The Functional Calculus of Hermitian Matrices and Product Formulas

Dustin Bryant

November 18, 2026

Abstract

For a Hermitian matrix A and a function $g : \mathbb{R} \to \mathbb{C}$ we define $g(A)$ as the matrix that multiplies every eigenvector of A with eigenvalue λ by $g(\lambda)$ (it exists and is unique), and derive the algebra of this functional calculus: it is linear in g, $g(A)h(A) = (gh)(A)$, $g(A)^* = \bar{g}(A)$, $g(A)^k = (g^k)(A)$, $g(f(A)) = (g \circ f)(A)$ for real f, $g(A)$ is unitary when $|g| = 1$ on the spectrum; special cases are the time evolution e^{itA} and the inverse. On top of it we verify the Lie–Trotter product formula $\|(e^{itH_1/r} \cdots e^{itH_m/r})^r - e^{it(H_1+\cdots+H_m)}\| \leq 2(mLt)^2/r$ and Suzuki's higher-order formulas: Suzuki's recursion S_{2k+2} agrees with e^{itH} up to degree $2k+2$ of the Taylor expansion, and r steps of it are within $r(2|F|+1)(\Lambda|t|/r)^{2k+3}e^{\Lambda|t|/r}/(2k+3)!$ of e^{itH}. The matrices are those of the AFP entry *Jordan Normal Form* [6]; the spectral theorem comes from the Schur decomposition of the AFP entry *Quantum Hoare Logic* [3], the adjoint, unitarity and the vector norm from *Isabelle Marries Dirac* [2]. The product formulas are due to Trotter [7] and Suzuki [5]; they are the basis of quantum simulation [4, 1].

Contents

1 Overview **3**

2 Using the entry **3**
2.1 Examples 4

3 Functional calculus for Hermitian matrices **4**
3.1 Adjoints 5
3.2 Vector norms 6
3.3 Spectral decompositions 9
3.4 The definition 11
3.5 Algebra 16
3.6 Time evolution 21
3.7 The inverse 22

4 Finite sums of matrices **24**

5 The Lie–Trotter product formula **24**
5.1 Scalar bounds 25
5.2 Bounds from the spectrum 26
5.3 Closeness of operators 29
5.4 First-order bounds 30
5.5 Products of evolutions 32

6 Order conditions for product formulas **40**

7 Higher-order product formulas **59**

1 Overview

The entry has five parts.

- `Hermitian_Functional_Calculus`: $g(A)$ for Hermitian A (*mat-fun-eigen*, *mat-fun-decomp*), its algebra ($g(A)h(A) = (gh)(A)$, $g(A)^* = \bar{g}(A)$, linearity, powers, composition), unitarity, time evolution, and the inverse (*herm-inv*).
- `Matrix_Sums`: finite sums of matrices.
- `Trotter_Product_Formula`: the Lie–Trotter formula with error bound (*trotter*).
- `Suzuki_Order`: Suzuki's recursion and its order conditions (*suzuki-order*).
- `Suzuki_Error`: the error of a product formula (*formula-product-approx*, *suzuki-error*).

2 Using the entry

For a Hermitian $A \in M_n$ ($A \in$ *carrier-mat n n* and *hermitian A*) and a function $g : \mathbb{R} \to \mathbb{C}$, *mat-fun g A* is $g(A)$.

- Computing $g(A)$: on eigenvectors (*mat-fun-eigen*), from a spectral decomposition (*spectral-decomp-exists*, *mat-fun-decomp*), or entrywise for real diagonal matrices (*mat-fun-diag*). It exists and is unique (*mat-fun-unique*); to show that a given matrix is $g(A)$, use *mat-fun-eqI*.
- Algebra: *mat-fun-add*, *mat-fun-smult*, *mat-fun-mult*, *mat-fun-pow*, *mat-fun-comp*, *mat-fun-dagger*, *mat-fun-hermitian*, *mat-fun-unitary*, and *mat-fun-cong* (only the values on the spectrum matter).
- *evolution A t* is e^{itA} and *herm-inv A* is A^{-1}.
- $U \approx_\eta V$ says $\|Uv - Vv\| \leq \eta\|v\|$ for all v; the product formulas (*trotter*, *suzuki-error*) are stated with it.

