp
next
  case (Cons A Al)
  hence hermitian A 0 < dim-row A by auto
  hence A ∈ carrier-mat (dim-row A) (dim-row A)
    by (simp add: hermitian-square)
  from this obtain B U where r: real-diag-decomp A B U
    using hermitian-real-diag-decomp ⟨hermitian A⟩ ⟨0 < dim-row A⟩ by blast
  have bcar: B ∈ carrier-mat (dim-row A) (dim-col A)
    using real-diag-decompD(1)
    by (metis ⟨A ∈ carrier-mat (dim-row A) (dim-row A)⟩ carrier-matD(2) r
      unitary-diag-carrier(1))
  have ucar: U ∈ carrier-mat (dim-row A) (dim-col A)
    using real-diag-decompD(1)
    by (metis ⟨A ∈ carrier-mat (dim-row A) (dim-row A)⟩ carrier-matD(2) r
      unitary-diag-carrier(2))
  have unit: unitary U
    by (meson r real-diag-decompD(1) unitary-diagD(3))
show ?case
proof (cases Al = [])
  case True
  hence diag-block-mat (Cons A Al) = A by auto
  moreover have diag-block-mat [B] = B by auto
  moreover have diag-block-mat [U] = U by auto
  moreover have unitary U
    using r real-diag-decompD(1) unitary-diagD(3) by blast
  ultimately have
    real-diag-decomp (diag-block-mat (Cons A Al))
      (diag-block-mat [B]) (diag-block-mat [U])
    using ⟨real-diag-decomp A B U⟩ by auto
  moreover have (∀ i < length (A # Al).
    [U]!i ∈ carrier-mat (dim-row ((A # Al) ! i)) (dim-col ((A # Al) ! i)) ∧
    Complex-Matrix.unitary ([U] ! i) ∧ [B] ! i ∈
    carrier-mat (dim-row ((A # Al) ! i)) (dim-col ((A # Al) ! i))) using True
    by (simp add: bcar ucar unit)
  ultimately show ?thesis
    using True ⟨Complex-Matrix.unitary U⟩ bcar less-one ucar
    by (metis length-list-update list-update-code(2))
next
  case False
  have list-all (λA. 0 < dim-row A ∧ hermitian A) Al using Cons by auto
  hence ∃ Bl Ul. length Ul = length Al ∧
    (∀ i < length Al.
      Ul ! i ∈ carrier-mat (dim-row (Al ! i)) (dim-col (Al ! i)) ∧
      unitary (Ul!i) ∧
      Bl ! i ∈ carrier-mat (dim-row (Al ! i)) (dim-col (Al ! i))) ∧
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl) (diag-block-mat Ul)

```

```

using Cons False by simp
from this obtain Bl Ul where length Ul = length Al and
  rl: real-diag-decomp (diag-block-mat Al)
  (diag-block-mat Bl) (diag-block-mat Ul)
and  $\forall i < \text{length } Al.$ 
   $Ul ! i \in \text{carrier-mat } (\text{dim-row } (Al ! i)) (\text{dim-col } (Al ! i)) \wedge$ 
   $\text{unitary } (Ul ! i) \wedge$ 
   $Bl ! i \in \text{carrier-mat } (\text{dim-row } (Al ! i)) (\text{dim-col } (Al ! i))$ 
by auto note bu = this
have real-diag-decomp (diag-block-mat (A # Al))
  (diag-block-mat (B # Bl)) (diag-block-mat (U # Ul))
using four-block-real-diag-decomp[OF r rl]
by (metis ⟨A ∈ carrier-mat (dim-row A) (dim-row A)⟩
  carrier-matD(2) diag-block-mat.simps(2) hermitian-square
  real-diag-decomp-hermitian rl)
moreover have  $\text{length } (U \# Ul) = \text{length } (A \# Al)$  using bu by simp
moreover have  $\forall i < \text{length } (A \# Al).$ 
   $(U \# Ul) ! i \in \text{carrier-mat } (\text{dim-row } ((A \# Al) ! i)) (\text{dim-col } ((A \# Al) !$ 
i))  $\wedge$ 
   $\text{unitary } ((U \# Ul) ! i) \wedge$ 
   $(B \# Bl) ! i \in \text{carrier-mat } (\text{dim-row } ((A \# Al) ! i)) (\text{dim-col } ((A \# Al) !$ 
i))
proof (intro allI impI)
  fix i
  assume  $i < \text{length } (A \# Al)$ 
  show  $(U \# Ul) ! i \in \text{carrier-mat } (\text{dim-row } ((A \# Al) ! i))$ 
   $(\text{dim-col } ((A \# Al) ! i)) \wedge \text{unitary } ((U \# Ul) ! i) \wedge$ 
   $(B \# Bl) ! i \in \text{carrier-mat } (\text{dim-row } ((A \# Al) ! i))$ 
   $(\text{dim-col } ((A \# Al) ! i))$ 
  proof (cases i = 0)
    case True
    then show ?thesis by (simp add: bcar ucar unit)
  next
    case False
    hence  $\exists j. i = \text{Suc } j$  by (simp add: not0-implies-Suc)
    from this obtain j where  $j: i = \text{Suc } j$  by auto
    hence  $j < \text{length } Al$  using  $\langle i < \text{length } (A \# Al) \rangle$  by simp
    have  $(A \# Al) ! i = Al ! j$   $(U \# Ul) ! i = Ul ! j$   $(B \# Bl) ! i = Bl ! j$ 
    using j by auto
    then show ?thesis using Cons ⟨j < length Al⟩ bu(3) by presburger
  qed
qed
ultimately show ?thesis by blast
qed
qed

```

**lemma** *diag-block-mat-adjoint:*  
**shows** *Complex-Matrix.adjoint (diag-block-mat Al) =*  
*diag-block-mat (map Complex-Matrix.adjoint Al)*

```

proof (induct Al)
  case Nil
  then show ?case using zero-adjoint by simp
next
  case (Cons a Al)
  have Complex-Matrix.adjoint (diag-block-mat (a # Al)) =
    Complex-Matrix.adjoint (four-block-diag a (diag-block-mat Al))
    using diag-block-mat.simps(2)[of a] unfolding Let-def by simp
  also have ... = four-block-diag (Complex-Matrix.adjoint a)
    (Complex-Matrix.adjoint (diag-block-mat Al))
    using four-block-diag-adjoint[of a] by simp
  also have ... = four-block-diag (Complex-Matrix.adjoint a)
    (diag-block-mat (map Complex-Matrix.adjoint Al)) using Cons by simp
  also have ... = diag-block-mat (map Complex-Matrix.adjoint (a#Al))
    using diag-block-mat.simps(2) unfolding Let-def
    by (metis (no-types) diag-block-mat.simps(2) list.map(2))
  finally show ?case .
qed

```

```

lemma diag-block-mat-mat-conj:
  assumes length Al = length Bl
  and  $\forall i < \text{length } Al. \text{dim-col } (Al!i) = \text{dim-row } (Bl!i)$ 
  and  $\forall i < \text{length } Al. \text{dim-row } (Bl!i) = \text{dim-col } (Bl!i)$ 
  shows mat-conj (diag-block-mat Al) (diag-block-mat Bl) =
    diag-block-mat (map2 mat-conj Al Bl)
proof -
  have mat-conj (diag-block-mat Al) (diag-block-mat Bl) =
    diag-block-mat Al * diag-block-mat Bl *
    diag-block-mat (map Complex-Matrix.adjoint Al)
    using diag-block-mat-adjoint[of Al] unfolding mat-conj-def by simp
  also have ... = diag-block-mat (map2 (*) Al Bl) *
    diag-block-mat (map Complex-Matrix.adjoint Al)
    using diag-block-mat-mult[OF assms(1) assms(2)] by simp
  also have ... = diag-block-mat (map2 (*) (map2 (*) Al Bl)
    (map Complex-Matrix.adjoint Al))
  proof (rule diag-block-mat-mult)
    show length (map2 (*) Al Bl) = length (map Complex-Matrix.adjoint Al)
      by (simp add: assms(1))
    show  $\forall i < \text{length } (map2 (*) Al Bl). \text{dim-col } (map2 (*) Al Bl ! i) =$ 
       $\text{dim-row } (map Complex-Matrix.adjoint Al ! i)$ 
      by (simp add: assms(2) assms(3))
  qed
  also have ... = diag-block-mat (map2 mat-conj Al Bl)
    using map2-mat-conj-exp[OF assms(1)] by simp
  finally show ?thesis .
qed

```

```

lemma diag-block-mat-commute:
  assumes length Al = length Bl

```

**and**  $\forall i < \text{length } Al. Al!i * (Bl!i) = Bl!i * (Al!i)$   
**and**  $\forall i < \text{length } Al. \text{dim-col } (Al ! i) = \text{dim-row } (Bl ! i)$   
**and**  $\forall i < \text{length } Al. \text{dim-col } (Bl ! i) = \text{dim-row } (Al ! i)$   
**shows**  $\text{diag-block-mat } Al * (\text{diag-block-mat } Bl) =$   
 $\text{diag-block-mat } Bl * (\text{diag-block-mat } Al)$   
**proof** –  
**have**  $\text{diag-block-mat } Al * \text{diag-block-mat } Bl =$   
 $\text{diag-block-mat } (\text{map2 } (*) Al Bl)$   
**using**  $\text{diag-block-mat-mult}[of Al Bl]$  **assms by simp**  
**also have**  $\dots = \text{diag-block-mat } (\text{map2 } (*) Bl Al)$   
**proof** –  
**have**  $\text{map2 } (*) Al Bl = \text{map2 } (*) Bl Al$   
**by**  $(\text{rule map2-commute, auto simp add: assms})$   
**thus**  $?thesis$  **by simp**  
**qed**  
**also have**  $\dots = \text{diag-block-mat } Bl * (\text{diag-block-mat } Al)$   
**using**  $\text{diag-block-mat-mult}[of Bl Al]$  **assms by simp**  
**finally show**  $?thesis$  .  
**qed**

**lemma**  $\text{diag-block-mat-length-1}$ :  
**assumes**  $\text{length } Al = 1$   
**shows**  $\text{diag-block-mat } Al = Al!0$   
**proof** –  
**have**  $Al = [Al!0]$  **using**  $assms$   
**by**  $(\text{metis One-nat-def length-0-conv length-Suc-conv nth-Cons-0})$   
**thus**  $?thesis$   
**by**  $(\text{metis diag-block-mat-singleton})$   
**qed**

**lemma**  $\text{diag-block-mat-cong-hd}$ :  
**assumes**  $0 < \text{length } Al$   
**and**  $\text{length } Al = \text{length } Bl$   
**and**  $\text{dim-row } (\text{hd } Al) = \text{dim-row } (\text{hd } Bl)$   
**and**  $\text{dim-col } (\text{hd } Al) = \text{dim-col } (\text{hd } Bl)$   
**and**  $\text{diag-block-mat } Al = \text{diag-block-mat } Bl$   
**shows**  $\text{hd } Al = \text{hd } Bl$   
**proof** –  
**have**  $Al \neq []$  **using**  $assms$  **by blast**  
**hence**  $Al = \text{hd } Al \# (\text{tl } Al)$  **by simp**  
**hence**  $\text{da:diag-block-mat } Al =$   
 $\text{four-block-diag } (\text{hd } Al) (\text{diag-block-mat } (\text{tl } Al))$   
**using**  $\text{diag-block-mat.simps}(2)[of hd Al tl Al]$  **unfolding Let-def by simp**  
**have**  $Bl \neq []$  **using**  $assms$  **by simp**  
**hence**  $Bl = \text{hd } Bl \# (\text{tl } Bl)$  **by simp**  
**hence**  $\text{diag-block-mat } Bl = \text{four-block-diag } (\text{hd } Bl) (\text{diag-block-mat } (\text{tl } Bl))$   
**using**  $\text{diag-block-mat.simps}(2)[of hd Bl tl Bl]$  **unfolding Let-def by simp**  
**hence**  $\text{four-block-diag } (\text{hd } Al) (\text{diag-block-mat } (\text{tl } Al)) =$   
 $\text{four-block-diag } (\text{hd } Bl) (\text{diag-block-mat } (\text{tl } Bl))$  **using da assms by simp**

thus ?thesis using four-block-diag-cong-comp assms by metis  
qed

lemma *diag-block-mat-cong-tl*:

assumes  $0 < \text{length } Al$   
and  $\text{length } Al = \text{length } Bl$   
and  $\text{dim-row } (\text{hd } Al) = \text{dim-row } (\text{hd } Bl)$   
and  $\text{dim-col } (\text{hd } Al) = \text{dim-col } (\text{hd } Bl)$   
and  $\text{diag-block-mat } Al = \text{diag-block-mat } Bl$   
shows  $\text{diag-block-mat } (\text{tl } Al) = \text{diag-block-mat } (\text{tl } Bl)$

proof –

have  $Al \neq []$  using *assms* by *blast*  
hence  $Al = \text{hd } Al \# (\text{tl } Al)$  by *simp*  
hence  $\text{da:diag-block-mat } Al =$   
 $\text{four-block-diag } (\text{hd } Al) (\text{diag-block-mat } (\text{tl } Al))$   
using *diag-block-mat.simps(2)[of hd Al tl Al]* unfolding *Let-def* by *simp*  
have  $Bl \neq []$  using *assms* by *simp*  
hence  $Bl = \text{hd } Bl \# (\text{tl } Bl)$  by *simp*  
hence  $\text{diag-block-mat } Bl = \text{four-block-diag } (\text{hd } Bl) (\text{diag-block-mat } (\text{tl } Bl))$   
using *diag-block-mat.simps(2)[of hd Bl tl Bl]* unfolding *Let-def* by *simp*  
hence  $\text{four-block-diag } (\text{hd } Al) (\text{diag-block-mat } (\text{tl } Al)) =$   
 $\text{four-block-diag } (\text{hd } Bl) (\text{diag-block-mat } (\text{tl } Bl))$  using *da assms* by *simp*  
thus ?thesis using *four-block-diag-cong-comp' assms* by *metis*

qed

lemma *diag-block-mat-cong-comp*:

assumes  $\text{length } Al = \text{length } Bl$   
and  $\forall i < \text{length } Al. \text{dim-row } (Al ! i) = \text{dim-row } (Bl ! i)$   
and  $\forall i < \text{length } Al. \text{dim-col } (Al ! i) = \text{dim-col } (Bl ! i)$   
and  $\text{diag-block-mat } Al = \text{diag-block-mat } Bl$

and  $j < \text{length } Al$

shows  $Al ! j = Bl ! j$  using *assms*

proof (induct *Al* arbitrary: *Bl j*)

case *Nil*

then show ?case by *simp*

next

case (*Cons a Al*)

hence  $0 < \text{length } Bl$  by *linarith*

hence  $Bl = \text{hd } Bl \# (\text{tl } Bl)$  by *simp*

then show ?case

proof (cases  $j = 0$ )

case *True*

hence  $(a \# Al) ! j = \text{hd } (a \# Al)$  by *simp*

have  $Bl ! j = \text{hd } Bl$  using  $\langle j = 0 \rangle$

by (*metis*  $\langle Bl = \text{hd } Bl \# \text{tl } Bl \rangle$  *nth-Cons-0*)

have *da*:  $\text{diag-block-mat } (a \# Al) = \text{four-block-diag } a (\text{diag-block-mat } Al)$

using *diag-block-mat.simps(2)[of a Al]* unfolding *Let-def* by *simp*

have *db*:  $\text{diag-block-mat } (\text{hd } Bl \# (\text{tl } Bl)) =$

$\text{four-block-diag } (\text{hd } Bl) (\text{diag-block-mat } (\text{tl } Bl))$

```

    using diag-block-mat.simps(2)[of hd Bl tl Bl]
  unfolding Let-def by simp
  have hd (a#Al) = hd Bl
  proof (rule diag-block-mat-cong-hd)
    show 0 < length (a # Al) by simp
    show length (a # Al) = length Bl using Cons by simp
    show diag-block-mat (a # Al) = diag-block-mat Bl using Cons by simp
    show dim-row (hd (a # Al)) = dim-row (hd Bl)
      by (metis True <0 < length Bl> <Bl ! j = hd Bl> list.sel(1) Cons(2)
          Cons(3) nth-Cons-0)
    show dim-col (hd (a # Al)) = dim-col (hd Bl)
      by (metis True <0 < length Bl> <Bl ! j = hd Bl> list.sel(1) Cons(2)
          Cons(4) nth-Cons-0)
  qed
  thus (a # Al) ! j = Bl ! j using <j = 0> <Bl ! j = hd Bl> by fastforce
next
case False
hence  $\exists k. j = \text{Suc } k$  by (simp add: not0-implies-Suc)
from this obtain k where j = Suc k by auto
hence (a#Al)!j = Al!k by simp
have Bl!j = (tl Bl)!k using <j = Suc k> <Bl = hd Bl#(tl Bl)>
  by (metis nth-Cons-Suc)
have Al!k = (tl Bl)!k
proof (rule Cons(1))
  show length Al = length (tl Bl) using Cons
    by (metis diff-Suc-1 length-Cons length-tl)
  show k < length Al
    by (metis Cons.prem5) Suc-less-SucD <j = Suc k> length-Cons)
  show  $\forall i < \text{length } Al. \text{dim-row } (Al ! i) = \text{dim-row } (tl Bl ! i)$ 
    by (metis Suc-less-eq <length Al = length (tl Bl)> length-Cons
        local.Cons(3) nth-Cons-Suc nth-tl)
  show  $\forall i < \text{length } Al. \text{dim-col } (Al ! i) = \text{dim-col } (tl Bl ! i)$ 
    by (metis Suc-mono <Bl = hd Bl # tl Bl> length-Cons local.Cons(4)
        nth-Cons-Suc)
  have diag-block-mat (tl (a#Al)) = diag-block-mat (tl Bl)
  proof (rule diag-block-mat-cong-tl)
    show length (a # Al) = length Bl using Cons by simp
    show dim-row (hd (a # Al)) = dim-row (hd Bl)
      by (metis <Bl = hd Bl # tl Bl> length-Cons list.sel(1) local.Cons(3)
          nth-Cons-0 zero-less-Suc)
    show dim-col (hd (a # Al)) = dim-col (hd Bl)
      by (metis <0 < length Bl> <Bl = hd Bl # tl Bl> list.sel(1)
          local.Cons(2) local.Cons(4) nth-Cons-0)
    show diag-block-mat (a # Al) = diag-block-mat Bl using Cons by simp
    show 0 < length (a#Al) by simp
  qed
  thus diag-block-mat Al = diag-block-mat (tl Bl) by simp
qed
then show ?thesis

```

by (simp add: ⟨a # Al⟩ ! j = Al ! k⟩ ⟨Bl ! j = tl Bl ! k⟩)  
 qed  
 qed

**lemma** *diag-block-mat-commute-comp*:

assumes  $\text{length } Al = \text{length } Bl$   
 and  $\forall i < \text{length } Al. \text{dim-row } (Al ! i) = \text{dim-col } (Al ! i)$   
 and  $\forall i < \text{length } Al. \text{dim-row } (Al ! i) = \text{dim-row } (Bl ! i)$   
 and  $\forall i < \text{length } Al. \text{dim-col } (Al ! i) = \text{dim-col } (Bl ! i)$   
 and  $\text{diag-block-mat } Al * (\text{diag-block-mat } Bl) =$   
 $\text{diag-block-mat } Bl * (\text{diag-block-mat } Al)$   
 and  $i < \text{length } Al$

shows  $Al ! i * Bl ! i = Bl ! i * Al ! i$

**proof** –

have  $\text{diag-block-mat } (\text{map2 } (*) Al Bl) = \text{diag-block-mat } Al * \text{diag-block-mat } Bl$   
 using *diag-block-mat-mult*[of  $Al$ ] *assms* **by** *simp*  
 also have  $\dots = \text{diag-block-mat } Bl * \text{diag-block-mat } Al$  **using** *assms* **by** *simp*  
 also have  $\dots = \text{diag-block-mat } (\text{map2 } (*) Bl Al)$   
 using *diag-block-mat-mult*[of  $Bl$ ] *assms* **by** *simp*  
 finally have *eq*:  $\text{diag-block-mat } (\text{map2 } (*) Al Bl) =$   
 $\text{diag-block-mat } (\text{map2 } (*) Bl Al)$  .

have  $(\text{map2 } (*) Al Bl) ! i = (\text{map2 } (*) Bl Al) ! i$

**proof** (*rule* *diag-block-mat-cong-comp*)

show  $\text{length } (\text{map2 } (*) Al Bl) = \text{length } (\text{map2 } (*) Bl Al)$

using *map2-length* *assms* **by** *metis*

show  $i < \text{length } (\text{map2 } (*) Al Bl)$  **using** *map2-length* *assms* **by** *metis*

show  $\text{diag-block-mat } (\text{map2 } (*) Al Bl) = \text{diag-block-mat } (\text{map2 } (*) Bl Al)$

using *eq* .

show  $\forall i < \text{length } (\text{map2 } (*) Al Bl). \text{dim-row } (\text{map2 } (*) Al Bl ! i) =$   
 $\text{dim-row } (\text{map2 } (*) Bl Al ! i)$

**by** (*simp* add: *assms*(3))

show  $\forall i < \text{length } (\text{map2 } (*) Al Bl). \text{dim-col } (\text{map2 } (*) Al Bl ! i) =$   
 $\text{dim-col } (\text{map2 } (*) Bl Al ! i)$

**by** (*simp* add: *assms*(4))

qed

moreover have  $(\text{map2 } (*) Al Bl) ! i = Al ! i * Bl ! i$  **using** *assms* **by** *simp*

moreover have  $(\text{map2 } (*) Bl Al) ! i = Bl ! i * Al ! i$  **using** *assms* **by** *simp*

ultimately show *?thesis* **by** *simp*

qed

**lemma** *diag-block-mat-dim-row-cong*:

assumes  $\text{length } Ul = \text{length } Bl$

and  $\forall i < \text{length } Bl. \text{dim-row } (Bl ! i) = \text{dim-row } (Ul ! i)$

shows  $\text{dim-row } (\text{diag-block-mat } Ul) = \text{dim-row } (\text{diag-block-mat } Bl)$

**proof** –

have  $\text{dim-row } (\text{diag-block-mat } Ul) = \text{sum-list } (\text{map } \text{dim-row } Ul)$

**by** (*simp* add: *dim-diag-block-mat*(1))

also have  $\dots = \text{sum-list } (\text{map } \text{dim-row } Bl)$  **using** *assms*

**by** (*metis* *nth-map-conv*)

```

also have ... = dim-row (diag-block-mat Bl)
  by (simp add: dim-diag-block-mat(1))
finally show ?thesis .
qed

lemma diag-block-mat-dim-col-cong:
  assumes length Ul = length Bl
  and  $\forall i < \text{length } Bl. \text{dim-col } (Bl!i) = \text{dim-col } (Ul!i)$ 
  shows dim-col (diag-block-mat Ul) = dim-col (diag-block-mat Bl)
proof -
  have dim-col (diag-block-mat Ul) = sum-list (map dim-col Ul)
    by (simp add: dim-diag-block-mat(2))
  also have ... = sum-list (map dim-col Bl) using assms
    by (metis nth-map-conv)
  also have ... = dim-col (diag-block-mat Bl)
    by (simp add: dim-diag-block-mat(2))
  finally show ?thesis .
qed

```

```

lemma diag-block-mat-dim-row-col-eq:
  assumes  $\forall i < \text{length } Al. \text{dim-row } (Al!i) = \text{dim-col } (Al!i)$ 
  shows dim-row (diag-block-mat Al) = dim-col (diag-block-mat Al)
proof -
  have dim-row (diag-block-mat Al) = sum-list (map dim-row Al)
    by (simp add: dim-diag-block-mat(1))
  also have ... = sum-list (map dim-col Al) using assms
    by (metis nth-map-conv)
  also have ... = dim-col (diag-block-mat Al)
    by (simp add: dim-diag-block-mat(2))
  finally show ?thesis .
qed

```

## 6 Block matrix decomposition

### 6.1 Subdiagonal extraction

`extract_subdiags` returns a list of diagonal sub-blocks, the sizes of which are specified by the list of integers provided as parameters.

