Simultaneous diagonalization of pairwise commuting Hermitian matrices

Mnacho Echenim

March 24, 2023

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 $\{A_1, \ldots, A_n\}$ 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 $i = 1, \ldots, n$, 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

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 *poly* $(\prod a{\leftarrow}L.\ [:-\ a,\ 1:])\ (c::'a :: field) = 0 \Longrightarrow c \in set\ L$
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 *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 ... = 0_v *n* + *w* **using** *assms* **by** *simp*
also have ... = *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*) ∈ *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* ∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
and *C*∈ *carrier-mat n n*
and *v*∈ *carrier-vec n*
shows *A* ∗ *B* ∗ *C* ∗$_v$ *v* = *A* ∗$_v$ (*B* ∗$_v$ (*C* ∗$_v$ *v*)) **using** *assms*
by (*smt* (*z3*) *assoc-mult-mat-vec mult-carrier-mat mult-mat-vec-carrier*)

lemma *adjoint-dim'*:
A ∈ *carrier-mat n m* ⟹ *Complex-Matrix.adjoint A* ∈ *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 ... = *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∈ carrier-mat n n
  and U∈ carrier-mat n n
shows mat-conj U A = A
proof −
  have mat-conj U A = A*U * Complex-Matrix.adjoint U using assms
    unfolding mat-conj-def by simp
  also have ... = A * (U * Complex-Matrix.adjoint U)
  proof (rule assoc-mult-mat, auto simp add: assms)
    show U ∈ carrier-mat (dim-col A) (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∈ carrier-mat n n
  and U ∈ carrier-mat n n
  and hermitian A
shows hermitian (mat-conj U A)
proof −
  have Complex-Matrix.adjoint (U * A * Complex-Matrix.adjoint U) =
    U * 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 * ((Complex-Matrix.adjoint A) *  Complex-Matrix.adjoint U)
    by (metis adjoint-mult assms(1) assms(2))
  also have ... = U * A * 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∈ carrier-mat n n
  and U ∈ 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 $A \in$ *carrier-mat n n*
and $U \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
and $A =$ *mat-conj U B*
shows $-A =$ *mat-conj U* $(-B)$ **using** *assms* **unfolding** *mat-conj-def* **by** *auto*

lemma *mat-conj-smult*:
assumes $A \in$ *carrier-mat n n*
and $U \in$ *carrier-mat n n*
and $B \in$ *carrier-mat n n*
and $A = U * B * ($*Complex-Matrix.adjoint U*$)$
shows $x \cdot_m A = U * (x \cdot_m B) * ($*Complex-Matrix.adjoint U*$)$ **using** *assms mult-smult-distrib*
by (*smt* (*z3*) *adjoint-dim' mult-carrier-mat mult-smult-assoc-mat*)

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

lemma *hermitian-square-hermitian*:
fixes $A::'a::$*conjugatable-field Matrix.mat*
assumes *hermitian A*
shows *hermitian* $(A * A)$
proof −
have *Complex-Matrix.adjoint* $(A * A) =$ *Complex-Matrix.adjoint A* $*$ (*Complex-Matrix.adjoint A*)
using *adjoint-mult* **by** (*metis assms hermitian-square*)
also have ... = $A * A$ **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 $A::'a::\{ring\}$ *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 $A::'a::\{ring\}$ *Matrix.mat*
assumes *diagonal-mat A*
shows *diagonal-mat* $(x \cdot_m A)$ **using** *assms* **unfolding** *diagonal-mat-def uminus-mat-def* **by** *auto*

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

lemma *set-diag-mat-uminus*:
assumes $A\in$ *carrier-mat n n*
shows *set* (*diag-mat* $(-A)$) = $\{-a \mid a.\ a\in$ *set* (*diag-mat A*)$\}$ (**is** *?L = ?R*)
proof
show *?L* $\subseteq$ *?R*
proof
fix *x*
assume $x \in$ *set* (*diag-mat* $(-A)$)
hence $\exists i <$ *length* (*diag-mat* $(-A)$). *nth* (*diag-mat* $(-A)$) $i = x$
using *in-set-conv-nth*[*of x*] **by** *simp*
from *this* **obtain** *i* **where** $i <$ *length* (*diag-mat* $(-A)$) **and** *nth* (*diag-mat* $(-A)$) $i = x$
by *auto* **note** *iprop* = *this*
hence $i <$ *dim-row* $(-A)$ **unfolding** *diag-mat-def* **by** *simp*
hence $i < n$ **using** *assms* **by** *simp*
have $x = (-A)\$\$(i,i)$ **using** *iprop* **unfolding** *diag-mat-def* **by** *simp*
also have ... = $-$ *A$$(i,i)* **using** ‹$i < n$› *assms* **unfolding** *uminus-mat-def*
by *auto*
also have ... $\in$ *?R* **using** *iprop assms* ‹$i < n$›
in-set-conv-nth[*of A$$(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∈ ?R
    hence ∃ i < length (diag-mat A). −(nth (diag-mat A)  i) = x
      using in-set-conv-nth[of x] by (smt (z3) in-set-conv-nth mem-Collect-eq)
    from this obtain i where i < length (diag-mat A) and −(nth (diag-mat A)
i) = x
      by auto note iprop = this
    hence i < dim-row (−A) unfolding diag-mat-def by simp
    hence i < n using assms by simp
    have x = −A$$(i,i) using iprop unfolding diag-mat-def by simp
    also have ... = (− A)$$(i,i) using ‹i < n› assms unfolding uminus-mat-def
by auto
    also have ... ∈ ?L using iprop assms ‹i < n›
        in-set-conv-nth[of A$$(i,i)]
      by (metis ‹i < dim-row (− A)› diag-elems-mem diag-elems-set-diag-mat)
    finally show x ∈ ?L .
  qed
qed

lemma set-diag-mat-smult:
  assumes A∈ carrier-mat n n
  shows set (diag-mat (x ·m A)) = {x ∗ a |a. a∈ set (diag-mat A)} (is ?L = ?R)
proof
  show ?L ⊆ ?R
  proof
    fix b
    assume b ∈ set (diag-mat (x ·m A))
    hence ∃ i < length (diag-mat (x ·m A)). nth (diag-mat (x ·m A))  i = b
      using in-set-conv-nth[of b] by simp
    from this obtain i where i < length (diag-mat (x ·m A)) and nth (diag-mat
(x ·m A))  i = b
      by auto note iprop = this
    hence i < dim-row (x ·m A) unfolding diag-mat-def by simp
    hence i < n using assms by simp
    have b = (x ·m A)$$(i,i) using iprop unfolding diag-mat-def by simp
    also have ... = x ∗ A$$(i,i) using ‹i < n› assms unfolding uminus-mat-def
by auto
    also have ... ∈ ?R using iprop assms ‹i < n›
        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 b ∈ ?R .
  qed
next
  show ?R ⊆ ?L
  proof
    fix b
```

assume *b∈ ?R*
hence ∃ *i* < *length* (*diag-mat A*). *x* ∗ (*nth* (*diag-mat A*) *i*) = *b*
using *in-set-conv-nth*[*of x*] **by** (*smt* (*z3*) *in-set-conv-nth mem-Collect-eq*)
from *this* **obtain** *i* **where** *i* < *length* (*diag-mat A*) **and** *x* ∗ (*nth* (*diag-mat A*) *i*) = *b*
by *auto* **note** *iprop* = *this*
hence *i* < *dim-row* (*x* ·$_m$ *A*) **unfolding** *diag-mat-def* **by** *simp*
hence *i* < *n* **using** *assms* **by** *simp*
have *b* = *x* ∗*A*$$(*i*,*i*) **using** *iprop* **unfolding** *diag-mat-def* **by** *simp*
also have ... = (*x* ·$_m$ *A*)$$(*i*,*i*) **using** ‹*i* < *n*› *assms* **unfolding** *uminus-mat-def* **by** *auto*
also have ... ∈ *?L* **using** *iprop assms* ‹*i* < *n*›
in-set-conv-nth[*of A*$$(*i*,*i*)]
by (*metis* ‹*i* < *dim-row* (*x* ·$_m$ *A*)› *diag-elems-mem diag-elems-set-diag-mat*)
finally show *b* ∈ *?L* **.**
qed
qed

lemma *diag-mat-diagonal-eq*:
assumes *diag-mat A* = *diag-mat B*
and *diagonal-mat A*
and *diagonal-mat B*
and *dim-col A* = *dim-col B*
shows *A* = *B*
proof
show *c*: *dim-col A* = *dim-col B* **using** *assms* **by** *simp*
show *r*: *dim-row A* = *dim-row B* **using** *assms* **unfolding** *diag-mat-def*
proof −
assume *map* (λ*i*. *A* $$ (*i*, *i*)) [*0*..<*dim-row A*] = *map* (λ*i*. *B* $$ (*i*, *i*)) [*0*..<*dim-row B*]
then show *?thesis*
by (*metis* (*lifting*) *length-map length-upt verit-minus-simplify*(*2*))
qed
fix *i j*
assume *i* < *dim-row B* **and** *j* < *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*: ‹*dim-row A* = *dim-row B*› ‹*i* ≠ *j*› ‹*i* < *dim-row B*› ‹*j* < *dim-col B*›)
next
case *True*
hence *A* $$ (*i*,*j*) = *A* $$ (*i*,*i*) **by** *simp*
also have ... = (*diag-mat A*)!*i* **using** *c r* ‹*i* < *dim-row B*› **unfolding** *diag-mat-def* **by** *simp*
also have ... = (*diag-mat B*)!*i* **using** *assms* **by** *simp*
also have ... = *B* $$(*i*,*i*) **using** *c r* ‹*i* < *dim-row B*› **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 $B \in$ *carrier-mat n n*
and $0 < n$
shows *diag-elems* $B \neq \{\}$
proof −
have B $$(0,0) \in$ *diag-elems B* **using** *assms* **by** *simp*
thus *?thesis* **by** *auto*
qed

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

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

```
  and B ∈ carrier-mat n n
  and i < n
  and j < n
  shows (A * B) $$ (i,j) = A$$(i,i) * B$$(i,j) unfolding diagonal-mat-def
proof −
  have dim-row (A * B) = n using assms by simp
  have dim-col (A * B) = n using  assms by simp
  have jvec: ⋀j. j < n ⟹ dim-vec (Matrix.col B j) = n using assms by simp
  have (A * B) $$ (i,j) = Matrix.scalar-prod (Matrix.row A i) (Matrix.col B j)
    using assms by (metis carrier-matD(1) carrier-matD(2) index-mult-mat(1))
  also have ... =
    (∑ k ∈ {0 ..< n}. vec-index (Matrix.row A i) k *
      vec-index (Matrix.col B j) k)
    using assms  jvec unfolding Matrix.scalar-prod-def by simp
  also have ... = vec-index (Matrix.row A i) i * vec-index (Matrix.col B j) i
  proof (rule sum-but-one)
    show i < n using assms ‹dim-row (A * B) = n› by simp
    show ∀ k<n. k ≠ i ⟶ vec-index (Matrix.row A i) k = 0 using assms ‹i <
n›
      unfolding diagonal-mat-def by auto
  qed
  also have ... = 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::comm-ring Matrix.mat
  assumes A ∈ carrier-mat n n
and B∈ 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) = Matrix.scalar-prod (Matrix.row A i) (Matrix.col B j)
using assms
      times-mat-def[of A] by simp
  also have ... = Matrix.scalar-prod (Matrix.col B j) (Matrix.row A i)
    using comm-scalar-prod[of Matrix.row A i n] assms by auto
 also have ... = (Matrix.vec-index (Matrix.col B j) j) * (Matrix.vec-index (Matrix.row
A i) j)
    unfolding Matrix.scalar-prod-def
  proof (rule sum-but-one)
    show j < dim-vec (Matrix.row A i) using assms by simp
    show ∀ ia<dim-vec (Matrix.row A i). ia ≠ j ⟶ Matrix.vec-index (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*∈ *carrier-mat n n*
and *B*∈ *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* < *dim-row* (*A* * *B*) **and** *j* < *dim-col* (*A* * *B*) **and** *i* ≠ *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*::{*comm-ring*} *Matrix.mat*
assumes *A*∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
and *diagonal-mat A*
and *diagonal-mat B*
shows *A* * *B* = *B* * *A*
proof (*rule eq-matI*)
show *dim-row* (*A* * *B*) = *dim-row* (*B* * *A*) **using** *assms* **by** *simp*
show *dim-col* (*A* * *B*) = *dim-col* (*B* * *A*) **using** *assms* **by** *simp*
have *bac*: *B*∗*A* ∈ *carrier-mat n n* **using** *assms* **by** *simp*
fix *i j*
assume *i* < *dim-row* (*B*∗*A*) **and** *j* < *dim-col* (*B*∗*A*) **note** *ij* = *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*::{*ring*} *Matrix.mat*
assumes *diagonal-mat B*
and *B* ∈ *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 ... = 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::{ring} Matrix.mat
  assumes diagonal-mat B
  and B ∈ 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 ‹B$$(i,j) = 0› by simp
  qed
qed

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

qed

lemma *real-diagonal-hermitian*:
fixes *B*::*complex Matrix.mat*
assumes *B*∈ *carrier-mat n n*
and *diagonal-mat B*
and ∀ *i* < *dim-row B*. *B*$$(*i*, *i*) ∈ *Reals*
shows *hermitian B* **unfolding** *hermitian-def*
proof (*rule eq-matI*)
show *dim-row* (*Complex-Matrix.adjoint B*) = *dim-row B* **using** *assms* **by** *auto*
show *dim-col* (*Complex-Matrix.adjoint B*) = *dim-col B* **using** *assms* **by** *auto*
next
fix *i j*
assume *i* < *dim-row B* **and** *j* < *dim-col B* **note** *ij* = *this*
show *Complex-Matrix.adjoint B* $$ (*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 ≡ (*unitary U* ∧
similar-mat-wit A B U (*Complex-Matrix.adjoint U*))

lemma *unitarily-equivD*:
assumes *unitarily-equiv A B U*
shows *unitary U*
similar-mat-wit A B U (*Complex-Matrix.adjoint U*) **using** *assms*
unfolding *unitarily-equiv-def* **by** *auto*

lemma *unitarily-equivI*:
assumes *similar-mat-wit A B U* (*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* = *mat-conj U B*
and *unitary U*
and *A*∈ *carrier-mat n n*
and *B*∈ *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* ·$_m$ *A*) (*x* ·$_m$ *B*) *U*
proof (*rule unitarily-equivI*)
show *similar-mat-wit* (*x* ·$_m$ *A*) (*x* ·$_m$ *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 $A\in$ *carrier-mat n n*
and *unitarily-equiv A B U*
shows *unitarily-equiv* $(-A)$ $(-B)$ *U*
proof (*rule unitarily-equivI*)
show *similar-mat-wit* $(-A)$ $(-B)$ *U* (*Complex-Matrix.adjoint U*)
using *mat-conj-uminus-eq assms*
by (*smt* (*z3*) *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 *unitary U* **using** *assms unitarily-equivD*(*1*)[*of A*] **by** *simp*
qed

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

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

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

lemma *unitarily-equiv-conjugate*:

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

lemma mat-conj-commute:
  assumes A ∈ carrier-mat n n
  and B∈ carrier-mat n n
  and U ∈ carrier-mat n n
  and unitary U
  and A*B = B*A
shows (mat-conj (Complex-Matrix.adjoint U) A) *
  (mat-conj (Complex-Matrix.adjoint U) B) =
  (mat-conj (Complex-Matrix.adjoint U) B) *
  (mat-conj (Complex-Matrix.adjoint U) A) (is ?L*?R = ?R* ?L)
proof −
  have u: Complex-Matrix.adjoint U ∈ carrier-mat n n using assms
    by (simp add: adjoint-dim')
  have ca: Complex-Matrix.adjoint U * A * U ∈ carrier-mat n n
    using assms by auto
  have cb: Complex-Matrix.adjoint U * B * U ∈ carrier-mat n n
    using assms by auto
  have  ?L * ?R =
```

```
    ?L * (Complex-Matrix.adjoint U * (B * U))
  proof −
    have Complex-Matrix.adjoint U * B * U =
      Complex-Matrix.adjoint U * (B * U)
      using assoc-mult-mat[of - n n B n U] assms
      by (meson adjoint-dim')
    thus ?thesis using mat-conj-adjoint by metis
  qed
  also have ... = ?L * Complex-Matrix.adjoint U * (B*U)
  proof −
    have ∃ na nb. Complex-Matrix.adjoint U ∈ carrier-mat n na ∧
      B * U ∈ carrier-mat na nb
    by (metis (no-types) assms(2) carrier-matD(1) carrier-mat-triv index-mult-mat(2)
u)
    then show ?thesis using ca
      by (metis assoc-mult-mat mat-conj-adjoint)
  qed
  also have ... = Complex-Matrix.adjoint U * A*
    (U * (Complex-Matrix.adjoint U)) * (B * U)
  proof −
    have Complex-Matrix.adjoint U * A * U * Complex-Matrix.adjoint U =
      Complex-Matrix.adjoint U * A * (U * Complex-Matrix.adjoint U)
      using assoc-mult-mat[of Complex-Matrix.adjoint U * A n n]
      by (metis assms(1) assms(3) mult-carrier-mat u)
    thus ?thesis by (simp add: mat-conj-adjoint)
  qed
  also have ... = Complex-Matrix.adjoint U * A*  (B * U)
    using assms by auto
  also have ... = Complex-Matrix.adjoint U * A * B * U
  proof (rule assoc-mult-mat[symmetric], auto simp add: assms)
    show B ∈ carrier-mat (dim-col A) (dim-row U) using assms by simp
  qed
  also have ... = Complex-Matrix.adjoint U * (A * B) * U
    using assms u by auto
  also have ... = Complex-Matrix.adjoint U * (B * A) * U using assms by simp
  also have ... = Complex-Matrix.adjoint U * B * A * U
    using assms u by auto
  also have ... = Complex-Matrix.adjoint U * B * (A * U)
  proof (rule assoc-mult-mat, auto simp add: assms)
    show A ∈ carrier-mat (dim-col B) (dim-row U)
      using assms by simp
  qed
  also have ... = Complex-Matrix.adjoint U * B*
    (U * (Complex-Matrix.adjoint U)) * (A * U)
    using assms by auto
  also have ... = Complex-Matrix.adjoint U * B*
    U * (Complex-Matrix.adjoint U) * (A * U)
  proof −
    have Complex-Matrix.adjoint U * B * U * 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 ∈ carrier-mat (dim-col B) (dim-col U) using assms by simp
   qed
   thus ?thesis by simp
  qed
  also have ... = 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 ∈ carrier-mat (dim-row U) n using assms by simp
  qed
  also have ... = 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 ∈ 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 ‹dim-row C = dim-col A› 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 ≡ unitarily-equiv A B U ∧  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*∈ *carrier-mat n n*
and *B*∈ *carrier-mat n n*
and *diagonal-mat B*
and *unitary U*
and *A* = *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*∈ *carrier-mat n n*
and *unitary-diag A B U*
shows *B* ∈ *carrier-mat n n U*∈ *carrier-mat n n*
proof −
show *B* ∈ *carrier-mat n n*
using *assms unitarily-equiv-carrier*(*1*)[*of A n B U*] **by** *simp*
show *U* ∈ *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*∈ *carrier-mat n n*
and *U*∈ *carrier-mat n n*
and *B* ∈ *carrier-mat n n*
and *A* = *mat-conj U B*
and (*Complex-Matrix.adjoint U*) ∗ *U* = 1_m *n*