Matrices are those of Jordan_Normal_Form, with the adjoint $M^\dagger$ and *unitary* of Isabelle_Marries_Dirac and *hermitian* of QHLProver. The theories hide Isabelle_Marries_Dirac's gate constants *X*, *Y*, *Z*, *H*, *S*, *T* (still available as *Quantum.X* etc.), so that these letters can name matrices.

2.1 Examples

The evolution under $Z = \mathrm{diag}(1, -1)$ is $e^{itZ} = \mathrm{diag}(e^{it}, e^{-it})$:

lemma *example-evolution-Z*: *evolution* (*diag-matrix* [*1*, *−1*]) *t* = *diag-matrix* [*cis t*, *cis* (*−* *t*)]
proof *−*
have *evolution* (*diag-matrix* (*map complex-of-real* [*1*, *−1*])) *t* = *diag-matrix* (*map* (*λx*. *cis* (*t* * *x*)) [*1*, *−1*])
unfolding *evolution-def* **by** (*rule mat-fun-diag*)
thus *?thesis* **by** *simp*
qed

The inverse of $\mathrm{diag}(2, 4)$ is $\mathrm{diag}(\frac{1}{2}, \frac{1}{4})$:

lemma *example-inverse*: *herm-inv* (*diag-matrix* [*2*, *4*]) = *diag-matrix* [*1/2*, *1/4*]
proof *−*
have *herm-inv* (*diag-matrix* (*map complex-of-real* [*2*, *4*])) = *diag-matrix* (*map* (*λx*. *complex-of-real* (*inverse x*)) [*2*, *4*])
unfolding *herm-inv-def* **by** (*rule mat-fun-diag*)
thus *?thesis* **by** (*simp add*: *inverse-eq-divide*)
qed

3 Functional calculus for Hermitian matrices

For Hermitian A and $g : \mathbb{R} \to \mathbb{C}$, $g(A)$ is the matrix that acts on every eigenvector v of A with eigenvalue λ as multiplication by $g(\lambda)$. The eigenvectors span $\mathbb{C}^n$, so this determines $g(A)$; for a spectral decomposition $A = U\,\mathrm{diag}(D)\,U^*$ it is $U\,\mathrm{diag}(g \circ D)\,U^*$.

$\mathrm{diag}(x_1, \ldots, x_n)$, the diagonal matrix with the entries of a list.

definition *diag-matrix* :: *complex list* ⇒ *complex Matrix.mat* **where**
diag-matrix xs = *mat-diag* (*length xs*) (*λi*. *xs* ! *i*)

lemma *diag-matrix-carrier* [*simp*]: *diag-matrix xs* ∈ *carrier-mat* (*length xs*) (*length xs*)
by (*simp add*: *diag-matrix-def mat-diag-def*)

lemma *dim-diag-matrix* [*simp*]: *dim-row* (*diag-matrix xs*) = *length xs dim-col* (*diag-matrix xs*) = *length xs*
by (*simp-all add*: *diag-matrix-def mat-diag-def*)

$$\mathrm{diag}(x_1, \ldots, x_n)\, w = (x_1 w_1, \ldots, x_n w_n)$$

lemma *diag-matrix-mult-vec*:
assumes *dim-vec w* = *length xs*
shows *diag-matrix xs* *$_v$ *w* = *Matrix.vec* (*length xs*) (*λi*. *xs* ! *i* * *w* $ *i*)