```

fun extract-subdiags where
  extract-subdiags B [] = []
| extract-subdiags B (x#xs) =
  (let (B1, B2, B3, B4) = (split-block B x x) in
   B1 # (extract-subdiags B4 xs))

```

```

lemma extract-subdiags-not-emp:
  fixes x::nat and l::nat list
  assumes (B1, B2, B3, B4) = (split-block B x x)
  shows hd (extract-subdiags B (x#l)) = B1

```

```

    tl (extract-subdiags B (x#l)) = extract-subdiags B4 l
proof –
  show hd (extract-subdiags B (x#l)) = B1 unfolding Let-def
    by (metis (no-types) assms extract-subdiags.simps(2) list.sel(1) split-conv)
  show tl (extract-subdiags B (x # l)) = extract-subdiags B4 l
    using assms extract-subdiags.simps(2) unfolding Let-def
    by (metis (no-types, lifting) list.sel(3) split-conv)
qed

lemma extract-subdiags-neq-Nil:
  shows extract-subdiags B (a#l) ≠ []
  using extract-subdiags.simps(2)[of B]
  unfolding Let-def split-block-def by simp

lemma extract-subdiags-length:
  shows length (extract-subdiags B l) = length l
proof (induct l arbitrary: B)
  case Nil
  then show ?case by simp
next
  case (Cons a l)
  define B1 where B1 = fst (split-block B a a)
  define B2 where B2 = fst (snd (split-block B a a))
  define B3 where B3 = fst (snd (snd (split-block B a a)))
  define B4 where B4 = snd (snd (snd (split-block B a a)))
  have sp: split-block B a a = (B1, B2, B3, B4) using fst-conv snd-conv
    unfolding B1-def B2-def B3-def B4-def by simp
  then show ?case using Cons extract-subdiags.simps(2)[of B a l]
    unfolding Let-def by simp
qed

lemma extract-subdiags-carrier:
  assumes i < length l
  shows (extract-subdiags B l)!i ∈ carrier-mat (!l i) (!l i) using assms
proof (induct i arbitrary: l B)
  case 0
  define B1 where B1 = fst (split-block B (hd l) (hd l))
  define B2 where B2 = fst (snd (split-block B (hd l) (hd l)))
  define B3 where B3 = fst (snd (snd (split-block B (hd l) (hd l))))
  define B4 where B4 = snd (snd (snd (split-block B (hd l) (hd l))))
  have sp: split-block B (hd l) (hd l) = (B1, B2, B3, B4) using fst-conv snd-conv

    unfolding B1-def B2-def B3-def B4-def by simp
  have l = hd l # (tl l) using 0 by auto
  have (extract-subdiags B l)!0 = B1
    using extract-subdiags.simps(2)[of B hd l tl l] ‹l = hd l # tl l› sp
    unfolding Let-def by auto
  also have ... ∈ carrier-mat (hd l) (hd l)
    unfolding B1-def split-block-def Let-def by simp

```

```

finally show ?case
  by (metis ‹l = hd l # tl ‹ hd-conv-nth list.sel(2) not-Cons-self)
next
  case (Suc i)
  define B1 where B1 = fst (split-block B (hd l) (hd l))
  define B2 where B2 = fst (snd (split-block B (hd l) (hd l)))
  define B3 where B3 = fst (snd (snd (split-block B (hd l) (hd l))))
  define B4 where B4 = snd (snd (snd (split-block B (hd l) (hd l))))
  have sp: split-block B (hd l) (hd l) = (B1, B2, B3, B4) using fst-conv snd-conv

  unfolding B1-def B2-def B3-def B4-def by simp
  have l = hd l # (tl l) using Suc
  by (metis Cons-nth-drop-Suc drop-Nil list.exhaust-sel not-Cons-self)
  hence ! Suc i = (tl l)!i by (metis nth-Cons-Suc)
  have tl (extract-subdiags B l) = extract-subdiags B4 (tl l)
    using extract-subdiags-not-emp(2)[OF sp[symmetric]] ‹l = hd l # (tl l)›
    by metis
  hence extract-subdiags B l = B1 # extract-subdiags B4 (tl l)
    using extract-subdiags-not-emp(1)[OF sp[symmetric]]
    by (metis ‹l = hd l # tl ‹ extract-subdiags-neq-Nil list.exhaust-sel)
  hence extract-subdiags B l ! Suc i = (extract-subdiags B4 (tl l))!i
    using nth-Cons-Suc by simp
  also have ... ∈ carrier-mat (tl l)!i (tl l)!i using Suc
    by (metis ‹l = hd l # tl ‹ length-Cons not-less-eq)
  also have ... = carrier-mat (!Suc i) (!Suc i)
    using nth-Cons-Suc[of hd l tl l i] ‹l = hd l # tl ‹ by simp
  finally show ?case .
qed

```

**lemma** extract-subdiags-diagonal:

```

assumes diagonal-mat B
and B ∈ carrier-mat n n
and l ≠ []
and sum-list l ≤ n
and i < length l
shows diagonal-mat ((extract-subdiags B l)!i) using assms
proof (induct i arbitrary: l B n)
  case 0
  define a where a = hd l
  have l = a#(tl l) unfolding a-def using 0 by simp
  have a ≤ n using 0 unfolding a-def
    by (metis a-def dual-order.strict-trans2 elem-le-sum-list
      hd-conv-nth less-le-not-le nat-le-linear)
  define B1 where B1 = fst (split-block B a a)
  define B2 where B2 = fst (snd (split-block B a a))
  define B3 where B3 = fst (snd (snd (split-block B a a)))
  define B4 where B4 = snd (snd (snd (split-block B a a)))
  have sp: split-block B a a = (B1, B2, B3, B4) using fst-conv snd-conv
    unfolding B1-def B2-def B3-def B4-def by simp

```

```

hence extract-subdiags B !0 = B1 unfolding a-def
using hd-conv-nth 0
by (metis ⟨l = a # tl l⟩ sp extract-subdiags-neq-Nil
      extract-subdiags-not-emp(1))
moreover have diagonal-mat B1 using sp split-block-diagonal assms ⟨a ≤ n⟩ 0
by blast
ultimately show ?case by simp
next
case (Suc i)
show ?case
proof (cases length l = 1)
  case True
    hence Suc i = 0 using Suc by presburger
    then show ?thesis by simp
  next
    case False
      define a where a = hd l
      have l = a#(tl l) unfolding a-def using Suc by simp
      have a ≤ n using Suc unfolding a-def
        by (metis dual-order.trans elem-le-sum-list hd-conv-nth
              length-greater-0-conv)
      define B1 where B1 = fst (split-block B a a)
      define B2 where B2 = fst (snd (split-block B a a))
      define B3 where B3 = fst (snd (snd (split-block B a a)))
      define B4 where B4 = snd (snd (snd (split-block B a a)))
      have sp: split-block B a a = (B1, B2, B3, B4) using fst-conv snd-conv
        unfolding B1-def B2-def B3-def B4-def by simp
      have extract-subdiags B l ! Suc i =
        extract-subdiags B4 (tl l)! i using sp
        by (metis Suc(6) Suc-less-SucD ⟨l = a # tl l⟩ length-Cons nth-tl
              extract-subdiags-length extract-subdiags-not-emp(2))
      moreover have diagonal-mat (extract-subdiags B4 (tl l)! i)
        proof (rule Suc(1))
          show tl l ≠ [] using False Suc
            by (metis ⟨l = a # tl l⟩ length-Cons list.size(3) numeral-nat(7))
          show i < length (tl l) using False Suc
            by (metis Suc-lessD ⟨l = a # tl l⟩ le-neq-implies-less length-Cons
                  less-Suc-eq-le)
          show B4 ∈ carrier-mat (n-a) (n-a)
            using sp split-block-diag-carrier(2) Suc(3) ⟨a ≤ n⟩ by blast
          show diagonal-mat B4
            using split-block-diagonal sp Suc ⟨a ≤ n⟩ by blast
          show sum-list (tl l) ≤ n - a using Suc(5) ⟨a ≤ n⟩ sum-list-tl-leq
            by (simp add: Suc(4) a-def)
        qed
      ultimately show ?thesis by simp
    qed
  qed
qed

```

```

lemma extract-subdiags-diag-elim:
  fixes  $B :: \text{complex Matrix.mat}$ 
  assumes  $B \in \text{carrier-mat } n \ n$ 
  and  $0 < n$ 
  and  $l \neq []$ 
  and  $i < \text{length } l$ 
  and  $j < !i$ 
  and  $\text{sum-list } l \leq n$ 
  and  $\forall j < \text{length } l. 0 < !j$ 
  shows  $\text{extract-subdiags } B \ !i \ \$\$ (j,j) =$ 
     $\text{diag-mat } B!(n-\text{sum } i \ l + j) \ \text{using } \text{assms}$ 
proof (induct i arbitrary: l B n)
  case 0
  define  $a$  where  $a = \text{hd } l$ 
  have  $l = a \# (\text{tl } l)$  unfolding  $a\text{-def}$  using 0 by simp
  have  $a \leq n$  using 0 unfolding  $a\text{-def}$ 
    by (metis a-def dual-order.strict-trans2 elem-le-sum-list
      hd-conv-nth less-le-not-le nat-le-linear)
  define  $B1$  where  $B1 = \text{fst } (\text{split-block } B \ a \ a)$ 
  define  $B2$  where  $B2 = \text{fst } (\text{snd } (\text{split-block } B \ a \ a))$ 
  define  $B3$  where  $B3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } B \ a \ a)))$ 
  define  $B4$  where  $B4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } B \ a \ a)))$ 
  have  $sp: \text{split-block } B \ a \ a = (B1, B2, B3, B4)$  using fst-conv snd-conv
    unfolding  $B1\text{-def } B2\text{-def } B3\text{-def } B4\text{-def}$  by simp
  hence  $\text{extract-subdiags } B \ !0 = B1$ 
    using hd-conv-nth unfolding  $Let\text{-def}$ 
    by (metis l = a # tl l extract-subdiags-neq-Nil
      extract-subdiags-not-emp(1))
  hence  $\text{extract-subdiags } B \ !0 \ \$\$ (j,j) = B \ \$\$ (j,j)$ 
    using  $sp \ 0$  unfolding  $\text{split-block-def}$ 
    by (metis (no-types, lifting) carrier-matD(2) dim-col-mat(1)
      index-mat(1) prod.sel(1) extract-subdiags-carrier)
  also have  $\dots = \text{diag-mat } B!j$ 
    using  $0 \ \langle a \leq n \rangle$  hd-conv-nth unfolding  $\text{diag-mat-def } a\text{-def}$ 
    by fastforce
  also have  $\dots = \text{diag-mat } B!(n-\text{sum } 0 \ l + j)$  by simp
  finally show ?case .
next
  case (Suc i)
  show ?case
  proof (cases length l = 1)
    case True
    hence  $\text{Suc } i < 0$  using Suc by simp
    then show ?thesis by simp
  next
  case False
  hence  $1 < \text{length } l$  using Suc by presburger
  define  $a$  where  $a = \text{hd } l$ 
  have  $l = a \# (\text{tl } l)$  unfolding  $a\text{-def}$  using Suc by simp

```

**have**  $a \leq n$  **using** *Suc* **unfolding** *a-def*  
**by** (*metis add-le-same-cancel1 elem-le-sum-list hd-conv-nth*  
*le-add2 le-trans verit-comp-simplify1 (3)*)  
**define**  $B_1$  **where**  $B_1 = \text{fst} (\text{split-block } B \ a \ a)$   
**define**  $B_2$  **where**  $B_2 = \text{fst} (\text{snd} (\text{split-block } B \ a \ a))$   
**define**  $B_3$  **where**  $B_3 = \text{fst} (\text{snd} (\text{snd} (\text{split-block } B \ a \ a)))$   
**define**  $B_4$  **where**  $B_4 = \text{snd} (\text{snd} (\text{snd} (\text{split-block } B \ a \ a)))$   
**have**  $sp$ :  $\text{split-block } B \ a \ a = (B_1, B_2, B_3, B_4)$  **using** *fst-conv snd-conv*  
**unfolding** *B1-def B2-def B3-def B4-def* **by** *simp*  
**have**  $B_4 \in \text{carrier-mat } (n-a) \ (n-a)$   
**using**  $sp$  *split-block-diag-carrier(2) Suc*  $\langle a \leq n \rangle$  **by** *blast*  
**have**  $B_1 \in \text{carrier-mat } a \ a$   
**using**  $sp$  *split-block-diag-carrier(1) Suc*  $\langle a \leq n \rangle$  **by** *blast*  
**have**  $n\text{-sum } (Suc \ i) \ l + j < n\text{-sum } (Suc \ (Suc \ i)) \ l$   
**using** *Suc n-sum-last-lt* **by** *metis*  
**hence**  $a + n\text{-sum } i \ (tl \ l) + j < n\text{-sum } (Suc \ (Suc \ i)) \ l$   
**unfolding** *a-def* **by** *simp*  
**also have**  $\dots \leq \text{sum-list } l$   
**proof** (*rule n-sum-sum-list*)  
**show**  $\forall j < \text{length } l. \ 0 \leq l \ ! \ j$  **using** *Suc* **by** *simp*  
**show**  $Suc \ (Suc \ i) \leq \text{length } l$  **using** *Suc* **by** *simp*  
**qed**  
**also have**  $\dots \leq n$  **using** *Suc* **by** *simp*  
**finally have**  $a + n\text{-sum } i \ (tl \ l) + j < n$  .  
**hence**  $n\text{-sum } i \ (tl \ l) + j < n - a$  **by** *simp*  
**have**  $\text{extract-subdiags } B \ !!(Suc \ i) =$   
 $\text{extract-subdiags } B_4 \ (tl \ l) ! i$   
**using**  $sp \ \langle l = a \ \# \ (tl \ l) \rangle$  **unfolding** *Let-def*  
**by** (*metis list.exhaust-sel nth-Cons-Suc extract-subdiags-neq-Nil*  
*extract-subdiags-not-emp(2)*)  
**hence**  $\text{extract-subdiags } B \ !!(Suc \ i) \ \$(j,j) =$   
 $\text{extract-subdiags } B_4 \ (tl \ l) ! i \ \$(j,j)$  **by** *simp*  
**also have**  $\dots = \text{diag-mat } B_4 \ !(n\text{-sum } i \ (tl \ l) + j)$   
**proof** (*rule Suc(1)*)  
**show**  $tl \ l \neq []$  **using** *False Suc*  
**by** (*metis*  $\langle l = a \ \# \ tl \ l \rangle$  *length-Cons list.size(3) numeral-nat(7)*)  
**show**  $i < \text{length } (tl \ l)$  **using** *False Suc*  
**by** (*metis Suc-lessD*  $\langle l = a \ \# \ tl \ l \rangle$  *le-neq-implies-less length-Cons*  
*less-Suc-eq-le*)  
**show**  $B_4 \in \text{carrier-mat } (n-a) \ (n-a)$   
**using**  $\langle B_4 \in \text{carrier-mat } (n-a) \ (n-a) \rangle$  .  
**show**  $\text{sum-list } (tl \ l) \leq n - a$  **using** *Suc(5)*  $\langle a \leq n \rangle$  *sum-list-tl-leq*  
**by** (*simp add: Suc a-def*)  
**show**  $0 < n - a$   
**by** (*metis Suc.prem(4) Suc.prem(7)*  $\langle i < \text{length } (tl \ l) \rangle$   
 $\langle l = a \ \# \ tl \ l \rangle$   $\langle \text{sum-list } (tl \ l) \leq n - a \rangle$  *bot-nat-0.extremum-uniqueI*  
*elem-le-sum-list gr-zeroI nth-Cons-Suc*)  
**show**  $\forall j < \text{length } (tl \ l). \ 0 < tl \ l \ ! \ j$   
**by** (*simp add: Suc(8) nth-tl*)

```

    show  $j < tl\ l\ !\ i$ 
      by (metis Suc(6)  $\langle i < length\ (tl\ l)\ \rangle\ nth\ tl$ )
    qed
  also have ... =  $B_4\ \$\$(n\ sum\ i\ (tl\ l)\ +\ j,\ n\ sum\ i\ (tl\ l)\ +\ j)$ 
  proof -
    have  $n\ sum\ i\ (tl\ l)\ +\ j < n - a$  using  $\langle n\ sum\ i\ (tl\ l)\ +\ j < n - a \rangle$  .
    thus ?thesis
      using  $\langle B_4 \in carrier\ mat\ (n-a)\ (n-a) \rangle$ 
      unfolding diag-mat-def by simp
    qed
  also have ... =  $B\ \$\$(n\ sum\ i\ (tl\ l)\ +\ j\ +\ a,\ n\ sum\ i\ (tl\ l)\ +\ j\ +\ a)$ 
    using sp  $\langle B_1 \in carrier\ mat\ a\ a \rangle\ \langle n\ sum\ i\ (tl\ l)\ +\ j < n - a \rangle$ 
       $\langle B_4 \in carrier\ mat\ (n-a)\ (n-a) \rangle\ carrier\ matD(2)\ dim\ col\ mat(1)\ Suc$ 
      index-mat(1) prod.sel
    unfolding split-block-def Let-def by force
  also have ... =  $diag\ mat\ B\ !\ (n\ sum\ i\ (tl\ l)\ +\ j\ +\ a)$ 
  proof -
    have  $n\ sum\ i\ (tl\ l)\ +\ j\ +\ a < n$  using  $\langle n\ sum\ i\ (tl\ l)\ +\ j < n - a \rangle$ 
      by simp
    thus ?thesis using Suc unfolding diag-mat-def by simp
  qed
  also have ... =  $diag\ mat\ B\ !\ (n\ sum\ (Suc\ i)\ l\ +\ j)$ 
  proof -
    have  $n\ sum\ i\ (tl\ l)\ +\ a = n\ sum\ (Suc\ i)\ l$  unfolding a-def by simp
    thus ?thesis
      by (simp add: add.commute add.left-commute)
  qed
  finally show ?thesis .
  qed
qed

```

**lemma** *hermitian-extract-subdiags*:

```

  assumes hermitian A
  and sum-list  $l \leq dim\ row\ A$ 
  and list-all  $(\lambda a.\ 0 < a)\ l$ 
  shows list-all  $(\lambda B.\ 0 < dim\ row\ B \wedge hermitian\ B)\ (extract\ subdiags\ A\ l)$ 
  using assms
  proof (induct l arbitrary: A)
    case Nil
    then show ?case by simp
  next
    case (Cons a l)
    define es where  $es = extract\ subdiags\ A\ (a\ \#\ l)$ 
    define B1 where  $B1 = fst\ (split\ block\ A\ a\ a)$ 
    define B2 where  $B2 = fst\ (snd\ (split\ block\ A\ a\ a))$ 
    define B3 where  $B3 = fst\ (snd\ (snd\ (split\ block\ A\ a\ a)))$ 
    define B4 where  $B4 = snd\ (snd\ (snd\ (split\ block\ A\ a\ a)))$ 
    have sp:  $split\ block\ A\ a\ a = (B1,\ B2,\ B3,\ B4)$  using fst-conv snd-conv
      unfolding B1-def B2-def B3-def B4-def by simp
  
```

```

have 0 < a using Cons by simp
have es ≠ [] using extract-subdiags-neq-Nil[of A]
  unfolding es-def by simp
hence es = hd es # (tl es) by simp
have hd es = B1 unfolding es-def
  using extract-subdiags-not-emp(1)[OF sp[symmetric]] by simp
have dim-row B1 = a unfolding B1-def split-block-def Let-def by simp
have tl es = extract-subdiags B4 l unfolding es-def
  using extract-subdiags-not-emp(2)[OF sp[symmetric]] by simp
have list-all (λB. 0 < dim-row B ∧ hermitian B) (hd es # (tl es))
proof (rule list-all-Cons-iff [THEN iffD2], intro conjI)
  show hermitian (hd es)
  proof (rule split-block-hermitian-1)
    show hermitian A using Cons by simp
    show ⟨hd es, B2, B3, B4⟩ = split-block A a a using sp ⟨hd es = B1⟩
      by simp
    show a ≤ dim-row A using Cons by simp
  qed
  have list-all (λB. 0 < dim-row B ∧ hermitian B) (extract-subdiags B4 l)
  proof (rule Cons(1))
    show hermitian B4
    proof (rule split-block-hermitian-4)
      show hermitian A using Cons by simp
      show a ≤ dim-row A using Cons by simp
      show ⟨B1, B2, B3, B4⟩ = split-block A a a using sp by simp
    qed
    show sum-list l ≤ dim-row B4 using Cons sp
      unfolding split-block-def Let-def by force
    show list-all ((<) 0) l using Cons(4) by auto
  qed
  thus list-all (λB. 0 < dim-row B ∧ hermitian B) (tl es)
    using ⟨tl es = extract-subdiags B4 l⟩ by simp
  show 0 < dim-row (hd es)
    using ⟨hd es = B1⟩ ⟨0 < a⟩ ⟨dim-row B1 = a⟩ by simp
  qed
  thus ?case using ⟨es = hd es # (tl es)⟩ unfolding es-def by metis
qed

```

## 6.2 Predicates on diagonal block matrices

The predicate `diag_compat` ensures that the provided matrix, when decomposed according to the list of integers provided as an input, is indeed a diagonal block matrix.