shows $A * A = mat\text{-}conj\ U\ (B*B)$
proof −
have $A * A = U * B * (Complex\text{-}Matrix.adjoint\ U) * (U * B * (Complex\text{-}Matrix.adjoint\ U))$
using *assms* **unfolding** *mat-conj-def* **by** *simp*
also have $... = U * B * ((Complex\text{-}Matrix.adjoint\ U) * U) * (B * (Complex\text{-}Matrix.adjoint\ U))$
by (*smt* (*z3*) *adjoint-dim-col adjoint-dim-row assms*(*3*) *assms*(*5*) *assoc-mult-mat carrier-matD*(*2*) *carrier-mat-triv index-mult-mat*(*2*) *index-mult-mat*(*3*) *right-mult-one-mat'*)
also have $... = U * B * (B * (Complex\text{-}Matrix.adjoint\ U))$ **using** *assms* **by** *simp*
also have $... = U * (B * B) * (Complex\text{-}Matrix.adjoint\ U)$
by (*smt* (*z3*) *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*::*complex Matrix.mat*
assumes *hermitian A*
and *A*∈ *carrier-mat n n*
and *unitary-diag A B U*
shows *similar-mat-wit* $(A * A)\ (B * B)\ U\ (Complex\text{-}Matrix.adjoint\ U)$
proof −
have *B*∈ *carrier-mat n n* **using** *unitary-diag-carrier*[*of A*] *assms* **by** *metis*
hence $B * B$∈ *carrier-mat n n* **by** *simp*
have *unitary U* **using** *assms unitary-diagD*[*of A*] **by** *simp*
have $A * A$= *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 $\{A * A,\ B * B,\ U,\ Complex\text{-}Matrix.adjoint\ U\} \subseteq$ *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 $U * Complex\text{-}Matrix.adjoint\ U = 1_m\ n \wedge Complex\text{-}Matrix.adjoint\ U * U = 1_m\ n$
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*∈ *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* (*z3*) ‹*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_m (*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_m (*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 *Complex-Matrix.adjoint V* ∈ *carrier-mat* (*dim-col B*) (*dim-col U*)
by (*metis* ‹*B* ∈ *carrier-mat n n*› ‹*U* ∈ *carrier-mat n n*› *adjoint-dim′* *assms*(*2*) *carrier-matD*(*2*))
qed
also have ... = *U* ∗ *V* ∗ (*B* ∗ (*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*))
proof (*rule assoc-mult-mat*, *auto simp add*: *assms*)
show *B* ∈ *carrier-mat* (*dim-col V*) (*dim-col V*)
by (*metis* ‹*B* ∈ *carrier-mat n n*› *assms*(*2*) *carrier-matD*(*2*))
qed
also have ... = *U* ∗ (*V* ∗ (*B* ∗ (*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*)))
proof (*rule assoc-mult-mat*, *auto simp add*: *assms*)
show *V* ∈ *carrier-mat* (*dim-col U*) (*dim-row B*)
using ‹*B* ∈ *carrier-mat n n*› ‹*U* ∈ *carrier-mat n n*› *assms*(*2*) **by** *auto*
qed
finally have *eq*: *U* ∗ *V* ∗ *B* ∗ *Complex-Matrix.adjoint* (*U* ∗ *V*) = *U* ∗ (*V* ∗ (*B* ∗ (*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*))) **.**
have *V* ∗ (*B* ∗ (*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*)) = *V* ∗ *B* ∗ (*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*)
proof (*rule assoc-mult-mat*[*symmetric*], *auto simp add*: *assms*)
show *B* ∈ *carrier-mat* (*dim-col V*) (*dim-col V*)
using ‹*B* ∈ *carrier-mat n n*› *assms*(*2*) **by** *auto*
qed
also have ... = *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*)
by (*metis* ‹*B* ∈ *carrier-mat n n*› ‹*U* ∈ *carrier-mat n n*› *adjoint-dim-row assms*(*2*) *assms*(*5*) *carrier-matD*(*2*) *carrier-mat-triv index-mult-mat*(*3*))
qed
also have ... = *B* ∗ *Complex-Matrix.adjoint U* **using** *assms* **by** *simp*
finally have *V* ∗(*B* ∗(*Complex-Matrix.adjoint V* ∗ *Complex-Matrix.adjoint U*)) =
B∗ *Complex-Matrix.adjoint U* **.**
hence *U* ∗ *V* ∗ *B* ∗ *Complex-Matrix.adjoint* (*U* ∗ *V*) = *U* ∗ *B* ∗ *Complex-Matrix.adjoint U* **using** *eq*
by (*metis* ‹*B* ∈ *carrier-mat n n*› ‹*U* ∈ *carrier-mat n n*› *adjoint-dim′* *assoc-mult-mat*)
also have ... = *A* **using** *assms unitarily-equiv-eq*[*of A B U*] **by** *simp*
finally show *A* = *U* ∗ *V* ∗ *B* ∗ *Complex-Matrix.adjoint* (*U* ∗ *V*) **by** *simp*
qed

definition *real-diag-decomp* **where**
real-diag-decomp A B U ≡ *unitary-diag A B U* ∧
(∀ *i* < *dim-row B*. *B*$$(*i*, *i*) ∈ *Reals*)

lemma *real-diag-decompD*[*simp*]:
assumes *real-diag-decomp A B U*
shows *unitary-diag A B U*
(∀ *i* < *dim-row B*. *B*$$(*i*, *i*) ∈ *Reals*) **using** *assms*
unfolding *real-diag-decomp-def unitary-diag-def* **by** *auto*

lemma *hermitian-decomp-decomp′*:
fixes *A*::*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*::*complex Matrix.mat*
assumes *real-diag-decomp A B U*
shows *hermitian A*
proof −
have *ud*: *unitary-diag A B U* **using** *assms real-diag-decompD* **by** *simp*
hence *A* = *U* ∗ *B* ∗ (*Complex-Matrix.adjoint U*)
by (*simp add*: *unitarily-equiv-eq*)
have *Complex-Matrix.adjoint A* =
Complex-Matrix.adjoint (*U* ∗ *B* ∗ (*Complex-Matrix.adjoint U*))
using *ud assms unitarily-equiv-eq unitary-diag-imp-unitarily-equiv* **by** *blast*
also have ... = *Complex-Matrix.adjoint* (*Complex-Matrix.adjoint U*) ∗
Complex-Matrix.adjoint B ∗ *Complex-Matrix.adjoint U*
by (*smt* (*z3*) *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 ... = *U* ∗ *Complex-Matrix.adjoint B* ∗ *Complex-Matrix.adjoint U*
by (*simp add*: *Complex-Matrix.adjoint-adjoint*)
also have ... = *U* ∗ *B* ∗ *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 ... = *A* **using** ‹*A* = *U* ∗ *B* ∗ (*Complex-Matrix.adjoint U*)› **by** *simp*
finally show *?thesis* **unfolding** *hermitian-def* **by** *simp*
qed

lemma *unitary-conjugate-real-diag-decomp*:
assumes *A*∈ *carrier-mat n n*
and *Us*∈ *carrier-mat n n*
and *unitary Us*
and *real-diag-decomp* (*mat-conj* (*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 $\bigwedge i.\ i < \textit{dim-row}\ B \Longrightarrow B\ \$\$\ (i,\ i) \in \mathbb{R}$ **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 $(\prod a{\leftarrow}(\textit{eigvals}\ A).\ [:-\ a,\ 1:]) = \textit{char-poly}\ A$
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 ... = $(\prod a{\leftarrow}\textit{diag-mat}\ B.\ [:-\ a,\ 1:])$
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 $(\prod a{\leftarrow}(\textit{eigvals}\ A).\ [:-\ a,\ 1:]) = (\prod a{\leftarrow}\textit{diag-mat}\ B.\ [:-\ a,\ 1:])$ **.**
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 ∃ *B U*. *real-diag-decomp A B U* **by** *auto*

```
  }
  thus ?thesis using that by auto
qed

lemma spectrum-smult:
  fixes A::complex Matrix.mat
  assumes hermitian A
  and A∈ carrier-mat n n
  and 0 < n
shows spectrum (x ·_m A) = {x * a|a. a∈ spectrum A}
proof −
  obtain B U where bu: real-diag-decomp A B U
    using assms hermitian-real-diag-decomp[of A] by auto
  hence spectrum (x ·_m A) = set (diag-mat (x ·_m B))
    using assms unitary-diag-spectrum-eq[of x ·_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 ... = {x * a |a. a ∈ set (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 ... = {x * a|a. a∈ 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::complex Matrix.mat
  assumes hermitian A
  and A∈ carrier-mat n n
  and 0 < n
shows spectrum (−A) = {−a|a. a∈ spectrum A}
proof −
  obtain B U where bu: real-diag-decomp A B U
    using assms hermitian-real-diag-decomp[of A] by auto
  hence spectrum (−A) = set (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 ... = {−a |a. a ∈ set (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 ... = {−a|a. a∈ 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 $z + cnj\ z \leq 2 * cmod\ z$
proof −
have *z*: $z + cnj\ z = 2 * Re\ z$ **by** (*simp add*: *complex-add-cnj*)
have $Re\ z \leq cmod\ z$ **by** (*simp add*: *complex-Re-le-cmod*)
hence $2 * complex\text{-}of\text{-}real\ (Re\ z) \leq 2 * complex\text{-}of\text{-}real\ (cmod\ z)$
using *less-eq-complex-def* **by** *auto*
thus *?thesis* **using** *z* **by** *simp*
qed

lemma *abs-real*:
fixes *x*::*complex*
assumes $x \in Reals$
shows $abs\ x \in Reals$ **unfolding** *abs-complex-def* **by** *auto*

lemma *csqrt-cmod-square*:
shows $csqrt\ ((cmod\ z)^2) = cmod\ z$
proof −
have $csqrt\ ((cmod\ z)^2) = sqrt\ (Re\ ((cmod\ z)^2))$ **by** *force*
also have ... = *cmod z* **by** *simp*
finally show *?thesis* **.**
qed

lemma *cpx-real-le*:
fixes *z*::*complex*
assumes $0 \leq z$
and $0 \leq u$
and $z^2 \leq u^2$
shows $z \leq u$
proof −
have $z\hat{}2 = Re\ (z\hat{}2)\ u\hat{}2 = Re\ (u\hat{}2)$ **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*: $Re\ (z\hat{}2) \leq Re\ (u\hat{}2)$ **using** *assms less-eq-complex-def* **by** *simp*
have $sqrt\ (Re\ (z\hat{}2)) = z\ sqrt\ (Re\ (u\hat{}2)) = u$ **using** *assms complex-eqI*
less-eq-complex-def **by** *auto*
have $z = sqrt\ (Re\ (z\hat{}2))$ **using** *assms complex-eqI less-eq-complex-def*
by *auto*
also have ... $\leq sqrt\ (Re\ (u\hat{}2))$ **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*::*complex*
shows $v * (conjugate\ v) \in Reals$
proof −

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

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

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

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

lemma *cmod-pos*:
fixes $z::complex$
assumes $0 \leq z$
shows $cmod\ z = z$
proof −
have $Im\ z = 0$ **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 $z::complex$
assumes $0 \leq z$
shows $0 \leq z^2$
proof −
have $Im\ z = 0$ **using** *assms* **by** (*simp add*: *complex-positive-Im*)
hence $Re\ (z^2) = (Re\ z)^2$ **by** *simp*
moreover have $Im\ (z^2) = 0$ **by** (*simp add*: ‹$Im\ z = 0$›)
ultimately show *?thesis* **by** (*simp add*: *less-eq-complex-def*)
qed

lemma *cmod-mult-pos*:
fixes $b::$ *complex*
fixes $z::complex$
assumes $0 \leq b$
shows $cmod\ (b * z) =\ Re\ b * cmod\ z$ **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* ≤ *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 ⋀*i*. *i* ∈ *I* ⟹ (*0*::*real*) ≤ *f i*
and *j*∈ *I*
and *c* < *f j*
shows *c* < *sum f I*
proof −
have *c* < *f j* **using** *assms* **by** *simp*
also have ... ≤ *f j* + *sum f* (*I* −{*j*})
by (*smt* (*z3*) *DiffD1 assms*(*2*) *sum-nonneg*)
also have ... = *sum f I* **using** *assms*
by (*simp add*: *sum-diff1*)
finally show *?thesis* **.**
qed

lemma *pos-sum-le-comp*:
assumes *finite I*
and ⋀*i*. *i* ∈ *I* ⟹ (*0*::*real*) ≤ *f i*
and *sum f I* ≤ *c*
shows ∀ *i* ∈ *I*. *f i* ≤ *c*
proof (*rule ccontr*)
assume ¬ (∀ *i*∈*I*. *f i* ≤ *c*)
hence ∃*j*∈ *I*. *c* < *f j* **by** *fastforce*
from *this* **obtain** *j* **where** *j*∈ *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 ⋀*i*. *i* ∈ *I* ⟹ ((*0*::*real*) ≤ *f i* ∧ *f i* ≤ *1*)
shows *sum* (λ*x*. *f x* ∗ *f x*) *I* ≤ *sum f I* **using** *assms*

proof (*induct rule:finite-induct*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert x F*)
have *sum* (λ*x*. *f x* ∗ *f x*) (*insert x F*) = *f x* ∗ *f x* + *sum* (λ*x*. *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 ⋀*i*. *i* ∈ *I* ⟹ ((*0*::*real*) ≤ *f i* ∧ *f i* ≤ *1*)
and *j* ∈ *I*
and *f j* < *1*
and *0* < *f j*
shows *sum* (λ*x*. *f x* ∗ *f x*) *I* < *sum f I* **using** *assms*
proof −
have *sum* (λ*x*. *f x* ∗ *f x*) *I* =
sum (λ*x*. *f x* ∗ *f x*) {*j*} + *sum* (λ*x*. *f x* ∗ *f x*) (*I*−{*j*})
using *assms sum.remove* **by** *fastforce*
also have ... = *f j* ∗ *f j* + *sum* (λ*x*. *f x* ∗ *f x*) (*I*−{*j*}) **by** *simp*
also have ... < *f j* + *sum* (λ*x*. *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-csqrt*)

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

lemma *rank-1-proj-trace-inner*:
fixes $A :: 'a$::*conjugatable-field Matrix.mat* **and** $v :: 'a$ *Matrix.vec*
assumes *A*: $A \in$ *carrier-mat n n*
and *v*: $v \in$ *carrier-vec n*
shows *Complex-Matrix.trace* $(A * (rank\text{-}1\text{-}proj\ v)) =$ *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*∈ *carrier-vec n*
and $w \in$ *carrier-vec n*
and $U \in$ *carrier-mat n n*
and *Complex-Matrix.unitary U*
shows *inner-prod* $(U *_v v)$ $(U *_v w) =$ *inner-prod v w*
proof −
have *inner-prod* $(U *_v v)$ $(U *_v w) =$
inner-prod $(((Complex\text{-}Matrix.adjoint\ U) * U) *_v v)\ w$
using *assms* **by** (*simp add*: *inner-prod-mult-mat-vec-left*)
also have ... = *inner-prod* $(1_m\ n *_v v)\ w$ **using** *assms* **by** *simp*
also have ... = *inner-prod v w* **using** *assms* **by** *simp*
finally show *?thesis* **.**
qed

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

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

lemma *unitary-col-norm*:
assumes *unitary U*
and *U*∈ *carrier-mat n n*
and $i < n$
shows $\|Matrix.col\ U\ i\| = 1$ **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*::*complex Matrix.mat*
assumes *diagonal-mat B*
and $B \in carrier\text{-}mat\ n\ n$
and $v \in carrier\text{-}vec\ n$
shows $inner\text{-}prod\ (B *_v v)\ v =$
$(\sum i \in \{0\ ..<\ n\}.\ (conjugate\ (B\ \$\$\ (i,i))) * (vec\text{-}index\ v\ i *$
$(conjugate\ (vec\text{-}index\ v\ i))))$
proof −
have *idx*: $\bigwedge i.\ i < n \Longrightarrow vec\text{-}index\ (B *_v v)\ i = B\ \$\$\ (i,i) * (vec\text{-}index\ v\ i)$
using *assms diagonal-mat-mult-vec* **by** *blast*
have $inner\text{-}prod\ (B *_v v)\ v =$
$(\sum i \in \{0\ ..<\ n\}.\ vec\text{-}index\ v\ i * vec\text{-}index\ (conjugate\ (B *_v v))\ i)$
unfolding *Matrix.scalar-prod-def* **using** *assms* **by** *fastforce*
also have ... = $(\sum i \in \{0\ ..<\ n\}.\ vec\text{-}index\ v\ i * conjugate\ (vec\text{-}index\ (B *_v v)$
$i))$
using *assms* **by** *force*
also have ... = $(\sum i \in \{0\ ..<\ n\}.\ (conjugate\ (B\ \$\$\ (i,i))) * (vec\text{-}index\ v\ i *$
$(conjugate\ (vec\text{-}index\ v\ i))))$
proof (*rule sum.cong*, *simp*)
show $\bigwedge i.\ i \in \{0\ ..<\ n\} \Longrightarrow vec\text{-}index\ v\ i * conjugate\ (vec\text{-}index\ (B *_v v)\ i)$
=
$(conjugate\ (B\ \$\$\ (i,i))) * (vec\text{-}index\ v\ i * (conjugate\ (vec\text{-}index\ v\ i)))$
by (*simp add*: *idx*)

```
  qed
  finally show ?thesis .
qed

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

lemma self-inner-prod-real:
  fixes v::complex Matrix.vec
  shows Complex-Matrix.inner-prod v v ∈ Reals
proof −
  have Im (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::complex Matrix.mat
  assumes diagonal-mat B
  and B ∈ carrier-mat n n
  and ∀ i < n. B$$(i, i) ∈ Reals
  and v ∈ carrier-vec n
shows inner-prod (B *_v v) v ∈ Reals
proof −
```

```
  have  inner-prod (B *v v) v =
    (∑ i ∈ {0 ..< n}. (conjugate (B $$ (i,i))) * (vec-index v i *
    (conjugate (vec-index v i)))) using inner-mult-diag-expand assms
    by simp
  also have ... ∈ Reals
  proof (rule real-sum-real)
    show ⋀i. i < n ⟹
      conjugate (B $$ (i, i)) *
      ((vec-index v i) * conjugate (vec-index v i)) ∈ ℝ
      using assms mult-conj-real by auto
  qed
  finally show ?thesis .
qed

lemma inner-mult-diag-real′:
  fixes B::complex Matrix.mat
  assumes diagonal-mat B
  and B ∈ carrier-mat n n
  and ∀ i < n. B$$(i, i) ∈ Reals
  and v ∈ carrier-vec n
shows inner-prod v (B *v v)  ∈ Reals
proof −
  have  inner-prod v (B *v v)  =
    (∑ i ∈ {0 ..< n}. B $$ (i,i) * (vec-index v i *
    (conjugate (vec-index v i))))
    using inner-mult-diag-expand′ assms by simp
  also have ... ∈ Reals
  proof (rule real-sum-real)
    show ⋀i. i < n ⟹
      B $$ (i, i) * ((vec-index v i) * conjugate (vec-index v i)) ∈ ℝ
      using assms mult-conj-real by auto
  qed
  finally show ?thesis .
qed

lemma inner-prod-mult-mat-vec-right:
assumes v ∈ carrier-vec n
  and w ∈ carrier-vec n′
  and A ∈ carrier-mat m n
  and B ∈ carrier-mat m n′
shows inner-prod (A *v v) (B *v w) =
  inner-prod v (((Complex-Matrix.adjoint A) * B) *v w)
proof −
  have inner-prod (A *v v) (B *v w) =
    inner-prod ((Complex-Matrix.adjoint (Complex-Matrix.adjoint A)) *v v)
      (B *v w)
    by (simp add: Complex-Matrix.adjoint-adjoint)
  also have ... = inner-prod v ((Complex-Matrix.adjoint A) *v (B *v w))
  proof (rule adjoint-def-alter[symmetric])
```

```
    show v ∈ carrier-vec n using assms by simp
    show B *v w ∈ carrier-vec m using assms by simp
    show Complex-Matrix.adjoint A ∈ carrier-mat n m
      using assms adjoint-dim'[of A] by simp
  qed
  also have ... = inner-prod v (((Complex-Matrix.adjoint A) * B) *v w)
    using assms
  proof −
    have (Complex-Matrix.adjoint A) *v (B *v w) =
      ((Complex-Matrix.adjoint A) * B) *v w
    proof (rule assoc-mult-mat-vec[symmetric], (auto simp add: assms))
      show w ∈ carrier-vec n' using assms by simp
      show B ∈ carrier-mat (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 dim-vec x = dim-vec y
shows cmod (inner-prod x y) ≤ vec-norm x * vec-norm y
proof −
  have x: x ∈ carrier-vec (dim-vec x) by simp
  moreover have y: y ∈ carrier-vec (dim-vec x) using assms by simp
  ultimately have (cmod (inner-prod x y))^2 = inner-prod x y * inner-prod y x
    using complex-norm-square by (metis inner-prod-swap mult-conj-cmod-square)
  also have ... ≤ inner-prod x x * inner-prod y y
    using Cauchy-Schwarz-complex-vec x y by blast
  finally have (cmod (inner-prod x y))^2 ≤ inner-prod x x * inner-prod y y .
  hence (cmod (inner-prod x y))^2 ≤ Re (inner-prod x x) * Re (inner-prod y y)
    using less-eq-complex-def by simp
  hence sqrt ((cmod (inner-prod x y))^2) ≤
    sqrt (Re (inner-prod x x) * Re (inner-prod y y))
    using real-sqrt-le-iff by blast
  also have ... =  sqrt (Re (inner-prod x x)) * sqrt (Re (inner-prod y y))
    by (simp add: real-sqrt-mult)
  finally have sqrt ((cmod (inner-prod x y))^2) ≤
    sqrt (Re (inner-prod x x)) * sqrt (Re (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::complex Matrix.vec
  assumes dim-vec u = dim-vec v
  shows (vec-norm (u+ v))^2 ≤ (vec-norm u + vec-norm v)^2
proof −
  have (vec-norm (u+v))^2 = inner-prod (u+v) (u+v)
    by (simp add: inner-prod-vec-norm-pow2)
```

also have ... = *inner-prod u u + inner-prod u v + inner-prod v u +*
inner-prod v v
using *assms add-scalar-prod-distrib conjugate-add-vec*
by (*smt* (*z3*) *ab-semigroup-add-class.add-ac*(*1*) *carrier-vec-dim-vec*
dim-vec-conjugate index-add-vec(*2*) *scalar-prod-add-distrib*)
also have ... = (*vec-norm u*)^2 + *inner-prod u v + inner-prod v u +*
(*vec-norm v*)^2
by (*simp add*: *inner-prod-vec-norm-pow2*)
also have ... ≤ (*vec-norm u*)^2 + *2* ∗ *cmod* (*inner-prod u v*) + (*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 ... ≤ (*vec-norm u*)^2 + *2* ∗ ((*vec-norm u*)∗(*vec-norm v*)) +
(*vec-norm v*)^2
using *Cauchy-Schwarz-complex-vec-norm*[*of u v*] *assms less-eq-complex-def*
by *auto*
also have ... = (*vec-norm u + vec-norm v*)^2 **by** (*simp add*: *power2-sum*)
finally show *?thesis* **.**
qed

lemma *vec-norm-triangle*:
fixes *u*::*complex Matrix.vec*
assumes *dim-vec u* = *dim-vec v*
shows *vec-norm* (*u*+ *v*) ≤ *vec-norm u + vec-norm v*
proof (*rule cpx-real-le*)
show (*vec-norm* (*u + v*))^2 ≤ (*vec-norm u + vec-norm v*)^2
using *assms vec-norm-triangle-sq* **by** *simp*
show *0* ≤ *vec-norm* (*u + v*) **using** *vec-norm-geq-0* **by** *simp*
show *0* ≤ *vec-norm u + 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* ∈ *fc-mats*
and *unitary-diag A B U*
shows *sum-mat* (λ*i*. (*diag-mat B* ! *i*) ·$_m$ *rank-1-proj* (*Matrix.col U i*))
{..< *dimR*} = *A*
proof −
have *similar-mat-wit A B U* (*Complex-Matrix.adjoint U*) ∧ *diagonal-mat B* ∧
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* ∈ *carrier-mat dimR dimR* **and** *dimP*: *U* ∈ *carrier-mat dimR*
dimR
and *unit*: *unitary U*
unfolding *similar-mat-wit-def Let-def* **using** *assms fc-mats-carrier* **by** *auto*