proof (*rule eq-vecI*)
fix *i* **assume** *i* < *dim-vec* (*Matrix.vec* (*length xs*) (λ*i*. *xs* ! *i* ∗ *w* $ *i*))
hence *i*: *i* < *length xs* **by** *simp*
have (*diag-matrix xs* ∗$_v$ *w*) $ *i* = (∑ *k* = *0*..<*length xs*. (*if i* = *k then xs* ! *i else 0*) ∗ *w* $ *k*)
using *i assms* **by** (*simp add*: *diag-matrix-def mat-diag-def mult-mat-vec-def scalar-prod-def cong*: *if-cong*)
also have ... = (∑ *k* = *0*..<*length xs*. *if k* = *i then xs* ! *i* ∗ *w* $ *i else 0*)
by (*rule sum.cong*) *auto*
also have ... = *xs* ! *i* ∗ *w* $ *i* **using** *i* **by** *simp*
finally show (*diag-matrix xs* ∗$_v$ *w*) $ *i* = *Matrix.vec* (*length xs*) (λ*i*. *xs* ! *i* ∗ *w* $ *i*) $ *i*
using *i* **by** *simp*
qed *simp*

$$\big(M\ \mathrm{diag}(x)\big)_{*j} = x_j\, M_{*j}$$

lemma *col-mult-diag*:
assumes *M*: *M* ∈ *carrier-mat n* (*length xs*) **and** *j*: *j* < *length xs*
shows *Matrix.col* (*M* ∗ *diag-matrix xs*) *j* = *xs* ! *j* ·$_v$ *Matrix.col M j*
proof (*rule eq-vecI*)
fix *i* **assume** *i* < *dim-vec* (*xs* ! *j* ·$_v$ *Matrix.col M j*)
hence *i*: *i* < *n* **using** *M* **by** *simp*
have *Matrix.col* (*M* ∗ *diag-matrix xs*) *j* $ *i* = (∑ *k* = *0*..<*length xs*. *M* $$ (*i*, *k*) ∗ (*if k* = *j then xs* ! *k else 0*))
using *M i j* **by** (*simp add*: *diag-matrix-def mat-diag-def scalar-prod-def atLeast0LessThan cong*: *if-cong*)
also have ... = (∑ *k* = *0*..<*length xs*. *if k* = *j then M* $$ (*i*, *j*) ∗ *xs* ! *j else 0*)
by (*rule sum.cong*) *auto*
also have ... = *M* $$ (*i*, *j*) ∗ *xs* ! *j* **using** *j* **by** *simp*
finally show *Matrix.col* (*M* ∗ *diag-matrix xs*) *j* $ *i* = (*xs* ! *j* ·$_v$ *Matrix.col M j*) $ *i*
using *M i j* **by** (*simp add*: *mult.commute*)
qed (*use M* **in** *simp*)

$$\mathrm{diag}(x)^* = \mathrm{diag}(\overline{x})$$

lemma *diag-matrix-dagger*: (*diag-matrix xs*)† = *diag-matrix* (*map cnj xs*)
by (*rule eq-matI*) (*auto simp*: *diag-matrix-def mat-diag-def dagger-def*)

3.1 Adjoints

QHLProver's *hermitian* and Schur decomposition, with IMD's adjoint $M^\dagger$, *Quantum.unitary* and norm $\|v\|$.

lemma *dagger-adjoint*: $M^\dagger$ = *Complex-Matrix.adjoint M*
by (*rule eq-matI*, *simp-all add*: *dagger-def adjoint-eval*)

lemma *hermitian-dagger*: *hermitian A* $\longleftrightarrow$ $A^\dagger = A$
by (*simp add*: *hermitian-def dagger-adjoint*)

lemma *dagger-carrier*: $M \in$ *carrier-mat m n* $\Longrightarrow$ $M^\dagger \in$ *carrier-mat n m*
by (*simp add*: *dagger-def*)

lemma *unitary-dagger*: *unitary A* $\Longrightarrow$ *unitary* $(A^\dagger)$
unfolding *unitary-def* **by** (*simp add*: *dagger-of-dagger-is-id*)

lemma *unitary-from-complex-unitary*:
assumes $P \in$ *carrier-mat n n Complex-Matrix.unitary P*
shows *unitary P*
using *assms* **unfolding** *unitary-def dagger-adjoint* **by** *simp*