```

fun diag_compat where
  diag_compat B [] = (dim-row B = 0 ∧ dim-col B = 0)
| diag_compat B (x#xs) =
  (x ≤ dim-row B ∧
   (let n = dim-row B; (B1, B2, B3, B4) = (split-block B x x) in
    B2 = (0m x (n - x)) ∧ B3 = (0m (n - x) x) ∧ diag_compat B4 xs))

```

When this is the case, the decomposition of a matrix leaves it unchanged.

**lemma** *diag-compat-extract-subdiag*:

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

**and** *diag-compat*  $B \ l$

**shows**  $B = \text{diag-block-mat } (\text{extract-subdiags } B \ l)$  **using** *assms*

**proof** (*induct l arbitrary:B n*)

**case** *Nil*

**have** *extract-subdiags*  $B \ \text{Nil} = []$  **by** *simp*

**have**  $B = 0_m \ 0 \ 0$

**proof** (*rule eq-matI, auto simp add: assms*)

**show**  $\text{dim-row } B = 0$  **using** *Nil* **by** *simp*

**show**  $\text{dim-col } B = 0$  **using** *Nil* **by** *simp*

**qed**

**then show** *?case* **using** *diag-block-mat-singleton[of B]* **by** *simp*

**next**

**case** (*Cons a l*)

**define**  $B1$  **where**  $B1 = \text{fst } (\text{split-block } B \ a \ a)$

**define**  $B2$  **where**  $B2 = \text{fst } (\text{snd } (\text{split-block } B \ a \ a))$

**define**  $B3$  **where**  $B3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } B \ a \ a)))$

**define**  $B4$  **where**  $B4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } B \ a \ a)))$

**have** *sp*: *split-block*  $B \ a \ a = (B1, B2, B3, B4)$  **using** *fst-conv snd-conv*

**unfolding** *B1-def B2-def B3-def B4-def* **by** *simp*

**have**  $a \leq n$  **using** *assms Cons* **by** *simp*

**have** *diag-compat*  $B4 \ l$  **using** *sp Cons* **by** (*simp add: Let-def*)

**have**  $B1 \in \text{carrier-mat } a \ a$  **using** *sp Cons split-block(1)[OF sp]*

**by** (*metis <a ≤ n> carrier-matD(1) carrier-matD(2) le-add-diff-inverse*)

**have**  $B2 \in \text{carrier-mat } a \ (n-a)$  **using** *sp Cons* **by** (*simp add: Let-def*)

**have**  $B3 \in \text{carrier-mat } (n-a) \ a$  **using** *sp Cons* **by** (*simp add: Let-def*)

**have**  $B4 \in \text{carrier-mat } (n-a) \ (n-a)$  **using** *assms <a ≤ n> Cons*

*split-block(4)[OF sp]* **by** *simp*

**have**  $b2: 0_m \ (\text{dim-row } B1) \ (\text{dim-col } B4) = B2$

**using** *diag-compat.simps(2)[THEN iffD1, OF <diag-compat B (a#l)>]*

$\langle B4 \in \text{carrier-mat } (n-a) \ (n-a) \rangle \langle B1 \in \text{carrier-mat } a \ a \rangle$

$\langle B2 \in \text{carrier-mat } a \ (n-a) \rangle$  *sp* **unfolding** *Let-def*

*Cons(2)* **by** *force*

**have**  $b3: 0_m \ (\text{dim-row } B4) \ (\text{dim-col } B1) = B3$

**using** *diag-compat.simps(2)[THEN iffD1, OF <diag-compat B (a#l)>]*

$\langle B4 \in \text{carrier-mat } (n-a) \ (n-a) \rangle \langle B1 \in \text{carrier-mat } a \ a \rangle$

$\langle B2 \in \text{carrier-mat } a \ (n-a) \rangle$  *sp* **unfolding** *Let-def*

*Cons(2)* **by** *force*

**have** *extract-subdiags*  $B \ (a\#l) = B1 \ \# \ (\text{extract-subdiags } B4 \ l)$

**using** *fst-conv snd-conv extract-subdiags.simps(2)[of B]*

**unfolding** *B1-def B4-def Let-def* **by** (*simp add: split-def*)

**also have** *diag-block-mat ... =*

(*let*

$C = \text{diag-block-mat } (\text{extract-subdiags } B4 \ l)$

*in* *four-block-mat*  $B1 \ (0_m \ (\text{dim-row } B1) \ (\text{dim-col } C))$

$(0_m \ (\text{dim-row } C) \ (\text{dim-col } B1)) \ C$ ) **by** *simp*

**also have**  $\dots = \text{four-block-mat } B1 \ (0_m \ (\text{dim-row } B1) \ (\text{dim-col } B4))$

$(0_m (\dim\text{-row } B_4) (\dim\text{-col } B_1)) B_4$  **using** *Cons*  $\langle \text{diag-compat } B_4 \rangle$   
 $\langle B_4 \in \text{carrier-mat } (n-a) (n-a) \rangle$  **by** (*simp add:Let-def*)  
**also have** ... = *four-block-mat*  $B_1 B_2 B_3 B_4$  **using**  $b_2 b_3$  **by** *simp*  
**also have** ... =  $B$  **using** *split-block(5)*[*OF sp, of n-a n-a*] *Cons* **by** *simp*  
**finally show** *?case* **by** *simp*  
**qed**

Predicate `diag_diff` holds when the decomposition of the considered matrix based on the list of integers provided as a parameter, is such that the diagonal elements of separate components are pairwise distinct.

**fun** *diag-diff* **where**

*diag-diff*  $D [] = (\dim\text{-row } D = 0 \wedge \dim\text{-col } D = 0)$   
| *diag-diff*  $D (x\#xs) =$   
 $(x \leq \dim\text{-row } D \wedge$   
 $(\text{let } (D_1, D_2, D_3, D_4) = (\text{split-block } D x) \text{ in}$   
 $(\forall i j. i < \dim\text{-row } D_1 \wedge j < \dim\text{-row } D_4 \longrightarrow D_1\$(i,i) \neq D_4\$(j,j)) \wedge$   
 $\text{diag-diff } D_4 xs))$

**lemma** *diag-diff-hd-diff*:

**assumes** *diag-diff*  $D (a\#xs)$   
**and**  $D \in \text{carrier-mat } n n$   
**and**  $i < a$   
**and**  $a \leq j$   
**and**  $j < n$

**shows**  $D\$(i,i) \neq D\$(j,j)$

**proof** –

**define**  $D_1$  **where**  $D_1 = \text{fst } (\text{split-block } D a a)$   
**define**  $D_2$  **where**  $D_2 = \text{fst } (\text{snd } (\text{split-block } D a a))$   
**define**  $D_3$  **where**  $D_3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } D a a)))$   
**define**  $D_4$  **where**  $D_4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } D a a)))$   
**have** *spd*: *split-block*  $D a a = (D_1, D_2, D_3, D_4)$  **using** *fst-conv snd-conv*  
**unfolding** *D1-def D2-def D3-def D4-def* **by** *simp*  
**have**  $c_1$ :  $D_1 \in \text{carrier-mat } a a$  **using** *split-block(1)*[*OF spd, of n-a n-a*]  
*assms* **by** *simp*  
**have**  $c_4$ :  $D_4 \in \text{carrier-mat } (n-a) (n-a)$  **using** *assms*  
*split-block(4)*[*OF spd*] **by** *simp*  
**hence**  $j - a < \dim\text{-row } D_4$  **using** *assms* **by** *simp*  
**have**  $D\$(i,i) = D_1\$(i,i)$  **using** *assms spd*  
**unfolding** *split-block-def Let-def* **by** *force*  
**moreover have**  $D\$(j,j) = D_4\$(j-a, j-a)$  **using** *assms spd*  
**unfolding** *split-block-def Let-def* **by** *force*  
**moreover have**  $D_1\$(i,i) \neq D_4\$(j-a, j-a)$   
**using** *assms*  $\langle j - a < \dim\text{-row } D_4 \rangle$  *spd c1 c4*  
*diag-diff.simps(2)*[*THEN iffD1, OF assms(1)*] **unfolding** *Let-def* **by** *simp*  
**ultimately show** *?thesis* **by** *simp*

**qed**

**lemma** *diag-compat-diagonal*:

**assumes**  $B \in \text{carrier-mat } (\dim\text{-row } B) (\dim\text{-row } B)$

```

and diagonal-mat B
and dim-row B = sum-list l
shows diag-compat B l using assms
proof (induct l arbitrary: B)
  case Nil
  then show ?case by simp
next
case (Cons a l)
define B1 where B1 = fst (split-block B a a)
define B2 where B2 = fst (snd (split-block B a a))
define B3 where B3 = fst (snd (snd (split-block B a a)))
define B4 where B4 = snd (snd (snd (split-block B a a)))
have sp: split-block B a a = (B1, B2, B3, B4) using fst-conv snd-conv
  unfolding B1-def B2-def B3-def B4-def by simp
have diagonal-mat B1  $\wedge$  diagonal-mat B4
proof (rule split-block-diagonal)
  show split-block B a a = (B1, B2, B3, B4) using sp .
  show diagonal-mat B using Cons by simp
  show B  $\in$  carrier-mat (dim-row B) (dim-row B) using Cons by simp
  show a  $\leq$  dim-row B using Cons by simp
qed
define n where n = dim-row B
have diag-compat B4 l
proof (rule Cons(1))
  show diagonal-mat B4 using  $\langle$ diagonal-mat B1  $\wedge$  diagonal-mat B4 $\rangle$  by simp
  show B4  $\in$  carrier-mat (dim-row B4) (dim-row B4) using sp Cons
    unfolding split-block-def Let-def by auto
  show dim-row B4 = sum-list l using Cons sp
    unfolding split-block-def Let-def by auto
qed
have B2 = 0m a (n - a)
proof (rule eq-matI, auto)
  show dim-row B2 = a using sp unfolding split-block-def Let-def n-def
    by auto
  show dim-col B2 = n - a using sp Cons
    unfolding split-block-def Let-def n-def by auto
  fix i j
  assume i < a and j < n - a
  thus B2 $$ (i, j) = 0 using sp Cons
    unfolding split-block-def Let-def n-def diagonal-mat-def by force
qed
have B3 = 0m (n - a) a
proof (rule eq-matI, auto)
  show dim-row B3 = n - a using sp Cons
    unfolding split-block-def Let-def n-def by auto
  show dim-col B3 = a using sp Cons
    unfolding split-block-def Let-def n-def by auto
  fix i j
  assume i < n - a and j < a

```

```

    thus  $B3 \text{ } \$(i,j) = 0$  using sp Cons
      unfolding split-block-def Let-def n-def diagonal-mat-def by force
    qed
  show ?case
  proof (rule diag-compat.simps(2)[THEN iffD2], intro conjI)
    show  $a \leq \text{dim-row } B$  using Cons by simp
    show let  $n = \text{dim-row } B$ ;
      ( $B1, B2, B3, B4$ ) = split-block  $B a a$  in  $B2 = 0_m a (n - a) \wedge$ 
       $B3 = 0_m (n - a) a \wedge \text{diag-compat } B4 l$ 
      using sp  $\langle B3 = 0_m (n - a) a \rangle \langle B2 = 0_m a (n - a) \rangle$ 
       $\langle \text{diag-compat } B4 l \rangle$  unfolding Let-def n-def by auto
    qed
  qed

```

The following lemma provides a sufficient condition for the `diag_compat` predicate to hold.

```

lemma commute-diag-compat:
  fixes  $D::'a::\{\text{field}\} \text{ Matrix.mat}$ 
  assumes diagonal-mat  $D$ 
  and  $D \in \text{carrier-mat } n n$ 
  and  $B \in \text{carrier-mat } n n$ 
  and  $B * D = D * B$ 
  and diag-diff  $D l$ 
shows diag-compat  $B l$  using assms
proof (induct l arbitrary: B D n)
  case Nil
    hence  $D \in \text{carrier-mat } 0 0$  using assms by simp
    hence  $n = 0$  using assms using Nil(2) by auto
    hence  $B \in \text{carrier-mat } 0 0$  using Nil by simp
    then show ?case by simp
  next
  case (Cons a l)
    define  $B1$  where  $B1 = \text{fst } (\text{split-block } B a a)$ 
    define  $B2$  where  $B2 = \text{fst } (\text{snd } (\text{split-block } B a a))$ 
    define  $B3$  where  $B3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } B a a)))$ 
    define  $B4$  where  $B4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } B a a)))$ 
    have spb: split-block  $B a a = (B1, B2, B3, B4)$  using fst-conv snd-conv
      unfolding B1-def B2-def B3-def B4-def by simp
    define  $D1$  where  $D1 = \text{fst } (\text{split-block } D a a)$ 
    define  $D2$  where  $D2 = \text{fst } (\text{snd } (\text{split-block } D a a))$ 
    define  $D3$  where  $D3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } D a a)))$ 
    define  $D4$  where  $D4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } D a a)))$ 
    have spd: split-block  $D a a = (D1, D2, D3, D4)$  using fst-conv snd-conv
      unfolding D1-def D2-def D3-def D4-def by simp
    have  $a \leq n$  using Cons by simp
    moreover have diag-compat  $B4 l$ 
    proof (rule Cons(1))
      show diagonal-mat  $D4$  using spd Cons  $\langle a \leq n \rangle$ 
        split-block-diagonal[of D n a] by blast

```

```

show  $D_4 \in \text{carrier-mat } (n-a) (n-a)$  using spd Cons(3)
  unfolding split-block-def Let-def by fastforce
show diag-diff  $D_4$   $l$  using spd Cons by simp
show  $B_4 \in \text{carrier-mat } (n-a) (n-a)$  using spb Cons(4)
  unfolding split-block-def Let-def by fastforce
show  $B_4 * D_4 = D_4 * B_4$  using spb Cons  $\langle a \leq n \rangle$ 
  split-block-commute-subblock[of D] by (meson spd)
qed
moreover have  $B_2 = 0_m$   $a (n-a)$ 
proof (rule commute-diag-mat-split-block(1) [of D n B a B1 B2 B3 B4],
  (auto simp add: spb Cons  $\langle a \leq n \rangle$ ))
  fix  $i j$ 
  assume  $i < a$  and  $a \leq j$  and  $j < n$ 
  thus  $D \ \$\$ (i, i) = D \ \$\$ (j, j) \implies \text{False}$ 
    using diag-diff-hd-diff[OF Cons(6) Cons(3), of i j] by simp
qed
moreover have  $B_3 = 0_m$   $(n-a) a$ 
proof (rule commute-diag-mat-split-block(2) [of D n B a B1 B2 B3 B4],
  (auto simp add: spb Cons  $\langle a \leq n \rangle$ ))
  fix  $i j$ 
  assume  $i < a$  and  $a \leq j$  and  $j < n$ 
  thus  $D \ \$\$ (i, i) = D \ \$\$ (j, j) \implies \text{False}$ 
    using diag-diff-hd-diff[OF Cons(6) Cons(3), of i j] by simp
qed
ultimately show ?case
  using spb diag-compat.simps(2)[THEN iffD2, of a B l] Cons
  unfolding Let-def by force
qed

```

### 6.3 Counting similar neighbours in a list

The function `eq_comps` takes a list as an input and counts the number of adjacent elements that are identical.

```

fun eq-comps ::  $\langle 'a \text{ list} \Rightarrow \text{nat list} \rangle$  where
  eq-comps [] = []
| eq-comps [x] = [1]
| eq-comps (x#y#l) = (let tmp = (eq-comps (y#l)) in
  if  $x = y$  then Suc (hd tmp) # (tl tmp)
  else 1 # tmp)

```

**lemma** *eq-comps-not-empty*:

```

assumes  $l \neq []$ 
shows eq-comps  $l \neq []$  using assms
proof (induct l rule: eq-comps.induct)
  case 1
  then show ?case by simp
next
  case (2 x)
  then show ?case by simp

```

```

next
  case ( $\exists x y l$ )
  then show ?case by (cases  $x = y$ , (auto simp add: Let-def))
qed

lemma eq-comps-empty-if:
  assumes eq-comps  $l = []$ 
  shows  $l = []$ 
proof (rule ccontr)
  assume  $l \neq []$ 
  hence eq-comps  $l \neq []$  using eq-comps-not-empty[of  $l$ ] by simp
  thus False using assms by simp
qed

lemma eq-comps-hd-eq-tl:
  assumes  $x = y$ 
  shows  $tl (eq-comps (x\#y\#l)) = tl (eq-comps (y\#l))$  using assms by (simp add: Let-def)

lemma eq-comps-hd-neq-tl:
  assumes  $x \neq y$ 
  shows  $tl (eq-comps (x\#y\#l)) = eq-comps (y\#l)$  using assms by (simp add: Let-def)

lemma eq-comps-drop:
  assumes  $x\#xs = eq-comps l$ 
  shows  $xs = eq-comps (drop x l)$  using assms
proof (induct  $l$  arbitrary:  $x xs$  rule: eq-comps.induct)
case 1
  then show ?case by simp
next
case ( $2 u$ )
  hence  $x = 1$  by simp
  hence  $drop x [u] = []$  by simp
  then show ?case using 2 by fastforce
next
case ( $3 u v l$ )
  define  $ec$  where  $ec = eq-comps (v\#l)$ 
  have  $ec = hd ec \# (tl ec)$  using eq-comps-not-empty[of  $v\#l$ ] unfolding ec-def

  by simp
show ?case
proof (cases  $u = v$ )
case True
  have  $xs = tl ec$  using 3 eq-comps-hd-eq-tl[OF True] ec-def
  by (metis list.sel(3))
  moreover have  $x = Suc (hd ec)$  using True 3 eq-comps.simps(3)[of  $u v$ ]
  unfolding ec-def Let-def by simp
  hence  $drop (hd ec) (v\#l) = drop x (u\#v\#l)$  by simp
  moreover have  $tl ec = eq-comps (drop (hd ec) (v\#l))$  using 3 ec-def

```

```

    ⟨ec = hd ec # (tl ec)⟩ by simp
  ultimately show ?thesis using β by simp
next
  case False
  hence x = 1 using β unfolding Let-def by simp
  moreover have xs = ec using β eq-comps-hd-neq-tl[OF False] ec-def
    by (metis list.sel(3))
  ultimately show ?thesis unfolding ec-def by simp
qed
qed

lemma eq-comps-neq-0:
  assumes a#m = eq-comps l
  shows a ≠ 0 using assms
proof (induct l rule: eq-comps.induct)
  case 1
  then show ?case by simp
next
  case (2 x)
  then show ?case by simp
next
  case (3 x y l)
  then show ?case by (cases x = y, (auto simp add: Let-def))
qed

lemma eq-comps-gt-0:
  assumes l ≠ []
  shows list-all (λa. 0 < a) (eq-comps l)
  apply (induction l rule: eq-comps.induct)
  apply (auto simp add: list-all-iff Let-def)
  apply (metis eq-comps-not-empty list.distinct(1) list.set-sel(2))
  done

lemma eq-comps-elem-le-length:
  assumes a#m = eq-comps l
  shows a ≤ length l using assms
proof (induct l arbitrary: a rule: eq-comps.induct)
  case 1
  then show ?case by simp
next
  case (2 x)
  then show ?case by auto
next
  case (3 x y l)
  then show ?case
  proof (cases x = y)
    case True
    define ec where ec = eq-comps (y#l)
    have ec = hd ec # (tl ec) using eq-comps-not-empty[of y#l] unfolding ec-def

```

```

    by simp
  have a = Suc (hd ec) using True  $\exists$  eq-comps.simps( $\exists$ )[of x y]
    unfolding ec-def Let-def by simp
  then show ?thesis using  $\exists$ 
    by (metis True  $\langle$ ec = hd ec # tl ec $\rangle$  ec-def eq-comps-hd-eq-tl length-Cons
        list.sel( $\exists$ ) not-less-eq-eq)
next
  case False
  hence a = 1 using  $\exists$  by (simp add: Let-def)
  then show ?thesis by simp
qed
qed

lemma eq-comps-length:
  shows length (eq-comps l)  $\leq$  length l
proof (induct l rule: eq-comps.induct)
  case 1
  then show ?case by simp
next
  case (2 x)
  then show ?case by auto
next
  case (3 x y l)
  define ec where ec = eq-comps (y#l)
  have ec: ec = hd ec # (tl ec) using eq-comps-not-empty[of y#l] unfolding
ec-def
    by simp
  then show ?case
proof (cases x = y)
  case True
  then show ?thesis using ec  $\exists$  eq-comps.simps( $\exists$ ) True unfolding Let-def
    by (metis ec-def le-SucI length-Cons)
  next
  case False
  then show ?thesis using ec  $\exists$  by simp
qed
qed

lemma eq-comps-eq:
  assumes a#m = eq-comps l
  and i < a
shows nth l i = hd l using assms
proof (induct l arbitrary: a m i rule: eq-comps.induct)
  case 1
  then show ?case by simp
next
  case (2 u)
  then show ?case by simp

```

```

next
  case ( $\exists u v l$ )
  show ?case
  proof (cases  $u = v$ )
    case False
    thus ?thesis using  $\exists$  by (simp add: Let-def)
  next
  case True
  define ec where  $ec = eq-comps (v\#l)$ 
  have  $ec = hd ec \# (tl ec)$  using eq-comps-not-empty[of  $v\#l$ ]
    unfolding ec-def by simp
  have  $a = Suc (hd ec)$  using True  $\exists eq-comps.simps(\exists)$ [of  $u v$ ]
    unfolding ec-def Let-def by simp
  hence  $i < hd ec$  using  $\exists$  by simp
  show ?thesis
  proof (cases  $i = 0$ )
    case True
    thus ?thesis by simp
  next
  case False
  hence  $\exists i'. i = Suc i'$  by (simp add: not0-implies-Suc)
  from this obtain  $i'$  where  $i = Suc i'$  by auto
  hence  $i' < hd ec$  using  $\langle i \leq hd ec \rangle$  by simp
  have  $(u \# v \# l) ! i = (v\#l) ! i'$  using  $\langle i = Suc i' \rangle$  by simp
  also have  $\dots = v$  using  $\exists \langle ec = eq-comps (v\#l) \rangle \langle ec = hd ec \# (tl ec) \rangle$ 
    by (metis  $\langle i' < hd ec \rangle list.sel(1)$ )
  also have  $\dots = hd (u\#v\#l)$  using  $\langle u = v \rangle$  by simp
  finally show ?thesis .
qed
qed
qed

lemma eq-comps-singleton:
  assumes  $[a] = eq-comps l$ 
  shows  $a = length l$  using assms
proof (induct  $l$  arbitrary: a rule: eq-comps.induct)
case 1
then show ?case by simp
next
case ( $2 x$ )
then show ?case by simp
next
case ( $\exists x y l$ )
define ec where  $ec = eq-comps (y\#l)$ 
have  $ec = hd ec \# (tl ec)$  using eq-comps-not-empty[of  $y\#l$ ]
  unfolding ec-def by simp
show ?case
proof (cases  $x = y$ )
case True

```

```

hence a = Suc (hd ec) using 3 eq-comps.simps(3)[of x y]
  unfolding ec-def Let-def by simp
have tl ec = [] using 3 True eq-comps.simps(3)[of x y]
  unfolding ec-def Let-def by simp
hence ec = [hd ec] using ⟨ec = hd ec # tl ec⟩ by simp
hence hd ec = length (y#l) using 3 ec-def by simp
then show ?thesis using ⟨a = Suc (hd ec)⟩ by simp
next
case False
then show ?thesis using eq-comps-hd-neq-tl 3
  ⟨ec = hd ec # tl ec⟩ ec-def by fastforce
qed
qed

lemma eq-comps-leq:
  assumes a#b#m = eq-comps l
  and sorted l
shows hd l < hd (drop a l) using assms
proof (induct l arbitrary: a b m rule: eq-comps.induct)
case 1
  then show ?case by simp
next
case (2 x)
  then show ?case by simp
next
case (3 x y l)
show ?case
proof (cases x = y)
case True
  hence hd (x#y#l) = y by simp
  define ec where ec = eq-comps (y#l)
  have a = Suc (hd (ec)) using True ec-def 3
    eq-comps.simps(3)[of x y] unfolding Let-def by simp
  have b#m = tl ec using True ec-def 3
    eq-comps.simps(3)[of x y] unfolding Let-def by simp
  hence eceq: ec = hd ec # (hd (tl ec)) # (tl (tl ec)) unfolding ec-def
    by (metis eq-comps-not-empty list.exhaust-sel list.simps(3))
  have dra: drop a (x#y#l) = drop (hd ec) (y#l) using ⟨a = Suc (hd (ec))⟩
    by simp
  have sorted (y#l) using 3 by simp
  hence y < hd (drop (hd ec) (y#l)) using 3(1) eceq unfolding ec-def
    by (metis list.sel(1))
  thus ?thesis using True dra by simp
next
case False
  hence a = 1 using 3 by (simp add: Let-def)
  have hd (x#y#l) = x by simp
  moreover have hd (drop a (x # y # l)) = y using ⟨a = 1⟩ by simp
  ultimately show ?thesis using False 3

```

by (*metis order-le-imp-less-or-eq sorted2-simps(2)*)  
qed  
qed

**lemma** *eq-comps-compare*:

assumes *sorted l*  
and  $a \# m = \text{eq-comps } l$   
and  $i < a$   
and  $a \leq j$   
and  $j < \text{length } l$   
shows  $\text{nth } l \ i < \text{nth } l \ j$  using *assms*  
**proof** (*cases m = []*)  
case *True*  
hence  $[a] = \text{eq-comps } l$  using *assms* by *simp*  
hence  $a = \text{length } l$  using *eq-comps-singleton[of a l]* by *simp*  
then show *?thesis* using *assms* by *simp*

next

case *False*  
hence  $m = \text{hd } m \# (\text{tl } m)$  by *simp*  
have  $!!i = \text{hd } l$  using *assms eq-comps-eq* by *metis*  
also have  $\dots < \text{hd } (\text{drop } a \ l)$  using *eq-comps-leq assms <m = hd m # (tl m)>*  
by *metis*  
also have  $\dots \leq !!j$  using *assms*  
by (*metis hd-drop-conv-nth le-less-trans sorted-nth-mono*)  
finally show *?thesis* .

qed

**lemma** *eq-comps-singleton-elems*:

assumes  $\text{eq-comps } l = [a]$   
shows  $\forall i < \text{length } l. !!i = !!0$  using *eq-comps-eq eq-comps-singleton*  
by (*metis assms bot-nat-0.not-eq-extremum eq-comps-neq-0*)

**lemma** *eq-comp-Re*:

assumes  $\forall z \in \text{set } l. z \in \text{Reals}$   
and  $m = \text{eq-comps } l$   
shows  $m = \text{eq-comps } (\text{map } \text{Re } l)$  using *assms*  
**proof** (*induct l arbitrary:m rule:eq-comps.induct*)

case *1*  
then show *?case* by *simp*

next

case (*2 x*)  
then show *?case* by *simp*

next

case (*3 x y l*)  
**define** *ec* where  $ec = \text{eq-comps } (y \# l)$   
have *ecr*:  $ec = \text{eq-comps } (\text{map } \text{Re } (y \# l))$  using *ec-def 3* by *simp*  
show *?case*  
**proof** (*cases x = y*)  
case *True*