```
  have pz: ⋀i. i < dimR ⟹ (Matrix.col U i) = zero-col U i
    unfolding zero-col-def by simp
  have sum-mat (λi. (diag-mat B ! i) ·_m
    rank-1-proj (Matrix.col U i)) {..< dimR} =
    U * B * 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 ∈ fc-mats using fc-mats-carrier dim-eq dimP by simp
    show B∈ 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 ∈ carrier-mat n n
  and 0 < n
and Complex-Matrix.unitary A
and j < n
and k < n
shows Complex-Matrix.inner-prod (Matrix.col A j) (Matrix.col A k) =
  (1_m n) $$ (j,k)
proof −
  have Complex-Matrix.inner-prod (Matrix.col A j) (Matrix.col A k) =
    (Complex-Matrix.adjoint A * A) $$ (j, k)
    using inner-prod-adjoint-comp[of A n A] assms
    by simp
  also have ... = (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 ∈ I
  and A j * A j = A j
  and ⋀i. i∈ I ⟹ A i ∈ fc-mats
  and ⋀ i . i∈ I ⟹ i≠ j ⟹ A i * (A j) = (0_m dimR dimR)
  shows (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 (sum-mat A (insert x F)) * (A j) =
    (A x + sum-mat A F) * (A j) using insert sum-mat-insert[of A]
    by (simp add: image-subset-iff)
  also have ... = A x * (A j) + sum-mat A F * (A j)
```

proof (*rule add-mult-distrib-mat*)
show $A\ x \in$ *carrier-mat dimR dimC* **using** *insert fc-mats-carrier* **by** *simp*
show *sum-mat A F* $\in$ *carrier-mat dimR dimC* **using** *insert*
by (*metis insert-iff local.fc-mats-carrier sum-mat-carrier*)
show $A\ j \in$ *carrier-mat dimC dimC* **using** *insert dim-eq fc-mats-carrier* **by** *force*
qed
also have ... = $A\ j$
proof (*cases* $x = j$)
case *True*
hence $j \notin F$ **using** *insert* **by** *auto*
hence *sum-mat A F* $*$ $A\ j = 0_m$ *dimR dimR* **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 $\bigwedge i.\ i \in F \Longrightarrow A\ i \in$ *fc-mats* **using** *insert* **by** *simp*
moreover have $\bigwedge i.\ i \in F \Longrightarrow i \neq j \Longrightarrow A\ i * A\ j = 0_m$ *dimR dimR* **using** *insert* **by** *simp*
ultimately have *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 $B \in$ *fc-mats*
and $\bigwedge i.\ i \in I \Longrightarrow A\ i \in$ *fc-mats*
and $\bigwedge i\ .\ i \in I \Longrightarrow i \neq j \Longrightarrow A\ i * B = (0_m\ dimR\ dimR)$
shows (*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 (*sum-mat A* (*insert x F*)) $*$ $B =$
($A\ x$ + *sum-mat A F*) $*$ B **using** *insert sum-mat-insert*[*of A*]
by (*simp add*: *image-subset-iff*)
also have ... = $A\ x * B$ + *sum-mat A F* $*$ B
proof (*rule add-mult-distrib-mat*)
show $A\ x \in$ *carrier-mat dimR dimC* **using** *insert fc-mats-carrier* **by** *simp*
show *sum-mat A F* $\in$ *carrier-mat dimR dimC* **using** *insert*
by (*metis insert-iff local.fc-mats-carrier sum-mat-carrier*)
show $B \in$ *carrier-mat dimC dimC* **using** *insert dim-eq fc-mats-carrier* **by** *force*
qed

also have ... = $A\ j * B$
proof (*cases* $x = j$)
case *True*
hence $j \notin F$ **using** *insert* **by** *auto*
hence *sum-mat A F * B = 0*$_m$ *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 $\bigwedge i.\ i \in F \Longrightarrow A\ i \in$ *fc-mats* **using** *insert* **by** *simp*
moreover have $\bigwedge i.\ i \in F \Longrightarrow i \neq j \Longrightarrow A\ i * B = 0_m$ *dimR dimR* **using** *insert* **by** *simp*
ultimately have *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 $A \in$ *carrier-mat n n*
and *unitarily-equiv A B U*
and $i < n$
shows *rank-1-proj* (*Matrix.col U i*) $\in$ *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*::*complex Matrix.mat*
assumes $A \in$ *carrier-mat n n*
and $0 < n$
and *hermitian A*
and *unitary-diag A B U*
and $j < n$
shows *Complex-Matrix.trace* ($A *$ (*rank-1-proj* (*Matrix.col U j*))) = *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*)
have *rf*: $\bigwedge i.\ i < n \Longrightarrow$ *rank-1-proj* (*Matrix.col U i*) $\in$ *fc* **using**
assms unitarily-equiv-rank-1-proj-col-carrier
by (*metis fc-def unitary-diag-imp-unitarily-equiv*)
hence *sm*: $\bigwedge i.\ i < n \Longrightarrow$ *diag-mat B* ! *i* $\cdot_m$ *rank-1-proj* (*Matrix.col U i*) $\in$ *fc*
using *fc-mats-carrier dim-eq* **by** *simp*

have $A * (\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j)) =$
$(\textit{sum-mat}\ (\lambda i.\ (\textit{diag-mat}\ B\ !\ i) \cdot_m (\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ i)))\ \{..< n\}) *$
$(\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j))$ **using** *assms sum-decomp-cols*
unfolding *fc-def* **by** *simp*
also have $... = \textit{diag-mat}\ B\ !\ j \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) *$
$\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j)$
proof (*rule sum-mat-ortho-one*, (*auto simp add*: *assms*))
show $\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) \in fc$ **by** (*simp add*: *assms rf*)
show $\bigwedge i.\ i < n \Longrightarrow \textit{diag-mat}\ B\ !\ i \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ i) \in fc$
by (*simp add*: *sm*)
show $\bigwedge i.\ i < n \Longrightarrow i \neq j \Longrightarrow$
$\textit{diag-mat}\ B\ !\ i \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ i) *$
$\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) = 0_m\ n\ n$
proof −
fix i
assume $i < n$ **and** $i \neq j$
define OP **where** $OP = \textit{outer-prod}\ (\textit{Matrix.col}\ U\ i)\ (\textit{Matrix.col}\ U\ j)$
have *cm*: $OP \in \textit{carrier-mat}\ n\ n$ **unfolding** *OP-def*
proof (*rule outer-prod-dim*)
have $\textit{dim-row}\ U = n$
using *assms unitary-diag-carrier*(*2*) *fc-mats-carrier*
by (*metis carrier-matD*(*1*))
thus $\textit{Matrix.col}\ U\ i \in \textit{carrier-vec}\ n$ **using**‹$i < n$›
by (*simp add*: *carrier-vecI*)
show $\textit{Matrix.col}\ U\ j \in \textit{carrier-vec}\ n$ **using** *assms* ‹$\textit{dim-row}\ U = n$›
by (*simp add*: *carrier-vecI*)
qed
have $\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ i) * \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) =$
$1_m\ n\ \$\$\ (i, j) \cdot_m \textit{outer-prod}\ (\textit{Matrix.col}\ U\ i)\ (\textit{Matrix.col}\ U\ j)$
proof (*rule rank-1-proj-unitary*, (*auto simp add*: ‹$i < n$› *assms*))
show $U \in fc$ **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 $... = 0 \cdot_m \textit{outer-prod}\ (\textit{Matrix.col}\ U\ i)\ (\textit{Matrix.col}\ U\ j)$
using ‹$i \neq j$›
by (*metis* ‹$i < n$› *assms*(*5*) *index-one-mat*(*1*))
also have $... = 0_m\ n\ n$ **using** *cm smult-zero* **unfolding** *OP-def* **by** *auto*
finally show $\textit{diag-mat}\ B\ !\ i \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ i) *$
$\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) = 0_m\ n\ n$
by (*metis* ‹$i < n$› ‹$\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) \in fc$›
fc-mats-carrier mult-smult-assoc-mat rf smult-zero-mat)
qed
qed
also have $... = \textit{diag-mat}\ B\ !\ j \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j)$
proof −
have $\textit{diag-mat}\ B\ !\ j \cdot_m \textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) *$
$\textit{rank-1-proj}\ (\textit{Matrix.col}\ U\ j) =$

```
        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* ≤ *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 ... = *Complex-Matrix.trace* (*A* ∗ (*rank-1-proj Uj*)) **using** *rank-1-proj-trace-inner uj*
assms **unfolding** *Uj-def* **by** *metis*
also have ... = *B* $$ (*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 *Complex-Matrix.trace* (*C* ∗ *A*) =
(∑ *i* = *0* ..< *n*. *B*$$(*i*,*i*) ∗
Complex-Matrix.trace (*C* ∗ *rank-1-proj* (*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 ∗ (*sum-mat* (λ*i*. (*diag-mat B* ! *i*) ·$_m$
rank-1-proj (*Matrix.col U i*)) {..< *n*})
using *sum-decomp-cols assms* ‹*hermitian A*›
unfolding *fc-def* **by** *simp*
also have ... = *sum-mat* (λ*i*. *C* ∗ ((*diag-mat B* ! *i*) ·$_m$
rank-1-proj (*Matrix.col U i*))) {..< *n*}
by (*rule sum-mat-distrib-left*[*symmetric*],
(*auto simp add*: *assms rf smult-mem fc-def*))
also have ... = *sum-mat* (λ*i*. (*diag-mat B* ! *i*) ·$_m$

```
    (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 ⋀i. i < n ⟹ diag-mat B ! i ·_m (C * rank-1-proj (Matrix.col U i)) ∈
      carrier-mat n n
     using assms unitarily-equiv-rank-1-proj-col-carrier cpx-sq-mat-mult smult-mem
      by (simp add: rf)
    show ⋀i. i < n ⟹ C * (diag-mat B ! i ·_m rank-1-proj (Matrix.col U i)) ∈
      carrier-mat n n
      using assms unitarily-equiv-rank-1-proj-col-carrier cpx-sq-mat-mult
        smult-mem by (simp add: rf)
    show ⋀i. i < n ⟹ C * (diag-mat B ! i ·_m rank-1-proj (Matrix.col U i)) =
      diag-mat B ! i ·_m (C * rank-1-proj (Matrix.col U i))
      by (metis assms(2) fc-mats-carrier mult-smult-distrib rf)
  qed
  finally have ceq: C * A = sum-mat (λi. (diag-mat B ! i) ·_m
    (C * rank-1-proj (Matrix.col U i))) {..< n} .
  have Complex-Matrix.trace (sum-mat (λi. (diag-mat B ! i) ·_m
    (C * rank-1-proj (Matrix.col U i))) {..< n}) = (∑ i = 0 ..< n.
      Complex-Matrix.trace ((diag-mat B ! i) ·_m
    (C * rank-1-proj (Matrix.col U i))))
     by (smt (z3) 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 ... = (∑ i = 0 ..< n. (diag-mat B ! i) *
      Complex-Matrix.trace  (C * rank-1-proj (Matrix.col U i)))
  proof (rule sum.cong, simp)
    show ⋀x. x ∈ {0..<n} ⟹
         Complex-Matrix.trace (diag-mat B ! x ·_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 ... = (∑ i = 0 ..< n. B $$ (i,i) *
      Complex-Matrix.trace  (C * rank-1-proj (Matrix.col U i)))
  proof (rule sum.cong, simp)
    have B ∈ carrier-mat n n using unitary-diag-carrier(1) assms
      fc-mats-carrier dim-eq by simp
   hence ⋀x. x ∈ {0..<n} ⟹ diag-mat B!x = B$$(x,x) unfolding diag-mat-def
by simp
    thus ⋀x. x ∈ {0..<n} ⟹
         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
```

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

lemma *unitarily-equiv-trace*:
assumes *A* ∈ *carrier-mat n n*
and *unitarily-equiv A B U*
shows *Complex-Matrix.trace A* = *Complex-Matrix.trace B*
proof −
have *Complex-Matrix.trace A* = *Complex-Matrix.trace* (*U* ∗ *B* ∗ (*Complex-Matrix.adjoint U*))
using *assms unitarily-equiv-eq*[*of A*] *unitary-diag-imp-unitarily-equiv*[*of A*] **by** *simp*
also have ... = *Complex-Matrix.trace* (*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 ... = *Complex-Matrix.trace B* **using** *assms*
by (*smt* (*z3*) *Complex-Matrix.unitary-def adjoint-dim-col assoc-mult-mat carrier-matD*(*2*) *carrier-mat-triv index-mult-mat*(*3*) *left-mult-one-mat unitarily-equivD*(*1*) *unitarily-equiv-carrier*(*1*) *unitarily-equiv-eq unitary-simps*(*1*))
finally show *?thesis* .
qed

lemma *unitarily-equiv-trace*′:
assumes *A* ∈ *carrier-mat n n*
and *unitarily-equiv A B U*
shows *Complex-Matrix.trace A* = (∑ *i* = *0* ..< *dim-row A*. *B* $$ (*i*,*i*))
proof −
have *Complex-Matrix.trace A* = *Complex-Matrix.trace B* **using** *assms unitarily-equiv-trace*[*of A*]
by (*meson unitary-diag-imp-unitarily-equiv*)
also have ... = (∑ *i* = *0* ..< *dim-row A*. *B* $$ (*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*::*complex Matrix.mat*
assumes *A*∈ *carrier-mat n n*
and *C*∈ *carrier-mat n n*
and *0* < *n*
and *Complex-Matrix.positive A*
and *unitary-diag A B U*
and ⋀*i*. *i* < *n* ⟹ *cmod* (*Complex-Matrix.trace* (*C* ∗ *rank-1-proj* (*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*$$(i,i)) *$ *cmod* (*Complex-Matrix.trace* $(C *$ *rank-1-proj* (*Matrix.col U i*))))
proof (*rule sum.cong*, *simp*)
show $\bigwedge x.\ x \in \{0..<n\} \Longrightarrow$
cmod (*B* $$ (*x*, *x*) *$ *Complex-Matrix.trace* $(C *$ *rank-1-proj* (*Matrix.col U x*))) =
Re (*B* $$ (*x*, *x*)) *$ *cmod* (*Complex-Matrix.trace* $(C *$ *rank-1-proj* (*Matrix.col U x*)))
using *cmod-mult-pos positive-unitary-diag-pos assms* **by** (*metis atLeastLessThan-iff*)
qed
also have $... \leq (\sum i = 0 ..< n.$ *Re* (*B*$$(i,i)) *$ $M)$
proof −
have $\bigwedge i.\ i < n \Longrightarrow 0 \leq$ *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 ‹*dim-row* $B = n$› **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 $D \in$ *carrier-mat n n*
and $a \leq n$
and *split-block D a a* $= (D1, D2, D3, D4)$
shows *D1*$\in$ *carrier-mat a a D4*$\in$ *carrier-mat* $(n-a)$ $(n-a)$
proof −
show *D1*$\in$ *carrier-mat a a* **using** *assms* **unfolding** *split-block-def*
by (*metis Pair-inject mat-carrier*)
show *D4* $\in$ *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 $D \in$ *carrier-mat n n*
and $a \leq n$
and *split-block D a a* $= (D1, D2, D3, D4)$
shows *diagonal-mat D1* $\wedge$ *diagonal-mat D4* **unfolding** *diagonal-mat-def*
proof (*intro allI conjI impI*)
have *D1* $\in$ *carrier-mat a a* **using** *assms* **unfolding** *split-block-def Let-def*
by *fastforce*
fix *i j*
assume $i <$ *dim-row D1*
and $j <$ *dim-col D1*
and $i \neq j$
have *D1* $$ $(i,j) =$ *D* $$(*i*,*j*) **using** *assms* **unfolding** *split-block-def Let-def*
using ‹$i <$ *dim-row D1*› ‹$j <$ *dim-col D1*› **by** *fastforce*
also have ... $=$ *0* **using** *assms* ‹$i \neq j$› ‹*D1* $\in$ *carrier-mat a a*›
‹$i <$ *dim-row D1*› ‹$j <$ *dim-col D1*› **unfolding** *diagonal-mat-def* **by** *fastforce*
finally show *D1* $$(*i*,*j*) = *0* **.**
next
have *D4* $\in$ *carrier-mat* $(n-a)$ $(n-a)$ **using** *assms*
unfolding *split-block-def Let-def* **by** *fastforce*
fix *i j*
assume $i <$ *dim-row D4*
and $j <$ *dim-col D4*
and $i \neq j$
have *D4* $$ $(i,j) =$ *D* $$(*i* + *a*,*j* + *a*) **using** *assms* **unfolding** *split-block-def Let-def*
using ‹$i <$ *dim-row D4*› ‹$j <$ *dim-col D4*› **by** *fastforce*
also have ... $=$ *0* **using** *assms* ‹$i \neq j$› ‹*D4* $\in$ *carrier-mat* $(n-a)$ $(n-a)$›
‹$i <$ *dim-row D4*› ‹$j <$ *dim-col D4*› **unfolding** *diagonal-mat-def* **by** *fastforce*
finally show *D4* $$(*i*,*j*) = *0* **.**
qed

lemma *split-block-times-diag-index*:
fixes *B*::*'a*::*comm-ring Matrix.mat*
assumes *diagonal-mat D*
and *D*∈ *carrier-mat n n*
and *B*∈ *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 $i <$ *dim-row* (*D4* ∗ *B4*)
and $j <$ *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* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*) **using** *assms*
split-block(*4*)[*of D*] **by** *simp*
have *b4*: *B4* ∈ *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** ‹*i* < *dim-row* (*D4* ∗ *B4*)› *b4 d4* **by** *simp*
have *j* < *n*−*a* **using** ‹*j* < *dim-col* (*D4* ∗ *B4*)› *b4 d4* **by** *simp*
have (*B4* ∗ *D4*) $$ (*i*, *j*) = *D4* $$ (*j*,*j*) ∗ *B4* $$ (*i*,*j*)
proof (*rule diagonal-mat-mult-index′*)
show *diagonal-mat D4* **using** ‹*diagonal-mat D4*› **.**
show *B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*) **using** *b4* **.**
show *D4* ∈ *carrier-mat* (*n* − *a*) (*n* − *a*) **using** *d4* **.**
show *i* < *n*−*a* **using** ‹*i* < *n*−*a*› **.**
show *j* < *n*−*a* **using** ‹*j* < *n*−*a*› **.**
qed
also have ... = *D* $$ (*j*+*a*, *j*+*a*) ∗ *B* $$ (*i*+*a*, *j*+*a*)
using *assms* ‹*i* < *n*−*a*› ‹*j* < *n*−*a*›
unfolding *split-block-def Let-def* **by** *fastforce*
also have ... = (*B*∗*D*) $$ (*i*+*a*, *j*+*a*) **using** *diagonal-mat-mult-index′ assms*
by (*metis* ‹*i* < *n* − *a*› ‹*j* < *n* − *a*› *less-diff-conv*)
finally show (*B4* ∗ *D4*) $$ (*i*, *j*) = (*B*∗*D*) $$ (*i*+*a*, *j*+*a*) **.**
have (*D4* ∗ *B4*) $$ (*i*, *j*) = *D4* $$ (*i*,*i*) ∗ *B4* $$ (*i*,*j*)
using *diagonal-mat-mult-index* ‹*diagonal-mat D4*› ‹*i* < *n* − *a*› ‹*j* < *n* − *a*› *b4*
d4
by *blast*
also have ... = *D* $$ (*i*+*a*, *i*+*a*) ∗ *B* $$ (*i*+*a*, *j*+*a*)
using *assms* ‹*i* < *n*−*a*› ‹*j* < *n*−*a*›
unfolding *split-block-def Let-def* **by** *fastforce*
also have ... = (*D*∗*B*) $$ (*i*+*a*, *j*+*a*) **using** *diagonal-mat-mult-index assms*
by (*metis* ‹*i* < *n* − *a*› ‹*j* < *n* − *a*› *less-diff-conv*)
finally show (*D4* ∗ *B4*) $$ (*i*, *j*) = (*D*∗*B*) $$ (*i*+*a*, *j*+*a*) **.**
qed

lemma *split-block-commute-subblock*:
fixes *B*::*'a*::*comm-ring Matrix.mat*

assumes *diagonal-mat D*
and *D∈ carrier-mat n n*
and *B∈ 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 $B * D = D * B$
shows $B4 * D4 = D4 * B4$
proof
have *d4*: $D4 \in$ *carrier-mat* $(n-a)$ $(n-a)$ **using** *assms*
split-block(*4*)[*of D*] **by** *simp*
have *b4*: $B4 \in$ *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*
show *dim-row* $(B4 * D4)$ = *dim-row* $(D4 * B4)$ **using** *d4 b4* **by** *simp*
show *dim-col* $(B4 * D4)$ = *dim-col* $(D4 * B4)$ **using** *d4 b4* **by** *simp*
fix *i j*
assume $i <$ *dim-row* $(D4 * B4)$
and $j <$ *dim-col* $(D4 * B4)$
have $(B4*D4)$ \$\$$(i,j) = (B*D)$ \$\$$(i+a, j+a)$
using *split-block-times-diag-index*[*of D n B a*] *assms*
‹$i <$ *dim-row* $(D4 * B4)$› ‹$j <$ *dim-col* $(D4 * B4)$› **by** *blast*
also have ... = $(D*B)$ \$\$ $(i+a, j+a)$ **using** *assms* **by** *simp*
also have ... = $(D4*B4)$ \$\$ (i, j)
using *split-block-times-diag-index*[*of D n B a*] *assms*
by (*metis* ‹$i <$ *dim-row* $(D4 * B4)$› ‹$j <$ *dim-col* $(D4 * B4)$›)
finally show $(B4*D4)$ \$\$$(i,j) = (D4*B4)$ \$\$ (i, j) **.**
qed