lemma *one-smult-mat*: (*1*::*complex*) $\cdot_m$ *M* = *M*
by (*rule eq-matI*, *simp-all*)

lemma *diag-conj-carrier*:
assumes *U*: $U \in$ *carrier-mat n n* **and** *xs*: *length xs* = *n*
shows $U *$ *diag-matrix xs* $* (U^\dagger) \in$ *carrier-mat n n*
proof −
have *diag-matrix xs* $\in$ *carrier-mat n n* **using** *diag-matrix-carrier*[*of xs*] *xs* **by** *simp*
thus *?thesis* **using** *mult-carrier-mat*[*OF mult-carrier-mat*[*OF U*] *dagger-carrier*[*OF U*]] **by** *blast*
qed

3.2 Vector norms

lemma *cpx-vec-length-smult*: $\|a \cdot_v v\|$ = *cmod a* $* \|v\|$
proof −
have $(\sum i$<*dim-vec v*. (*cmod* (*a* $*$ *v* $ *i*))^2) = (*cmod a*)^2 $*$ $(\sum i$<*dim-vec v*. (*cmod* (*v* $ *i*))^2)
by (*simp add*: *norm-mult power-mult-distrib sum-distrib-left*)
thus *?thesis* **by** (*simp add*: *cpx-vec-length-def real-sqrt-mult*)
qed

The Euclidean norm $\|v\| = \left(\sum_i |v_i|^2\right)^{1/2}$, with $\|v\|^2 = \mathrm{Re}\langle v|v\rangle$, is positive for $v \neq 0$.

lemma *vnorm-L2*: $\|v\|$ = *L2-set* (λi. *cmod* (*v* $ *i*)) {..<*dim-vec v*}
by (*simp add*: *cpx-vec-length-def L2-set-def*)

lemma *vnorm-ge0*: *0* $\leq \|v\|$
by (*simp add*: *cpx-vec-length-def sum-nonneg*)

lemma *vnorm-pos*:
assumes *v*: *dim-vec v* = *n* $v \neq 0_v$ *n*
shows *0* $< \|v\|$
proof −

obtain i **where** i: $i < n$ $v \$ i \neq 0$
proof (*rule ccontr*)
assume $\neg$ *thesis*
hence $v = 0_v\ n$ **using** *that* $v(1)$ **by** (*intro eq-vecI*) *auto*
thus *False* **using** $v(2)$ **by** *simp*
qed
have $0 < (cmod\ (v \$ i))^2$ **using** i **by** *simp*
also have $\ldots \leq (\sum j{<}dim\text{-}vec\ v.\ (cmod\ (v \$ j))^2)$ **using** i v **by** (*intro member-le-sum*) *auto*
finally show *?thesis* **by** (*simp add*: *cpx-vec-length-def*)
qed

lemma *vnorm-sq*: $\|v\|^2 = (\sum i{<}dim\text{-}vec\ v.\ (cmod\ (v \$ i))^2)$
by (*simp add*: *cpx-vec-length-def sum-nonneg*)

lemma *vnorm-Re-inner*: $Re\ \langle v|v\rangle = \|v\|^2$
unfolding *cpx-vec-length-inner-prod*[*symmetric*] **by** *simp*

The triangle inequality, in the forms used later, and its reverse $|\|a\| - \|b\|| \leq \|a - b\|$:

lemma *vnorm-add-le*:
assumes *dim-vec* a = *dim-vec* b
shows $\|a + b\| \leq \|a\| + \|b\|$
proof −
have $\|a + b\| = L2\text{-}set\ (\lambda i.\ cmod\ (a \$ i + b \$ i))\ \{..{<}dim\text{-}vec\ b\}$
unfolding *vnorm-L2* **using** *assms* **by** (*intro L2-set-cong*) *simp-all*
also have $\ldots \leq L2\text{-}set\ (\lambda i.\ cmod\ (a \$ i) + cmod\ (b \$ i))\ \{..{<}dim\text{-}vec\ b\}$
by (*rule L2-set-mono*) (*simp-all add*: *norm-triangle-ineq*)
also have $\ldots \leq L2\text{-}set\ (\lambda i.\ cmod\ (a \$ i))\ \{..{<}dim\text{-}vec\ b\} + L2\text{-}set\ (\lambda i.\ cmod\ (b \$ i))\ \{..{<}dim\text{-}vec\ b\}$
by (*rule L2-set-triangle-ineq*)
finally show *?thesis* **unfolding** *vnorm-L2 assms* **.**
qed