```

hence  $Re\ x = Re\ y$  by simp
have  $m = Suc\ (hd\ ec)\ \#\ (tl\ ec)$  using ec-def 3 True
  by (simp add: Let-def)
also have  $\dots = eq\ comps\ (map\ Re\ (x\ \#\ y\ \#\ l))$  using ecr  $\langle Re\ x = Re\ y \rangle$ 
  by (simp add: Let-def)
finally show ?thesis .
next
case False
hence  $Re\ x \neq Re\ y$  using 3
  by (metis list.set-intros(1) list.set-intros(2) of-real-Re)
have  $m = 1\ \#\ ec$  using ec-def 3 False
  by (simp add: Let-def)
also have  $\dots = eq\ comps\ (map\ Re\ (x\ \#\ y\ \#\ l))$  using ecr  $\langle Re\ x \neq Re\ y \rangle$ 
  by (simp add: Let-def)
finally show ?thesis using ecr unfolding Let-def by simp
qed
qed

```

```

lemma eq-comps-sum-list:
  shows  $sum\ list\ (eq\ comps\ l) = length\ l$ 
proof (induct l rule: eq-comps.induct)
  case 1
  then show ?case unfolding diag-mat-def by simp
next
  case (2 x)
  have  $eq\ comps\ [x] = [1]$  using eq-comps.simps(2)[of x] by simp
  then show ?case by simp
next
  case (3 x y l)
  then show ?case
  proof (cases  $x = y$ )
    case True
    then show ?thesis using eq-comps.simps(3)[of x y l] 3
      by (cases  $\langle eq\ comps\ (y\ \#\ l) \rangle$ ) simp-all
  next
    case False
    then show ?thesis using eq-comps.simps(3)[of x y l] 3
      unfolding Let-def by simp
  qed
qed
qed

```

```

lemma eq-comps-elem-lt:
  assumes  $1 < length\ (eq\ comps\ l)$ 
  shows  $hd\ (eq\ comps\ l) < length\ l$ 
proof -
  define a where  $a = hd\ (eq\ comps\ l)$ 
  define b where  $b = hd\ (tl\ (eq\ comps\ l))$ 
  define c where  $c = tl\ (tl\ (eq\ comps\ l))$ 
  have  $eq\ comps\ l = a\ \#\ b\ \#\ c$  using assms unfolding a-def b-def c-def

```

by (metis eq-comps.simps(2) eq-comps-singleton length-0-conv  
 less-irrefl-nat less-nat-zero-code list.exhaust-sel)  
 hence  $b\#c = \text{eq-comps } (\text{drop } a \ l)$  using eq-comps-drop by metis  
 hence  $0 < b$  using eq-comps-neq-0 by auto  
 moreover have  $0 < a$  using  $\langle \text{eq-comps } l = a\#b\#c \rangle$  eq-comps-neq-0  
 by (metis gr0I)  
 moreover have  $a+b \leq \text{length } l$  using eq-comps-sum-list  
 by (metis  $\langle \text{eq-comps } l = a \ \# \ b \ \# \ c \rangle$  le-add1 nat-add-left-cancel-le  
 sum-list-simps(2))  
 ultimately show ?thesis unfolding a-def by auto  
 qed

lemma eq-comp-sum-diag-mat:  
 shows  $\text{sum-list } (\text{eq-comps } (\text{diag-mat } A)) = \text{dim-row } A$   
 using eq-comps-sum-list[of diag-mat A] diag-mat-length by simp

lemma nsum-Suc-elem:  
 assumes  $1 < \text{length } (\text{eq-comps } l)$   
 shows  $l!(n\text{-sum } (\text{Suc } i) (\text{eq-comps } l)) =$   
 $(\text{drop } (\text{hd } (\text{eq-comps } l)) \ l)!(n\text{-sum } i (\text{tl } (\text{eq-comps } l)))$  using assms  
 proof (induct i arbitrary: l)  
 case 0  
 hence  $1 < \text{length } l$  using eq-comps-length[of l] by presburger  
 hence  $l \neq []$  by fastforce  
 hence  $l! \ n\text{-sum } (\text{Suc } 0) (\text{eq-comps } l) = l! \ \text{hd } (\text{eq-comps } l)$   
 by (simp add: 0.premis eq-comps-not-empty hd-conv-nth)  
 also have  $\dots = \text{hd } (\text{drop } (\text{hd } (\text{eq-comps } l)) \ l)$   
 by (metis 0.premis eq-comps-elem-lt hd-drop-conv-nth)  
 finally show ?case using 0  
 by (metis (no-types, opaque-lifting)  $\langle l! \ \text{hd } (\text{eq-comps } l) =$   
 $\text{hd } (\text{drop } (\text{hd } (\text{eq-comps } l)) \ l) \rangle$   $\langle l \neq [] \rangle$  append-Nil2  
 eq-comps.simps(1) eq-comps-drop eq-comps-empty-if eq-comps-singleton  
 hd-conv-nth list.exhaust-sel n-sum.simps(1) nat-arith.rule0  
 nth-append-length-plus)

next  
 case (Suc i)  
 have  $l!(n\text{-sum } (\text{Suc } (\text{Suc } i)) (\text{eq-comps } l)) =$   
 $l!(\text{hd } (\text{eq-comps } l) + (n\text{-sum } (\text{Suc } i) (\text{tl } (\text{eq-comps } l))))$  by simp  
 also have  $\dots = (\text{drop } (\text{hd } (\text{eq-comps } l)) \ l)!$   
 $(n\text{-sum } (\text{Suc } i) (\text{tl } (\text{eq-comps } l)))$   
 using less-or-eq-imp-le  
 by (metis Suc.premis eq-comps-elem-lt nth-drop)  
 finally show ?case .  
 qed

lemma eq-comps-elems-eq:  
 assumes  $l \neq []$   
 and  $i < \text{length } (\text{eq-comps } l)$   
 and  $j < (\text{eq-comps } l)!i$

**shows**  $l!(n\text{-sum } i \text{ (eq-comps } l)) = l!(n\text{-sum } i \text{ (eq-comps } l) + j)$  **using** *assms*  
**proof** (*induct i arbitrary: l*)  
  **case** 0  
  **hence**  $\text{eq-comps } l = \text{hd (eq-comps } l) \# (\text{tl (eq-comps } l))$  **by** *simp*  
  **have**  $l! \text{ n-sum } 0 \text{ (eq-comps } l) = \text{hd } l$   
  **by** (*simp add: 0(1) hd-conv-nth*)  
  **also have**  $\dots = l!j$  **using** 0 *eq-comps-eq*  
  **by** (*metis <eq-comps l = hd (eq-comps l) # tl (eq-comps l)> nth-Cons-0*)  
  **finally show** ?*case* **by** *simp*  
**next**  
  **case** (*Suc i*)  
  **show** ?*case*  
  **proof** (*cases length (eq-comps l) = 1*)  
  **case** *True*  
  **hence**  $\text{Suc } i = 0$  **using** *Suc.premis(2)* **by** *fastforce*  
  **then show** ?*thesis* **by** *simp*  
  **next**  
  **case** *False*  
  **hence**  $1 < \text{length (eq-comps } l)$  **using** *Suc eq-comps-not-empty[of l]*  
  **by** *presburger*  
  **hence**  $l!(n\text{-sum (Suc } i) \text{ (eq-comps } l)) =$   
   $(\text{drop (hd (eq-comps } l)) } l)!(n\text{-sum } i \text{ (tl (eq-comps } l)))$   
  **using** *nsum-Suc-elem* **by** *simp*  
  **also have**  $\dots = (\text{drop (hd (eq-comps } l)) } l)!$   
   $(n\text{-sum } i \text{ (eq-comps (drop (hd (eq-comps } l)) } l)))$   
  **using** *eq-comps-drop[of hd (eq-comps l)] eq-comps-empty-if list.collapse*  
  **by** *fastforce*  
  **also have**  $\dots = (\text{drop (hd (eq-comps } l)) } l)!$   
   $(n\text{-sum } i \text{ (eq-comps (drop (hd (eq-comps } l)) } l)) + j)$   
  **proof** (*rule Suc(1)*)  
  **show**  $\text{drop (hd (eq-comps } l)) } l \neq []$   
  **by** (*metis Cons-nth-drop-Suc <1 < length (eq-comps l)> eq-comps-elem-lt*  
  *list.distinct(1)*)  
  **show**  $i < \text{length (eq-comps (drop (hd (eq-comps } l)) } l))$  **using** *Suc*  
  **by** (*metis (no-types, lifting) Suc-lessD eq-comps-drop*  
  *eq-comps-not-empty length-Suc-conv list.collapse*  
  *not-less-less-Suc-eq*)  
  **show**  $j < \text{eq-comps (drop (hd (eq-comps } l)) } l) ! i$  **using** *Suc*  
  **by** (*metis eq-comps-drop length-Suc-conv less-natE list.exhaust-sel*  
  *list.simps(3) nth-Cons-Suc*)  
  **qed**  
  **also have**  $\dots = (\text{drop (hd (eq-comps } l)) } l)!$   
   $(n\text{-sum } i \text{ (tl (eq-comps } l)) + j)$   
  **by** (*metis Suc(2) eq-comps-drop eq-comps-not-empty hd-Cons-tl*)  
  **also have**  $\dots = l!(n\text{-sum (Suc } i) \text{ (eq-comps } l) + j)$   
  **by** (*metis (no-types, opaque-lifting) Groups.add-ac(2)*  
  *Groups.add-ac(3) <1 < length (eq-comps l)> eq-comps-elem-lt*  
  *less-or-eq-imp-le n-sum.simps(2) nth-drop*)  
  **finally show** ?*thesis* .

qed  
qed

When the diagonal block matrices are extracted using `eq_comp`, each extracted matrix is a multiple of the identity.

**lemma** *extract-subdiags-eq-comp*:

**fixes** *A::complex Matrix.mat*

**assumes** *diagonal-mat A*

**and** *A ∈ carrier-mat n n*

**and**  $0 < n$

**and**  $i < \text{length } (\text{eq-comps } (\text{diag-mat } A))$

**shows**  $\exists k. (\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)))!i =$   
 $k \cdot_m (1_m ((\text{eq-comps } (\text{diag-mat } A))!i))$

**proof**

**define** *l* **where**  $l = \text{diag-mat } A$

**define** *k* **where**  $k = l!(n\text{-sum } i (\text{eq-comps } l))$

**show**  $\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i =$

$k \cdot_m 1_m (\text{eq-comps } (\text{diag-mat } A) ! i)$

**proof** (rule *eq-matI*, auto simp add: *assms*)

**show** *dr*:  $\text{dim-row } (\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i) =$   
 $\text{eq-comps } (\text{diag-mat } A) ! i$

**using** *extract-subdiags-carrier assms carrier-matD(1)* **by** *blast*

**show** *dc*:  $\text{dim-col } (\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i) =$   
 $\text{eq-comps } (\text{diag-mat } A) ! i$

**using** *extract-subdiags-carrier assms carrier-matD* **by** *blast*

**fix** *m np*

**assume**  $m < \text{eq-comps } (\text{diag-mat } A)!i$  **and**  $np < \text{eq-comps } (\text{diag-mat } A)!i$

**and**  $m \neq np$  **note** *mnp=this*

**have** *diagonal-mat*  $(\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i)$

**proof** (rule *extract-subdiags-diagonal*)

**show** *diagonal-mat A* **using** *assms* **by** *simp*

**show**  $A \in \text{carrier-mat } n \ n$  **using** *assms* **by** *simp*

**show**  $\text{eq-comps } (\text{diag-mat } A) \neq []$  **using** *assms* **unfolding** *diag-mat-def*  
**by** *auto*

**show**  $\text{sum-list } (\text{eq-comps } (\text{diag-mat } A)) \leq n$

**using** *assms eq-comps-sum-list* **unfolding** *diag-mat-def*

**by** (*metis carrier-matD(1) carrier-matD(2) length-cols-mat-to-cols-list*  
*length-map order.eq-iff*)

**show**  $i < \text{length } (\text{eq-comps } (\text{diag-mat } A))$  **using** *assms* **by** *simp*

qed

**thus**  $\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i \ \S\S (m, np) = 0$

**using** *mnp dr dc* **by** (*metis diagonal-mat-def*)

**next**

**fix** *p*

**assume**  $p < \text{eq-comps } (\text{diag-mat } A) ! i$

**have**  $\text{extract-subdiags } A (\text{eq-comps } (\text{diag-mat } A)) ! i \ \S\S (p, p) =$

$\text{diag-mat } A! (n\text{-sum } i (\text{eq-comps } (\text{diag-mat } A)) + p)$

**proof** (rule *extract-subdiags-diag-elem*)

**show**  $A \in \text{carrier-mat } n \ n \ 0 < n \ i < \text{length } (\text{eq-comps } (\text{diag-mat } A))$

```

    using assms by auto
  show ne: eq-comps (diag-mat A) ≠ [] using assms by auto
  show p < eq-comps (diag-mat A) ! i
    using ⟨p < eq-comps (diag-mat A) ! i⟩ .
  show sum-list (eq-comps (diag-mat A)) ≤ n
    using assms eq-comps-sum-list[of diag-mat A]
    unfolding diag-mat-def by simp
  show ∀ j < length (eq-comps (diag-mat A)). 0 < eq-comps (diag-mat A) ! j
    using eq-comps-gt-0 ne
    by (metis eq-comps.simps(1) list-all-length)
qed
also have ... = k unfolding k-def l-def
proof (rule eq-comps-elems-eq[symmetric])
  show diag-mat A ≠ [] using assms unfolding diag-mat-def by simp
  show p < eq-comps (diag-mat A) ! i
    using ⟨p < eq-comps (diag-mat A) ! i⟩ .
  show i < length (eq-comps (diag-mat A)) using assms by simp
qed
finally show
  extract-subdiags A (eq-comps (diag-mat A)) ! i $$$ (p, p) = k .
qed
qed

```

**lemma** *extract-subdiags-comp-commute*:

```

  fixes A::complex Matrix.mat
  assumes diagonal-mat A
  and A ∈ carrier-mat n n
  and 0 < n
  and i < length (eq-comps (diag-mat A))
  and B ∈ carrier-mat ((eq-comps (diag-mat A))! i) ((eq-comps (diag-mat A))! i)
  shows (extract-subdiags A (eq-comps (diag-mat A))! i) * B =
    B * (extract-subdiags A (eq-comps (diag-mat A))! i)
proof -
  define m where m = (eq-comps (diag-mat A))! i
  have ∃ k. (extract-subdiags A (eq-comps (diag-mat A))! i) =
    k ·m (1m ((eq-comps (diag-mat A))! i))
    using assms extract-subdiags-eq-comp by simp
  from this obtain k where
    (extract-subdiags A (eq-comps (diag-mat A))! i) =
    k ·m (1m m) unfolding m-def by auto note kprop = this
  hence (extract-subdiags A (eq-comps (diag-mat A))! i) * B =
    k ·m (1m m) * B by simp
  also have ... = B * (k ·m (1m m)) using assms m-def
    by (metis left-mult-one-mat mult-smult-assoc-mat
      mult-smult-distrib one-carrier-mat right-mult-one-mat)
  finally show ?thesis using kprop by simp
qed

```

In particular, extracting the diagonal sub-blocks of a diagonal matrix leaves

it unchanged.

**lemma** *diagonal-extract-eq*:  
**assumes**  $B \in \text{carrier-mat } n \ n$   
**and** *diagonal-mat*  $B$   
**shows**  $B = \text{diag-block-mat } (\text{extract-subdiags } B \ (\text{eq-comps } (\text{diag-mat } B)))$   
**proof** (*rule diag-compat-extract-subdiag*)  
**define** *eqcl* **where**  $\text{eqcl} = \text{eq-comps } (\text{diag-mat } B)$   
**show**  $B \in \text{carrier-mat } n \ n$  **using** *assms* **by** *simp*  
**show** *diag-compat*  $B \ \text{eqcl}$   
**proof** (*rule diag-compat-diagonal*)  
**show**  $B \in \text{carrier-mat } (\text{dim-row } B) \ (\text{dim-row } B)$   
**using** *assms* **by** *simp*  
**show** *diagonal-mat*  $B$  **using** *assms* **by** *simp*  
**have**  $\text{dim-row } B = \text{length } (\text{diag-mat } B)$  **unfolding** *diag-mat-def* **by** *simp*  
**also have**  $\dots = \text{sum-list } \text{eqcl}$  **using** *eq-comps-sum-list* [*of* *diag-mat*  $B$ ]  
**unfolding** *eqcl-def* **by** *simp*  
**finally show**  $\text{dim-row } B = \text{sum-list } \text{eqcl}$ .  
**qed**  
**qed**

**fun** *lst-diff* **where**  
 $\text{lst-diff } l \ [] = (l = [])$   
 $|\ \text{lst-diff } l \ (x\#\text{xs}) = (x \leq \text{length } l \wedge$   
 $(\forall i \ j. \ i < x \wedge x \leq j \wedge j < \text{length } l \longrightarrow \text{nth } l \ i < \text{nth } l \ j) \wedge$   
 $\text{lst-diff } (\text{drop } x \ l) \ \text{xs})$

**lemma** *sorted-lst-diff*:  
**assumes** *sorted*  $l$   
**and**  $m = \text{eq-comps } l$   
**shows** *lst-diff*  $l \ m$  **using** *assms*  
**proof** (*induct m arbitrary: l*)  
**case** *Nil*  
**hence**  $l = []$  **using** *eq-comps-empty-if* [*of*  $l$ ] **by** *simp*  
**then show** *?case* **by** *simp*  
**next**  
**case** (*Cons*  $a \ m$ )  
**have** *sorted* ( $\text{drop } a \ l$ ) **using** *Cons sorted-wrt-drop* **by** *simp*  
**moreover have**  $m = \text{eq-comps } (\text{drop } a \ l)$  **using** *eq-comps-drop* *Cons* **by** *simp*  
**ultimately have** *lst-diff* ( $\text{drop } a \ l$ )  $m$  **using** *Cons* **by** *simp*  
**have**  $a \leq \text{length } l$  **using** *eq-comps-elem-le-length* *Cons* **by** *simp*  
**have**  $(\forall i \ j. \ i < a \wedge a \leq j \wedge j < \text{length } l \longrightarrow \text{nth } l \ i < \text{nth } l \ j)$   
**using** *Cons eq-comps-compare* **by** *blast*  
**then show** *?case* **using**  $\langle a \leq \text{length } l \rangle \langle \text{lst-diff } (\text{drop } a \ l) \ m \rangle$  **by** *fastforce*  
**qed**

**lemma** *lst-diff-imp-diag-diff*:  
**fixes**  $D :: 'a :: \text{preorder Matrix.mat}$   
**assumes**  $D \in \text{carrier-mat } n \ n$   
**and** *lst-diff* (*diag-mat*  $D$ )  $m$

**shows** *diag-diff*  $D$   $m$  **using** *assms*  
**proof** (*induct arbitrary: n rule:diag-diff.induct*)  
**case** (1  $D$ )  
**hence** *diag-mat*  $D = []$  **by** *simp*  
**hence** *dim-row*  $D = 0$  **unfolding** *diag-mat-def* **by** *simp*  
**hence**  $n = 0$  **using** 1 **by** *simp*  
**hence** *dim-col*  $D = 0$  **using** 1 **by** *simp*  
**then show** ?*case* **using**  $\langle \text{dim-row } D = 0 \rangle$  **by** *simp*  
**next**  
**case** (2  $D$   $a$   $xs$ )  
**define**  $D1$  **where**  $D1 = \text{fst } (\text{split-block } D \ a \ a)$   
**define**  $D2$  **where**  $D2 = \text{fst } (\text{snd } (\text{split-block } D \ a \ a))$   
**define**  $D3$  **where**  $D3 = \text{fst } (\text{snd } (\text{snd } (\text{split-block } D \ a \ a)))$   
**define**  $D4$  **where**  $D4 = \text{snd } (\text{snd } (\text{snd } (\text{split-block } D \ a \ a)))$   
**have** *spd*: *split-block*  $D \ a \ a = (D1, D2, D3, D4)$  **using** *fst-conv* *snd-conv*  
**unfolding** *D1-def* *D2-def* *D3-def* *D4-def* **by** *simp*  
**have** *length* (*diag-mat*  $D$ ) =  $n$  **using** 2 **unfolding** *diag-mat-def* **by** *simp*  
**hence**  $a \leq n$  **using** 2 **by** *simp*  
**hence**  $c1$ :  $D1 \in \text{carrier-mat } a \ a$   
**using** *split-block*(1)[*OF spd, of n-a n-a*] 2 **by** *simp*  
**have**  $c4$ :  $D4 \in \text{carrier-mat } (n-a) \ (n-a)$   
**using** 2 *split-block*(4)[*OF spd*]  $\langle a \leq n \rangle$  **by** *simp*  
**have** *diag-mat*  $D = \text{diag-mat } D1 \ @ \ (\text{diag-mat } D4)$   
**using** *diag-four-block-mat* *split-block*(5)  
**by** (*metis* 2(2)  $\langle a \leq n \rangle$   $c1 \ c4$  *carrier-matD*(1) *carrier-matD*(2))  
*le-Suc-ex spd*)  
**have** *length* (*diag-mat*  $D1$ ) =  $a$  **using**  $c1$  **unfolding** *diag-mat-def* **by** *simp*  
**hence** *diag-mat*  $D4 = \text{drop } a \ (\text{diag-mat } D)$   
**using**  $\langle \text{diag-mat } D = \text{diag-mat } D1 \ @ \ (\text{diag-mat } D4) \rangle$  **by** *simp*  
**hence** *lst-diff* (*diag-mat*  $D4$ )  $xs$  **using** 2 **by** *simp*  
**hence** *diag-diff*  $D4$   $xs$  **using** 2(1)[*OF spd[symmetric]*] *spd*  $c4$   
**by** *blast*  
**have** ( $\forall i \ j. i < \text{dim-row } D1 \ \wedge \ j < \text{dim-row } D4 \ \longrightarrow \ D1 \ \$\$ (i,i) < D4 \ \$\$ (j,j)$ )  
**proof** (*intro allI impI*)  
**fix**  $i \ j$   
**assume**  $ijp$ :  $i < \text{dim-row } D1 \ \wedge \ j < \text{dim-row } D4$   
**hence**  $i < a$  **using**  $c1$  **by** *simp*  
**have**  $j+a < n$  **using**  $c4 \ ijp$  **by** (*metis* *carrier-matD*(1) *less-diff-conv*)  
**have**  $D1 \ \$\$ (i,i) = D \ \$\$ (i,i)$  **using** *spd*  $\langle i < a \rangle$   
**unfolding** *split-block-def* *Let-def* **by** *force*  
**also have**  $\dots = (\text{diag-mat } D)!i$  **using**  $\langle i < a \rangle \ \langle a \leq n \rangle$  2  
**unfolding** *diag-mat-def* **by** *simp*  
**also have**  $\dots < (\text{diag-mat } D)!(j+a)$  **using** 2  $\langle j+a < n \rangle$   
**by** (*metis*  $\langle i < a \rangle \ \langle \text{length } (\text{diag-mat } D) = n \rangle$ )  
*le-add2 lst-diff.simps*(2))  
**also have**  $\dots = D \ \$\$ (j+a, j+a)$  **using**  $\langle j+a < n \rangle$  2  
**unfolding** *diag-mat-def* **by** *simp*  
**also have**  $\dots = D4 \ \$\$ (j,j)$  **using** *spd*  $ijp$  2  
**unfolding** *split-block-def* *Let-def* **by** *force*

finally show  $D1\$(i,i) < D4\$(j,j)$  .  
**qed**  
 hence  $(\forall i j. i < \text{dim-row } D1 \wedge j < \text{dim-row } D4 \longrightarrow D1\$(i,i) \neq D4\$(j,j))$   
 by (metis order-less-irrefl)  
 thus ?case using ‹diag-diff D4 xs› ‹a ≤ n› 2 spd by simp  
**qed**

**lemma sorted-diag-diff:**  
 fixes  $D::'a::\text{linorder Matrix.mat}$   
 assumes  $D \in \text{carrier-mat } n\ n$   
 and sorted (diag-mat D)  
 shows diag-diff D (eq-comps (diag-mat D))  
**proof** (rule lst-diff-imp-diag-diff)  
 show  $D \in \text{carrier-mat } n\ n$  using assms by simp  
 show lst-diff (diag-mat D) (eq-comps (diag-mat D))  
 using sorted-lst-diff[of diag-mat D] assms by simp  
**qed**

**lemma Re-sorted-lst-diff:**  
 fixes  $l::\text{complex list}$   
 assumes  $\forall z \in \text{set } l. z \in \text{Reals}$   
 and sorted (map Re l)  
 and  $m = \text{eq-comps } l$   
 shows lst-diff l m using assms  
**proof** (induct m arbitrary: l)  
 case Nil  
 hence  $l = []$  using eq-comps-empty-iff[of l] by simp  
 then show ?case by simp  
**next**  
 case (Cons a m)  
 have sorted (map Re (drop a l)) using Cons sorted-wrt-drop  
 by (metis drop-map)  
 moreover have  $m = \text{eq-comps } (\text{drop } a\ l)$  using eq-comps-drop Cons by simp  
 ultimately have lst-diff (drop a l) m using Cons  
 by (metis in-set-dropD)  
 have  $a \leq \text{length } l$  using eq-comps-elem-le-length Cons by simp  
 have  $(\forall i j. i < a \wedge a \leq j \wedge j < \text{length } l \longrightarrow \text{nth } l\ i < \text{nth } l\ j)$   
**proof** (intro allI impI)  
 fix  $i j$   
 assume asm:  $i < a \wedge a \leq j \wedge j < \text{length } l$   
 hence  $\text{Re } (!i) < \text{Re } (!j)$   
 using Cons eq-comps-compare eq-comp-Re  
 by (smt (verit) dual-order.strict-trans dual-order.strict-trans1 length-map  
 nth-map)  
 moreover have  $!i \in \text{Reals}$  using asm Cons by simp  
 moreover have  $!j \in \text{Reals}$  using asm Cons by simp  
 ultimately show  $\text{nth } l\ i < \text{nth } l\ j$  using less-complex-def  
 by (simp add: complex-is-Real-iff)  
**qed**

**then show**  $?case$  **using**  $\langle a \leq length\ l \rangle \langle lst\_diff\ (drop\ a\ l)\ m \rangle$  **by** *fastforce*  
**qed**

The following lemma states a sufficient condition for the `diag_diff` predicate to hold.

**lemma** *cpx-sorted-diag-diff*:  
**fixes**  $D::complex\ Matrix.mat$   
**assumes**  $D \in carrier\_mat\ n\ n$   
**and**  $\forall\ i < n. D\ \$\$(i,i) \in Reals$   
**and** *sorted*  $(map\ Re\ (diag\_mat\ D))$   
**shows** *diag-diff*  $D\ (eq\_comps\ (diag\_mat\ D))$   
**proof** *(rule lst-diff-imp-diag-diff)*  
**show**  $D \in carrier\_mat\ n\ n$  **using** *assms* **by** *simp*  
**have**  $\forall\ z \in set\ (diag\_mat\ D). z \in \mathbb{R}$  **using** *assms* **unfolding** *diag-mat-def* **by** *auto*  
**thus** *lst-diff*  $(diag\_mat\ D)\ (eq\_comps\ (diag\_mat\ D))$   
**using** *Re-sorted-lst-diff*  $[of\ diag\_mat\ D]$  *assms* **by** *simp*  
**qed**

## 7 Sorted hermitian decomposition

We prove that any Hermitian matrix  $A$  can be decomposed into a product  $U^\dagger \cdot B \cdot U$ , where  $U$  is a unitary matrix and  $B$  is a diagonal matrix containing only real components which are ordered along the diagonal.

**definition** *per-col* **where**  
*per-col*  $A\ f = Matrix.mat\ (dim\_row\ A)\ (dim\_col\ A)\ (\lambda\ (i,j). A\ \$\$(i,(f\ j)))$

**lemma** *per-col-carrier*:  
**assumes**  $A \in carrier\_mat\ n\ m$   
**shows** *per-col*  $A\ f \in carrier\_mat\ n\ m$  **using** *assms* **unfolding** *per-col-def*  
**by** *simp*

**lemma** *per-col-col*:  
**assumes**  $A \in carrier\_mat\ n\ m$   
**and**  $j < m$   
**shows**  $Matrix.col\ (per\_col\ A\ f)\ j = Matrix.col\ A\ (f\ j)$   
**proof**  
**show**  $dim: dim\_vec\ (Matrix.col\ (per\_col\ A\ f)\ j) =$   
 $dim\_vec\ (Matrix.col\ A\ (f\ j))$   
**using** *per-col-carrier* **by**  $(metis\ assms(1)\ carrier\_matD(1)\ dim\_col)$   
**fix**  $i$   
**assume**  $i < dim\_vec\ (Matrix.col\ A\ (f\ j))$   
**hence**  $i < dim\_vec\ (Matrix.col\ (per\_col\ A\ f)\ j)$  **using** *dim* **by** *simp*  
**hence**  $vec\_index\ (Matrix.col\ (per\_col\ A\ f)\ j)\ i = (per\_col\ A\ f)\ \$\$(i,j)$   
**unfolding** *Matrix.col-def* **by** *simp*  
**also have**  $\dots = A\ \$\$(i,(f\ j))$  **unfolding** *per-col-def*  
**using**  $\langle i < dim\_vec\ (Matrix.col\ A\ (f\ j)) \rangle$  *assms* **by** *fastforce*  
**also have**  $\dots = vec\_index\ (Matrix.col\ A\ (f\ j))\ i$  **unfolding** *Matrix.col-def*

**using**  $\langle i < \dim\text{-vec } (\text{Matrix.col } A \text{ } (f \ j)) \rangle$  **by** *auto*  
**finally show**  $\text{vec-index } (\text{Matrix.col } (\text{per-col } A \ f) \ j) \ i =$   
 $\text{vec-index } (\text{Matrix.col } A \ (f \ j)) \ i .$   
**qed**

**lemma** *per-col-adjoint-row*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**and**  $f \ i < n$   
**shows**  $\text{Matrix.row } (\text{Complex-Matrix.adjoint } (\text{per-col } A \ f)) \ i =$   
 $\text{Matrix.row } (\text{Complex-Matrix.adjoint } A) \ (f \ i)$   
**proof** –  
**have**  $\text{per-col } A \ f \in \text{carrier-mat } n \ n$  **using** *assms per-col-carrier[of A]*  
**by** *simp*  
**hence**  $\text{Matrix.row } (\text{Complex-Matrix.adjoint } (\text{per-col } A \ f)) \ i =$   
 $\text{conjugate } (\text{Matrix.col } (\text{per-col } A \ f) \ i)$   
**using** *assms adjoint-row[of i per-col A f]* **by** *simp*  
**also have**  $\dots = \text{conjugate } (\text{Matrix.col } A \ (f \ i))$  **using** *assms per-col-col*  
**by** *simp*  
**also have**  $\dots = \text{Matrix.row } (\text{Complex-Matrix.adjoint } A) \ (f \ i)$  **using** *assms*  
*adjoint-row[of f i A]* **by** *simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *per-col-mult-adjoint*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**and**  $j < n$   
**and**  $f \ i < n$   
**and**  $f \ j < n$   
**shows**  $((\text{Complex-Matrix.adjoint } (\text{per-col } A \ f)) * (\text{per-col } A \ f))\$\$(i,j) =$   
 $((\text{Complex-Matrix.adjoint } A) * A)\$(f \ i, f \ j)$   
**proof** –  
**have**  $((\text{Complex-Matrix.adjoint } (\text{per-col } A \ f)) * (\text{per-col } A \ f))\$(i,j) =$   
 $\text{Matrix.scalar-prod } (\text{Matrix.row } (\text{Complex-Matrix.adjoint } (\text{per-col } A \ f)) \ i)$   
 $(\text{Matrix.col } A \ (f \ j))$  **using** *assms per-col-col unfolding times-mat-def*  
**by** (*metis adjoint-dim-row carrier-matD(2) dim-col-mat(1) index-mat(1)*)  
*old.prod.case per-col-def*  
**also have**  $\dots = \text{Matrix.scalar-prod}$   
 $(\text{Matrix.row } (\text{Complex-Matrix.adjoint } A) \ (f \ i))$   
 $(\text{Matrix.col } A \ (f \ j))$  **using** *assms per-col-adjoint-row by metis*  
**also have**  $\dots = ((\text{Complex-Matrix.adjoint } A) * A)\$(f \ i, f \ j)$  **using** *assms*  
*unfolding times-mat-def by simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *idty-index*:  
**assumes** *bij-betw*  $f \ \{.. < n\} \ \{.. < n\}$   
**and**  $i < n$