lemma *commute-diag-mat-zero-comp*:
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 $i < n$
and $j < n$
and \$\$$(i,i) \neq$ \$\$$(j,j)$
shows \$\$$(i,j) = 0$
proof −
have \$\$$(i,j) *$ \$\$$(j,j) = (B*D)$ \$\$$(i,j)$
using *diagonal-mat-mult-index'*[*of B n D*] *assms* **by** *simp*
also have ... = $(D*B)$ \$\$ (i,j) **using** *assms* **by** *simp*
also have ... = \$\$$(i,j) *$ \$\$$(i,i)$
using *diagonal-mat-mult-index assms*
by (*metis Groups.mult-ac*(*2*))
finally have \$\$$(i,j) *$ \$\$$(j,j) =$ \$\$$(i,j) *$ \$\$$(i,i)$ **.**
hence \$\$$(i,j) *$ (\$\$$(j,j) -$ \$\$$(i,i)) = 0$ **by** *auto*
thus \$\$$(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

```
    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 ≤ dim-row A
and (A1, A2, A3, A4) = split-block A n n
shows hermitian A1  unfolding hermitian-def
proof (rule eq-matI, auto)
  have dim-row A = dim-col A using assms
    by (metis carrier-matD(2) hermitian-square)
  show dim-col A1 = dim-row A1 using assms unfolding split-block-def Let-def
    by simp
  thus dim-row A1 = dim-col A1 by simp
  show ⋀i j. i < dim-row A1 ⟹ j < dim-col A1 ⟹
    Complex-Matrix.adjoint A1 $$ (i, j) = A1 $$ (i, j)
  proof −
    fix i j
    assume i < dim-row A1 and j < dim-col A1 note ij = this
    have r: dim-row A1 = n using assms unfolding split-block-def Let-def
      by simp
    have c: dim-col A1 = n using assms unfolding split-block-def Let-def
      by simp
    have Complex-Matrix.adjoint A1 $$ (i, j) = conjugate (A1 $$ (j,i))
      using ij r c unfolding Complex-Matrix.adjoint-def by simp
    also have ... = conjugate (A $$ (j,i)) using assms ij r c
      unfolding split-block-def Let-def by simp
    also have ... = A $$ (i,j) using assms ij r c ‹dim-row A = dim-col A›
      unfolding hermitian-def Complex-Matrix.adjoint-def
      by (metis adjoint-eval assms(1) hermitian-def order-less-le-trans)
    also have ... = A1 $$(i,j) using assms ij r c
      unfolding split-block-def Let-def by simp
    finally show Complex-Matrix.adjoint A1 $$ (i, j) = A1 $$ (i, j) .
  qed
qed

lemma split-block-hermitian-4:
  assumes hermitian A
  and n ≤ dim-row A
and (A1, A2, A3, A4) = split-block A n n
shows hermitian A4  unfolding hermitian-def
proof (rule eq-matI, auto)
  have arc: dim-row A = dim-col A using assms
    by (metis carrier-matD(2) hermitian-square)
  thus dim-col A4 = dim-row A4 using assms unfolding split-block-def Let-def
    by simp
  thus dim-row A4 = dim-col A4 by simp
  show ⋀i j. i < dim-row A4 ⟹ j < dim-col A4 ⟹
```

```
    Complex-Matrix.adjoint A4 $$ (i, j) = A4 $$ (i, j)
  proof −
    fix i j
    assume i < dim-row A4 and j < dim-col A4 note ij = this
    have r: dim-row A4 = dim-row A − n using assms
      unfolding split-block-def Let-def by simp
    have c: dim-col A4 = dim-col A − n using assms
      unfolding split-block-def Let-def by simp
    have Complex-Matrix.adjoint A4 $$ (i, j) = conjugate (A4 $$ (j,i))
      using ij r c arc unfolding Complex-Matrix.adjoint-def by simp
    also have ... = conjugate (A $$ (j +n ,i+n)) using assms ij r c arc
      unfolding split-block-def Let-def by simp
    also have ... = A $$ (i+n,j+n) using assms ij r c arc
      unfolding hermitian-def Complex-Matrix.adjoint-def
      by (metis index-mat(1) less-diff-conv split-conv)
    also have ... = A4 $$(i,j) using assms ij r c
      unfolding split-block-def Let-def by simp
    finally show Complex-Matrix.adjoint A4 $$ (i, j) = A4 $$ (i, j) .
  qed
qed

lemma diag-block-split-block:
  assumes B∈ carrier-mat n n
  and k < n
  and (B1, B2, B3, B4) = split-block B k k
  and B2 = 0_m k (n−k)
  and B3 = 0_m (n−k) k
shows B = diag-block-mat [B1,B4]
proof −
  have dr: dim-row B = k + (n−k) using assms by simp
  have dc: dim-col B = k + (n−k) using assms by simp
  have c1: B1 ∈ carrier-mat k k using assms
    split-block(1)[of B, OF - dr dc] by metis
  have c4: B4 ∈ carrier-mat (n−k) (n−k) using assms
    split-block(4)[of B, OF - dr dc] by metis
  have d4: diag-block-mat [B4] = B4 using diag-block-mat-singleton[of B4]
    by simp
  have B = four-block-mat B1 B2 B3 B4 using assms split-block(3)[of B k ]
    by (metis carrier-matD(1) carrier-matD(2) diff-is-0-eq
        le-add-diff-inverse nat-le-linear semiring-norm(137)
        split-block(5) zero-less-diff)
  also have ... = four-block-mat B1 (0_m k (n−k)) (0_m (n−k) k) B4
    using assms by simp
  also have ... = four-block-mat B1 (0_m k (n−k)) (0_m (n−k) k)
    (diag-block-mat [B4]) using diag-block-mat-singleton[of B4] by simp
  also have ... = diag-block-mat [B1, B4]
    using diag-block-mat.simps(2)[of B1 [B4]] c1 c4
    unfolding Let-def by auto
  finally show ?thesis .
```

qed

5.2 Diagonal block matrices

abbreviation *four-block-diag* **where**
four-block-diag B1 B2 ≡
  (*four-block-mat B1* (*0*$_m$ (*dim-row B1*) (*dim-col B2*))
  (*0*$_m$ (*dim-row B2*) (*dim-col B1*)) *B2*)

lemma *four-block-diag-cong-comp*:
  assumes *dim-row A1* = *dim-row B1*
  and *dim-col A1* = *dim-col B1*
  and *four-block-diag A1 A2* = *four-block-diag B1 B2*
shows *A1* = *B1*
proof (*rule eq-matI*, *auto simp*:*assms*)
  define *A* **where** *A* = *four-block-diag A1 A2*
  define *B* **where** *B* = *four-block-diag B1 B2*
  fix *i j*
  assume *i* < *dim-row B1* **and** *j*<*dim-col B1* **note** *ij*=*this*
  hence *i* <*dim-row A1 j*<*dim-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 ... = *B*$$(*i*, *j*) **using** *assms* **unfolding** *A-def B-def* **by** *simp*
  also have ... = *B1*$$(*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-row A1* = *dim-row B1*
  and *dim-col A1* = *dim-col B1*
  and *four-block-diag A1 A2* = *four-block-diag B1 B2*
shows *A2* = *B2*
proof (*rule eq-matI*)
  define *n* **where** *n*=*dim-row A1*
  define *m* **where** *m* = *dim-col A1*
  define *A* **where** *A* = *four-block-diag A1 A2*
  define *B* **where** *B* = *four-block-diag B1 B2*
  show *dim-row A2* = *dim-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-col A2* = *dim-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-row B2* **and** *j*<*dim-col B2* **note** *ij*=*this*
  hence *i*+*n* < *dim-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-col A
    unfolding A-def n-def m-def four-block-mat-def Let-def
    by (simp add: ‹dim-col A2 = dim-col B2› ij)
  {
    have n ≤ i+n by simp
    have m≤ j+m by simp
    have i + n − n = i by simp
    have j + m − m = j by simp
  } note ijeq = this
  have A2$$(i,j) = A$$(i+n, j+m) using ijeq
    using A-def ‹i + n < dim-row A› ‹j + m < dim-col A› m-def n-def by force
  also have ... = B$$(i+n, j+m) using assms unfolding A-def B-def by simp
  also have ... = B2$$(i,j) using ijeq
    by (metis A-def B-def ‹i + n < dim-row A› ‹j + m < dim-col A›
        add-implies-diff assms(1) assms(2) assms(3) index-mat-four-block(1)
        index-mat-four-block(2) index-mat-four-block(3) m-def n-def
        not-add-less2)
  finally show A2$$(i,j) = B2$$(i,j) .
qed

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

lemma four-block-diagonal:
  assumes dim-row B1 = dim-col B1
  and dim-row B2 = dim-col B2
  and diagonal-mat B1
  and diagonal-mat B2
shows diagonal-mat (four-block-diag B1 B2) unfolding diagonal-mat-def
proof (intro allI impI)
  fix i j
  assume i < dim-row (four-block-diag B1 B2)
  and j < dim-col (four-block-diag B1 B2)
  and i ≠ j note ijprops = this
  show (four-block-diag B1 B2) $$ (i,j) = 0
  proof (cases i < dim-row B1)
    case 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 ‹dim-col (four-block-diag B A) = dim-col A›
      ‹dim-row (four-block-diag B A) = dim-row A›
  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 (0_m nr1 n2) (0_m nr2 n1) D1
  define fb2 where fb2 = four-block-mat A2 (0_m n1 nc2) (0_m n2 nc1) D2
  have fb1 * fb2 = four-block-mat (A1 * A2 + 0_m nr1 n2 * 0_m n2 nc1)
    (A1 * 0_m n1 nc2 + 0_m nr1 n2 * D2) (0_m nr2 n1 * A2 + D1 * 0_m 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 ∈ carrier-mat nr1 n1 using assms by simp
    show D1 ∈ carrier-mat nr2 n2 using assms by simp
   show A2 ∈ carrier-mat n1 nc1 D2 ∈ 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 ∈ carrier-mat (dim-row ?fU) (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 = 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 * (Complex-Matrix.adjoint ?fU) =
      ?fU * (four-block-diag (Complex-Matrix.adjoint U1)
      (Complex-Matrix.adjoint U2))  by simp
    also have ... = four-block-diag
      (U1 * (Complex-Matrix.adjoint U1))
      (U2 * (Complex-Matrix.adjoint U2))
      by (rule mult-four-block-diag, (auto simp add: assms))
    also have ... = four-block-mat
```

```
    (1_m (dim-row U1))
    (0_m (dim-row U1) (dim-row U2))
    (0_m (dim-row U2) (dim-row U1))
    (1_m (dim-row U2)) using assms
    unfolding unitary-def inverts-mat-def
    by simp
  also have ... = 1_m (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) =
  1_m 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 =
  1_m 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 ∈ 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 ∈ 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 ∈ 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 ... = 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 ∈ carrier-mat (dim-row A1) (dim-row A1)
        by (metis assms(1) assms(3) carrier-mat-triv
          unitarily-equiv-carrier(2))
      show B1 ∈ carrier-mat (dim-row A1) (dim-row A1)
        by (metis assms(1) assms(3) carrier-mat-triv
          unitarily-equiv-carrier(1))
      show U2 ∈ carrier-mat (dim-row A2) (dim-row A2)
        by (metis assms(2) assms(4) carrier-mat-triv
          unitarily-equiv-carrier(2))
      show B2 ∈ 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 $i < dim\text{-}row\ ?fB$
show $?fB\ \$\$\ (i,i) \in Reals$
proof (*rule four-block-mat-real-diag*)
show $i < dim\text{-}row\ ?fB$ **using** ‹$i < dim\text{-}row\ ?fB$› **.**
show $\forall i{<}dim\text{-}row\ B1.\ B1\ \$\$\ (i,\ i) \in \mathbb{R}$ **using** *assms*
unfolding *real-diag-decomp-def* **by** *simp*
show $\forall i{<}dim\text{-}row\ B2.\ B2\ \$\$\ (i,\ i) \in \mathbb{R}$ **using** *assms*
unfolding *real-diag-decomp-def* **by** *simp*
show $dim\text{-}row\ B1 = dim\text{-}col\ B1$ **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 $dim\text{-}row\ B2 = dim\text{-}col\ B2$ **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 $length\ Al = length\ Bl$
and $\forall i < length\ Al.\ dim\text{-}col\ (Al!i) = dim\text{-}row\ (Bl!i)$
shows $diag\text{-}block\text{-}mat\ Al * (diag\text{-}block\text{-}mat\ Bl) =$
$(diag\text{-}block\text{-}mat\ (map2\ (*)\ Al\ Bl))$ **using** *assms*
proof (*induct Al arbitrary*: *Bl*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a Al*)
define *A* **where** $A = diag\text{-}block\text{-}mat\ Al$
define *B* **where** $B = diag\text{-}block\text{-}mat\ (tl\ Bl)$
have $0 < length\ Bl$ **using** *Cons* **by** *auto*
hence $Bl = hd\ Bl\ \#\ (tl\ Bl)$ **by** *simp*
have $length\ (tl\ Bl) = length\ Al$ **using** *Cons* **by** *simp*
have *dim*: $\forall i{<}length\ Al.\ dim\text{-}col\ (Al\ !\ i) = dim\text{-}row\ (tl\ Bl\ !\ i)$
proof (*intro allI impI*)
fix *i*
assume $i < length\ Al$
hence $dim\text{-}col\ (Al\ !\ i) = dim\text{-}col\ ((a\#Al)!(Suc\ i))$ **by** *simp*
also have $... = dim\text{-}row\ (Bl!(Suc\ i))$ **using** *Cons*
by (*metis Suc-lessI* ‹$i < length\ Al$› *length-Cons less-Suc-eq*)
also have $... = dim\text{-}row\ (tl\ Bl!i)$
by (*metis* ‹$Bl = hd\ Bl\ \#\ tl\ Bl$› *nth-Cons-Suc*)
finally show $dim\text{-}col\ (Al\ !\ i) = dim\text{-}row\ (tl\ Bl!i)$ **.**
qed
define *C* **where** $C = map2\ (*)\ (a\ \#\ Al)\ Bl$
have $hd\ C = a * hd\ Bl$ **using** ‹$Bl = hd\ Bl\ \#\ tl\ Bl$› **unfolding** *C-def*
by (*metis list.map*(2) *list.sel*(1) *prod.simps*(2) *zip-Cons-Cons*)
have $tl\ C = map2\ (*)\ Al\ (tl\ Bl)$
by (*metis* (*no-types*, *lifting*) *C-def* ‹$Bl = hd\ Bl\ \#\ tl\ Bl$› *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* (λ*A*. *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* (λ*A*. *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 ∀ 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))
    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 length (U#Ul) = length (A#Al) using bu by simp
  moreover have ∀ i<length (A # Al).
        (U#Ul) ! i ∈ carrier-mat (dim-row ((A # Al) ! i)) (dim-col ((A # Al) !
i)) ∧
         unitary ((U#Ul)!i) ∧
         (B#Bl) ! i ∈ carrier-mat (dim-row ((A # Al) ! i)) (dim-col ((A # Al) !
i))
  proof (intro allI impI)
    fix i
    assume i < length (A#Al)
    show (U # Ul) ! i ∈ carrier-mat (dim-row ((A # Al) ! i))
      (dim-col ((A # Al) ! i)) ∧ unitary ((U#Ul)!i) ∧
      (B # Bl) ! i ∈ carrier-mat (dim-row ((A # Al) ! i))
      (dim-col ((A # Al) ! i))
    proof (cases i = 0)
      case True
      then show ?thesis by (simp add: bcar ucar unit)
    next
      case False
      hence ∃ j. i = Suc j by (simp add: not0-implies-Suc)
      from this obtain j where j: i = Suc j by auto
      hence j < length Al using ‹i < length (A#Al)› 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 ∀ *i* < *length Al*. *dim-col* (*Al*!*i*) = *dim-row* (*Bl*!*i*)
and ∀ *i* < *length Al*. *dim-row* (*Bl*!*i*) = *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 ∀ *i*<*length* (*map2* (∗) *Al Bl*). *dim-col* (*map2* (∗) *Al Bl* ! *i*) =
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 ∀ i < length Al. Al!i ∗ (Bl!i) = Bl!i ∗ (Al!i)
  and ∀ i<length Al. dim-col (Al ! i) = dim-row (Bl ! i)
  and ∀ i<length Al. dim-col (Bl ! i) = dim-row (Al ! i)
shows diag-block-mat Al ∗ (diag-block-mat Bl) =
  diag-block-mat Bl ∗ (diag-block-mat Al)
proof −
  have diag-block-mat Al ∗ diag-block-mat Bl =
    diag-block-mat (map2 (∗) Al Bl)
    using diag-block-mat-mult[of Al Bl] assms by simp
  also have ... = diag-block-mat (map2 (∗) Bl Al)
  proof −
    have map2 (∗) Al Bl = map2 (∗) Bl Al
      by (rule map2-commute, auto simp add: assms)
    thus ?thesis by simp
  qed
  also have ... = diag-block-mat Bl ∗ (diag-block-mat Al)
    using diag-block-mat-mult[of Bl Al] assms by simp
  finally show ?thesis .
qed