lemma *vnorm-minus-commute*:
assumes *dim-vec* a = *dim-vec* b
shows $\|a - b\| = \|b - a\|$
using *assms* **by** (*simp add*: *cpx-vec-length-def norm-minus-commute*)

lemma *vnorm-diff-le*:
assumes a: *dim-vec* $a = n$ **and** b: *dim-vec* $b = n$ **and** c: *dim-vec* $c = n$
shows $\|a - c\| \leq \|a - b\| + \|b - c\|$
proof −
have $a - c = (a - b) + (b - c)$ **using** a b c **by** (*intro eq-vecI*) *simp-all*
thus *?thesis* **using** *vnorm-add-le*[*of* $a - b$ $b - c$] a b c **by** *simp*
qed

lemma *vnorm-le-diff-add*:
assumes *dim-vec* a = *dim-vec* b

shows $\|a\| \leq \|a - b\| + \|b\|$
proof −
have $(a - b) + b = a$ **using** *assms* **by** (*intro eq-vecI*) *simp-all*
thus *?thesis* **using** *vnorm-add-le*[*of a − b b*] *assms* **by** *simp*
qed

lemma *vnorm-reverse*:
assumes *dim-vec a* = *dim-vec b*
shows $|\|a\| - \|b\|| \leq \|a - b\|$
using *vnorm-le-diff-add*[*OF assms*] *vnorm-le-diff-add*[*of b a*] *assms vnorm-minus-commute*[*OF assms*] **by** *simp*

lemma *vnorm-zero-vec*: $\|0_v\ m\| = 0$
by (*simp add*: *cpx-vec-length-def*)

A unitary is square: comparing $\mathrm{Tr}(U^*U) = \mathrm{Tr}(I_n) = n$ with $\mathrm{Tr}(UU^*) = \mathrm{Tr}(I_m) = m$, both being $\sum_{i,j} |u_{ij}|^2$.

lemma *unitary-square-dim*:
assumes *unitary A*
shows *dim-row A* = *dim-col A*
proof −
let *?m* = *dim-row A* **and** *?n* = *dim-col A*
have *l*: $(A^\dagger) * A = 1_m\ ?n$ **and** *r*: $A * (A^\dagger) = 1_m\ ?m$ **using** *assms* **unfolding** *unitary-def* **by** *auto*
have *of-nat ?n* = $(\sum c{<}?n.\ ((A^\dagger) * A)\ \$\$\ (c,\ c))$ **by** (*simp add*: *l*)
also have ... = $(\sum c{<}?n.\ \sum r{<}?m.\ cnj\ (A\ \$\$\ (r,\ c)) * A\ \$\$\ (r,\ c))$
by (*rule sum.cong*) (*auto simp*: *scalar-prod-def dagger-def atLeast0LessThan*)
also have ... = $(\sum r{<}?m.\ \sum c{<}?n.\ A\ \$\$\ (r,\ c) * cnj\ (A\ \$\$\ (r,\ c)))$
by (*subst sum.swap*) (*simp add*: *mult.commute*)
also have ... = $(\sum r{<}?m.\ (A * (A^\dagger))\ \$\$\ (r,\ r))$
by (*rule sum.cong*) (*auto simp*: *scalar-prod-def dagger-def atLeast0LessThan*)
also have ... = *of-nat ?m* **by** (*simp add*: *r*)
finally show *?thesis* **by** *simp*
qed

Unitaries preserve the norm, $\|Wy\| = \|y\|$; only the number of columns of W has to match y.