```

    and  $j < n$ 
  shows  $(1_m \ n) \$(i,j) = (1_m \ n) \$(f \ i, f \ j)$ 
  proof -
    have  $f \ i < n \ f \ j < n$  using assms bij-betwE by auto
    show ?thesis
    proof (cases  $i = j$ )
      case True
        then show ?thesis using  $\langle f \ i < n \rangle$  assms by simp
      next
        case False
          hence  $f \ i \neq f \ j$  using assms
          by (metis bij-betw-iff-bijections lessThan-iff)
          then show ?thesis
            by (metis  $\langle f \ i < n \rangle \langle f \ j < n \rangle$  assms(2) assms(3) index-one-mat(1))
    qed
  qed

```

lemma *per-col-unitary*:

```

  assumes  $A \in$  carrier-mat  $n \ n$ 
  and unitary  $A$ 
  and bij-betw  $f \ \{.. < n\} \ \{.. < n\}$ 
  shows unitary (per-col  $A \ f$ ) unfolding unitary-def
  proof
    show pc: per-col  $A \ f \in$ 
      carrier-mat (dim-row (per-col  $A \ f$ )) (dim-row (per-col  $A \ f$ ))
      using assms per-col-carrier by (metis carrier-matD(1))
    have dim-row (per-col  $A \ f$ ) =  $n$  using assms per-col-carrier
      by (metis carrier-matD(1))
    moreover have (Complex-Matrix.adjoint (per-col  $A \ f$ )) * (per-col  $A \ f$ ) =  $1_m \ n$ 

  proof (rule eq-matI)
    show dim-row (Complex-Matrix.adjoint (per-col  $A \ f$ ) * per-col  $A \ f$ ) =
      dim-row ( $1_m \ n$ ) using pc
      by (metis adjoint-dim-row calculation carrier-matD(2) index-mult-mat(2))
      index-one-mat(2))
    thus dim-col (Complex-Matrix.adjoint (per-col  $A \ f$ ) * per-col  $A \ f$ ) =
      dim-col ( $1_m \ n$ ) by auto
    fix  $i \ j$ 
    assume  $i < \text{dim-row } (1_m \ n)$  and  $j < \text{dim-col } (1_m \ n)$  note  $ij = \text{this}$ 
    have (Complex-Matrix.adjoint (per-col  $A \ f$ ) * per-col  $A \ f$ )  $\$(i, j) =$ 
      (Complex-Matrix.adjoint  $A * A$ )  $\$(f \ i, f \ j)$ 
    proof (rule per-col-mult-adjoint)
      show  $A \in$  carrier-mat  $n \ n$  using assms by simp
      show  $i < n \ j < n$  using  $ij$  by auto
      thus  $f \ i < n$  using assms by (meson bij-betw-apply lessThan-iff)
      show  $f \ j < n$  using  $\langle j < n \rangle$  assms by (meson bij-betw-apply lessThan-iff)
    qed
    also have ... =  $(1_m \ n) \$(f \ i, f \ j)$  using assms
      unfolding Complex-Matrix.unitary-def

```

by (*metis* *assms*(2) *unitary-simps*(1))  
 also have ... =  $(1_m \ n) \ \$(i,j)$  using *idty-index*[*of f n i j*] *assms ij*  
 by *auto*  
 finally show (*Complex-Matrix.adjoint* (*per-col A f*) \*  
*per-col A f*)  $\$(i,j) = 1_m \ n \ \$(i,j)$  .  
 qed  
 ultimately show *inverts-mat* (*per-col A f*)  
 (*Complex-Matrix.adjoint* (*per-col A f*))  
 using *inverts-mat-symm* *inverts-mat-def*  
 by (*metis* (*no-types*, *lifting*) *adjoint-dim-col* *adjoint-dim-row*  
*carrier-mat-triv* *index-mult-mat*(3) *index-one-mat*(3))  
 qed

**definition** *per-diag* where  
*per-diag A f* = *Matrix.mat* (*dim-row A*) (*dim-col A*) ( $\lambda (i,j). A \ \$(f \ i, (f \ j))$ )

**lemma** *per-diag-carrier*:  
 shows *per-diag A f*  $\in$  *carrier-mat* (*dim-row A*) (*dim-col A*)  
 unfolding *per-diag-def* by *simp*

**lemma** *per-diag-diagonal*:  
 assumes  $D \in$  *carrier-mat*  $n \ n$   
 and *diagonal-mat*  $D$   
 and *bij-betw*  $f \ \{.. < n\} \ \{.. < n\}$   
 shows *diagonal-mat* (*per-diag D f*) unfolding *diagonal-mat-def*  
 proof (*intro allI impI*)  
 fix  $i \ j$   
 assume  $i < \text{dim-row}(\text{per-diag } D \ f)$  and  $j < \text{dim-col}(\text{per-diag } D \ f)$   
 and  $i \neq j$  note *asm* = *this*  
 hence  $f \ i \neq f \ j$  using *assms*  
 by (*metis* *bij-betw-iff-bijections* *carrier-matD*(1) *carrier-matD*(2)  
*lessThan-iff per-diag-carrier*)  
 moreover have  $f \ i < n$  using *assms* *asm*  
 by (*metis* *bij-betwE* *carrier-matD*(1) *lessThan-iff per-diag-carrier*)  
 moreover have  $f \ j < n$  using *assms* *asm*  
 by (*metis* *bij-betwE* *carrier-matD*(2) *lessThan-iff per-diag-carrier*)  
 ultimately show *per-diag D f*  $\$(i,j) = 0$  using *assms*  
 unfolding *per-diag-def* *diagonal-mat-def*  
 by (*metis* *asm*(1) *asm*(2) *carrier-matD*(1) *carrier-matD*(2)  
*dim-col-mat*(1) *dim-row-mat*(1) *index-mat*(1) *old.prod.case per-diag-def*)  
 qed

**lemma** *per-diag-diag-mat*:  
 assumes  $A \in$  *carrier-mat*  $n \ n$   
 and  $i < n$   
 and  $f \ i < n$   
 shows *diag-mat* (*per-diag A f*)!  $i = \text{diag-mat } A \ ! (f \ i)$   
 using *assms* unfolding *diag-mat-def* *per-diag-def* by *auto*

**lemma** *per-diag-diag-mat-Re*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $i < n$   
**and**  $f \ i < n$   
**shows**  $\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } A \ f))!i = \text{map } \text{Re } (\text{diag-mat } A) ! (f \ i)$   
**proof** –  
**have**  $\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } A \ f))!i = \text{Re } (\text{diag-mat } (\text{per-diag } A \ f))!i$   
**proof** (*rule nth-map*)  
**show**  $i < \text{length } (\text{diag-mat } (\text{per-diag } A \ f))$   
**using** *assms unfolding diag-mat-def*  
**by** (*metis carrier-matD(1) carrier-matD(2) length-cols-mat-to-cols-list length-map per-diag-carrier*)  
**qed**  
**also have**  $\dots = \text{Re } (\text{diag-mat } A ! (f \ i))$  **using** *assms per-diag-diag-mat*  
**by** *metis*  
**also have**  $\dots = \text{map } \text{Re } (\text{diag-mat } A) ! (f \ i)$  **unfolding** *diag-mat-def*  
**using** *assms by auto*  
**finally show** *?thesis* .  
**qed**

**lemma** *per-diag-real*:  
**fixes**  $B::\text{complex Matrix.mat}$   
**assumes**  $B \in \text{carrier-mat } n \ n$   
**and**  $\forall i < n. B \ \$\$ (i,i) \in \text{Reals}$   
**and** *bij-betw*  $f \ \{..<n\} \ \{..<n\}$   
**shows**  $\forall j < n. (\text{per-diag } B \ f) \ \$\$ (j,j) \in \text{Reals}$   
**proof** (*intro allI impI*)  
**fix**  $j$   
**assume**  $j < n$   
**hence**  $\text{per-diag } B \ f \ \$\$ (j,j) = B \ \$\$ (f \ j, f \ j)$   
**using** *assms unfolding per-diag-def by simp*  
**also have**  $\dots \in \text{Reals}$  **using** *assms*  $\langle j < n \rangle$  *bij-betwE* **by** *blast*  
**finally show**  $\text{per-diag } B \ f \ \$\$ (j,j) \in \text{Reals}$  .  
**qed**

**lemma** *per-col-mult-unitary*:  
**fixes**  $A::\text{complex Matrix.mat}$   
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and** *unitary*  $A$   
**and**  $D \in \text{carrier-mat } n \ n$   
**and** *diagonal-mat*  $D$   
**and**  $0 < n$   
**and** *bij-betw*  $f \ \{..<n\} \ \{..<n\}$   
**shows**  $A * D * (\text{Complex-Matrix.adjoint } A) =$   
 $(\text{per-col } A \ f) * (\text{per-diag } D \ f) * (\text{Complex-Matrix.adjoint } (\text{per-col } A \ f))$   
**(is**  $?L = ?R$ **)**  
**proof** –  
**have** *row: dim-row*  $?L = \text{dim-row } ?R$  **using** *per-col-carrier assms*  
**by** (*metis carrier-matD(1) index-mult-mat(2)*)

```

have col: dim-col ?L = dim-col ?R using per-col-carrier assms
  by (metis adjoint-dim carrier-matD(2) index-mult-mat(3))
define fc::complex Matrix.mat set where fc = carrier-mat n n
interpret cpx-sq-mat n n fc
proof
  show 0 < n using assms by simp
qed (auto simp add: fc-def)
define h where
  h = (λi. (if i < n then diag-mat D ! i ·m rank-1-proj (Matrix.col A i)
    else (0m n n)))
define g where
  g = (λi. (if i < n
    then diag-mat D ! (f i) ·m rank-1-proj (Matrix.col A (f i))
    else (0m n n)))
have f ' {..using assms
  by (simp add: bij-betw-imp-surj-on)
have g: ∀ i. g i ∈ fc unfolding g-def fc-def
  by (metis adjoint-dim-col assms(1) carrier-matD(1) carrier-matI
    dim-col fc-mats-carrier rank-1-proj-adjoint rank-1-proj-dim
    smult-carrier-mat zero-mem)
moreover have h:∀ i. h i ∈ fc unfolding h-def fc-def
  by (metis assms(1) carrier-matD(1) dim-col fc-mats-carrier
    rank-1-proj-carrier smult-carrier-mat zero-mem)
moreover have inj-on f {..using assms(6) bij-betw-def by auto
moreover have ∧ x. x ∈ {..unfolding h-def g-def
  by (meson assms(6) bij-betwE lessThan-iff)
ultimately have sum-mat g {..using sum-with-reindex-cong[of h g f {..unfolding sum-mat-def by simp
also have ... = sum-mat h {..using ⟨f ' {..by simp
also have ... = sum-mat (λi. diag-mat D ! i ·m rank-1-proj (Matrix.col A i))
  {..proof (rule sum-mat-cong, (auto simp add:h h-def))
  show ∧ i. i < n ⇒ diag-mat D ! i ·m rank-1-proj (Matrix.col A i) ∈ fc
    using assms unfolding fc-def by (metis fc-mats-carrier h h-def)
  show ∧ i. i < n ⇒ diag-mat D ! i ·m rank-1-proj (Matrix.col A i) ∈ fc
    using assms unfolding fc-def by (metis fc-mats-carrier h h-def)
qed
also have ... = A * D * (Complex-Matrix.adjoint A)
  using weighted-sum-rank-1-proj-unitary assms unfolding fc-def by simp
finally have sg: sum-mat g {..have (per-col A f) * (per-diag D f) *
  (Complex-Matrix.adjoint (per-col A f)) =
  sum-mat (λi. diag-mat (per-diag D f) ! i ·m
    rank-1-proj (Matrix.col (per-col A f) i)) {..proof (rule weighted-sum-rank-1-proj-unitary[symmetric])
  show per-col A f ∈ fc using per-col-carrier[of A] assms unfolding fc-def
  by simp
  show per-diag D f ∈ fc using per-diag-carrier[of D] assms

```

**unfolding** *fc-def* **by** *simp*  
**show** *diagonal-mat* (*per-diag*  $D$   $f$ ) **using** *assms per-diag-diagonal*[*of D*]  
**by** *simp*  
**show** *unitary* (*per-col*  $A$   $f$ ) **using** *per-col-unitary*[*of A*] *assms* **by** *simp*  
**qed**  
**also have**  $\dots = \text{sum-mat } g \{..<n\}$   
**proof** (*rule sum-mat-cong*, (*auto simp add: g-def*))  
**show**  $\bigwedge i. i < n \implies \text{diag-mat } (\text{per-diag } D \ f) \ ! \ i \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) \in \text{fc}$   
**proof** –  
**fix**  $i$   
**assume**  $i < n$   
**have**  $\text{dim-vec } (\text{Matrix.col } (\text{per-col } A \ f) \ i) = n$  **using** *assms per-col-col*  
**by** (*metis carrier-matD(1) dim-col*)  
**hence**  $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) \in \text{fc}$  **unfolding** *fc-def*  
**using** *rank-1-proj-carrier* **by** *blast*  
**thus**  $\text{diag-mat } (\text{per-diag } D \ f) \ ! \ i \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) \in \text{fc}$  **unfolding** *fc-def* **by** *simp*  
**qed**  
**show**  $\bigwedge i. i < n \implies \text{diag-mat } D \ ! \ f \ i \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } A \ (f \ i)) \in \text{fc}$   
**proof** –  
**fix**  $i$   
**assume**  $i < n$   
**hence**  $f \ i < n$  **using**  $\langle f \ ' \ \{..<n\} = \{..<n\} \rangle$  **by** *auto*  
**hence**  $\text{dim-vec } (\text{Matrix.col } A \ (f \ i)) = n$  **using** *assms*  
**by** (*metis carrier-matD(1) dim-col*)  
**hence**  $\text{rank-1-proj } (\text{Matrix.col } A \ (f \ i)) \in \text{fc}$  **unfolding** *fc-def*  
**using** *rank-1-proj-carrier* **by** *blast*  
**thus**  $\text{diag-mat } D \ ! \ f \ i \cdot_m$   $\text{rank-1-proj } (\text{Matrix.col } A \ (f \ i)) \in \text{fc}$   
**unfolding** *fc-def* **by** *simp*  
**qed**  
**show**  $\bigwedge i. i < n \implies$   
 $\text{diag-mat } (\text{per-diag } D \ f) \ ! \ i \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) =$   
 $\text{diag-mat } D \ ! \ f \ i \cdot_m \ \text{rank-1-proj } (\text{Matrix.col } A \ (f \ i))$   
**proof**–  
**fix**  $i$   
**assume**  $i < n$   
**hence**  $f \ i < n$  **using**  $\langle f \ ' \ \{..<n\} = \{..<n\} \rangle$  **by** *auto*  
**have**  $\text{Matrix.col } (\text{per-col } A \ f) \ i = \text{Matrix.col } A \ (f \ i)$   
**using** *per-col-col* *assms*  $\langle i < n \rangle$  **by** *simp*  
**hence**  $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) =$   
 $\text{rank-1-proj } (\text{Matrix.col } A \ (f \ i))$  **by** *simp*  
**thus**  $\text{diag-mat } (\text{per-diag } D \ f) \ ! \ i \cdot_m$   
 $\text{rank-1-proj } (\text{Matrix.col } (\text{per-col } A \ f) \ i) =$   
 $\text{diag-mat } D \ ! \ f \ i \cdot_m \ \text{rank-1-proj } (\text{Matrix.col } A \ (f \ i))$   
**using** *assms per-diag-diag-mat*[*of D*]  $\langle i < n \rangle$   $\langle f \ i < n \rangle$  **by** *simp*  
**qed**

**qed**  
**also have**  $\dots = A * D * (\text{Complex-Matrix.adjoint } A)$  **using** *sg* **by** *simp*  
**finally have**  $(\text{per-col } A f) * (\text{per-diag } D f) * (\text{Complex-Matrix.adjoint } (\text{per-col } A f)) = A * D * (\text{Complex-Matrix.adjoint } A)$  .  
**thus** *?thesis* **by** *simp*  
**qed**

**lemma** *sort-permutation*:

**assumes**  $m = \text{sort } l$   
**obtains**  $f$  **where**  $\text{bij-betw } f \{..<\text{length } l\} \{..<\text{length } l\} \wedge (\forall i < \text{length } l. l ! f i = m ! i)$   
**proof** –  
**have**  $\text{length } l = \text{length } m$  **using** *assms* **by** *simp*  
**have**  $\text{mset } l = \text{mset } m$  **using** *assms* **by** *simp*  
**from** *this* **obtain**  $p$  **where**  $p$  *permutes*  $\{..<\text{length } m\}$  *permute-list*  $p l = m$  **by** *(metis <length l = length m> mset-eq-permutation)* **note**  $\text{pprop} = \text{this}$   
**have**  $\text{bij-betw } p \{..<\text{length } l\} \{..<\text{length } l\}$  **using**  $\text{pprop}$   $\langle \text{length } l = \text{length } m \rangle$  **by** *(simp add: permutes-imp-bij)*  
**moreover have**  $\forall i < \text{length } l. l ! p i = m ! i$  **using**  $\text{pprop}$  **by** *(metis <length l = length m> permute-list-nth)*  
**ultimately have**  $\exists f. \text{bij-betw } f \{..<\text{length } l\} \{..<\text{length } l\} \wedge (\forall i < \text{length } l. l ! f i = m ! i)$  **by** *auto*  
**thus** *?thesis* **using** *that* **by** *auto*  
**qed**

**lemma** *per-diag-sorted-Re*:

**fixes**  $B :: \text{complex Matrix.mat}$   
**assumes**  $B \in \text{carrier-mat } n \ n$   
**obtains**  $f$  **where**  $\text{bij-betw } f \{..<n\} \{..<n\} \wedge \text{map } \text{Re } (\text{diag-mat } (\text{per-diag } B f)) = \text{sort } (\text{map } \text{Re } (\text{diag-mat } B))$   
**proof** –  
**define**  $m$  **where**  $m = \text{sort } (\text{map } \text{Re } (\text{diag-mat } B))$   
**have**  $\text{length } m = \text{length } (\text{map } \text{Re } (\text{diag-mat } B))$  **unfolding** *m-def* **by** *simp*  
**also have**  $\dots = \text{length } (\text{diag-mat } B)$  **by** *simp*  
**also have**  $\dots = n$  **using** *assms* **unfolding** *diag-mat-def* **by** *simp*  
**finally have**  $\text{length } m = n$  .  
**from** *this* **obtain**  $f$  **where**  $\text{bij-betw } f \{..<n\} \{..<n\} \wedge (\forall i < n. (\text{map } \text{Re } (\text{diag-mat } B)) ! f i = m ! i)$  **using** *sort-permutation[of m map Re (diag-mat B)]* *m-def* **by** *auto*  
**note**  $\text{fprop} = \text{this}$   
**have**  $l: \text{length } (\text{diag-mat } (\text{per-diag } B f)) = \text{length } m$  **using** *per-diag-carrier* *assms*  $\langle \text{length } m = n \rangle$  **unfolding** *diag-mat-def* **by** *(metis carrier-matD(1) length-map map-nth)*  
**have**  $\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } B f)) = m$   
**proof** *(rule list-eq-iff-nth-eq[THEN iffD2], intro conjI)*  
**show**  $\text{length } (\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } B f))) = \text{length } m$  **using** *l* **by** *simp*  
**have**  $\forall i < n. (\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } B f))) ! i = m ! i$