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

lemma diag-block-mat-cong-hd:
  assumes 0 < length Al
  and length Al = length Bl
  and dim-row (hd Al) = dim-row (hd Bl)
  and dim-col (hd Al) = dim-col (hd Bl)
  and diag-block-mat Al = diag-block-mat Bl
shows hd Al = hd Bl
proof −
  have Al ≠ [] using assms by blast
  hence Al = hd Al#(tl Al) by simp
  hence da:diag-block-mat Al =
    four-block-diag (hd Al) (diag-block-mat (tl Al))
    using diag-block-mat.simps(2)[of hd Al tl Al] unfolding Let-def by simp
  have  Bl ≠ [] using assms by simp
  hence Bl = hd Bl#(tl Bl) by simp
  hence diag-block-mat Bl = four-block-diag (hd Bl) (diag-block-mat (tl Bl))
    using diag-block-mat.simps(2)[of hd Bl tl Bl] unfolding Let-def by simp
  hence four-block-diag (hd Al) (diag-block-mat (tl Al)) =
    four-block-diag (hd Bl) (diag-block-mat (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 < length Al*
and *length Al = length Bl*
and *dim-row (hd Al) = dim-row (hd Bl)*
and *dim-col (hd Al) = dim-col (hd Bl)*
and *diag-block-mat Al = diag-block-mat Bl*
shows *diag-block-mat (tl Al) = diag-block-mat (tl Bl)*
proof −
have *Al ≠ []* **using** *assms* **by** *blast*
hence *Al = hd Al#(tl Al)* **by** *simp*
hence *da:diag-block-mat Al =*
four-block-diag (hd Al) (diag-block-mat (tl Al))
using *diag-block-mat.simps(2)[of hd Al tl Al]* **unfolding** *Let-def* **by** *simp*
have *Bl ≠ []* **using** *assms* **by** *simp*
hence *Bl = hd Bl#(tl Bl)* **by** *simp*
hence *diag-block-mat Bl = four-block-diag (hd Bl) (diag-block-mat (tl Bl))*
using *diag-block-mat.simps(2)[of hd Bl tl Bl]* **unfolding** *Let-def* **by** *simp*
hence *four-block-diag (hd Al) (diag-block-mat (tl Al)) =*
four-block-diag (hd Bl) (diag-block-mat (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 *length Al = length Bl*
and *∀ i<length Al. dim-row (Al ! i) = dim-row (Bl ! i)*
and *∀ i<length Al. dim-col (Al ! i) = dim-col (Bl ! i)*
and *diag-block-mat Al = diag-block-mat Bl*
and *j < 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 <length Bl* **by** *linarith*
hence *Bl = hd Bl#(tl Bl)* **by** *simp*
then show *?case*
proof (*cases j = 0*)
case *True*
hence *(a#Al)!j = hd(a#Al)* **by** *simp*
have *Bl!j= hd Bl* **using** ‹*j = 0*›
by (*metis* ‹*Bl = hd Bl # tl Bl*› *nth-Cons-0*)
have *da*: *diag-block-mat (a#Al) = four-block-diag a (diag-block-mat Al)*
using *diag-block-mat.simps(2)[of a Al]* **unfolding** *Let-def* **by** *simp*
have *db*: *diag-block-mat (hd Bl#(tl Bl)) =*
four-block-diag (hd Bl) (diag-block-mat (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 ∃ k. j = 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.prems(5) Suc-less-SucD ‹j = Suc k› length-Cons)
    show ∀ i<length Al. dim-row (Al ! i) = 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 ∀ i<length Al. dim-col (Al ! i) = 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 length Al = length Bl
  and ∀ i<length Al. dim-row (Al ! i) = dim-col (Al ! i)
  and ∀ i<length Al. dim-row (Al ! i) = dim-row (Bl ! i)
  and ∀ i<length Al. dim-col (Al ! i) = dim-col (Bl ! i)
  and diag-block-mat Al ∗ (diag-block-mat Bl) =
    diag-block-mat Bl ∗ (diag-block-mat Al)
  and i < length Al
shows Al!i ∗ Bl!i = Bl!i ∗ Al!i
proof −
  have diag-block-mat (map2 (∗) Al Bl)=diag-block-mat Al ∗ diag-block-mat Bl
    using diag-block-mat-mult[of Al] assms by simp
  also have ... = diag-block-mat Bl ∗ diag-block-mat Al using assms by simp
  also have ... = diag-block-mat (map2 (∗) Bl Al)
    using diag-block-mat-mult[of Bl] assms by simp
  finally have eq: diag-block-mat (map2 (∗) Al Bl) =
    diag-block-mat (map2 (∗) Bl Al) .
  have (map2 (∗) Al Bl)!i = (map2 (∗) Bl Al)!i
  proof (rule diag-block-mat-cong-comp)
    show length (map2 (∗) Al Bl) = length (map2 (∗) Bl Al)
      using map2-length assms by metis
    show i < length (map2 (∗) Al Bl) using map2-length assms by metis
    show diag-block-mat (map2 (∗) Al Bl) = diag-block-mat (map2 (∗) Bl Al)
      using eq .
    show ∀ i<length (map2 (∗) Al Bl). dim-row (map2 (∗) Al Bl ! i) =
      dim-row (map2 (∗) Bl Al ! i)
      by (simp add: assms(3))
    show ∀ i<length (map2 (∗) Al Bl). dim-col (map2 (∗) Al Bl ! i) =
      dim-col (map2 (∗) Bl Al ! i)
      by (simp add: assms(4))
  qed
  moreover have (map2 (∗) Al Bl)!i = Al!i ∗ Bl!i using assms by simp
  moreover have (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 length Ul = length Bl
  and ∀ i < length Bl. dim-row (Bl!i) = dim-row (Ul!i)
  shows dim-row (diag-block-mat Ul) = dim-row (diag-block-mat Bl)
proof −
  have dim-row (diag-block-mat Ul) = sum-list (map dim-row Ul)
    by (simp add: dim-diag-block-mat(1))
  also have ... = sum-list (map 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 ∀ *i* < *length Bl*. *dim-col* (*Bl*!*i*) = *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 ∀ *i* < *length Al*. *dim-row* (*Al*!*i*) = *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 l› 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 l! 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 l› 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 l› length-Cons not-less-eq)
  also have ... = carrier-mat (l!Suc i) (l! Suc i)
    using nth-Cons-Suc[of hd l tl l i] ‹l = hd l # tl l› 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 l!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
```

lemma *extract-subdiags-diag-elem*:
fixes *B*::*complex Matrix.mat*
assumes *B*∈ *carrier-mat n n*
and *0* < *n*
and *l* ≠ []
and *i* < *length l*
and *j*< *l*!*i*
and *sum-list l* ≤ *n*
and ∀ *j* < *length l*. *0* < *l*!*j*
shows *extract-subdiags B l*!*i* $$ (*j*,*j*) =
diag-mat B!(*n-sum i l* + *j*) **using** *assms*
proof (*induct i arbitrary*: *l B n*)
case *0*
define *a* **where** *a* = *hd l*
have *l* = *a*#(*tl l*) **unfolding** *a-def* **using** *0* **by** *simp*
have *a* ≤ *n* **using** *0* **unfolding** *a-def*
by (*metis a-def dual-order.strict-trans2 elem-le-sum-list*
hd-conv-nth less-le-not-le nat-le-linear)
define *B1* **where** *B1* = *fst* (*split-block B a a*)
define *B2* **where** *B2* = *fst* (*snd* (*split-block B a a*))
define *B3* **where** *B3* = *fst* (*snd* (*snd* (*split-block B a a*)))
define *B4* **where** *B4* = *snd* (*snd* (*snd* (*split-block B a a*)))
have *sp*: *split-block B a a* = (*B1*, *B2*, *B3*, *B4*) **using** *fst-conv snd-conv*
unfolding *B1-def B2-def B3-def B4-def* **by** *simp*
hence *extract-subdiags B l*!*0* = *B1*
using *hd-conv-nth* **unfolding** *Let-def*
by (*metis* ‹*l* = *a* # *tl l*› *extract-subdiags-neq-Nil*
extract-subdiags-not-emp(*1*))
hence *extract-subdiags B l*!*0* $$ (*j*,*j*) = *B*$$ (*j*,*j*)
using *sp 0* **unfolding** *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 ... = *diag-mat B*!*j*
using *0* ‹*a* ≤ *n*› *hd-conv-nth* **unfolding** *diag-mat-def a-def*
by *fastforce*
also have ... = *diag-mat B*!(*n-sum 0 l* + *j*) **by** *simp*
finally show *?case* **.**
next
case (*Suc i*)
show *?case*
proof (*cases length l* = *1*)
case *True*
hence *Suc i* < *0* **using** *Suc* **by** *simp*
then show *?thesis* **by** *simp*
next
case *False*
hence *1* < *length l* **using** *Suc* **by** *presburger*
define *a* **where** *a* = *hd l*
have *l* = *a*#(*tl l*) **unfolding** *a-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 *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 *B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*)
using *sp split-block-diag-carrier*(*2*) *Suc* ‹*a* ≤ *n*› **by** *blast*
have *B1* ∈ *carrier-mat a a*
using *sp split-block-diag-carrier*(*1*) *Suc* ‹*a* ≤ *n*› **by** *blast*
have *n-sum* (*Suc i*) *l* + *j* < *n-sum* (*Suc* (*Suc i*)) *l*
using *Suc n-sum-last-lt* **by** *metis*
hence *a* + *n-sum i* (*tl l*) + *j* < *n-sum* (*Suc* (*Suc i*)) *l*
unfolding *a-def* **by** *simp*
also have ... ≤ *sum-list l*
proof (*rule n-sum-sum-list*)
show ∀ *j*<*length l*. *0* ≤ *l* ! *j* **using** *Suc* **by** *simp*
show *Suc* (*Suc i*) ≤ *length l* **using** *Suc* **by** *simp*
qed
also have ... ≤ *n* **using** *Suc* **by** *simp*
finally have *a* + *n-sum i* (*tl l*) + *j* < *n* **.**
hence *n-sum i* (*tl l*) +*j* < *n* − *a* **by** *simp*
have *extract-subdiags B l*!(*Suc i*) =
extract-subdiags B4 (*tl l*)!*i*
using *sp* ‹*l* = *a*#(*tl l*)› **unfolding** *Let-def*
by (*metis list.exhaust-sel nth-Cons-Suc extract-subdiags-neq-Nil extract-subdiags-not-emp*(*2*))
hence *extract-subdiags B l*!(*Suc i*) $$(*j*,*j*) =
extract-subdiags B4 (*tl l*)!*i* $$(*j*,*j*) **by** *simp*
also have ... = *diag-mat B4*!(*n-sum i* (*tl l*) + *j*)
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 ‹*B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*)› **.**
show *sum-list* (*tl l*) ≤ *n* − *a* **using** *Suc*(*5*) ‹*a* ≤ *n*› *sum-list-tl-leq*
by (*simp add*: *Suc a-def*)
show *0* < *n* − *a*
by (*metis Suc.prems*(*4*) *Suc.prems*(*7*) ‹*i* < *length* (*tl l*)›
‹*l* = *a* # *tl l*› ‹*sum-list* (*tl l*) ≤ *n* − *a*› *bot-nat-0.extremum-uniqueI elem-le-sum-list gr-zeroI nth-Cons-Suc*)
show ∀ *j*<*length* (*tl l*). *0* < *tl l* ! *j*
by (*simp add*: *Suc*(*8*) *nth-tl*)

```
    show j < tl l ! i
      by (metis Suc(6) ‹i < length (tl l)› nth-tl)
  qed
  also have ... = B4$$(n-sum i (tl l)+j, n-sum i (tl l)+j)
  proof −
    have n-sum i (tl l) +j < n − a using ‹n-sum i (tl l) +j < n − a› .
    thus ?thesis
      using ‹B4 ∈ carrier-mat (n−a) (n−a)›
      unfolding diag-mat-def by simp
  qed
  also have ... = B$$(n-sum i (tl l) + j + a, n-sum i (tl l) + j + a)
    using sp ‹B1 ∈ carrier-mat a a› ‹n-sum i (tl l) +j < n − a›
      ‹B4 ∈ carrier-mat (n−a) (n−a)› 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 ‹n-sum i (tl l) +j < n − a›
      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 ≤ dim-row A
 and list-all (λa. 0 < a) l
 shows list-all (λB. 0 < dim-row B ∧ 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-simps(1)[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 = (0_m x (n − x)) ∧ B3 = (0_m (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*∈ *carrier-mat n n*
and *diag-compat B l*
shows *B* = *diag-block-mat* (*extract-subdiags B l*) **using** *assms*
proof (*induct l arbitrary*:*B n*)
case *Nil*
have *extract-subdiags B Nil* = [] **by** *simp*
have *B* = 0_m *0 0*
proof (*rule eq-matI*, *auto simp add*: *assms*)
show *dim-row B* = *0* **using** *Nil* **by** *simp*
show *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* = *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 *a* ≤ *n* **using** *assms Cons* **by** *simp*
have *diag-compat B4 l* **using** *sp Cons* **by** (*simp add*: *Let-def*)
have *B1* ∈ *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* ∈ *carrier-mat a* (*n*−*a*) **using** *sp Cons* **by** (*simp add*: *Let-def*)
have *B3* ∈ *carrier-mat* (*n*−*a*) *a* **using** *sp Cons* **by** (*simp add*: *Let-def*)
have *B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*) **using** *assms* ‹*a* ≤ *n*› *Cons*
split-block(*4*)[*OF sp*] **by** *simp*
have *b2*: 0_m (*dim-row B1*) (*dim-col B4*) = *B2*
using *diag-compat.simps*(*2*)[*THEN iffD1*, *OF* ‹*diag-compat B* (*a*#*l*)›]
‹*B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*)› ‹*B1* ∈ *carrier-mat a a*›
‹*B2* ∈ *carrier-mat a* (*n*−*a*)› *sp* **unfolding** *Let-def*
Cons(*2*) **by** *force*
have *b3*: 0_m (*dim-row B4*) (*dim-col B1*) = *B3*
using *diag-compat.simps*(*2*)[*THEN iffD1*, *OF* ‹*diag-compat B* (*a*#*l*)›]
‹*B4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*)› ‹*B1* ∈ *carrier-mat a a*›
‹*B2* ∈ *carrier-mat a* (*n*−*a*)› *sp* **unfolding** *Let-def*
Cons(*2*) **by** *force*
have *extract-subdiags B* (*a*#*l*) = *B1* # (*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 = *diag-block-mat* (*extract-subdiags B4 l*)
in four-block-mat B1 (0_m (*dim-row B1*) (*dim-col C*))
(0_m (*dim-row C*) (*dim-col B1*)) *C*) **by** *simp*
also have ... = *four-block-mat B1* (0_m (*dim-row B1*) (*dim-col B4*))

```
    (0_m (dim-row B4) (dim-col B1)) B4 using Cons ‹diag-compat B4 l›
    ‹B4 ∈ carrier-mat (n−a) (n−a)› by (simp add:Let-def)
  also have ... = four-block-mat B1 B2 B3 B4 using b2 b3 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-row D = 0 ∧ dim-col D = 0)
| diag-diff D (x#xs) =
    (x ≤ dim-row D ∧
      (let (D1, D2, D3, D4) = (split-block D x x) in
        (∀ i j. i < dim-row D1 ∧ j < dim-row D4 ⟶ D1$$(i,i) ≠ D4 $$ (j,j)) ∧
        diag-diff D4 xs))

lemma diag-diff-hd-diff:
  assumes diag-diff D (a#xs)
  and D∈ carrier-mat n n
  and i < a
  and a ≤ j
  and j < n
shows D$$(i,i) ≠ D $$ (j,j)
proof −
  define D1 where D1 = fst (split-block D a a)
  define D2 where D2 = fst (snd (split-block D a a))
  define D3 where D3 = fst (snd (snd (split-block D a a)))
  define D4 where D4 = snd (snd (snd (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 c1: D1 ∈ carrier-mat a a using split-block(1)[OF spd, of n−a n−a]
      assms by simp
  have c4: D4 ∈ carrier-mat (n−a) (n−a) using assms
      split-block(4)[OF spd] by simp
  hence j − a < dim-row D4 using assms by simp
  have D $$ (i,i) = D1 $$ (i,i) using assms spd
    unfolding split-block-def Let-def by force
  moreover have D $$ (j,j) = D4 $$ (j−a, j − a) using assms spd
    unfolding split-block-def Let-def by force
  moreover have D1 $$ (i,i) ≠ D4 $$ (j−a, j − a)
    using assms ‹j − a < dim-row D4› 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∈ carrier-mat (dim-row B) (dim-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 ∧ 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∈ carrier-mat (dim-row B) (dim-row B) using Cons by simp
    show a ≤ 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 ‹diagonal-mat B1 ∧ diagonal-mat B4› by simp
    show B4 ∈ 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 = 0_m 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 = 0_m (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 $$(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 ≤ dim-row B using Cons by simp
    show let n = dim-row B;
      (B1, B2, B3, B4) = split-block B a a in B2 = 0_m a (n − a) ∧
      B3 = 0_m (n − a) a ∧ diag-compat B4 l
      using sp ‹B3 = 0_m (n − a) a› ‹B2 = 0_m a (n − a)›
        ‹diag-compat B4 l› 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::{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 diag-diff D l
shows diag-compat B l using assms
proof (induct l arbitrary: B D n)
  case Nil
  hence D ∈ carrier-mat 0 0 using assms by simp
  hence n = 0 using assms using Nil(2) by auto
  hence B ∈ carrier-mat 0 0 using Nil by simp
  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 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 = fst (split-block D a a)
  define D2 where D2 = fst (snd (split-block D a a))
  define D3 where D3 = fst (snd (snd (split-block D a a)))
  define D4 where D4 = snd (snd (snd (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 ≤ n using Cons by simp
  moreover have diag-compat B4 l
  proof (rule Cons(1))
    show diagonal-mat D4 using spd Cons ‹a ≤ n›
      split-block-diagonal[of D n a] by blast
```

```
    show D4∈ carrier-mat (n−a) (n−a) using spd Cons(3)
      unfolding split-block-def Let-def by fastforce
    show diag-diff D4 l using spd Cons by simp
    show B4 ∈ carrier-mat (n − a) (n − a) using spb Cons(4)
      unfolding split-block-def Let-def by fastforce
    show B4 ∗ D4 = D4 ∗ B4 using spb Cons ‹a ≤ n›
      split-block-commute-subblock[of D] by (meson spd)
  qed
  moreover have B2 = 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 ‹a ≤ n›))
    fix i j
    assume i < a and a ≤ j and j < n
    thus D $$ (i, i) = D $$ (j, j) ⟹ False
      using diag-diff-hd-diff[OF Cons(6) Cons(3), of i j] by simp
  qed
  moreover have B3 = 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 ‹a ≤ n›))
    fix i j
    assume i < a and a ≤ j and j < n
    thus D $$ (i, i) = D $$ (j, j) ⟹ 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 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≠ []
  shows eq-comps l ≠ [] 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-empty-if*:
assumes *eq-comps l* = []
shows *l* = []
proof (*rule ccontr*)
assume *l*≠ []
hence *eq-comps l* ≠ [] **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* ≠ *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 3 by simp
  next
   case False
   hence x = 1 using 3 unfolding Let-def by simp
   moreover have xs = ec using 3 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)
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
  proof (cases x = y)
   case True
   then show ?thesis
    using 3 eq-comps.simps(3)[of x y l] list-all-simps(1) unfolding Let-def
    by (metis eq-comps-not-empty hd-Cons-tl list.discI zero-less-Suc)
  next
   case False
   then show ?thesis
    using 3 eq-comps.simps(3)[of x y l] list-all-simps(1) unfolding Let-def
    by auto
  qed
qed
```

```
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 3 eq-comps.simps(3)[of x y]
      unfolding ec-def Let-def by simp
    then show ?thesis using 3
      by (metis True ‹ec = hd ec # tl ec› ec-def eq-comps-hd-eq-tl length-Cons
          list.sel(3) not-less-eq-eq)
  next
    case False
    hence a = 1 using 3 by (simp add: Let-def)
    then show ?thesis by simp
  qed
qed

lemma eq-comps-length:
  shows length (eq-comps l) ≤ 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 3 eq-comps.simps(3) True unfolding Let-def
```

```
    by (metis ec-def le-SucI length-Cons)
  next
    case False
    then show ?thesis using ec 3 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 (3 u v l)
  show ?case
  proof (cases u = v)
    case False
    thus ?thesis using 3 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 3 eq-comps.simps(3)[of u v]
      unfolding ec-def Let-def by simp
    hence i ≤ hd ec using 3 by simp
    show ?thesis
    proof (cases i = 0)
      case True
      thus ?thesis by simp
    next
      case False
      hence ∃ 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 ‹i ≤ hd ec› by simp
      have (u # v # l) ! i = (v#l) ! i' using ‹i = Suc i'› by simp
      also have ... = v using 3 ‹ec = eq-comps (v#l)› ‹ec = hd ec # (tl ec)›
        by (metis ‹i' < hd ec› list.sel(1))
      also have ... = hd (u#v#l) using ‹u = v› 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 (3 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 = eq-comps l
 and i < a
 and a ≤ j
 and j < length l
shows nth l i < nth l j using assms
proof (cases m =[])
 case True
 hence [a] = eq-comps l using assms by simp
  hence a = length l using eq-comps-singleton[of a l] by simp
  then show ?thesis using assms by simp
next
 case False
 hence m = hd m # (tl m) by simp
 have l!i = hd l using assms eq-comps-eq by metis
 also have ... < hd (drop a l) using eq-comps-leq assms ‹m = hd m # (tl m)›
  by metis
 also have ... ≤ l!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 eq-comps l = [a]
 shows ∀ i < length l. l!i = l!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 ∀ z ∈ set l. z ∈ Reals
  and m = eq-comps l
shows m = eq-comps (map 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 = eq-comps (y#l)
  have ecr: ec = eq-comps (map 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 ... = eq-comps (map Re (x#y # l)) using ecr ‹Re x = Re y›
      by (simp add: Let-def)
    finally show ?thesis .
  next
    case False
    hence Re x ≠ 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 ... = eq-comps (map Re (x#y # l)) using ecr ‹Re x ≠ Re y›
      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 ‹eq-comps (y # l)›) 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

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 = eq-comps (drop a l) using eq-comps-drop by metis
  hence 0 < b using eq-comps-neq-0 by auto
  moreover have 0 < a using ‹eq-comps l = a#b#c› eq-comps-neq-0
    by (metis gr0I)
  moreover have a+b ≤ length l using eq-comps-sum-list
    by (metis ‹eq-comps l = a # b # c› 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 sum-list (eq-comps (diag-mat A)) = dim-row A
  using eq-comps-sum-list[of diag-mat A] diag-mat-length by simp

lemma nsum-Suc-elem:
  assumes 1 < length (eq-comps l)
  shows l!(n-sum (Suc i) (eq-comps l)) =
    (drop (hd (eq-comps l)) l)!(n-sum i (tl (eq-comps l))) using assms
proof (induct i arbitrary: l)
  case 0
  hence 1 < length l using eq-comps-length[of l] by presburger
  hence l ≠ [] by fastforce
  hence l ! n-sum (Suc 0) (eq-comps l) = l ! hd (eq-comps l)
    by (simp add: 0.prems eq-comps-not-empty hd-conv-nth)
  also have ... = hd (drop (hd (eq-comps l)) l)
    by (metis 0.prems eq-comps-elem-lt hd-drop-conv-nth)
  finally show ?case using 0
    by (metis (no-types, opaque-lifting) ‹l ! hd (eq-comps l) =
      hd (drop (hd (eq-comps l)) l)› ‹l ≠ []› 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-sum (Suc (Suc i)) (eq-comps l)) =
    l!(hd (eq-comps l) + (n-sum (Suc i) (tl (eq-comps l)))) by simp
  also have ... = (drop (hd (eq-comps l)) l)!
    (n-sum (Suc i) (tl (eq-comps l)))
    using less-or-eq-imp-le
    by (metis Suc.prems eq-comps-elem-lt nth-drop)
  finally show ?case .
qed

lemma eq-comps-elems-eq:
  assumes l≠[]
  and i < length (eq-comps l)
  and j < (eq-comps l)!i
shows l!(n-sum i (eq-comps l)) = l!(n-sum i (eq-comps l) + j) using assms
proof (induct i arbitrary: l)
  case 0
  hence eq-comps l = hd (eq-comps l)#(tl (eq-comps l)) by simp
  have l ! n-sum 0 (eq-comps l) = hd l
    by (simp add: 0(1) hd-conv-nth)
  also have ... = 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 Suc i = 0 using Suc.prems(2) by fastforce
    then show ?thesis by simp
  next
    case False
    hence 1 < length (eq-comps l) using Suc eq-comps-not-empty[of l]
      by presburger
    hence l!(n-sum (Suc i) (eq-comps l)) =
    (drop (hd (eq-comps l)) l)!(n-sum i (tl (eq-comps l)))
      using nsum-Suc-elem by simp
    also have ... = (drop (hd (eq-comps l)) l)!
      (n-sum i (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 ... = (drop (hd (eq-comps l)) l)!
      (n-sum i (eq-comps (drop (hd (eq-comps l)) l)) + j)
    proof (rule Suc(1))
      show drop (hd (eq-comps l)) l ≠ []
        by (metis Cons-nth-drop-Suc ‹1 < length (eq-comps l)› eq-comps-elem-lt
```

```
        list.distinct(1))
    show i < 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 < 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 ... = (drop (hd (eq-comps l)) l)!
    (n-sum i (tl (eq-comps l)) + j)
    by (metis Suc(2) eq-comps-drop eq-comps-not-empty hd-Cons-tl)
  also have ... = l!(n-sum (Suc i) (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 < length (eq-comps (diag-mat A))
 shows ∃k. (extract-subdiags A (eq-comps (diag-mat A)))!i =
   k ·m (1m ((eq-comps (diag-mat A))!i))
proof
 define l where l = diag-mat A
 define k where k = l!(n-sum i (eq-comps l))
 show extract-subdiags A (eq-comps (diag-mat A)) ! i =
   k ·m 1m (eq-comps (diag-mat A) ! i)
 proof (rule eq-matI, auto simp add: assms)
   show dr: dim-row (extract-subdiags A (eq-comps (diag-mat A)) ! i) =
     eq-comps (diag-mat A) ! i
     using extract-subdiags-carrier assms carrier-matD(1) by blast
   show dc: dim-col (extract-subdiags A (eq-comps (diag-mat A)) ! i) =
     eq-comps (diag-mat A) ! i
     using extract-subdiags-carrier assms carrier-matD by blast
   fix m np
   assume m < eq-comps (diag-mat A)!i and np < eq-comps (diag-mat A)!i
     and m≠ np note mnp=this
   have diagonal-mat (extract-subdiags A (eq-comps (diag-mat A)) !i)
   proof (rule extract-subdiags-diagonal)
     show diagonal-mat A using assms by simp
     show A∈ carrier-mat n n using assms by simp
```

```
    show eq-comps (diag-mat A) ≠ [] using assms unfolding diag-mat-def
      by auto
    show sum-list (eq-comps (diag-mat A)) ≤ 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 < length (eq-comps (diag-mat A)) using assms by simp
  qed
  thus extract-subdiags A (eq-comps (diag-mat A)) ! i $$ (m, np) = 0
    using mnp dr dc by (metis diagonal-mat-def)
 next
  fix p
  assume p < eq-comps (diag-mat A) ! i
  have extract-subdiags A (eq-comps (diag-mat A)) ! i $$ (p, p) =
    diag-mat A! (n-sum i (eq-comps (diag-mat A)) + p)
  proof (rule extract-subdiags-diag-elem)
    show A∈ carrier-mat n n 0 < n i < length (eq-comps (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** $m = (eq\text{-}comps\ (diag\text{-}mat\ A))!i$
have $\exists k.\ (extract\text{-}subdiags\ A\ (eq\text{-}comps\ (diag\text{-}mat\ A)))!i =$
$k \cdot_m (1_m\ ((eq\text{-}comps\ (diag\text{-}mat\ A))!i))$
using *assms extract-subdiags-eq-comp* **by** *simp*
from *this* **obtain** k **where**
$(extract\text{-}subdiags\ A\ (eq\text{-}comps\ (diag\text{-}mat\ A)))!i =$
$k \cdot_m (1_m\ m)$ **unfolding** *m-def* **by** *auto* **note** *kprop* = *this*
hence $(extract\text{-}subdiags\ A\ (eq\text{-}comps\ (diag\text{-}mat\ A)))!i * B =$
$k \cdot_m (1_m\ m) * B$ **by** *simp*
also have $... = B * (k \cdot_m (1_m\ m))$ **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 $B \in carrier\text{-}mat\ n\ n$
and *diagonal-mat B*
shows $B = diag\text{-}block\text{-}mat\ (extract\text{-}subdiags\ B\ (eq\text{-}comps\ (diag\text{-}mat\ B)))$
proof (*rule diag-compat-extract-subdiag*)
define *eqcl* **where** $eqcl = eq\text{-}comps\ (diag\text{-}mat\ B)$
show $B \in carrier\text{-}mat\ n\ n$ **using** *assms* **by** *simp*
show *diag-compat B eqcl*
proof (*rule diag-compat-diagonal*)
show $B \in carrier\text{-}mat\ (dim\text{-}row\ B)\ (dim\text{-}row\ B)$
using *assms* **by** *simp*
show *diagonal-mat B* **using** *assms* **by** *simp*
have $dim\text{-}row\ B = length\ (diag\text{-}mat\ B)$ **unfolding** *diag-mat-def* **by** *simp*
also have $... = sum\text{-}list\ eqcl$ **using** *eq-comps-sum-list*[*of diag-mat B*]
unfolding *eqcl-def* **by** *simp*
finally show $dim\text{-}row\ B = sum\text{-}list\ eqcl$.
qed
qed

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

lemma *sorted-lst-diff*:
assumes *sorted l*
and $m = eq\text{-}comps\ l$
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* (*drop a l*) **using** *Cons sorted-wrt-drop* **by** *simp*
moreover have m = *eq-comps* (*drop a l*) **using** *eq-comps-drop Cons* **by** *simp*
ultimately have *lst-diff* (*drop a l*) m **using** *Cons* **by** *simp*
have $a \leq$ *length l* **using** *eq-comps-elem-le-length Cons* **by** *simp*
have $(\forall\, i\ j.\ i < a \wedge a \leq j \wedge j <$ *length l* $\longrightarrow$ *nth l i* $<$ *nth l j*$)$
using *Cons eq-comps-compare* **by** *blast*
then show *?case* **using** ‹$a \leq$ *length l*› ‹*lst-diff* (*drop a l*) m› **by** *fastforce*
qed

lemma *lst-diff-imp-diag-diff*:
fixes D::$'a$::*preorder Matrix.mat*
assumes $D\in$ *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** ‹*dim-row D* = *0*› **by** *simp*
next
case (*2 D a xs*)
define *D1* **where** *D1* = *fst* (*split-block D a a*)
define *D2* **where** *D2* = *fst* (*snd* (*split-block D a a*))
define *D3* **where** *D3* = *fst* (*snd* (*snd* (*split-block D a a*)))
define *D4* **where** *D4* = *snd* (*snd* (*snd* (*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* ∈ *carrier-mat a a*
using *split-block*(*1*)[*OF spd*, *of n*−*a n*−*a*] *2* **by** *simp*
have *c4*: *D4* ∈ *carrier-mat* (*n*−*a*) (*n*−*a*)
using *2* *split-block*(*4*)[*OF spd*] ‹$a \leq n$› **by** *simp*
have *diag-mat D* = *diag-mat D1* @ (*diag-mat D4*)
using *diag-four-block-mat split-block*(*5*)
by (*metis 2*(*2*) ‹$a \leq n$› *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* = *drop a* (*diag-mat D*)
using ‹*diag-mat D* = *diag-mat D1* @ (*diag-mat D4*)› **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  (∀ i j. i < dim-row D1 ∧ j < dim-row D4 ⟶ D1$$(i,i) < D4 $$ (j,j))
  proof (intro allI impI)
    fix i j
    assume ijp: i < dim-row D1 ∧ j < 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 ‹i < a›
      unfolding split-block-def Let-def by force
    also have ... = (diag-mat D)!i using ‹i < a› ‹a ≤ n› 2
      unfolding diag-mat-def by simp
    also have ... < (diag-mat D)!(j+a) using 2 ‹j+a < n›
      by (metis ‹i < a› ‹length (diag-mat D) = n›
          le-add2 lst-diff.simps(2))
    also have ... = D $$ (j+a, j+a) using ‹j+a < n› 2
      unfolding diag-mat-def by simp
    also have ... = D4 $$ (j,j) using spd ijp 2
      unfolding split-block-def Let-def by force
    finally show D1$$(i,i) < D4 $$ (j,j) .
  qed
  hence (∀ i j. i < dim-row D1 ∧ j < dim-row D4 ⟶ D1$$(i,i) ≠ D4 $$ (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::linorder Matrix.mat
  assumes D∈ 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 ∈ 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::complex list
  assumes ∀ z ∈ set l. z ∈ Reals
  and sorted (map Re l)
  and m = 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 (map Re (drop a l)) using Cons sorted-wrt-drop
    by (metis drop-map)
```