lemma *unitary-vnorm*:
assumes *W*: $W \in$ *carrier-mat m n unitary W* **and** *y*: *dim-vec y* = *n*
shows $\|W *_v y\| = \|y\|$
proof −
have *Matrix.col* $(W * |y\rangle)\ 0 = W *_v y$
using *W*(*1*) *y* **by** (*intro eq-vecI*) (*auto simp*: *ket-vec-def scalar-prod-def*)
thus *?thesis* **using** *unitary-is-length-preserving*[*OF W*(*2*), *of y*] *W*(*1*) *y* **by** *simp*
qed

Products of unitaries are unitary.

lemma *unitary-mult*:

assumes *unitary A unitary B A* ∈ *carrier-mat n n B* ∈ *carrier-mat n n*
shows *unitary* (*A* ∗ *B*)
proof −
have *d*: $A^\dagger \in$ *carrier-mat n n* $B^\dagger \in$ *carrier-mat n n* **using** *assms*(*3*,*4*) **by** (*simp-all add*: *dagger-carrier*)
have *u*: $A^\dagger * A = 1_m\ n\ A * (A^\dagger) = 1_m\ n\ B^\dagger * B = 1_m\ n\ B * (B^\dagger) = 1_m\ n$
using *assms* **unfolding** *unitary-def* **by** *auto*
have $(A * B)^\dagger = (B^\dagger) * (A^\dagger)$ **using** *assms*(*3*,*4*) **by** (*simp add*: *dagger-of-prod*)
moreover have $(B^\dagger) * (A^\dagger) * (A * B) = (B^\dagger) * ((A^\dagger) * A) * B$
$A * B * ((B^\dagger) * (A^\dagger)) = A * (B * (B^\dagger)) * (A^\dagger)$
using *assms*(*3*,*4*) *d* **by** (*mat-assoc n*)+
ultimately show *?thesis* **unfolding** *unitary-def* **using** *u assms*(*3*,*4*) *d* **by** *simp*
qed

Powers of a unitary are unitary.

lemma *unitary-pow*:
assumes *U*: *U* ∈ *carrier-mat m m* **and** *u*: *unitary U*
shows *unitary* (*U* ^$_m$ *k*)
proof (*induction k*)
case *0*
show *?case* **by** *simp*
next
case (*Suc k*)
have *unitary* (*U* ^$_m$ *k* ∗ *U*) **by** (*rule unitary-mult*[*OF Suc u pow-carrier-mat*[*OF U*] *U*])
thus *?case* **by** *simp*
qed

3.3 Spectral decompositions

spectral-decomp A U D: *U* unitary and $U^*AU = \mathrm{diag}(D)$.

definition *spectral-decomp* :: *complex Matrix.mat* ⇒ *complex Matrix.mat* ⇒ *real list* ⇒ *bool* **where**
spectral-decomp A U D ⟷ *U* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*) ∧ *unitary U* ∧
length D = *dim-row A* ∧ $U^\dagger * A * U$ = *diag-matrix* (*map complex-of-real D*)

The spectral theorem, from QHLProver's Schur decomposition of Hermitian matrices.

lemma *spectral-decomp-exists*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows ∃ *U D*. *spectral-decomp A U D*
proof −
obtain *es* **where** *c*: *char-poly A* = (∏ *e*←*es*. [:− *e*, *1*:])
using *complex-mat-char-poly-factorizable*[*OF A*(*1*)] **by** *blast*
obtain *B P Q* **where** *s*: *unitary-schur-decomposition A es* = (*B*, *P*, *Q*)
by (*cases unitary-schur-decomposition A es*) *auto*
have *h*: *similar-mat-wit A B P* (*Complex-Matrix.adjoint P*) *diagonal-mat B Complex-Matrix.unitary P*