```

proof (intro allI impI)
  fix i
  assume i < n
  have (map Re (diag-mat (per-diag B f)))!i = (map Re (diag-mat B))!(f i)
  proof (rule per-diag-diag-mat-Re)
    show B ∈ carrier-mat n n using assms by simp
    show i < n using ⟨i < n⟩ .
    thus f i < n using fprop bij-betwE by blast
  qed
  also have ... = m!i using fprop ⟨i < n⟩ by simp
  finally show (map Re (diag-mat (per-diag B f)))!i = m!i .
  qed
  thus ∀ i < length (map Re (diag-mat (per-diag B f))).
    (map Re (diag-mat (per-diag B f)))!i = m!i using l ⟨length m = n⟩ by simp
  qed
  thus ?thesis using that fprop unfolding m-def by auto
qed

```

**lemma** *bij-unitary-diag*:

```

fixes A::complex Matrix.mat
assumes unitary-diag A B U
and A ∈ carrier-mat n n
and bij-betw f {.. $n$ } {.. $n$ }
and 0 < n
shows unitary-diag A (per-diag B f) (per-col U f)
proof (intro unitary-diagI)
  show unitary (per-col U f) using assms unitary-diagD(1)
    unitary-diagD(3) per-col-unitary by (metis similar-mat-witD2(6))
  show diagonal-mat (per-diag B f)
    using assms unitary-diagD(2) per-diag-diagonal
    by (metis similar-mat-witD2(5) unitary-diagD(1))
  have A = U * B * (Complex-Matrix.adjoint U) using assms
    unitary-diagD similar-mat-witD2(3) unitary-diagD(1) by blast
  also have ... = (per-col U f) * (per-diag B f)
    * (Complex-Matrix.adjoint (per-col U f)) using assms per-col-mult-unitary
    by (meson similar-mat-witD2(5) similar-mat-witD2(6) unitary-diagD(1)
      unitary-diagD(2) unitary-diagD(3))
  finally have eq: A = per-col U f * per-diag B f *
    Complex-Matrix.adjoint (per-col U f) .
  show similar-mat-wit A (per-diag B f) (per-col U f)
    (Complex-Matrix.adjoint (per-col U f))
    unfolding similar-mat-wit-def Let-def
  proof (intro conjI)
    have A ∈ carrier-mat n n using assms by simp
    moreover have per-diag B f ∈ carrier-mat n n using assms unitary-diagD(1)

      per-diag-carrier[of B]
      by (metis carrier-matD(1) carrier-matD(2) similar-mat-witD2(5))
    moreover have per-col U f ∈ carrier-mat n n using assms unitary-diagD(1)

```

$\text{per-col-carrier[of } U]$   
**by**  $(\text{metis similar-mat-witD2}(6))$   
**moreover hence**  $\text{Complex-Matrix.adjoint } (\text{per-col } U f) \in \text{carrier-mat } n \ n$   
**by**  $(\text{simp add: adjoint-dim})$   
**ultimately show**  
 $\{A, \text{per-diag } B f, \text{per-col } U f, \text{Complex-Matrix.adjoint } (\text{per-col } U f)\} \subseteq$   
 $\text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A)$  **by**  $\text{auto}$   
**show**  $\text{per-col } U f * \text{Complex-Matrix.adjoint } (\text{per-col } U f) = 1_m (\text{dim-row } A)$   
**using**  $\langle \text{Complex-Matrix.unitary } (\text{per-col } U f) \rangle \text{ assms}(2)$   
 $\langle \text{per-col } U f \in \text{carrier-mat } n \ n \rangle$  **by**  $\text{auto}$   
**show**  $\text{Complex-Matrix.adjoint } (\text{per-col } U f) * \text{per-col } U f = 1_m (\text{dim-row } A)$   
**using**  $\langle \text{Complex-Matrix.unitary } (\text{per-col } U f) \rangle \text{ assms}$   
 $\langle \text{per-col } U f \in \text{carrier-mat } n \ n \rangle$  **by**  $\text{auto}$   
**qed**  $(\text{simp add: eq})$   
**qed**

**lemma hermitian-real-diag-sorted:**  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $0 < n$   
**and**  $\text{hermitian } A$   
**obtains**  $Bs \ Us$  **where**  $\text{real-diag-decomp } A \ Bs \ Us \wedge \text{sorted } (\text{map } \text{Re } (\text{diag-mat } Bs))$   
**proof** –  
**obtain**  $U1 \ B1$  **where**  $\text{real-diag-decomp } A \ B1 \ U1$   
**using**  $\text{hermitian-real-diag-decomp[of } A]$  **assms** **by**  $\text{auto}$  **note**  $\text{ubprop} = \text{this}$   
**hence**  $B1 \in \text{carrier-mat } n \ n$  **using**  $\text{assms}$  **unfolding**  $\text{real-diag-decomp-def}$   
**by**  $(\text{meson unitary-diag-carrier}(1))$   
**from this obtain**  $f$  **where**  $\text{bij-betw } f \ \{..<n\} \ \{..<n\} \wedge$   
 $\text{map } \text{Re } (\text{diag-mat } (\text{per-diag } B1 f)) = \text{sort } (\text{map } \text{Re } (\text{diag-mat } B1))$   
**using**  $\text{per-diag-sorted-Re}$  **by**  $\text{auto}$  **note**  $\text{fprop} = \text{this}$   
**define**  $Bs$  **where**  $Bs = \text{per-diag } B1 f$   
**define**  $Us$  **where**  $Us = \text{per-col } U1 f$   
**have**  $\text{unitary-diag } A \ Bs \ Us$  **unfolding**  $Bs\text{-def } Us\text{-def}$   
**proof**  $(\text{rule bij-unitary-diag})$   
**show**  $\text{unitary-diag } A \ B1 \ U1$  **using**  $\text{ubprop}$  **unfolding**  $\text{real-diag-decomp-def}$   
**by**  $\text{simp}$   
**show**  $A \in \text{carrier-mat } n \ n$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**show**  $\text{bij-betw } f \ \{..<n\} \ \{..<n\}$  **using**  $\text{fprop}$  **by**  $\text{simp}$   
**show**  $0 < n$  **using**  $\text{assms}$  **by**  $\text{simp}$   
**qed**  
**have**  $\text{real-diag-decomp } A \ Bs \ Us$  **unfolding**  $\text{real-diag-decomp-def}$   
**proof**  $(\text{simp add: } \langle \text{unitary-diag } A \ Bs \ Us \rangle)$   
**show**  $\forall i < \text{dim-row } Bs. Bs \ \$\$ (i, i) \in \mathbb{R}$  **unfolding**  $Bs\text{-def}$   
**proof**  $(\text{rule per-diag-real})$   
**show**  $\text{br: } B1 \in \text{carrier-mat } (\text{dim-row } (\text{per-diag } B1 f))$   
 $(\text{dim-row } (\text{per-diag } B1 f))$   
**by**  $(\text{metis } \langle B1 \in \text{carrier-mat } n \ n \rangle \text{ carrier-matD}(1) \text{ per-diag-carrier})$   
**thus**  $\forall i < \text{dim-row } (\text{per-diag } B1 f). B1 \ \$\$ (i, i) \in \mathbb{R}$  **using**  $\text{ubprop}$  **by**  $\text{auto}$   
**show**  $\text{bij-betw } f \ \{..<\text{dim-row } (\text{per-diag } B1 f)\}$

```

    {..dim-row (per-diag B1 f)} using fprop
  by (metis br  $\langle B1 \in \text{carrier-mat } n \ n \rangle \text{carrier-matD}(1)$ )
qed
qed
moreover have sorted (map Re (diag-mat Bs)) using fprop unfolding Bs-def
  by simp
ultimately show ?thesis using that by simp
qed

```

## 8 Commuting Hermitian families

This part is devoted to the proof that a finite family of commuting Hermitian matrices is simultaneously diagonalizable.

### 8.1 Intermediate properties

```

lemma real-diag-decomp-mult-dbm-unit:
  assumes  $A \in \text{carrier-mat } n \ n$ 
  and real-diag-decomp A B U
  and  $B = \text{diag-block-mat } Bl$ 
  and  $\text{length } Ul = \text{length } Bl$ 
  and  $\forall i < \text{length } Bl. \text{dim-col } (Bl!i) = \text{dim-row } (Bl!i)$ 
  and  $\forall i < \text{length } Bl. \text{dim-row } (Bl!i) = \text{dim-row } (Ul!i)$ 
  and  $\forall i < \text{length } Bl. \text{dim-col } (Bl!i) = \text{dim-col } (Ul!i)$ 
  and unitary (diag-block-mat Ul)
  and  $\forall i < \text{length } Ul. Ul!i * Bl!i = Bl!i * Ul!i$ 
shows real-diag-decomp A B ( $U * (\text{diag-block-mat } Ul)$ )
  unfolding real-diag-decomp-def
proof (intro conjI allI impI)
  have  $B \in \text{carrier-mat } n \ n$ 
  by (meson assms(1) assms(2) real-diag-decompD(1) unitary-diag-carrier(1))
  have  $\text{dim-row } (\text{diag-block-mat } Ul) = \text{dim-row } B$ 
  using diag-block-mat-dim-row-cong assms by blast
  moreover have  $\text{dim-col } (\text{diag-block-mat } Ul) = \text{dim-col } B$ 
  using diag-block-mat-dim-col-cong assms by blast
  ultimately have  $\text{diag-block-mat } Ul \in \text{carrier-mat } n \ n$  using assms
  by (metis  $\langle B \in \text{carrier-mat } n \ n \rangle \text{carrier-matD}(1) \text{carrier-matD}(2)$ 
    carrier-mat-triv)
  define Uf where  $Uf = U * (\text{diag-block-mat } Ul)$ 
  show  $\bigwedge i. i < \text{dim-row } B \implies B \$\$ (i, i) \in \mathbf{R}$ 
  using assms real-diag-decompD(2)  $\langle B \in \text{carrier-mat } n \ n \rangle$  by auto
  show unitary-diag A B Uf unfolding unitary-diag-def
proof
  show diagonal-mat B using assms real-diag-decompD(2)
    real-diag-decompD(1) unitary-diagD(2) by blast
  show unitarily-equiv A B Uf unfolding Uf-def
  proof (rule conjugate-eq-unitarily-equiv)
  show  $A \in \text{carrier-mat } n \ n$  using assms by simp

```

```

show unitarily-equiv  $A B U$  using assms real-diag-decompD(1)
  unfolding unitary-diag-def by simp
show diag-block-mat  $Ul \in$  carrier-mat  $n n$ 
  using  $\langle$ diag-block-mat  $Ul \in$  carrier-mat  $n n \rangle$  .
show unitary (diag-block-mat  $Ul$ ) using assms by simp
have mat-conj (diag-block-mat  $Ul$ ) (diag-block-mat  $Bl$ ) =
  diag-block-mat  $Bl$ 
proof (rule mat-conj-unit-commute)
  show unitary (diag-block-mat  $Ul$ ) using  $\langle$ unitary (diag-block-mat  $Ul$ ) $\rangle$  .
  show diag-block-mat  $Bl \in$  carrier-mat  $n n$ 
    using assms  $\langle B \in$  carrier-mat  $n n \rangle$  by simp
  show diag-block-mat  $Ul \in$  carrier-mat  $n n$ 
    using  $\langle$ diag-block-mat  $Ul \in$  carrier-mat  $n n \rangle$  .
  show diag-block-mat  $Ul *$  diag-block-mat  $Bl =$ 
    diag-block-mat  $Bl *$  diag-block-mat  $Ul$ 
  proof (rule diag-block-mat-commute)
    show length  $Ul =$  length  $Bl$  using assms
      by simp
    show comm:  $\forall i <$  length  $Ul$ .  $Ul ! i * Bl ! i = Bl ! i * Ul ! i$ 
      using assms by simp
    show  $\forall i <$  length  $Ul$ . dim-col ( $Ul ! i$ ) = dim-row ( $Bl ! i$ )
      using assms by presburger
    thus  $\forall i <$  length  $Ul$ . dim-col ( $Bl ! i$ ) = dim-row ( $Ul ! i$ )
      by (metis comm index-mult-mat(2) index-mult-mat(3))
  qed
qed
thus diag-block-mat  $Ul * B *$ 
  Complex-Matrix.adjoint (diag-block-mat  $Ul$ ) =  $B$ 
using  $\langle B =$  diag-block-mat  $Bl \rangle$  unfolding mat-conj-def by simp
qed
qed
qed

```

```

lemma real-diag-decomp-block-set:
  assumes  $Als \neq \{\}$ 
  and  $0 < n$ 
  and  $\forall Al \in Als$ . length  $Al = n$ 
  and  $\forall i < n$ .  $\forall Al \in Als$ . dim-row ( $Al ! i$ ) = dim-col ( $Al ! i$ )
  and  $\forall i < n$ .  $\exists U$ .  $\forall Al \in Als$ .  $\exists B$ . real-diag-decomp ( $Al ! i$ )  $B U$ 
shows  $\exists Ul$ . (length  $Ul = n \wedge (\forall i < n$ .  $\forall Al \in Als$ .
  (dim-row ( $Ul ! i$ ) = dim-row ( $Al ! i$ )  $\wedge$  dim-col ( $Ul ! i$ ) = dim-col ( $Al ! i$ )))  $\wedge$ 
  ( $\forall Al \in Als$ .  $\exists Bl$ . (length  $Bl = n \wedge$ 
  real-diag-decomp (diag-block-mat  $Al$ ) (diag-block-mat  $Bl$ ) (diag-block-mat  $Ul$ ))))
  using assms
proof (induct  $n$  arbitrary:  $Als$ )
  case 0
  then show ?case by simp
next
  case (Suc  $n$ )

```

**hence**  $\exists U. \forall Al \in Als. \exists B. \text{real-diag-decomp } (Al!0) B U$  **by simp**  
**from this obtain**  $U0$  **where**  $\forall Al \in Als. \exists B. \text{real-diag-decomp } (Al!0) B U0$   
**by auto note**  $u0 = this$   
**have**  $u0\text{-dim}: \forall Al \in Als.$   
 $\quad \text{dim-row } ([U0] ! 0) = \text{dim-row } (Al ! 0) \wedge$   
 $\quad \text{dim-col } ([U0] ! 0) = \text{dim-col } (Al ! 0)$   
**proof** (*intro allI impI ballI*)  
**fix**  $Al$   
**assume**  $Al \in Als$   
**have**  $[U0]!0 = U0$  **by simp**  
**have**  $\exists B. \text{real-diag-decomp } (Al!0) B U0$  **using**  $u0 \langle Al \in Als \rangle$   
**by simp**  
**from this obtain**  $B$  **where**  $\text{real-diag-decomp } (Al!0) B U0$  **by auto**  
**moreover have**  $\text{dim-row } (Al!0) = \text{dim-col } (Al!0)$  **using**  $\langle Al \in Als \rangle Suc$   
**by simp**  
**ultimately have**  $\text{dim-row } U0 = \text{dim-row } (Al!0)$   
**using** *unitary-diag-carrier(2)*  
**by** (*metis carrier-matD(1) carrier-matI real-diag-decompD(1)*)  
**moreover have**  $\text{dim-col } U0 = \text{dim-col } (Al!0)$   
**using**  $\langle \text{real-diag-decomp } (Al!0) B U0 \rangle \langle \text{dim-row } (Al!0) = \text{dim-col } (Al!0) \rangle$   
**using** *real-diag-decompD(1) unitary-diag-carrier(2)*  
**by** (*metis carrier-matD(2) carrier-mat-triv*)  
**ultimately show**  $\text{dim-row } ([U0] ! 0) = \text{dim-row } (Al ! 0) \wedge$   
 $\text{dim-col } ([U0] ! 0) = \text{dim-col } (Al ! 0)$   
**by simp**  
**qed**  
**show** *?case*  
**proof** (*cases n = 0*)  
**case** *True*  
**hence**  $\forall Al \in Als. \text{diag-block-mat } Al = Al!0$  **using** *Suc*  
**by** (*simp add: diag-block-mat-length-1*)  
**have**  $\forall Al \in Als. \exists Bl. (\text{length } Bl = 1 \wedge$   
 $\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$   
 $(\text{diag-block-mat } [U0]))$   
**proof**  
**fix**  $Al$   
**assume**  $Al \in Als$   
**hence**  $\exists B. \text{real-diag-decomp } (Al!0) B U0$  **using**  $u0$  **by simp**  
**from this obtain**  $B$  **where**  $\text{real-diag-decomp } (Al!0) B U0$  **by auto**  
**hence**  $\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } [B])$   
 $(\text{diag-block-mat } [U0])$   
**by** (*metis*  $\langle Al \in Als \rangle \langle \forall Al \in Als. \text{diag-block-mat } Al = Al ! 0 \rangle$   
*diag-block-mat-singleton*)  
**moreover have**  $\text{length } [B] = 1$  **by simp**  
**ultimately show**  $\exists Bl. (\text{length } Bl = 1 \wedge$   
 $\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$   
 $(\text{diag-block-mat } [U0]))$   
**by blast**  
**qed**

**moreover have**  $\text{length } [U0] = 1$  **by** *simp*  
**moreover have**  $\forall i < \text{Suc } n. \forall Al \in \text{Als}. \text{dim-row } ([U0] ! i) = \text{dim-row } (Al ! i) \wedge$   
 $\text{dim-col } ([U0] ! i) = \text{dim-col } (Al ! i)$   
**using**  $\langle u0\text{-dim } \langle n = 0 \rangle \rangle$  **by** *simp*  
**ultimately have**  $\text{length } [U0] = \text{Suc } 0 \wedge$   
 $(\forall i < \text{Suc } n. \forall Al \in \text{Als}. \text{dim-row } ([U0] ! i) = \text{dim-row } (Al ! i) \wedge$   
 $\text{dim-col } ([U0] ! i) = \text{dim-col } (Al ! i)) \wedge$   
 $(\forall Al \in \text{Als}. \exists Bl. \text{length } Bl = \text{Suc } n \wedge \text{real-diag-decomp}$   
 $(\text{diag-block-mat } Al) (\text{diag-block-mat } Bl) (\text{diag-block-mat } [U0]))$   
**using**  $\langle n=0 \rangle$  *One-nat-def* **by** *metis*  
**thus** *?thesis* **by** (*metis True*)  
**next**  
**case** *False*  
**hence**  $0 < n$  **by** *simp*  
**define** *tAls* **where**  $tAls = tl \text{ 'Als}$   
**have**  $tex: \forall tAl \in tAls. \exists Al \in \text{Als}. tAl = tl \text{ } Al$  **unfolding** *tAls-def* **by** *auto*  
**have**  $\exists Ul. \text{length } Ul = n \wedge$   
 $(\forall i < n. \forall Al \in tAls.$   
 $(\text{dim-row } (Ul ! i) = \text{dim-row } (Al ! i) \wedge \text{dim-col } (Ul ! i) = \text{dim-col } (Al ! i))) \wedge$   
 $(\forall Al \in tAls. \exists Bl. \text{length } Bl = n \wedge$   
 $\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$   
 $(\text{diag-block-mat } Ul))$   
**proof** (*rule Suc(1)*)  
**show**  $tAls \neq \{\}$   $0 < n$  **unfolding** *tAls-def* **by** (*auto simp add: \langle 0 < n \rangle Suc*)  
**show**  $\forall tAl \in tAls. \text{length } tAl = n$   
**using** *Suc(4)* *tex* **by** *fastforce*  
**show**  $\forall i < n. \exists U. \forall Al \in tAls. \exists B. \text{real-diag-decomp } (Al ! i) B U$   
**proof** (*intro allI impI*)  
**fix** *i*  
**assume**  $i < n$   
**hence**  $\exists U. \forall Al \in \text{Als}. \exists B. \text{real-diag-decomp } (Al ! (\text{Suc } i)) B U$   
**using** *Suc Suc-mono* **by** *presburger*  
**from this obtain** *U* **where**  
 $tu: \forall Al \in \text{Als}. \exists B. \text{real-diag-decomp } (Al ! (\text{Suc } i)) B U$  **by** *auto*  
**have**  $\forall tAl \in tAls. \exists B. \text{real-diag-decomp } (tAl ! i) B U$   
**proof**  
**fix** *tAl*  
**assume**  $tAl \in tAls$   
**hence**  $\exists Al \in \text{Als}. tAl = tl \text{ } Al$  **using** *tex* **by** *simp*  
**from this obtain** *Al* **where**  $Al \in \text{Als}$  **and**  $tAl = tl \text{ } Al$  **by** *auto*  
**hence**  $tAl ! i = Al ! (\text{Suc } i)$  **by** (*simp add: Suc(4) \langle i < n \rangle nth-tl*)  
**moreover have**  $\exists B. \text{real-diag-decomp } (Al ! (\text{Suc } i)) B U$   
**using** *tu \langle Al \in \text{Als} \rangle* **by** *simp*  
**ultimately show**  $\exists B. \text{real-diag-decomp } (tAl ! i) B U$  **by** *simp*  
**qed**  
**thus**  $\exists U. \forall Al \in tAls. \exists B. \text{real-diag-decomp } (Al ! i) B U$  **by** *auto*  
**qed**  
**show**  $\forall i < n. \forall Al \in tAls. \text{dim-row } (Al ! i) = \text{dim-col } (Al ! i)$

by (metis Suc(5)  $\langle \forall tAl \in tAls. \text{length } tAl = n \rangle$  not-less-eq nth-tl tex)

**qed**

**from this obtain**  $Ul$  **where**  $\text{length } Ul = n$  **and**

$\forall i < n. \forall Al \in tAls.$

$(\text{dim-row } (Ul!i) = \text{dim-row } (Al!i) \wedge \text{dim-col } (Ul!i) = \text{dim-col } (Al!i))$

$(\forall Al \in tAls. \exists Bl. \text{length } Bl = n \wedge$

$\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$

$(\text{diag-block-mat } Ul))$  **by auto** **note**  $ulprop = \text{this}$

**have**  $\forall Al \in Als. \exists Bl. \text{length } Bl = \text{Suc } n \wedge$

$\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$

$(\text{diag-block-mat } (U0 \# Ul))$

**proof**

**fix**  $Al$

**assume**  $Al \in Als$

**hence**  $0 < \text{length } Al$  **using**  $\text{Suc}$  **by**  $\text{simp}$

**hence**  $Al = \text{hd } Al \# (\text{tl } Al)$  **by**  $\text{simp}$

**have**  $\exists B. \text{real-diag-decomp } (Al!0) B U0$  **using**  $u0 \langle Al \in Als \rangle$  **by**  $\text{simp}$

**from this obtain**  $B0$  **where**  $b0: \text{real-diag-decomp } (Al!0) B0 U0$  **by auto**

**hence**  $\text{real-diag-decomp } (\text{hd } Al) B0 U0$

**by** (metis  $\langle Al = \text{hd } Al \# \text{tl } Al \rangle$  nth-Cons-0)

**have**  $\text{tl } Al \in tAls$  **using**  $\langle Al \in Als \rangle$  **unfolding**  $tAls\text{-def}$  **by**  $\text{simp}$

**hence**  $\exists Bl. \text{length } Bl = n \wedge$

$\text{real-diag-decomp } (\text{diag-block-mat } (\text{tl } Al)) (\text{diag-block-mat } Bl)$

$(\text{diag-block-mat } Ul)$  **using**  $ulprop$  **by**  $\text{simp}$

**from this obtain**  $B1$  **where**  $\text{length } B1 = n$  **and**

$rl: \text{real-diag-decomp } (\text{diag-block-mat } (\text{tl } Al)) (\text{diag-block-mat } B1)$

$(\text{diag-block-mat } Ul)$  **by auto**

**have**  $\text{dim-row } (\text{diag-block-mat } (\text{tl } Al)) = \text{dim-col } (\text{diag-block-mat } (\text{tl } Al))$

**using**  $\text{Suc } \langle Al \in Als \rangle$   $\text{diag-block-mat-dim-row-col-eq}$

**by** (metis (no-types, lifting)  $\langle Al = \text{hd } Al \# \text{tl } Al \rangle$  length-Cons lessI less-trans-Suc nth-tl)

**moreover have**  $\text{dim-row } (Al ! 0) = \text{dim-col } (Al ! 0)$

**using**  $\text{Suc } \langle Al \in Als \rangle$  **by**  $\text{simp}$

**ultimately have**  $\text{real-diag-decomp } (\text{diag-block-mat } ((\text{hd } Al) \# (\text{tl } Al)))$

$(\text{diag-block-mat } (B0 \# B1)) (\text{diag-block-mat } (U0 \# Ul))$

**using**  $\text{four-block-real-diag-decomp}[OF rl b0]$   $\text{diag-block-mat.simps}(2)$

$\text{real-diag-decomp-hermitian}$

**by** (metis  $\langle Al = \text{hd } Al \# \text{tl } Al \rangle$   $b0$   $\text{four-block-real-diag-decomp}$  nth-Cons-0)

$rl)$

**moreover have**  $\text{length } (B0 \# B1) = \text{Suc } n$  **using**  $\langle \text{length } B1 = n \rangle$  **by**  $\text{simp}$

**ultimately show**  $\exists Bl. \text{length } Bl = \text{Suc } n \wedge$

$\text{real-diag-decomp } (\text{diag-block-mat } Al) (\text{diag-block-mat } Bl)$

$(\text{diag-block-mat } (U0 \# Ul))$  **using**  $\langle Al = \text{hd } Al \# \text{tl } Al \rangle$  **by**  $\text{metis}$

**qed**

**moreover have**  $\text{length } (U0 \# Ul) = \text{Suc } n$  **using**  $ulprop$  **by**  $\text{simp}$

**moreover have**  $\forall i < \text{Suc } n. \forall Al \in Als. \text{dim-row } ((U0 \# Ul) ! i) = \text{dim-row } (Al !$

$i) \wedge$

$\text{dim-col } ((U0 \# Ul) ! i) = \text{dim-col } (Al ! i)$

**proof** (intro allI impI ballI)