moreover have m = *eq-comps* (*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 $a \leq$ *length l* **using** *eq-comps-elem-le-length Cons* **by** *simp*
have ($\forall\, i\ j.\ i < a \wedge a \leq j \wedge j <$ *length l* $\longrightarrow$ *nth l i* $<$ *nth l j*)
proof (*intro allI impI*)
fix *i j*
assume *asm*: $i < a \wedge a \leq j \wedge j <$ *length l*
hence *Re* (*l*!*i*) $<$ *Re* (*l*!*j*)
using *Cons eq-comps-compare eq-comp-Re*
by (*smt* (*z3*) *dual-order.strict-trans dual-order.strict-trans1 length-map nth-map*)
moreover have *l*!*i* $\in$ *Reals* **using** *asm Cons* **by** *simp*
moreover have *l*!*j* $\in$ *Reals* **using** *asm Cons* **by** *simp*
ultimately show *nth l i* $<$ *nth l j* **using** *less-complex-def*
by (*simp add*: *complex-is-Real-iff*)
qed
then show *?case* **using** ‹$a \leq$ *length l*› ‹*lst-diff* (*drop a l*) *m*› **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*) (λ (*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*)$$(*i*,*j*)
unfolding *Matrix.col-def* **by** *simp*
also have ... = *A* $$(*i*, (*f j*)) **unfolding** *per-col-def*
using ‹*i* < *dim-vec* (*Matrix.col A* (*f j*))› *assms* **by** *fastforce*
also have ... = *vec-index* (*Matrix.col A* (*f j*)) *i* **unfolding** *Matrix.col-def*
using ‹*i* < *dim-vec* (*Matrix.col A* (*f j*))› **by** *auto*
finally show *vec-index* (*Matrix.col* (*per-col A f*) *j*) *i* =
vec-index (*Matrix.col A* (*f j*)) *i* **.**
qed

lemma *per-col-adjoint-row*:
assumes $A \in$ *carrier-mat n n*
and $i < n$
and $f\ i < n$
shows *Matrix.row* (*Complex-Matrix.adjoint* (*per-col A f*)) *i* =
Matrix.row (*Complex-Matrix.adjoint A*) (*f i*)
proof −
have *per-col A f* ∈ *carrier-mat n n* **using** *assms per-col-carrier*[*of A*]
by *simp*
hence *Matrix.row* (*Complex-Matrix.adjoint* (*per-col A f*)) *i* =
conjugate (*Matrix.col* (*per-col A f*) *i*)
using *assms adjoint-row*[*of i per-col A f*] **by** *simp*
also have ... = *conjugate* (*Matrix.col A* (*f i*)) **using** *assms per-col-col*
by *simp*
also have ... = *Matrix.row* (*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 $A \in$ *carrier-mat n n*
and $i < n$
and $j < n$
and $f\ i < n$
and $f\ j < n$
shows ((*Complex-Matrix.adjoint* (*per-col A f*)) ∗ (*per-col A f*))$$(*i*,*j*) =
((*Complex-Matrix.adjoint A*) ∗ *A*)$$(*f i*, *f j*)

proof −
have ((*Complex-Matrix.adjoint* (*per-col A f*)) ∗ (*per-col A f*))$$(*i*,*j*) =
Matrix.scalar-prod (*Matrix.row* (*Complex-Matrix.adjoint* (*per-col A f*)) *i*)
(*Matrix.col A* (*f j*)) **using** *assms per-col-col* **unfolding** *times-mat-def*
by (*metis adjoint-dim-row carrier-matD*(*2*) *dim-col-mat*(*1*) *index-mat*(*1*)
old.prod.case per-col-def)
also have ... = *Matrix.scalar-prod*
(*Matrix.row* (*Complex-Matrix.adjoint A*) (*f i*))
(*Matrix.col A* (*f j*)) **using** *assms per-col-adjoint-row* **by** *metis*
also have ... = ((*Complex-Matrix.adjoint A*) ∗ *A*)$$(*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** ‹*f i* < *n*› *assms* **by** *simp*
next
case *False*
hence *f i* ≠ *f j* **using** *assms*
by (*metis bij-betw-iff-bijections lessThan-iff*)
then show *?thesis*
by (*metis* ‹*f i* < *n*› ‹*f j* < *n*› *assms*(*2*) *assms*(*3*) *index-one-mat*(*1*))
qed
qed

lemma *per-col-unitary*:
assumes *A*∈ *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* ∈
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 < dim-row (1_m n) and j < dim-col (1_m n) note ij = 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∈ 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 ‹j < n› 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) (λ (i,j). A $$(f i, (f j)))

lemma per-diag-carrier:
 shows per-diag A f ∈ carrier-mat (dim-row A) (dim-col A)
 unfolding per-diag-def by simp

lemma per-diag-diagonal:
 assumes D∈ 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 < dim-row (per-diag D f) and j < dim-col (per-diag D f)
 and i≠ j note asm = this
 hence f i ≠ 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 ∈ carrier-mat n n
  and i < n
  and f i < n
shows diag-mat (per-diag A f)!i = diag-mat A ! (f i)
  using assms unfolding diag-mat-def per-diag-def by auto

lemma per-diag-diag-mat-Re:
  assumes A ∈ carrier-mat n n
  and i < n
  and f i < n
shows map Re (diag-mat (per-diag A f))!i = map Re (diag-mat A) ! (f i)
proof −
  have map Re (diag-mat (per-diag A f))!i = Re (diag-mat (per-diag A f)!i)
  proof (rule nth-map)
    show i < length (diag-mat (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 ... = Re (diag-mat A ! (f i)) using assms per-diag-diag-mat
    by metis
  also have ... = map Re (diag-mat A) ! (f i) unfolding diag-mat-def
    using assms by auto
  finally show ?thesis .
qed

lemma per-diag-real:
  fixes B::complex Matrix.mat
  assumes B∈ carrier-mat n n
  and ∀ i< n. B$$(i,i) ∈ Reals
  and bij-betw f {..<n} {..<n}
shows ∀ j<n. (per-diag B f)$$(j,j) ∈ Reals
proof (intro allI impI)
  fix j
  assume j < n
  hence per-diag B f $$ (j,j) = B $$ (f j, f j)
    using assms unfolding per-diag-def by simp
  also have ... ∈ Reals using assms ‹j < n› bij-betwE by blast
```

finally show *per-diag B f* $$ (*j*,*j*) ∈ *Reals* **.**
qed

lemma *per-col-mult-unitary*:
fixes *A*::*complex Matrix.mat*
assumes *A* ∈ *carrier-mat n n*
and *unitary A*
and *D*∈ *carrier-mat n n*
and *diagonal-mat D*
and *0* < *n*
and *bij-betw f* {..< *n*} {..< *n*}
shows *A* ∗ *D* ∗ (*Complex-Matrix.adjoint A*) =
(*per-col A f*) ∗ (*per-diag D f*) ∗ (*Complex-Matrix.adjoint* (*per-col A f*))
(**is** *?L* = *?R*)
proof −
have *row*: *dim-row ?L* = *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 = (λ*i*. (*if i* < *n then diag-mat D* ! *i* ·$_m$ *rank-1-proj* (*Matrix.col A i*)
else (*0*$_m$ *n n*)))
define *g* **where**
g = (λ*i*. (*if i* < *n*
then diag-mat D ! (*f i*) ·$_m$ *rank-1-proj* (*Matrix.col A* (*f i*))
else (*0*$_m$ *n n*)))
have *f* ‘ {..<*n*} = {..<*n*} **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* {..<*n*} **using** *assms*(*6*) *bij-betw-def* **by** *auto*
moreover have ⋀*x*. *x* ∈ {..<*n*} ⟹ *h* (*f x*) = *g x* **unfolding** *h-def g-def*
by (*meson assms*(*6*) *bij-betwE lessThan-iff*)
ultimately have *sum-mat g* {..<*n*} = *sum-mat h* (*f*‘{..<*n*})
using *sum-with-reindex-cong*[*of h g f* {..<*n*}]
unfolding *sum-mat-def* **by** *simp*
also have ... = *sum-mat h* {..<*n*} **using** ‹*f* ‘ {..<*n*} = {..<*n*}› **by** *simp*
also have ... =*sum-mat* (λ*i*. *diag-mat D* ! *i* ·$_m$ *rank-1-proj* (*Matrix.col A i*))
{..<*n*}

proof (*rule sum-mat-cong*, (*auto simp add:h h-def*))
show $\bigwedge i.\ i < n \Longrightarrow$ *diag-mat D* ! *i* $\cdot_m$ *rank-1-proj* (*Matrix.col A i*) $\in$ *fc*
using *assms* **unfolding** *fc-def* **by** (*metis fc-mats-carrier h h-def*)
show $\bigwedge i.\ i < n \Longrightarrow$ *diag-mat D* ! *i* $\cdot_m$ *rank-1-proj* (*Matrix.col A i*) $\in$ *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* {..<*n*} = *A* $*$ *D* $*$ (*Complex-Matrix.adjoint A*) **.**
have (*per-col A f*) $*$ (*per-diag D f*) $*$
(*Complex-Matrix.adjoint* (*per-col A f*)) =
sum-mat (λ*i*. *diag-mat* (*per-diag D f*) ! *i* $\cdot_m$
rank-1-proj (*Matrix.col* (*per-col A f*) *i*)) {..<*n*}
proof (*rule weighted-sum-rank-1-proj-unitary*[*symmetric*])
show *per-col A f* $\in$ *fc* **using** *per-col-carrier*[*of A*] *assms* **unfolding** *fc-def*
by *simp*
show *per-diag D f* $\in$ *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 ... = *sum-mat g* {..<*n*}
proof (*rule sum-mat-cong*, (*auto simp add*: *g-def*))
show $\bigwedge i.\ i < n \Longrightarrow$ *diag-mat* (*per-diag D f*) ! *i* $\cdot_m$
rank-1-proj (*Matrix.col* (*per-col A f*) *i*) $\in$ *fc*
proof −
fix *i*
assume *i* < *n*
have *dim-vec* (*Matrix.col* (*per-col A f*) *i*) = *n* **using** *assms per-col-col*
by (*metis carrier-matD*(*1*) *dim-col*)
hence *rank-1-proj* (*Matrix.col* (*per-col A f*) *i*) $\in$ *fc* **unfolding** *fc-def*
using *rank-1-proj-carrier* **by** *blast*
thus *diag-mat* (*per-diag D f*) ! *i* $\cdot_m$
rank-1-proj (*Matrix.col* (*per-col A f*) *i*) $\in$ *fc* **unfolding** *fc-def* **by** *simp*
qed
show $\bigwedge i.\ i < n \Longrightarrow$ *diag-mat D* ! *f i* $\cdot_m$
rank-1-proj (*Matrix.col A* (*f i*)) $\in$ *fc*
proof −
fix *i*
assume *i* < *n*
hence *f i* < *n* **using** ‹*f* ‘ {..<*n*} = {..<*n*}› **by** *auto*
hence *dim-vec* (*Matrix.col A* (*f i*)) = *n* **using** *assms*
by (*metis carrier-matD*(*1*) *dim-col*)
hence *rank-1-proj* (*Matrix.col A* (*f i*)) $\in$ *fc* **unfolding** *fc-def*
using *rank-1-proj-carrier* **by** *blast*
thus *diag-mat D* ! *f i* $\cdot_m$ *rank-1-proj* (*Matrix.col A* (*f i*)) $\in$ *fc*
unfolding *fc-def* **by** *simp*
qed