```

fix  $i$   $Al$ 
assume  $i < \text{Suc } n$  and  $Al \in Als$ 
show  $\text{dim-row } ((U0 \# Ul) ! i) = \text{dim-row } (Al ! i) \wedge$ 
 $\text{dim-col } ((U0 \# Ul) ! i) = \text{dim-col } (Al ! i)$ 
proof (cases  $i = 0$ )
  case True
    then show ?thesis using  $\langle Al \in Als \rangle$  u0-dim by simp
  next
    case False
      hence  $\exists j. i = \text{Suc } j$  by (simp add: not0-implies-Suc)
      from this obtain  $j$  where  $i = \text{Suc } j$  by auto
      hence  $(U0 \# Ul) ! i = Ul ! j$  by simp
      have  $tl \ Al \in tAls$  using  $\langle Al \in Als \rangle$  unfolding tAls-def by simp
      moreover have  $Al ! i = (tl \ Al) ! j$  using  $\langle i = \text{Suc } j \rangle$ 
        by (metis Suc.premis(3) Zero-not-Suc  $\langle Al \in Als \rangle$  diag-block-mat.cases
          list.sel(3) list.size(3) nth-Cons-Suc)
      ultimately show ?thesis using  $\langle (U0 \# Ul) ! i = Ul ! j \rangle$ 
        by (metis Suc-less-SucD  $\langle i < \text{Suc } n \rangle$   $\langle i = \text{Suc } j \rangle$  ulprop(2))
    qed
  qed
ultimately show ?thesis by blast
qed

```

```

lemma real-diag-decomp-eq-comps-props:
  assumes  $Ap \in \text{carrier-mat } n \ n$ 
  and  $0 < n$ 
  and  $\text{real-diag-decomp } Ap \ Bs \ Us \wedge \text{sorted } (\text{map } Re \ (\text{diag-mat } Bs))$ 
shows  $Bs \in \text{carrier-mat } n \ n$  diagonal-mat  $Bs$  unitary  $Us$ 
 $Us \in \text{carrier-mat } n \ n$  diag-diff  $Bs$  (eq-comps (diag-mat  $Bs$ ))
 $\text{eq-comps } (\text{diag-mat } Bs) \neq []$  diag-mat  $Bs \neq []$ 
proof –
  show  $Bs \in \text{carrier-mat } n \ n$ 
    using assms real-diag-decompD(1) unitary-diag-carrier(1)
    by blast
  thus  $\text{diag-mat } Bs \neq []$  using  $\langle 0 < n \rangle$  unfolding diag-mat-def by simp
  show diagonal-mat  $Bs$  using assms
    using real-diag-decompD(1) unitary-diagD(2) by blast
  show unitary  $Us$ 
    using unitary-diagD(3) assms unfolding real-diag-decomp-def by auto
  show  $Us \in \text{carrier-mat } n \ n$ 
    using assms real-diag-decompD(1) unitary-diag-carrier(2)
    by blast
  define eqcl where  $\text{eqcl} = \text{eq-comps } (\text{diag-mat } Bs)$ 
  show diag-diff  $Bs$  eqcl unfolding eqcl-def
proof (rule cpx-sorted-diag-diff)
  show  $Bs \in \text{carrier-mat } n \ n$  using  $\langle Bs \in \text{carrier-mat } n \ n \rangle$  .
  show  $\text{sorted } (\text{map } Re \ (\text{diag-mat } Bs))$  using assms by simp
  show  $\forall i < n. Bs \ \$\$ (i, i) \in \mathbb{R}$ 

```

```

    using assms real-diag-decompD(2) ⟨Bs ∈ carrier-mat n n⟩ by auto
  qed
  show eqcl ≠ [] using eq-comps-not-empty[of diag-mat Bs]
    ⟨Bs ∈ carrier-mat n n⟩ assms
    unfolding eqcl-def diag-mat-def
    by (simp add: assms)
qed

lemma commuting-conj-mat-set-props:
  fixes As::'a::conjugatable-field Matrix.mat set
    and U::'a Matrix.mat
  assumes finite As
    and card As ≤ i
    and ∀ A ∈ As. hermitian A ∧ A ∈ carrier-mat n n
    and ∀ A ∈ As. ∀ B ∈ As. A*B = B*A
    and unitary U
    and U ∈ carrier-mat n n
    and CjA = (λA2. mat-conj (Complex-Matrix.adjoint U) A2) 'As
  shows finite CjA card CjA ≤ i
    ∀ A ∈ CjA. A ∈ carrier-mat n n ∧ hermitian A
    ∀ C1 ∈ CjA. ∀ C2 ∈ CjA. C1*C2 = C2*C1
  proof -
    define Cj where Cj = (λA2. mat-conj (Complex-Matrix.adjoint U) A2)
    have CjA = Cj 'As using assms unfolding Cj-def by simp
    show finite CjA using assms by simp
    show card CjA ≤ i
      using ⟨card As ≤ i⟩ ⟨finite As⟩ card-image-le dual-order.trans assms
      by blast
    show ∀ A ∈ CjA. A ∈ carrier-mat n n ∧ hermitian A
    proof (intro ballI conjI)
      fix A
      assume A ∈ CjA
      hence ∃ nA ∈ As. A = Cj nA using assms unfolding Cj-def by auto
      from this obtain nA where nA ∈ As and A = Cj nA by auto
      have hermitian nA using assms ⟨nA ∈ As⟩ by auto
      thus hermitian A
        using assms ⟨A = Cj nA⟩ hermitian-mat-conj'[of nA n U] ⟨nA ∈ As⟩ Cj-def
        mat-conj-adjoint by fastforce
      show A ∈ carrier-mat n n using ⟨nA ∈ As⟩ ⟨A = Cj nA⟩ unfolding Cj-def
        by (metis ⟨hermitian A⟩ adjoint-dim-row assms(6) carrier-matD(2)
            hermitian-square index-mult-mat(2) mat-conj-adjoint)
    qed
    show ∀ C1 ∈ CjA. ∀ C2 ∈ CjA. C1*C2 = C2*C1
    proof (intro ballI)
      fix C1 C2
      assume C1 ∈ CjA and C2 ∈ CjA
      hence ∃ A1 ∈ As. C1 = mat-conj (Complex-Matrix.adjoint U) A1
        using assms unfolding Cj-def by auto
      from this obtain A1 where A1 ∈ As and

```

```

    C1 = mat-conj (Complex-Matrix.adjoint U) A1
  by auto
have  $\exists A2 \in As. C2 = mat-conj (Complex-Matrix.adjoint U) A2$ 
  using  $\langle C2 \in CjA \rangle$  assms unfolding Cj-def by auto
from this obtain A2 where  $A2 \in As$  and
  C2 = mat-conj (Complex-Matrix.adjoint U) A2
  by auto
have mat-conj (Complex-Matrix.adjoint U) A1 *
  mat-conj (Complex-Matrix.adjoint U) A2 =
  mat-conj (Complex-Matrix.adjoint U) A2 *
  mat-conj (Complex-Matrix.adjoint U) A1
proof (rule mat-conj-commute)
  show unitary U using  $\langle unitary U \rangle$  .
  show  $A1 \in carrier-mat\ n\ n$  using  $\langle A1 \in As \rangle$  assms by simp
  show  $A2 \in carrier-mat\ n\ n$  using  $\langle A2 \in As \rangle$  assms by simp
  show  $U \in carrier-mat\ n\ n$  using  $\langle U \in carrier-mat\ n\ n \rangle$  .
  show  $A1 * A2 = A2 * A1$  using  $\langle A1 \in As \rangle \langle A2 \in As \rangle$  assms by simp
qed
thus  $C1 * C2 = C2 * C1$ 
  using  $\langle C1 = mat-conj (Complex-Matrix.adjoint U) A1 \rangle$ 
   $\langle C2 = mat-conj (Complex-Matrix.adjoint U) A2 \rangle$ 
  by simp
qed
qed

```

**lemma** *commute-extract-diag-block-eq:*

```

fixes Ap::complex Matrix.mat
assumes Ap  $\in carrier-mat\ n\ n$ 
and  $0 < n$ 
and real-diag-decomp Ap Bs Us  $\wedge sorted (map\ Re (diag-mat\ Bs))$ 
and finite Afp
and card Afp  $\leq i$ 
and  $\forall A \in Afp. hermitian\ A \wedge A \in carrier-mat\ n\ n$ 
and  $\forall A \in Afp. \forall B \in Afp. A * B = B * A$ 
and  $\forall A \in Afp. Ap * A = A * Ap$ 
and CjA =  $(\lambda A2. mat-conj (Complex-Matrix.adjoint\ Us)\ A2)$  'Afp
and eqcl = eq-comps (diag-mat Bs)
shows  $\forall C \in CjA. C = diag-block-mat (extract-subdiags\ C\ eqcl)$ 
proof
note ubprops = real-diag-decomp-eq-comps-props[OF assms(1) assms(2) assms(3)]
note cjprops = commuting-conj-mat-set-props[OF assms(4) assms(5) assms(6)]
  assms(7) ubprops(3) ubprops(4) assms(9)]
fix C
assume C  $\in CjA$ 
hence  $\exists Ac \in Afp. C = mat-conj (Complex-Matrix.adjoint\ Us)\ Ac$ 
  using assms by auto
from this obtain Ac where  $Ac \in Afp$  and
  C = mat-conj (Complex-Matrix.adjoint Us) Ac by auto
show C = diag-block-mat (extract-subdiags C eqcl)

```

**proof** (rule *diag-compat-extract-subdiag*)  
**show**  $C \in \text{carrier-mat } n \ n$  **using**  $\langle \text{CjA} \rangle$  **by** *simp*  
**show** *diag-compat*  $C$  *eqcl*  
**proof** (rule *commute-diag-compat*)  
**show**  $Bs \in \text{carrier-mat } n \ n$  **using**  $\langle Bs \in \text{carrier-mat } n \ n \rangle$  .  
**show** *diag-diff*  $Bs$  *eqcl* **using** *ubprops* *assms* **by** *simp*  
**show** *diagonal-mat*  $Bs$  **using**  $\langle \text{diagonal-mat } Bs \rangle$  .  
**show**  $C \in \text{carrier-mat } n \ n$  **using**  $\langle C \in \text{carrier-mat } n \ n \rangle$  .  
**have**  $Bs * (\text{Complex-Matrix.adjoint } Us * Ac * Us) =$   
 $\text{Complex-Matrix.adjoint } Us * Ac * Us * Bs$   
**proof** (rule *unitarily-equiv-commute*)  
**show** *unitarily-equiv*  $Ap$   $Bs$   $Us$  **using** *assms* *real-diag-decompD*(1)  
**by** *simp*  
**show**  $Ap * Ac = Ac * Ap$  **using** *assms*  $\langle Ac \in \text{Afp} \rangle$  **by** *simp*  
**qed**  
**thus**  $C * Bs = Bs * C$   
**using**  $\langle C = \text{mat-conj } (\text{Complex-Matrix.adjoint } Us) \ Ac \rangle$   
**by** (*metis* *mat-conj-adjoint*)  
**qed**  
**qed**  
**qed**

**lemma** *extract-dbm-eq-component-commute*:  
**assumes**  $\forall C \in Cs. C = \text{diag-block-mat } (\text{extract-subdiags } C \ l)$   
**and**  $\forall C1 \in Cs. \forall C2 \in Cs. C1 * C2 = C2 * C1$   
**and**  $ExC = (\lambda A. \text{extract-subdiags } A \ l) ' Cs$   
**and**  $j < \text{length } l$   
**and**  $Exi = (\lambda A. (A!j)) ' ExC$   
**and**  $Al \in Exi$   
**and**  $Bl \in Exi$   
**shows**  $Al * Bl = Bl * Al$   
**proof** –  
**define** *ncl* **where**  $ncl = \text{length } l$   
**have**  $\forall Al \in ExC. \text{length } Al = ncl$   
**by** (*simp* *add*: *assms* *extract-subdiags-length* *ncl-def*)  
**have**  $\exists Ea \in ExC. Al = Ea!j$  **using** *assms* **by** *auto*  
**from** *this* **obtain**  $Ea$  **where**  $Ea \in ExC$  **and**  $Al = Ea!j$  **by** *auto*  
**have**  $\exists Eb \in ExC. Bl = Eb!j$  **using** *assms* **by** *auto*  
**from** *this* **obtain**  $Eb$  **where**  $Eb \in ExC$  **and**  $Bl = Eb!j$  **by** *auto*  
**have**  $\forall j < ncl. \forall E \in ExC. E ! j \in \text{carrier-mat } (l ! j) \ (l ! j)$   
**by** (*metis* (*no-types*, *lifting*) *assms*(3) *extract-subdiags-carrier* *imageE* *ncl-def*)  
**hence**  $\forall i < ncl. \forall Al \in ExC. \text{dim-row } (Al ! i) = \text{dim-col } (Al ! i)$   
**by** (*metis* *carrier-matD*(1) *carrier-matD*(2))  
**have**  $Ea!j * Eb!j = Eb!j * Ea!j$   
**proof** (rule *diag-block-mat-commute-comp*)  
**show**  $\text{length } Ea = \text{length } Eb$   
**by** (*simp* *add*:  $\langle Ea \in ExC \rangle \langle Eb \in ExC \rangle \langle \forall Al \in ExC. \text{length } Al = ncl \rangle$ )  
**show**  $j < \text{length } Ea$

**by** (*metis*  $\langle Eb \in ExC \rangle \langle \forall Al \in ExC. \text{length } Al = ncl \rangle \langle j < \text{length } l \rangle$   
*ncl-def*  $\langle \text{length } Ea = \text{length } Eb \rangle$ )  
**show**  $\forall i < \text{length } Ea. \text{dim-row } (Ea ! i) = \text{dim-col } (Ea ! i)$   
**by** (*simp add:*  $\langle Ea \in ExC \rangle \langle \forall Al \in ExC. \text{length } Al = ncl \rangle$   
 $\langle \forall i < ncl. \forall Al \in ExC. \text{dim-row } (Al ! i) = \text{dim-col } (Al ! i) \rangle$ )  
**show**  $\forall i < \text{length } Ea. \text{dim-row } (Ea ! i) = \text{dim-row } (Eb ! i)$   
**by** (*metis*  $\langle Ea \in ExC \rangle \langle Eb \in ExC \rangle \langle \forall Al \in ExC. \text{length } Al = ncl \rangle$   
 $\langle \forall j < ncl. \forall E \in ExC. E ! j \in \text{carrier-mat } (l ! j) (l ! j) \rangle$   
*carrier-matD*(1))  
**show**  $\forall i < \text{length } Ea. \text{dim-col } (Ea ! i) = \text{dim-col } (Eb ! i)$   
**using**  $\langle Ea \in ExC \rangle \langle Eb \in ExC \rangle \langle \forall Al \in ExC. \text{length } Al = ncl \rangle$   
 $\langle \forall i < \text{length } Ea. \text{dim-row } (Ea ! i) = \text{dim-row } (Eb ! i) \rangle$   
 $\langle \forall i < ncl. \forall Al \in ExC. \text{dim-row } (Al ! i) = \text{dim-col } (Al ! i) \rangle$  **by** *auto*  
**have**  $\exists Cea \in Cs. Ea = \text{extract-subdiags } Cea l$   
**using**  $\langle Ea \in ExC \rangle$  *assms* **by** *auto*  
**from this obtain** *Cea* **where**  $Cea \in Cs$  **and**  
 $Ea = \text{extract-subdiags } Cea l$  **by** *auto*  
**hence** *cea*:  $Cea = \text{diag-block-mat } Ea$   
**by** (*simp add:*  $\langle \forall C \in Cs. C =$   
 $\text{diag-block-mat } (\text{extract-subdiags } C l) \rangle$ )  
**have**  $\exists Ceb \in Cs. Eb = \text{extract-subdiags } Ceb l$   
**using**  $\langle Eb \in ExC \rangle$  *assms* **by** *auto*  
**from this obtain** *Ceb* **where**  $Ceb \in Cs$  **and**  
 $Eb = \text{extract-subdiags } Ceb l$  **by** *auto*  
**hence**  $Ceb = \text{diag-block-mat } Eb$   
**by** (*simp add:*  $\langle \forall C \in Cs. C =$   
 $\text{diag-block-mat } (\text{extract-subdiags } C l) \rangle$ )  
**moreover have**  $Cea * Ceb = Ceb * Cea$   
**by** (*simp add:*  $\langle Cea \in Cs \rangle \langle Ceb \in Cs \rangle$   
 $\langle \forall C1 \in Cs. \forall C2 \in Cs. C1 * C2 = C2 * C1 \rangle$ )  
**ultimately show**  $\text{diag-block-mat } Ea * \text{diag-block-mat } Eb =$   
 $\text{diag-block-mat } Eb * \text{diag-block-mat } Ea$  **using** *cea* **by** *simp*  
**qed**  
**thus**  $Al * Bl = Bl * Al$  **using**  $\langle Al = Ea ! j \rangle \langle Bl = Eb ! j \rangle$  **by** *simp*  
**qed**

**lemma** *extract-comm-real-diag-decomp*:

**fixes**  $CjA::\text{complex Matrix.mat set}$   
**assumes**  $\bigwedge (Af::\text{complex Matrix.mat set}) n . \text{finite } Af \implies$   
 $\text{card } Af \leq i \implies$   
 $Af \neq \{\}$   $\implies$   
 $(\bigwedge A. A \in Af \implies A \in \text{carrier-mat } n n) \implies$   
 $0 < n \implies (\bigwedge A. A \in Af \implies \text{hermitian } A) \implies$   
 $(\bigwedge A B. A \in Af \implies B \in Af \implies A * B = B * A) \implies$   
 $\exists U. \forall A \in Af. \exists B. \text{real-diag-decomp } A B U$   
**and** *finite*  $CjA$   
**and**  $CjA \neq \{\}$   
**and**  $\text{card } CjA \leq i$   
**and**  $\forall C \in CjA. C = \text{diag-block-mat } (\text{extract-subdiags } C \text{ eqcl})$

**and**  $\forall C1 \in CjA. \forall C2 \in CjA. C1 * C2 = C2 * C1$   
**and**  $Exc = (\lambda A. extract-subdiags A eqcl) \text{ `CjA}$   
**and**  $\forall E \in Exc. list-all (\lambda B. 0 < dim-row B \wedge hermitian B) E$   
**and**  $\forall i < length eqcl. 0 < eqcl!i$   
**shows**  $\forall i < length eqcl. \exists U. \forall Al \in Exc. \exists B. real-diag-decomp (Al ! i) B U$   
**proof** (*intro allI impI*)  
**define**  $ncl$  **where**  $ncl = length eqcl$   
**fix**  $j$   
**assume**  $j < ncl$   
**define**  $Exi$  **where**  $Exi = (\lambda l. l!j) \text{ `Exc}$   
**have** *finite Exi* **using** *assms unfolding Exi-def* **by** *simp*  
**have**  $card Exi \leq i$  **using** *assms unfolding Exi-def*  
**by** (*metis card-image-le image-image le-trans*)  
**have** *exfl*:  $\forall Ej \in Exi. \exists Fj \in CjA. Ej = (extract-subdiags Fj eqcl)!j$   
**proof**  
**fix**  $Ej$   
**assume**  $Ej \in Exi$   
**hence**  $\exists El \in Exc. Ej = El!j$  **unfolding** *Exi-def* **by** *auto*  
**from this** **obtain**  $El$  **where**  $El \in Exc$  **and**  $Ej = El!j$  **by** *auto*  
**hence**  $\exists Fl \in CjA. El = extract-subdiags Fl eqcl$   
**using** *assms* **by** *auto*  
**from this** **obtain**  $Fl$  **where**  $Fl \in CjA$  **and**  
 $El = extract-subdiags Fl eqcl$  **by** *auto*  
**thus**  $\exists Fj \in CjA. Ej = (extract-subdiags Fj eqcl)!j$  **using**  $\langle Ej = El!j \rangle$   
**by** *auto*  
**qed**  
**have**  $\exists U. \forall Al \in Exi. \exists B. real-diag-decomp (Al) B U$   
**proof** (*rule assms(1)*)  
**show** *finite Exi* **using**  $\langle finite Exi \rangle$  .  
**show**  $card Exi \leq i$  **using**  $\langle card Exi \leq i \rangle$  .  
**show**  $Exi \neq \{\}$  **using**  $\langle CjA \neq \{\} \rangle$  **using** *assms unfolding Exi-def* **by** *auto*  
**show**  $0 < eqcl!j$  **using**  $\langle j < ncl \rangle$  *ncl-def* *assms* **by** *simp*  
**show**  $\bigwedge Al. Al \in Exi \implies Al \in carrier-mat (eqcl ! j) (eqcl ! j)$   
**proof** –  
**fix**  $Al$   
**assume**  $Al \in Exi$   
**hence**  $\exists Fl \in CjA. Al = (extract-subdiags Fl eqcl)!j$  **using** *exfl*  
**by** *simp*  
**from this** **obtain**  $Fl$  **where**  $Fl \in CjA$  **and**  
 $Al = (extract-subdiags Fl eqcl)!j$  **by** *auto*  
**thus**  $Al \in carrier-mat (eqcl ! j) (eqcl ! j)$   
**using** *extract-subdiags-carrier[of j eqcl]*  $\langle j < ncl \rangle$   
**unfolding** *Exi-def* *ncl-def* **by** *simp*  
**qed**  
**show**  $\bigwedge Al. Al \in Exi \implies hermitian Al$   
**proof** –  
**fix**  $Al$   
**assume**  $Al \in Exi$   
**hence**  $\exists El \in Exc. Al = El!j$  **unfolding** *Exi-def* **by** *auto*