```
    show ⋀i. i < n ⟹
        diag-mat (per-diag D f) ! i ·_m
          rank-1-proj (Matrix.col (per-col A f) i) =
        diag-mat D ! f i ·_m rank-1-proj (Matrix.col A (f i))
    proof−
      fix i
      assume i < n
      hence f i < n using ‹f ‘ {..<n} = {..<n}› by auto
      have Matrix.col (per-col A f) i = Matrix.col A (f i)
        using per-col-col assms ‹i < n› by simp
      hence rank-1-proj (Matrix.col (per-col A f) i) =
        rank-1-proj (Matrix.col A (f i)) by simp
      thus diag-mat (per-diag D f) ! i ·_m
        rank-1-proj (Matrix.col (per-col A f) i) =
        diag-mat D ! f i ·_m rank-1-proj (Matrix.col A (f i))
        using assms per-diag-diag-mat[of D] ‹i < n› ‹f i < n› by simp
    qed
  qed
  also have ... = A * D * (Complex-Matrix.adjoint A) using sg by simp
  finally have (per-col A f) * (per-diag D f) *
    (Complex-Matrix.adjoint (per-col A f)) =
    A * D * (Complex-Matrix.adjoint A) .
  thus ?thesis by simp
qed

lemma sort-permutation:
  assumes m = sort l
  obtains f where bij-betw f {..<length l} {..<length l} ∧
    (∀ i<length l. l ! f i = m ! i)
proof −
  have length l = length m using assms by simp
   have mset l = mset m using assms by simp
  from this obtain p where p permutes {..<length m} permute-list p l = m
    by (metis ‹length l = length m› mset-eq-permutation) note pprop = this
  have bij-betw p {..<length l} {..<length l}
    using pprop ‹length l = length m›
    by (simp add: permutes-imp-bij)
  moreover have ∀ i<length l. l ! p i = m ! i using pprop
    by (metis ‹length l = length m› permute-list-nth)
  ultimately have ∃f. bij-betw f {..<length l} {..<length l} ∧
    (∀ i<length l. l ! f i = m ! i) by auto
  thus ?thesis using that by auto
qed

lemma per-diag-sorted-Re:
  fixes B::complex Matrix.mat
  assumes B∈ carrier-mat n n
  obtains f where bij-betw f {..< n} {..< n} ∧
    map Re (diag-mat (per-diag B f)) = sort (map Re (diag-mat B))
```

proof −
define m **where** $m = sort\ (map\ Re\ (diag\text{-}mat\ B))$
have $length\ m = length\ (map\ Re\ (diag\text{-}mat\ B))$ **unfolding** $m\text{-}def$ **by** $simp$
also have $... = length\ (diag\text{-}mat\ B)$ **by** $simp$
also have $... = n$ **using** $assms$ **unfolding** $diag\text{-}mat\text{-}def$ **by** $simp$
finally have $length\ m = n$ **.**
from $this$ **obtain** f **where** $bij\text{-}betw\ f\ \{..<n\}\ \{..<n\} \wedge$
$(\forall\, i{<}n.\ (map\ Re\ (diag\text{-}mat\ B))\ !\ f\ i = m\ !\ i)$
using $sort\text{-}permutation[of\ m\ map\ Re\ (diag\text{-}mat\ B)]\ m\text{-}def$ **by** $auto$
note $fprop = this$
have l: $length\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)) = length\ m$
using $per\text{-}diag\text{-}carrier\ assms\ \langle length\ m = n\rangle$ **unfolding** $diag\text{-}mat\text{-}def$
by $(metis\ carrier\text{-}matD(1)\ length\text{-}map\ map\text{-}nth)$
have $map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)) = m$
proof $(rule\ list\text{-}eq\text{-}iff\text{-}nth\text{-}eq[THEN\ iffD2],\ intro\ conjI)$
show $length\ (map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f))) = length\ m$ **using** l **by** $simp$
have $\forall\, i{<}n.\ (map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)))!i = m!i$
proof $(intro\ allI\ impI)$
fix i
assume $i{<}\ n$
have $(map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)))!i = (map\ Re\ (diag\text{-}mat\ B))!(f\ i)$
proof $(rule\ per\text{-}diag\text{-}diag\text{-}mat\text{-}Re)$
show $B{\in}\ carrier\text{-}mat\ n\ n$ **using** $assms$ **by** $simp$
show $i < n$ **using** $\langle i < n\rangle$ **.**
thus $f\ i < n$ **using** $fprop\ bij\text{-}betwE$ **by** $blast$
qed
also have $... = m!i$ **using** $fprop\ \langle i < n\rangle$ **by** $simp$
finally show $(map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)))!i = m!i$ **.**
qed
thus $\forall\, i{<}length\ (map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)))$.
$(map\ Re\ (diag\text{-}mat\ (per\text{-}diag\ B\ f)))!i = m\ !\ i$ **using** $l\ \langle length\ m = n\rangle$ **by** $simp$
qed
thus $?thesis$ **using** $that\ fprop$ **unfolding** $m\text{-}def$ **by** $auto$
qed

lemma $bij\text{-}unitary\text{-}diag$:
fixes A::$complex\ Matrix.mat$
assumes $unitary\text{-}diag\ A\ B\ U$
and $A \in carrier\text{-}mat\ \ n\ n$
and $bij\text{-}betw\ f\ \{..<n\}\ \{..<n\}$
and $0 < n$
shows $unitary\text{-}diag\ A\ (per\text{-}diag\ B\ f)\ (per\text{-}col\ U\ f)$
proof $(intro\ unitary\text{-}diagI)$
show $unitary\ (per\text{-}col\ U\ f)$ **using** $assms\ unitary\text{-}diagD(1)$
$unitary\text{-}diagD(3)\ per\text{-}col\text{-}unitary$ **by** $(metis\ similar\text{-}mat\text{-}witD2(6))$
show $diagonal\text{-}mat\ (per\text{-}diag\ B\ f)$
using $assms\ unitary\text{-}diagD(2)\ per\text{-}diag\text{-}diagonal$
by $(metis\ similar\text{-}mat\text{-}witD2(5)\ unitary\text{-}diagD(1))$
have $A = U * \ B * (Complex\text{-}Matrix.adjoint\ U)$ **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)
      per-col-carrier[of U]
      by (metis similar-mat-witD2(6))
    moreover hence Complex-Matrix.adjoint (per-col U f) ∈ carrier-mat n n
      by (simp add: adjoint-dim)
    ultimately show
      {A, per-diag B f, per-col U f, Complex-Matrix.adjoint (per-col U f)} ⊆
      carrier-mat (dim-row A) (dim-row A) by auto
    show per-col U f * Complex-Matrix.adjoint (per-col U f) = 1_m (dim-row A)
      using ‹Complex-Matrix.unitary (per-col U f)› assms(2)
        ‹per-col U f ∈ carrier-mat n n› by auto
    show Complex-Matrix.adjoint (per-col U f) * per-col U f = 1_m (dim-row A)
      using ‹Complex-Matrix.unitary (per-col U f)› assms
        ‹per-col U f ∈ carrier-mat n n›  by auto
  qed (simp add: eq)
qed

lemma hermitian-real-diag-sorted:
  assumes A ∈ carrier-mat n n
  and 0 < n
  and hermitian A
  obtains Bs Us where real-diag-decomp A Bs Us ∧ sorted (map Re (diag-mat
Bs))
proof −
  obtain U1 B1 where real-diag-decomp A B1 U1
    using hermitian-real-diag-decomp[of A] assms by auto note ubprop = this
  hence B1 ∈ carrier-mat n n using assms unfolding real-diag-decomp-def
    by (meson unitary-diag-carrier(1))
  from this obtain f where bij-betw f {..< n} {..< n} ∧
      map Re (diag-mat (per-diag B1 f)) = sort (map Re (diag-mat B1))
    using per-diag-sorted-Re by auto note fprop = this
  define Bs where Bs = per-diag B1 f
  define Us where Us = per-col U1 f
```

have *unitary-diag A Bs Us* **unfolding** *Bs-def Us-def*
proof (*rule bij-unitary-diag*)
show *unitary-diag A B1 U1* **using** *ubprop* **unfolding** *real-diag-decomp-def*
by *simp*
show $A \in$ *carrier-mat n n* **using** *assms* **by** *simp*
show *bij-betw f* $\{..<n\}$ $\{..<n\}$ **using** *fprop* **by** *simp*
show $0 < n$ **using** *assms* **by** *simp*
qed
have *real-diag-decomp A Bs Us* **unfolding** *real-diag-decomp-def*
proof (*simp add*: ‹*unitary-diag A Bs Us*›)
show $\forall i<$*dim-row Bs*. *Bs* $\$\$ (i, i) \in \mathbb{R}$ **unfolding** *Bs-def*
proof (*rule per-diag-real*)
show *br*: *B1* $\in$ *carrier-mat* (*dim-row* (*per-diag B1 f*))
(*dim-row* (*per-diag B1 f*))
by (*metis* ‹*B1* $\in$ *carrier-mat n n*› *carrier-matD*(*1*) *per-diag-carrier*)
thus $\forall i<$*dim-row* (*per-diag B1 f*). *B1* $\$\$ (i, i) \in \mathbb{R}$ **using** *ubprop* **by** *auto*
show *bij-betw f* $\{..<$*dim-row* (*per-diag B1 f*)$\}$
$\{..<$*dim-row* (*per-diag B1 f*)$\}$ **using** *fprop*
by (*metis br* ‹*B1* $\in$ *carrier-mat n n*› *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 $A \in$ *carrier-mat n n*
and *real-diag-decomp A B U*
and $B =$ *diag-block-mat Bl*
and *length Ul* = *length Bl*
and $\forall i <$ *length Bl*. *dim-col* (*Bl*!*i*) = *dim-row* (*Bl*!*i*)
and $\forall i <$ *length Bl*. *dim-row* (*Bl*!*i*) = *dim-row* (*Ul*!*i*)
and $\forall i <$ *length Bl*. *dim-col* (*Bl*!*i*) = *dim-col* (*Ul*!*i*)
and *unitary* (*diag-block-mat Ul*)
and $\forall i<$*length Ul*. *Ul* ! *i* $*$ *Bl* ! *i* = *Bl* ! *i* $*$ *Ul* ! *i*
shows *real-diag-decomp A B* (*U* $*$ (*diag-block-mat Ul*))
unfolding *real-diag-decomp-def*
proof (*intro conjI allI impI*)
have *B*$\in$ *carrier-mat n n*
by (*meson assms*(*1*) *assms*(*2*) *real-diag-decompD*(*1*) *unitary-diag-carrier*(*1*))

```
  have dim-row (diag-block-mat Ul) = dim-row B
    using diag-block-mat-dim-row-cong assms by blast
  moreover have dim-col (diag-block-mat Ul) = dim-col B
    using diag-block-mat-dim-col-cong assms by blast
  ultimately have diag-block-mat Ul ∈ carrier-mat n n using assms
    by (metis ‹B ∈ carrier-mat n n› carrier-matD(1) carrier-matD(2)
        carrier-mat-triv)
  define Uf where Uf = U * (diag-block-mat Ul)
  show ⋀i. i < dim-row B ⟹ B $$ (i, i) ∈ ℝ
    using assms real-diag-decompD(2) ‹B ∈ carrier-mat n n› 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∈ 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 ∈ carrier-mat n n
        using ‹diag-block-mat Ul ∈ carrier-mat n n› .
      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 ‹unitary (diag-block-mat Ul)› .
        show diag-block-mat Bl ∈ carrier-mat n n
          using assms ‹B ∈ carrier-mat n n› by simp
        show diag-block-mat Ul ∈ carrier-mat n n
          using ‹diag-block-mat Ul ∈ carrier-mat n n› .
        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: ∀ i<length Ul. Ul ! i * Bl ! i = Bl ! i * Ul ! i
            using assms by simp
          show ∀ i<length Ul. dim-col (Ul ! i) = dim-row (Bl ! i)
            using assms by presburger
          thus ∀ 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 ‹B = diag-block-mat Bl› 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 $\forall\, i < n.\ \forall\ Al \in Als.\ dim\text{-}row\ (Al!i) = dim\text{-}col\ (Al!i)$
and $\forall\, i < n.\ \exists\, U.\ \forall\ Al \in Als.\ \exists\, B.\ real\text{-}diag\text{-}decomp\ (Al!i)\ B\ U$
shows $\exists\, Ul.\ (length\ Ul = n \wedge (\forall\, i < n.\ \forall\, Al \in Als.$
$(dim\text{-}row\ (Ul!i) = dim\text{-}row\ (Al!i) \wedge dim\text{-}col\ (Ul!i) = dim\text{-}col\ (Al!i))) \wedge$
$(\forall\ Al \in Als.\ \exists\, Bl.\ (length\ Bl = n \wedge$
$real\text{-}diag\text{-}decomp\ (diag\text{-}block\text{-}mat\ Al)\ (diag\text{-}block\text{-}mat\ Bl)\ (diag\text{-}block\text{-}mat\ Ul))))$
using *assms*
proof (*induct n arbitrary*: *Als*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
hence $\exists\, U.\ \forall\ Al \in Als.\ \exists\, B.\ real\text{-}diag\text{-}decomp\ (Al!0)\ B\ U$ **by** *simp*
from *this* **obtain** *U0* **where** $\forall\ Al \in Als.\ \exists\, B.\ real\text{-}diag\text{-}decomp\ (Al!0)\ B\ U0$
by *auto* **note** *u0* = *this*
have *u0-dim*: $\forall\, Al \in Als.$
$dim\text{-}row\ ([U0]\ !\ 0) = dim\text{-}row\ (Al\ !\ 0) \wedge$
$dim\text{-}col\ ([U0]\ !\ 0) = dim\text{-}col\ (Al\ !\ 0)$
proof (*intro allI impI ballI*)
fix *Al*
assume $Al \in Als$
have $[U0]!0 = U0$ **by** *simp*
have $\exists\, B.\ real\text{-}diag\text{-}decomp\ (Al!0)\ B\ U0$ **using** *u0* ‹$Al \in Als$›
by *simp*
from *this* **obtain** *B* **where** $real\text{-}diag\text{-}decomp\ (Al!0)\ B\ U0$ **by** *auto*
moreover have $dim\text{-}row\ (Al!0) = dim\text{-}col\ (Al!0)$ **using** ‹$Al \in Als$› *Suc*
by *simp*
ultimately have $dim\text{-}row\ U0 = dim\text{-}row\ (Al!0)$
using *unitary-diag-carrier*(*2*)
by (*metis carrier-matD*(*1*) *carrier-matI real-diag-decompD*(*1*))
moreover have $dim\text{-}col\ U0 = dim\text{-}col\ (Al!0)$
using ‹$real\text{-}diag\text{-}decomp\ (Al!0)\ B\ U0$› ‹$dim\text{-}row\ (Al!0) = dim\text{-}col\ (Al!0)$›
using *real-diag-decompD*(*1*) *unitary-diag-carrier*(*2*)
by (*metis carrier-matD*(*2*) *carrier-mat-triv*)
ultimately show $dim\text{-}row\ ([U0]\ !\ 0) = dim\text{-}row\ (Al\ !\ 0) \wedge$
$dim\text{-}col\ ([U0]\ !\ 0) = dim\text{-}col\ (Al\ !\ 0)$
by *simp*
qed
show *?case*
proof (*cases n* = *0*)
case *True*
hence $\forall\ Al \in Als.\ diag\text{-}block\text{-}mat\ Al = Al!0$ **using** *Suc*
by (*simp add*: *diag-block-mat-length-1*)
have $\forall\ Al \in Als.\ \exists\, Bl.\ (length\ Bl = 1 \wedge$

```
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
    (diag-block-mat [U0]))
  proof
    fix Al
    assume Al ∈ Als
    hence ∃ B. real-diag-decomp (Al!0) B U0 using u0 by simp
    from this obtain B where real-diag-decomp (Al!0) B U0 by auto
    hence real-diag-decomp (diag-block-mat Al) (diag-block-mat [B])
      (diag-block-mat [U0])
      by (metis ‹Al ∈ Als› ‹∀ Al∈Als. diag-block-mat Al = Al ! 0›
          diag-block-mat-singleton)
    moreover have length [B] = 1 by simp
    ultimately show ∃ Bl. (length Bl = 1 ∧
      real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
      (diag-block-mat [U0]))
      by blast
  qed
  moreover have length [U0] = 1 by simp
  moreover have ∀ i<Suc n. ∀ Al∈Als.
    dim-row ([U0] ! i) = dim-row (Al ! i) ∧
    dim-col ([U0] ! i) = dim-col (Al ! i)
    using u0-dim ‹n = 0› by simp
  ultimately have length [U0] = Suc 0 ∧
    (∀ i<Suc n. ∀ Al∈Als. dim-row ([U0] ! i) = dim-row (Al ! i) ∧
      dim-col ([U0] ! i) = dim-col (Al ! i)) ∧
    (∀ Al∈Als. ∃ Bl. length Bl = Suc n ∧ real-diag-decomp
      (diag-block-mat Al) (diag-block-mat Bl) (diag-block-mat [U0]))
    using ‹n=0› One-nat-def by metis
  thus ?thesis by (metis True)
next
  case False
  hence 0 < n by simp
  define tAls where tAls = tl ‘Als
  have tex: ∀ tAl∈ tAls. ∃ Al ∈ Als. tAl = tl Al unfolding tAls-def by auto
  have ∃ Ul. length Ul = n ∧
    (∀ i <n. ∀ Al ∈ tAls.
    (dim-row (Ul!i) = dim-row (Al!i) ∧ dim-col (Ul!i) = dim-col (Al!i)))∧
      (∀ Al∈tAls. ∃ Bl. length Bl = n ∧
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
    (diag-block-mat Ul))
  proof (rule Suc(1))
    show tAls ≠ {} 0 < n unfolding tAls-def by (auto simp add: ‹0 < n› Suc)
    show ∀ tAl∈tAls. length tAl = n
      using Suc(4) tex by fastforce
    show ∀ i<n. ∃ U. ∀ Al∈tAls. ∃ B. real-diag-decomp (Al ! i) B U
    proof (intro allI impI)
      fix i
      assume i < n
      hence ∃ U. ∀ Al∈Als. ∃ B. real-diag-decomp (Al ! (Suc i)) B U
```

```
    using Suc Suc-mono by presburger
  from this obtain U where
    tu: ∀ Al∈Als. ∃ B. real-diag-decomp (Al ! (Suc i)) B U by auto
  have ∀ tAl∈tAls. ∃ B. real-diag-decomp (tAl ! i) B U
  proof
    fix tAl
    assume tAl ∈ tAls
    hence ∃ Al ∈ Als. tAl = tl Al using tex by simp
    from this obtain Al where Al ∈ Als and tAl = tl Al by auto
    hence tAl!i = Al!(Suc i) by (simp add: Suc(4) ‹i < n› nth-tl)
    moreover have ∃ B. real-diag-decomp (Al!(Suc i)) B U
      using tu ‹Al ∈ Als› by simp
    ultimately show ∃ B. real-diag-decomp (tAl ! i) B U by simp
  qed
  thus ∃ U. ∀ Al∈tAls. ∃ B. real-diag-decomp (Al ! i) B U by auto
 qed
 show ∀ i < n. ∀ Al∈tAls.  dim-row (Al!i) = dim-col (Al!i)
  by (metis Suc(5) ‹∀ tAl∈tAls. length tAl = n› not-less-eq nth-tl tex)
qed
from this obtain Ul where length Ul = n and
 ∀ i <n. ∀ Al ∈ tAls.
  (dim-row (Ul!i) = dim-row (Al!i) ∧ dim-col (Ul!i) = dim-col (Al!i))
 (∀ Al∈tAls. ∃ Bl. length Bl = n ∧
 real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
 (diag-block-mat Ul)) by auto note ulprop = this
have ∀ Al∈Als. ∃ Bl. length Bl = Suc n ∧
 real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
 (diag-block-mat (U0#Ul))
proof
 fix Al
 assume Al ∈ Als
 hence 0 < length Al using Suc by simp
 hence Al = hd Al # (tl Al) by simp
 have ∃ B. real-diag-decomp (Al!0) B U0 using u0 ‹Al∈ Als› by simp
 from this obtain B0 where b0: real-diag-decomp (Al!0) B0 U0 by auto
 hence real-diag-decomp (hd Al) B0 U0
  by (metis ‹Al = hd Al # tl Al› nth-Cons-0)
 have tl Al ∈ tAls using ‹Al ∈ Als› unfolding tAls-def by simp
 hence ∃ Bl. length Bl = n ∧
  real-diag-decomp (diag-block-mat (tl Al)) (diag-block-mat Bl)
  (diag-block-mat Ul) using ulprop by simp
 from this obtain Bl where length Bl = n and
  rl: real-diag-decomp (diag-block-mat (tl Al)) (diag-block-mat Bl)
  (diag-block-mat Ul) by auto
 have dim-row (diag-block-mat (tl Al)) = dim-col (diag-block-mat (tl Al))
  using Suc ‹Al ∈ Als› diag-block-mat-dim-row-col-eq
  by (metis (no-types, lifting) ‹Al = hd Al # tl Al› length-Cons lessI
     less-trans-Suc nth-tl)
 moreover have dim-row (Al ! 0) = dim-col (Al ! 0)
```

```
      using Suc ‹Al ∈ Als› by simp
    ultimately have real-diag-decomp (diag-block-mat ((hd Al)#(tl Al)))
      (diag-block-mat (B0#Bl)) (diag-block-mat (U0#Ul))
      using  four-block-real-diag-decomp[OF rl b0] diag-block-mat.simps(2)
        real-diag-decomp-hermitian
       by (metis ‹Al = hd Al # tl Al› b0 four-block-real-diag-decomp nth-Cons-0
rl)
    moreover have length (B0#Bl) = Suc n using ‹length Bl = n› by simp
    ultimately show ∃ Bl. length Bl = Suc n ∧
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
    (diag-block-mat (U0#Ul)) using ‹Al = hd Al # tl Al› by metis
  qed
  moreover have length (U0#Ul) = Suc n using ulprop by simp
  moreover have ∀ i<Suc n. ∀ Al∈Als. dim-row ((U0#Ul) ! i) = dim-row (Al !
i) ∧
    dim-col ((U0#Ul) ! i) = dim-col (Al ! i)
  proof (intro allI impI ballI)
    fix i Al
    assume i < Suc n and Al ∈ Als
    show dim-row ((U0 # Ul) ! i) = dim-row (Al ! i) ∧
      dim-col ((U0 # Ul) ! i) = dim-col (Al ! i)
    proof (cases i = 0)
      case True
      then show ?thesis using ‹Al ∈ Als› u0-dim by simp
    next
      case False
      hence ∃ j. i = Suc j by (simp add: not0-implies-Suc)
      from this obtain j where i = Suc j by auto
      hence (U0#Ul)!i = Ul!j by simp
      have tl Al ∈ tAls using ‹Al ∈ Als› unfolding tAls-def by simp
      moreover have Al!i = (tl Al)!j using ‹i = Suc j›
        by (metis Suc.prems(3) Zero-not-Suc ‹Al ∈ Als› diag-block-mat.cases
            list.sel(3) list.size(3) nth-Cons-Suc)
      ultimately show ?thesis using ‹(U0#Ul)!i = Ul!j›
        by (metis Suc-less-SucD ‹i < Suc n› ‹i = Suc j› ulprop(2))
    qed
  qed
  ultimately show ?thesis by blast
 qed
qed

lemma real-diag-decomp-eq-comps-props:
 assumes Ap∈ carrier-mat n n
 and 0 < n
 and real-diag-decomp Ap Bs Us ∧ sorted (map Re (diag-mat Bs))
shows Bs ∈ carrier-mat n n diagonal-mat Bs unitary Us
  Us ∈ carrier-mat n n diag-diff Bs (eq-comps (diag-mat Bs))
  eq-comps (diag-mat Bs) ≠ [] diag-mat Bs ≠ []
proof −
```

```
  show  Bs ∈ carrier-mat n n
    using assms real-diag-decompD(1) unitary-diag-carrier(1)
    by blast
  thus diag-mat Bs ≠ [] using ‹0 < n› 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 ∈ carrier-mat n n
    using assms real-diag-decompD(1) unitary-diag-carrier(2)
    by blast
  define eqcl where eqcl = eq-comps (diag-mat Bs)
  show diag-diff Bs eqcl unfolding eqcl-def
  proof (rule cpx-sorted-diag-diff)
    show Bs ∈ carrier-mat n n using ‹Bs ∈ carrier-mat n n› .
    show sorted (map Re (diag-mat Bs)) using assms by simp
    show ∀ i<n. Bs $$ (i, i) ∈ 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 ∃ A2∈ As. C2 = mat-conj (Complex-Matrix.adjoint U) A2
      using ‹C2∈ CjA› assms unfolding Cj-def by auto
    from this obtain A2 where A2∈ 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 ‹unitary U› .
      show A1∈ carrier-mat n n using ‹A1∈ As› assms by simp
      show A2∈ carrier-mat n n using ‹A2∈ As› assms by simp
      show U ∈ carrier-mat n n using ‹U ∈ carrier-mat n n› .
      show A1 * A2 = A2 * A1 using ‹A1∈ As› ‹A2∈ As› assms by simp
    qed
    thus C1 * C2 = C2 * C1
      using ‹C1 = mat-conj (Complex-Matrix.adjoint U) A1›
        ‹C2 = mat-conj (Complex-Matrix.adjoint U) A2›
      by simp
  qed
qed

lemma commute-extract-diag-block-eq:
  fixes Ap::complex Matrix.mat
  assumes Ap∈ carrier-mat n n
  and 0 < n
  and real-diag-decomp Ap Bs Us ∧ sorted (map Re (diag-mat Bs))
  and finite Afp
  and card Afp ≤ i
```