```

from this obtain  $El$  where  $El \in Exc$  and  $Al = El!j$  by auto
thus hermitian  $Al$  using assms  $\langle j < ncl \rangle$  ncl-def
  by (metis (no-types, lifting) extract-subdiags-length image-iff
    list-all-length)
qed
show  $\bigwedge Al Bl. Al \in Exi \implies Bl \in Exi \implies Al * Bl = Bl * Al$ 
proof –
  fix  $Al Bl$ 
  assume  $Al \in Exi$  and  $Bl \in Exi$ 
  show  $Al * Bl = Bl * Al$ 
  proof (rule extract-dbm-eq-component-commute[of CjA eqcl])
    show  $Al \in Exi$  using  $\langle Al \in Exi \rangle$  .
    show  $Bl \in Exi$  using  $\langle Bl \in Exi \rangle$  .
    show  $\forall C \in CjA. C = \text{diag-block-mat } (extract-subdiags C eqcl)$ 
      using  $\langle \forall C \in CjA. C = \text{diag-block-mat } (extract-subdiags C eqcl) \rangle$  .
    show  $\forall C1 \in CjA. \forall C2 \in CjA. C1 * C2 = C2 * C1$ 
      using  $\langle \forall C1 \in CjA. \forall C2 \in CjA. C1 * C2 = C2 * C1 \rangle$  .
    show  $j < \text{length } eqcl$  using  $\langle j < ncl \rangle$  ncl-def by simp
    show  $Exi = (\lambda A. A ! j)$  ‘Exc’ using Exi-def by simp
    show  $Exc = (\lambda A. \text{extract-subdiags } A eqcl)$  ‘CjA’
      using assms by simp
  qed
qed
qed
thus  $\exists U. \forall Al \in Exc. \exists B. \text{real-diag-decomp } (Al ! j) B U$  unfolding Exi-def
  by simp
qed

```

## 8.2 The main result

```

theorem commuting-hermitian-family-diag:
  fixes  $Af :: \text{complex Matrix.mat set}$ 
  assumes finite Af
  and  $Af \neq \{\}$ 
  and  $\bigwedge A. A \in Af \implies A \in \text{carrier-mat } n n$ 
  and  $0 < n$ 
  and  $\bigwedge A. A \in Af \implies \text{hermitian } A$ 
  and  $\bigwedge A B. A \in Af \implies B \in Af \implies A * B = B * A$ 
shows  $\exists U. \forall A \in Af. \exists B. \text{real-diag-decomp } A B U$  using assms
proof –
  define  $i$  where  $i = \text{card } Af$ 
  have  $\text{card } Af \leq i$ 
    by (simp add: i-def)
  from assms(1) this assms(2–) show ?thesis
proof (induct i arbitrary: Af n)
  case 0
    then have  $Af = \{\}$  by simp
    then show ?case using 0 by simp
  next

```

```

case (Suc i)
hence  $\exists A. A \in Af$  by blast
from this obtain Ap where  $Ap \in Af$  by auto
define Afp where  $Afp = Af - \{Ap\}$ 
have finite Afp using Suc unfolding Afp-def by simp
have  $\text{card } Afp \leq i$  using  $\langle \text{card } Af \leq \text{Suc } i \rangle \langle Ap \in Af \rangle$ 
  unfolding Afp-def by simp
have  $\forall A \in Afp. \text{hermitian } A \wedge A \in \text{carrier-mat } n \ n$  using Suc
  by (metis Afp-def Diff-subset subset-iff)
have  $\forall A \in Afp. \forall B \in Afp. A * B = B * A$  using Suc
  by (metis Afp-def Diff-subset subset-iff)
have  $\forall A \in Afp. Ap * A = A * Ap$  using Suc
  by (simp add: Afp-def  $\langle Ap \in Af \rangle$ )
have hermitian Ap  $Ap \in \text{carrier-mat } n \ n$   $0 < n$  using  $\langle Ap \in Af \rangle$  Suc by auto
from this obtain Bs Us where rd:  $\text{real-diag-decomp } Ap \ Bs \ Us \wedge$ 
  sorted (map Re (diag-mat Bs))
  using hermitian-real-diag-sorted[of Ap] by auto note ub = this
note ubprops =  $\text{real-diag-decomp-eq-comps-props}[OF \langle Ap \in \text{carrier-mat } n \ n \rangle$ 
 $\langle 0 < n \rangle \text{ ub}]$ 
define eqcl where  $\text{eqcl} = \text{eq-comps } (\text{diag-mat } Bs)$ 
have  $\text{diag-diff } Bs \ \text{eqcl}$  using ubprops unfolding eqcl-def by simp
have  $\text{eqcl} \neq []$  using ubprops unfolding eqcl-def by simp
hence  $\text{eqcl} = \text{hd } \text{eqcl} \# (\text{tl } \text{eqcl})$  by simp
define esubB where  $\text{esubB} = \text{extract-subdiags } Bs \ \text{eqcl}$ 
have ebcars:  $\forall i < \text{length } \text{esubB}. \text{esubB } ! \ i \in \text{carrier-mat } (\text{eqcl}!i) \ (\text{eqcl}!i)$ 
  using  $\text{extract-subdiags-carrier}[of - \text{eqcl } Bs]$ 
  by (simp add: esubB-def  $\text{extract-subdiags-length}$ )
have  $Bs = \text{diag-block-mat } \text{esubB}$  unfolding esubB-def eqcl-def
proof (rule diagonal-extract-eq)
  show  $Bs \in \text{carrier-mat } n \ n$  using  $\langle Bs \in \text{carrier-mat } n \ n \rangle$  .
  show  $\text{diagonal-mat } Bs$  using ubprops  $\text{real-diag-decompD}(2)$ 
     $\text{real-diag-decompD}(1)$   $\text{unitary-diagD}(2)$  by blast
qed
show ?case
proof (cases  $Afp = \{\}$ )
  case True
  hence  $Af = \{Ap\}$  using  $\langle Afp = Af - \{Ap\} \rangle$ 
    by (simp add:  $\text{Suc}(4)$   $\text{subset-singleton-iff}$ )
  then show ?thesis using rd  $\langle Af = \{Ap\} \rangle$  by auto
next
  case False
  define Cj where  $Cj = (\lambda A2. \text{mat-conj } (\text{Complex-Matrix.adjoint } Us) \ A2)$ 
  define CjA where  $CjA = Cj \ Afp$ 
  have  $CjA = (\lambda A2. (\text{mat-conj } (\text{Complex-Matrix.adjoint } Us) \ A2)) \ Afp$ 
    using CjA-def Cj-def by simp
  note cjprops =  $\text{commuting-conj-mat-set-props}[OF \langle \text{finite } Afp \rangle \langle \text{card } Afp \leq i \rangle$ 
 $\langle \forall A \in Afp. \text{hermitian } A \wedge A \in \text{carrier-mat } n \ n \rangle$ 
 $\langle \forall A \in Afp. \forall B \in Afp. A * B = B * A \rangle$ 
 $\langle \text{unitary } Us \rangle \langle Us \in \text{carrier-mat } n \ n \rangle$ 

```

```

  ⟨CjA = (λA2. mat-conj (Complex-Matrix.adjoint Us) A2) ‘Afp’]
have ∀ C ∈ CjA. C = diag-block-mat (extract-subdiags C eqcl)
proof (rule commute-extract-diag-block-eq[OF ⟨Ap ∈ carrier-mat n n⟩
  ⟨0 < n⟩ rd ⟨finite Afp⟩ -
  ⟨∀ A ∈ Afp. hermitian A ∧ A ∈ carrier-mat n n⟩],
  auto simp add: eqcl-def CjA-def Cj-def)
show ∧ A B. A ∈ Afp ⇒ B ∈ Afp ⇒ A * B = B * A
  by (simp add: ⟨∀ A ∈ Afp. ∀ B ∈ Afp. A * B = B * A⟩)
show ∧ A. A ∈ Afp ⇒ Ap * A = A * Ap
  using ⟨∀ A ∈ Afp. Ap * A = A * Ap⟩ by simp
qed
define Ex where Ex = (λA. extract-subdiags A eqcl)‘CjA
have finite Ex using ⟨finite CjA⟩ unfolding Ex-def by simp
have Ex ≠ {} using False unfolding Ex-def CjA-def by simp
have card Ex ≤ i using ⟨card CjA ≤ i⟩ unfolding Ex-def
  by (metis ⟨finite CjA⟩ basic-trans-rules(23) card-image-le)
have exall: ∀ E ∈ Ex. list-all (λB. 0 < dim-row B ∧ hermitian B) E
proof
  fix E
  assume E ∈ Ex
hence ∃ nA ∈ CjA. E = extract-subdiags nA eqcl unfolding Ex-def by auto
from this obtain nA where nA ∈ CjA and E = extract-subdiags nA eqcl
  by auto
have list-all (λB. 0 < dim-row B ∧ hermitian B)
  (extract-subdiags nA eqcl)
proof (rule hermitian-extract-subdiags)
  show hermitian nA using ⟨∀ A ∈ CjA. A ∈ carrier-mat n n ∧ hermitian
A⟩
  ⟨nA ∈ CjA⟩ by simp
show list-all ((< 0) eqcl) unfolding eqcl-def
  by (metis ⟨eqcl ≠ []⟩ eq-comps.simps(1) eq-comps-gt-0 eqcl-def)
show sum-list eqcl ≤ dim-row nA
  using ⟨∀ A ∈ CjA. A ∈ carrier-mat n n ∧ hermitian A⟩
  ⟨nA ∈ CjA⟩ unfolding eqcl-def
  by (metis ⟨Bs ∈ carrier-mat n n⟩ carrier-matD(1)
  eq-comp-sum-diag-mat le-refl)
qed
thus list-all (λB. 0 < dim-row B ∧ hermitian B) E
  using ⟨E = extract-subdiags nA eqcl⟩ by simp
qed
define ncl where ncl = length eqcl
have ∀ j < ncl. ∀ E ∈ Ex. E!j ∈ carrier-mat (eqcl!j) (eqcl!j)
proof (intro allI impI ballI)
  fix E j
  assume j < ncl and E ∈ Ex
  thus E!j ∈ carrier-mat (eqcl!j) (eqcl!j) unfolding Ex-def
  using extract-subdiags-carrier ncl-def by blast
qed
have ∃ Ul. (length Ul = ncl ∧

```

$(\forall i < ncl. \forall Al \in Ex.$   
 $(dim\text{-row } (Ul!i) = dim\text{-row } (Al!i) \wedge dim\text{-col } (Ul!i) = dim\text{-col } (Al!i))) \wedge$   
 $(\forall Al \in Ex. \exists Bl. (length\ Bl = ncl \wedge$   
 $real\text{-diag}\text{-decomp } (diag\text{-block}\text{-mat } Al) (diag\text{-block}\text{-mat } Bl)$   
 $(diag\text{-block}\text{-mat } Ul))))$   
**proof** (rule real-diag-decomp-block-set)  
**show**  $Ex \neq \{\}$  **using**  $\langle Afp \neq \{\} \rangle$  **unfolding**  $Ex\text{-def } CjA\text{-def}$  **by** *auto*  
**show**  $0 < ncl$  **unfolding**  $ncl\text{-def}$  **using**  $\langle eqcl \neq [] \rangle$  **by** *simp*  
**show**  $\forall Al \in Ex. length\ Al = ncl$  **unfolding**  $ncl\text{-def } Ex\text{-def}$   
**by** (*simp add: extract-subdiags-length*)  
**show**  $\forall i < ncl. \forall Al \in Ex. dim\text{-row } (Al!i) = dim\text{-col } (Al!i)$   
**proof** (*intro allI impI ballI*)  
**fix**  $i\ Al$   
**assume**  $i < ncl$  **and**  $Al \in Ex$   
**thus**  $dim\text{-row } (Al!i) = dim\text{-col } (Al!i)$  **using** *exall*  
**by** (*metis (mono-tags, lifting)  $\langle \forall Al \in Ex. length\ Al = ncl \rangle$*   
*carrier-matD(2) hermitian-square list-all-length*)  
**qed**  
**show**  $\forall i < ncl. \exists U. \forall Al \in Ex. \exists B. real\text{-diag}\text{-decomp } (Al!i) B U$  **unfolding**  
*ncl-def*  
**proof** (rule *extract-comm-real-diag-decomp[of i CjA, OF Suc(1)]*),  
*auto simp add: exall Ex-def*)  
**show** *finite CjA* **using**  $\langle finite\ CjA \rangle$  .  
**show**  $card\ CjA \leq i$  **using**  $\langle card\ CjA \leq i \rangle$  .  
**show**  $\bigwedge C. C \in CjA \implies C = diag\text{-block}\text{-mat } (extract\text{-subdiags } C\ eqcl)$   
**using**  $\langle \forall C \in CjA. C = diag\text{-block}\text{-mat } (extract\text{-subdiags } C\ eqcl) \rangle$  **by**  
*simp*  
**show**  $\bigwedge C1\ C2. C1 \in CjA \implies C2 \in CjA \implies C1 * C2 = C2 * C1$   
**using** *cjprops* **by** *simp*  
**show**  $\bigwedge i. i < length\ eqcl \implies 0 < eqcl!i$   
**proof** –  
**fix**  $il$   
**assume**  $il < length\ eqcl$   
**thus**  $0 < eqcl!il$  **using** *eq-comps-gt-0[OF  $\langle diag\text{-mat } Bs \neq [] \rangle$*   
*list-all-length[of (<) 0 eq-comps (diag-mat Bs)]*  
**unfolding** *eqcl-def* **by** *simp*  
**qed**  
**show**  $CjA = \{\} \implies False$  **by** (*simp add: CjA-def False*)  
**qed**  
**qed**  
**from this obtain**  $Ul$  **where**  $length\ Ul = ncl$  **and**  
 $dimul: (\forall i < ncl. \forall Al \in Ex.$   
 $(dim\text{-row } (Ul!i) = dim\text{-row } (Al!i) \wedge dim\text{-col } (Ul!i) = dim\text{-col } (Al!i)))$  **and**  
 $ul: \forall Al \in Ex. \exists Bl. (length\ Bl = ncl \wedge$   
 $real\text{-diag}\text{-decomp } (diag\text{-block}\text{-mat } Al) (diag\text{-block}\text{-mat } Bl)$   
 $(diag\text{-block}\text{-mat } Ul))$   
**by** *auto*  
**define**  $Uf$  **where**  $Uf = Us * (diag\text{-block}\text{-mat } Ul)$   
**have** *afp*:  $\forall A \in Afp. \exists Bl. real\text{-diag}\text{-decomp } A (diag\text{-block}\text{-mat } Bl) Uf$

```

proof
  fix A
  assume A ∈ Afp
  define Ca where Ca = mat-conj (Complex-Matrix.adjoint Us) A
  define Eca where Eca = extract-subdiags Ca eqcl
  have Ca ∈ CjA using ⟨A ∈ Afp⟩
    unfolding Ca-def CjA-def Cj-def by simp
  hence Ca = diag-block-mat Eca unfolding Eca-def
    using ⟨∀ C ∈ CjA. C = diag-block-mat (extract-subdiags C eqcl)⟩ by simp
  have Eca ∈ Ex unfolding Ex-def Eca-def using ⟨Ca ∈ CjA⟩ by simp
  hence ∃ Bl. (length Bl = ncl ∧
    real-diag-decomp (diag-block-mat Eca) (diag-block-mat Bl)
    (diag-block-mat Ul)) using ul by simp
  from this obtain Ecb where length Ecb = ncl and
    real-diag-decomp (diag-block-mat Eca) (diag-block-mat Ecb)
    (diag-block-mat Ul) by auto
  hence real-diag-decomp Ca (diag-block-mat Ecb)
    (diag-block-mat Ul) using ⟨Ca = diag-block-mat Eca⟩ by simp
  have real-diag-decomp A (diag-block-mat Ecb) Uf unfolding Uf-def
  proof (rule unitary-conjugate-real-diag-decomp)
    show A ∈ carrier-mat n n using ⟨A ∈ Afp⟩ unfolding Afp-def
      by (simp add: Suc(5))
    show Us ∈ carrier-mat n n using ⟨Us ∈ carrier-mat n n⟩ .
    show unitary Us using ⟨unitary Us⟩ .
    show real-diag-decomp (mat-conj (Complex-Matrix.adjoint Us) A)
      (diag-block-mat Ecb) (diag-block-mat Ul)
      using ⟨real-diag-decomp Ca (diag-block-mat Ecb)
        (diag-block-mat Ul)⟩ unfolding Ca-def by simp
  qed
  thus ∃ Bl. real-diag-decomp A (diag-block-mat Bl) Uf by blast
qed
have real-diag-decomp Ap Bs Uf unfolding Uf-def
proof (rule real-diag-decomp-mult-dbm-unit)
  show Ap ∈ carrier-mat n n using ⟨Ap ∈ carrier-mat n n⟩ .
  show real-diag-decomp Ap Bs Us using ub by simp
  show Bs = diag-block-mat esubB using ⟨Bs = diag-block-mat esubB⟩ .
  show length Ul = length esubB using ⟨length Ul = ncl⟩
    by (simp add: esubB-def extract-subdiags-length ncl-def)
  show ∀ i < length esubB. dim-col (esubB ! i) = dim-row (esubB ! i)
    by (metis carrier-matD(1) carrier-matD(2) ebcarr)
  have length esubB = ncl using ⟨length Ul = length esubB⟩
    ⟨length Ul = ncl⟩ ncl-def by auto
  show roweq: ∀ i < length esubB. dim-row (esubB ! i) = dim-row (Ul ! i)
    using ebcarr dimul ⟨length esubB = ncl⟩ ⟨Ex ≠ {}⟩
    ⟨∀ j < ncl. ∀ E ∈ Ex. E ! j ∈ carrier-mat (eqcl ! j) (eqcl ! j)⟩
    by (metis all-not-in-conv carrier-matD(1))
  show coleq: ∀ i < length esubB. dim-col (esubB ! i) = dim-col (Ul ! i)
    using ebcarr dimul ⟨length esubB = ncl⟩ ⟨Ex ≠ {}⟩
    ⟨∀ j < ncl. ∀ E ∈ Ex. E ! j ∈ carrier-mat (eqcl ! j) (eqcl ! j)⟩

```

```

    by (metis all-not-in-conv carrier-matD(2))
  show unitary (diag-block-mat Ul) using ul
    by (metis CjA-def False all-not-in-conv image-is-empty Ex-def
      real-diag-decompD(1) unitary-diagD(3))
  show  $\forall i < \text{length } Ul. Ul ! i * \text{esubB} ! i = \text{esubB} ! i * Ul ! i$ 
  proof (intro allI impI)
    fix i
    assume  $i < \text{length } Ul$ 
    show  $Ul ! i * \text{esubB} ! i = \text{esubB} ! i * Ul ! i$ 
      unfolding esubB-def eqcl-def
    proof (rule extract-subdiags-comp-commute[symmetric])
      show diagonal-mat Bs using  $\langle \text{diagonal-mat } Bs \rangle$  .
      show  $Bs \in \text{carrier-mat } n \ n$  using  $\langle Bs \in \text{carrier-mat } n \ n \rangle$  .
      show  $0 < n$  using  $\langle 0 < n \rangle$  .
      show  $i < \text{length } (eq\text{-comps } (diag\text{-mat } Bs))$ 
        using  $\langle i < \text{length } Ul \rangle \langle \text{length } Ul = \text{length } \text{esubB} \rangle$ 
        extract-subdiags-length
      unfolding esubB-def eqcl-def by metis
    show  $Ul ! i \in \text{carrier-mat } (eq\text{-comps } (diag\text{-mat } Bs) ! i)$ 
      (eq-comps (diag-mat Bs) ! i)
      using dimul  $\langle \forall j < ncl. \forall E \in Ex. E ! j \in \text{carrier-mat } (eqcl!j) (eqcl!j) \rangle$ 
       $\langle Ex \neq \{\} \rangle$  unfolding ncl-def eqcl-def
    by (metis coleq roweq
       $\langle \forall i < \text{length } \text{esubB}. \text{dim-col } (\text{esubB} ! i) = \text{dim-row } (\text{esubB} ! i) \rangle$ 
       $\langle i < \text{length } Ul \rangle \langle \text{length } Ul = \text{length } \text{esubB} \rangle$  carrier-matD(2)
      carrier-matI ebcarr eqcl-def)
  qed
qed
qed
hence  $\exists B. \text{real-diag-decomp } Ap \ B \ Uf$  by blast
hence  $\forall A \in Af. \exists B. \text{real-diag-decomp } A \ B \ Uf$  using afp
  unfolding Afp-def by auto
thus  $\exists U. \forall A \in Af. \exists B. \text{real-diag-decomp } A \ B \ U$  by blast
qed
qed
qed
end

```