```
  and ∀ A∈Afp. hermitian A ∧ A ∈ carrier-mat n n
  and ∀ A∈Afp. ∀ B∈Afp. A * B = B * A
  and ∀ A∈ Afp. Ap * A = A * Ap
  and CjA = (λA2. mat-conj (Complex-Matrix.adjoint Us) A2)‘Afp
  and eqcl = eq-comps (diag-mat Bs)
  shows  ∀ C ∈ 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∈ CjA
 hence ∃ Ac ∈ Afp. C = mat-conj (Complex-Matrix.adjoint Us) Ac
   using assms by auto
 from this obtain Ac where Ac ∈ 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 ∈ carrier-mat n n using cjprops ‹C∈ CjA› by simp
   show  diag-compat C eqcl
   proof (rule commute-diag-compat)
     show Bs ∈ carrier-mat n n using ‹Bs ∈ carrier-mat n n› .
     show diag-diff Bs eqcl using ubprops assms by simp
     show diagonal-mat Bs using ‹diagonal-mat Bs› .
     show C ∈ carrier-mat n n using ‹C ∈ carrier-mat n n› .
     have Bs * (Complex-Matrix.adjoint Us * Ac * Us) =
       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 ‹Ac ∈ Afp› by simp
     qed
     thus C * Bs = Bs * C
       using ‹C = mat-conj (Complex-Matrix.adjoint Us) Ac›
       by (metis mat-conj-adjoint)
   qed
 qed
qed

lemma extract-dbm-eq-component-commute:
 assumes ∀ C∈Cs. C = diag-block-mat (extract-subdiags C l)
 and ∀ C1∈Cs. ∀ C2∈Cs. C1 * C2 = C2 * C1
 and ExC = (λA. extract-subdiags A l)‘Cs
 and j < length l
 and Exi = (λA. (A!j))‘ ExC
 and Al ∈ Exi
 and Bl∈ Exi
 shows Al * Bl = Bl * Al
proof −
```

define *ncl* **where** *ncl = length l*
have ∀ *Al*∈*ExC*. *length Al = ncl*
by (*simp add*: *assms extract-subdiags-length ncl-def*)
have ∃ *Ea* ∈ *ExC*. *Al = Ea*!*j* **using** *assms* **by** *auto*
from *this* **obtain** *Ea* **where** *Ea*∈ *ExC* **and** *Al = Ea*!*j* **by** *auto*
have ∃ *Eb* ∈ *ExC*. *Bl = Eb*!*j* **using** *assms* **by** *auto*
from *this* **obtain** *Eb* **where** *Eb*∈ *ExC* **and** *Bl = Eb*!*j* **by** *auto*
have ∀ *j*<*ncl*. ∀ *E*∈*ExC*. *E* ! *j* ∈ *carrier-mat* (*l* ! *j*) (*l* ! *j*)
by (*metis* (*no-types*, *lifting*) *assms*(*3*) *extract-subdiags-carrier*
imageE ncl-def)
hence ∀ *i* < *ncl*. ∀ *Al*∈*ExC*. *dim-row* (*Al* ! *i*) = *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 *length Ea* = *length Eb*
by (*simp add*: ‹*Ea* ∈ *ExC*› ‹*Eb* ∈ *ExC*› ‹∀ *Al*∈*ExC*. *length Al* = *ncl*›)
show *j* < *length Ea*
by (*metis* ‹*Eb* ∈ *ExC*› ‹∀ *Al*∈*ExC*. *length Al* = *ncl*› ‹*j* < *length l*›
ncl-def ‹*length Ea* = *length Eb*›)
show ∀ *i*<*length Ea*. *dim-row* (*Ea* ! *i*) = *dim-col* (*Ea* ! *i*)
by (*simp add*: ‹*Ea* ∈ *ExC*› ‹∀ *Al*∈*ExC*. *length Al* = *ncl*›
‹∀ *i* < *ncl*. ∀ *Al*∈*ExC*. *dim-row* (*Al* ! *i*) = *dim-col* (*Al* ! *i*)›)
show ∀ *i*<*length Ea*. *dim-row* (*Ea* ! *i*) = *dim-row* (*Eb* ! *i*)
by (*metis* ‹*Ea* ∈ *ExC*› ‹*Eb* ∈ *ExC*› ‹∀ *Al*∈*ExC*. *length Al* = *ncl*›
‹∀ *j*<*ncl*. ∀ *E*∈*ExC*. *E* ! *j* ∈ *carrier-mat* (*l* ! *j*) (*l* ! *j*)›
carrier-matD(*1*))
show ∀ *i*<*length Ea*. *dim-col* (*Ea* ! *i*) = *dim-col* (*Eb* ! *i*)
using ‹*Ea* ∈ *ExC*› ‹*Eb* ∈ *ExC*› ‹∀ *Al*∈*ExC*. *length Al* = *ncl*›
‹∀ *i*<*length Ea*. *dim-row* (*Ea* ! *i*) = *dim-row* (*Eb* ! *i*)›
‹∀ *i*<*ncl*. ∀ *Al*∈*ExC*. *dim-row* (*Al* ! *i*) = *dim-col* (*Al* ! *i*)› **by** *auto*
have ∃ *Cea* ∈ *Cs*. *Ea = extract-subdiags Cea l*
using ‹*Ea* ∈ *ExC*› *assms* **by** *auto*
from *this* **obtain** *Cea* **where** *Cea*∈ *Cs* **and**
Ea = extract-subdiags Cea l **by** *auto*
hence *cea*: *Cea = diag-block-mat Ea*
by (*simp add*: ‹∀ *C*∈*Cs*. *C* =
diag-block-mat (*extract-subdiags C l*)›)
have ∃ *Ceb* ∈ *Cs*. *Eb = extract-subdiags Ceb l*
using ‹*Eb* ∈ *ExC*› *assms* **by** *auto*
from *this* **obtain** *Ceb* **where** *Ceb*∈ *Cs* **and**
Eb = extract-subdiags Ceb l **by** *auto*
hence *Ceb = diag-block-mat Eb*
by (*simp add*: ‹∀ *C*∈*Cs*. *C* =
diag-block-mat (*extract-subdiags C l*)›)
moreover have *Cea* ∗ *Ceb* = *Ceb* ∗ *Cea*
by (*simp add*: ‹*Cea* ∈ *Cs*› ‹*Ceb* ∈ *Cs*›
‹∀ *C1*∈*Cs*. ∀ *C2*∈*Cs*. *C1* ∗ *C2* = *C2* ∗ *C1*›)
ultimately show *diag-block-mat Ea* ∗ *diag-block-mat Eb* =
diag-block-mat Eb ∗ *diag-block-mat Ea* **using** *cea* **by** *simp*

```
  qed
  thus Al * Bl = Bl * Al using ‹Al = Ea!j› ‹Bl = Eb!j› by simp
qed

lemma extract-comm-real-diag-decomp:
  fixes CjA::complex Matrix.mat set
  assumes ⋀(Af::complex Matrix.mat set) n . finite Af ⟹
    card Af ≤ i ⟹
    Af ≠ {} ⟹
    (⋀A. A ∈ Af ⟹ A ∈ carrier-mat n n) ⟹
    0 < n ⟹ (⋀A. A ∈ Af ⟹ hermitian A) ⟹
    (⋀A B. A ∈ Af ⟹ B ∈ Af ⟹ A * B = B * A) ⟹
    ∃ U. ∀ A∈Af. ∃ B. real-diag-decomp A B U
  and finite CjA
  and CjA ≠ {}
  and card CjA ≤ i
  and ∀ C∈CjA. C = diag-block-mat (extract-subdiags C eqcl)
  and ∀ C1∈CjA. ∀ C2∈CjA. C1 * C2 = C2 * C1
  and Exc = (λA. extract-subdiags A eqcl) ‘CjA
  and ∀ E∈ Exc. list-all (λB. 0 < dim-row B ∧ hermitian B) E
  and ∀ i < length eqcl. 0 < eqcl!i
  shows ∀ i<length eqcl. ∃ U. ∀ Al∈Exc. ∃ 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 = (λl. l!j) ‘Exc
  have finite Exi using assms unfolding Exi-def by simp
  have card Exi ≤ i using assms unfolding Exi-def
    by (metis card-image-le image-image le-trans)
  have exfl: ∀ Ej ∈ Exi. ∃ Fj ∈ CjA. Ej = (extract-subdiags Fj eqcl)!j
  proof
    fix Ej
    assume Ej ∈ Exi
    hence ∃ El ∈ Exc. Ej = El!j unfolding Exi-def by auto
    from this obtain El where El ∈ Exc and Ej = El!j by auto
    hence ∃ Fl ∈ CjA. El = extract-subdiags Fl eqcl
      using assms by auto
    from this obtain Fl where Fl ∈ CjA and
      El = extract-subdiags Fl eqcl by auto
    thus ∃ Fj ∈ CjA. Ej = (extract-subdiags Fj eqcl)!j using ‹Ej = El!j›
      by auto
  qed
  have ∃ U. ∀ Al∈Exi. ∃ B. real-diag-decomp (Al) B U
  proof (rule assms(1))
    show finite Exi using ‹finite Exi› .
    show card Exi ≤ i using ‹card Exi <= i› .
    show Exi ≠ {} using ‹CjA ≠ {}› using assms unfolding Exi-def by auto
    show 0 < eqcl!j using ‹j < ncl› ncl-def assms by simp
```

show $\bigwedge Al.\ Al \in Exi \Longrightarrow Al \in carrier\text{-}mat\ (eqcl\ !\ j)\ (eqcl\ !\ j)$
proof −
fix *Al*
assume $Al \in Exi$
hence $\exists\, Fl \in CjA.\ Al = (extract\text{-}subdiags\ Fl\ eqcl)!j$ **using** *exfl*
by *simp*
from *this* **obtain** *Fl* **where** $Fl \in CjA$ **and**
$Al = (extract\text{-}subdiags\ Fl\ eqcl)!j$ **by** *auto*
thus $Al \in carrier\text{-}mat\ (eqcl\ !\ j)\ (eqcl\ !\ j)$
using *extract-subdiags-carrier*[*of j eqcl*] ‹$j < ncl$›
unfolding *Exi-def ncl-def* **by** *simp*
qed
show $\bigwedge Al.\ Al \in Exi \Longrightarrow hermitian\ Al$
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* ‹$j < ncl$› *ncl-def*
by (*metis* (*no-types*, *lifting*) *extract-subdiags-length image-iff*
list-all-length)
qed
show $\bigwedge Al\ Bl.\ Al \in Exi \Longrightarrow Bl \in Exi \Longrightarrow Al * Bl = Bl * Al$
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** ‹$Al \in Exi$› **.**
show $Bl \in Exi$ **using** ‹$Bl \in Exi$› **.**
show $\forall\, C{\in}CjA.\ C = diag\text{-}block\text{-}mat\ (extract\text{-}subdiags\ C\ eqcl)$
using ‹$\forall\, C{\in}CjA.\ C = diag\text{-}block\text{-}mat\ (extract\text{-}subdiags\ C\ eqcl)$› **.**
show $\forall\, C1{\in}CjA.\ \forall\, C2{\in}CjA.\ C1 * C2 = C2 * C1$
using ‹$\forall\, C1{\in}CjA.\ \forall\, C2{\in}CjA.\ C1 * C2 = C2 * C1$› **.**
show $j < length\ eqcl$ **using** ‹$j < ncl$› *ncl-def* **by** *simp*
show $Exi = (\lambda A.\ A\ !\ j)\ `\ Exc$ **using** *Exi-def* **by** *simp*
show $Exc = (\lambda A.\ extract\text{-}subdiags\ A\ eqcl)\ `\ CjA$
using *assms* **by** *simp*
qed
qed
qed
thus $\exists\, U.\ \forall\, Al{\in}Exc.\ \exists\, B.\ real\text{-}diag\text{-}decomp\ (Al\ !\ j)\ B\ U$ **unfolding** *Exi-def*
by *simp*
qed

8.2 The main result

theorem *commuting-hermitian-family-diag*:
fixes *Af*::*complex Matrix.mat set*

assumes *finite Af*
and $Af \neq \{\}$
and $\bigwedge A.\ A \in Af \Longrightarrow A \in$ *carrier-mat n n*
and $0 < n$
and $\bigwedge A.\ A \in Af \Longrightarrow$ *hermitian A*
and $\bigwedge A\ B.\ A \in Af \Longrightarrow B \in Af \Longrightarrow A * B = B * A$
shows $\exists\ U.\ \forall\ A \in Af.\ \exists B.$ *real-diag-decomp A B U* **using** *assms*
proof −
define *i* **where** $i =$ *card Af*
have *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 *card Afp* $\leq i$ **using** ‹*card Af* $\leq$ *Suc i*› ‹*Ap* $\in$ *Af*›
unfolding *Afp-def* **by** *simp*
have $\forall A \in Afp.$ *hermitian A* $\wedge$ $A \in$ *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* ‹*Ap* $\in$ *Af*›)
have *hermitian Ap Ap* $\in$ *carrier-mat n n* $0 < n$ **using** ‹*Ap* $\in$ *Af*› *Suc* **by** *auto*
from *this* **obtain** *Bs Us* **where** *rd*: *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* = *real-diag-decomp-eq-comps-props*[*OF* ‹*Ap* $\in$ *carrier-mat n n*›
‹$0 < n$› *ub*]
define *eqcl* **where** *eqcl* = *eq-comps* (*diag-mat Bs*)
have *diag-diff Bs eqcl* **using** *ubprops* **unfolding** *eqcl-def* **by** *simp*
have $eqcl \neq []$ **using** *ubprops* **unfolding** *eqcl-def* **by** *simp*
hence *eqcl* = *hd eqcl* # (*tl eqcl*) **by** *simp*
define *esubB* **where** *esubB* = *extract-subdiags Bs eqcl*
have *ebcar*: $\forall i <$ *length esubB*. *esubB* ! $i \in$ *carrier-mat* (*eqcl*!*i*) (*eqcl*!*i*)
using *extract-subdiags-carrier*[*of* - *eqcl Bs*]
by (*simp add*: *esubB-def extract-subdiags-length*)
have *Bs* = *diag-block-mat esubB* **unfolding** *esubB-def eqcl-def*
proof (*rule diagonal-extract-eq*)
show *Bs* $\in$ *carrier-mat n n* **using** ‹*Bs* $\in$ *carrier-mat n n*› **.**
show *diagonal-mat Bs* **using** *ubprops real-diag-decompD*(*2*)
real-diag-decompD(*1*) *unitary-diagD*(*2*) **by** *blast*

qed
show *?case*
proof (*cases Afp* = {})
case *True*
hence *Af* = {*Ap*} **using** ‹*Afp* = *Af* − {*Ap*}›
by (*simp add*: *Suc*(*4*) *subset-singleton-iff*)
then show *?thesis* **using** *rd* ‹*Af* = {*Ap*}› **by** *auto*
next
case *False*
define *Cj* **where** *Cj* = (λ*A2*. *mat-conj* (*Complex-Matrix.adjoint Us*) *A2*)
define *CjA* **where** *CjA* = *Cj*‘*Afp*
have *CjA* = (λ*A2*. (*mat-conj* (*Complex-Matrix.adjoint Us*) *A2*)) ‘ *Afp*
using *CjA-def Cj-def* **by** *simp*
note *cjprops* = *commuting-conj-mat-set-props*[*OF* ‹*finite Afp*› ‹*card Afp* ≤ *i*›
‹∀ *A*∈*Afp*. *hermitian A* ∧ *A* ∈ *carrier-mat n n*›
‹∀ *A*∈*Afp*. ∀ *B*∈*Afp*. *A* ∗ *B* = *B* ∗ *A*›
‹*unitary Us*› ‹*Us* ∈ *carrier-mat n n*›
‹*CjA* = (λ*A2*. *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* = (λ*A*. *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* (λ*B*. *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* (λ*B*. *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 ∧
    (∀ i <ncl. ∀ Al ∈ Ex.
    (dim-row (Ul!i) = dim-row (Al!i) ∧ dim-col (Ul!i) = dim-col (Al!i)))∧
    (∀  Al ∈ Ex. ∃ Bl. (length Bl = ncl ∧
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
    (diag-block-mat Ul))))
  proof (rule real-diag-decomp-block-set)
    show Ex ≠ {} using ‹Afp ≠ {}› unfolding Ex-def CjA-def by auto
    show 0 < ncl unfolding ncl-def using ‹eqcl ≠[]› by simp
    show ∀ Al∈Ex. length Al = ncl unfolding ncl-def Ex-def
      by (simp add: extract-subdiags-length)
    show ∀ i<ncl. ∀ Al∈Ex. dim-row (Al ! i) = dim-col (Al ! i)
    proof (intro allI impI ballI)
      fix i Al
      assume i < ncl and Al ∈ Ex
      thus dim-row (Al ! i) = dim-col (Al ! i) using exall
        by (metis (mono-tags, lifting) ‹∀ Al∈Ex. length Al = ncl›
            carrier-matD(2) hermitian-square list-all-length)
    qed
   show  ∀ i<ncl. ∃ U. ∀ Al∈Ex. ∃ B. real-diag-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 ‹finite CjA› .
      show card CjA ≤ i using ‹card CjA ≤ i› .
      show ⋀C. C ∈ CjA ⟹ C = diag-block-mat (extract-subdiags C eqcl)
        using ‹∀ C ∈ CjA. C = diag-block-mat (extract-subdiags C eqcl)› by
simp
      show ⋀C1 C2. C1 ∈ CjA ⟹ C2 ∈ CjA ⟹ C1 * C2 = C2 * C1
        using cjprops by simp
      show ⋀i. i < length eqcl ⟹ 0 < eqcl!i
      proof −
        fix il
```

```
      assume il < length eqcl
      thus 0 < eqcl!il using eq-comps-gt-0[OF ‹diag-mat Bs ≠ []›]
          list-all-length[of (<) 0 eq-comps (diag-mat Bs)]
        unfolding eqcl-def by simp
    qed
    show CjA = {} ⟹ False by (simp add: CjA-def False)
  qed
qed
from this obtain Ul where length Ul = ncl and
  dimul: (∀ i <ncl. ∀ Al ∈ Ex.
  (dim-row (Ul!i) = dim-row (Al!i) ∧ dim-col (Ul!i) = dim-col (Al!i))) and
  ul: ∀ Al ∈ Ex. ∃ Bl. (length Bl = ncl ∧
    real-diag-decomp (diag-block-mat Al) (diag-block-mat Bl)
    (diag-block-mat Ul))
  by auto
define Uf where Uf = Us * (diag-block-mat Ul)
have afp:∀ A ∈ Afp. ∃ Bl. real-diag-decomp A (diag-block-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*) *ebcar*)
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 *ebcar 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 *ebcar 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 ∀ *i*<*length Ul*. *Ul* ! *i* ∗ *esubB* ! *i* = *esubB* ! *i* ∗ *Ul* ! *i*
proof (*intro allI impI*)
fix *i*
assume *i* < *length Ul*
show *Ul* ! *i* ∗ *esubB* ! *i* = *esubB* ! *i* ∗ *Ul* ! *i*
unfolding *esubB-def eqcl-def*
proof (*rule extract-subdiags-comp-commute*[*symmetric*])
show *diagonal-mat Bs* **using** ‹*diagonal-mat Bs*› **.**
show *Bs*∈ *carrier-mat n n* **using** ‹*Bs* ∈ *carrier-mat n n*› **.**
show *0* < *n* **using** ‹*0* < *n*› **.**
show *i* < *length* (*eq-comps* (*diag-mat Bs*))
using ‹*i* < *length Ul*› ‹*length Ul* = *length esubB*›
extract-subdiags-length
unfolding *esubB-def eqcl-def* **by** *metis*
show *Ul* ! *i* ∈ *carrier-mat* (*eq-comps* (*diag-mat Bs*) ! *i*)
(*eq-comps* (*diag-mat Bs*) ! *i*)
using *dimul* ‹∀ *j* < *ncl*. ∀ *E* ∈ *Ex*. *E*!*j* ∈ *carrier-mat* (*eqcl*!*j*) (*eqcl*!*j*)›
‹*Ex*≠ {}› **unfolding** *ncl-def eqcl-def*
by (*metis coleq roweq*
‹∀ *i*<*length esubB*. *dim-col* (*esubB* ! *i*) = *dim-row* (*esubB* ! *i*)›
‹*i* < *length Ul*› ‹*length Ul* = *length esubB*› *carrier-matD*(*2*)
carrier-matI ebcar eqcl-def)
qed
qed
qed
hence ∃ *B*. *real-diag-decomp Ap B Uf* **by** *blast*
hence ∀ *A*∈ *Af*. ∃ *B*. *real-diag-decomp A B Uf* **using** *afp*
unfolding *Afp-def* **by** *auto*

thus $\exists\, U.\ \forall\, A{\in}Af.\ \exists\, B.$ *real-diag-decomp A B U* **by** *blast*
qed
qed
qed

end