$\forall i{<}n.\ B\ \$\$\ (i,\ i) \in \mathbb{R}$
using *hermitian-eigenvalue-real*[*OF A c s*] **by** *auto*
have *P*: $P \in$ *carrier-mat n n B* $\in$ *carrier-mat n n A* = *P* $*$ *B* $*$ *Complex-Matrix.adjoint P*
using *h*(*1*) *A*(*1*) **unfolding** *similar-mat-wit-def Let-def* **by** *auto*
have *u*: *unitary P* **by** (*rule unitary-from-complex-unitary*[*OF P*(*1*) *h*(*3*)])
hence *PP*: $P^\dagger * P = 1_m\ n$ **using** *P*(*1*) **unfolding** *unitary-def* **by** *simp*
define *D* **where** *D* = *map* (λ*i*. *Re* (*B* $$ (*i*, *i*))) [*0*..<*n*]
have *BD*: *B* = *diag-matrix* (*map complex-of-real D*)
proof (*rule eq-matI*)
fix *i j* **assume** *i* < *dim-row* (*diag-matrix* (*map complex-of-real D*)) *j* < *dim-col* (*diag-matrix* (*map complex-of-real D*))
hence *ij*: *i* < *n j* < *n* **by** (*simp-all add*: *D-def*)
show *B* $$ (*i*, *j*) = *diag-matrix* (*map complex-of-real D*) $$ (*i*, *j*)
using *h*(*2*,*4*) *ij P*(*2*) **by** (*cases i* = *j*) (*auto simp*: *diag-matrix-def mat-diag-def D-def diagonal-mat-def*)
qed (*use P*(*2*) **in** ‹*simp-all add*: *D-def*›)
have *Pd*: $P^\dagger \in$ *carrier-mat n n* **by** (*rule dagger-carrier*[*OF P*(*1*)])
have $P^\dagger * A * P = (P^\dagger * P) * B * (P^\dagger * P)$
unfolding *P*(*3*) *dagger-adjoint*[*symmetric*] **using** *P*(*1*,*2*) *Pd* **by** (*mat-assoc n*)
hence $P^\dagger * A * P = B$ **unfolding** *PP* **using** *P*(*2*) **by** *simp*
hence *spectral-decomp A P D* **unfolding** *spectral-decomp-def BD* **using** *P*(*1*) *A*(*1*) *u* **by** (*simp add*: *D-def*)
thus *?thesis* **by** *blast*
qed

Unpacking a spectral decomposition: $UU^* = U^*U = I$ and $A = U\ \mathrm{diag}(D)\, U^*$.

lemma *spectral-decompD*:
assumes *A*: *A* $\in$ *carrier-mat n n* **and** *sd*: *spectral-decomp A U D*
shows *U* $\in$ *carrier-mat n n unitary U length D* = *n* $U^\dagger * U = 1_m\ n\ U * (U^\dagger) = 1_m\ n$
A = *U* $*$ *diag-matrix* (*map complex-of-real D*) $*$ ($U^\dagger$) *A* $*$ *U* = *U* $*$ *diag-matrix* (*map complex-of-real D*)
proof −
show *U*: *U* $\in$ *carrier-mat n n* **and** *u*: *unitary U* **and** *D*: *length D* = *n*
using *sd A* **unfolding** *spectral-decomp-def* **by** *auto*
show *UU*: $U^\dagger * U = 1_m\ n$ **and** *UU'*: *U* $*$ ($U^\dagger$) $= 1_m\ n$ **using** *u U* **unfolding** *unitary-def* **by** *auto*
have *e*: $U^\dagger * A * U$ = *diag-matrix* (*map complex-of-real D*) **using** *sd* **unfolding** *spectral-decomp-def* **by** *auto*
have *Ud*: $U^\dagger \in$ *carrier-mat n n* **by** (*rule dagger-carrier*[*OF U*])
have *U* $*$ *diag-matrix* (*map complex-of-real D*) $*$ ($U^\dagger$) = (*U* $*$ ($U^\dagger$)) $*$ *A* $*$ (*U* $*$ ($U^\dagger$))
unfolding *e*[*symmetric*] **using** *U Ud A* **by** (*mat-assoc n*)
thus *A* = *U* $*$ *diag-matrix* (*map complex-of-real D*) $*$ ($U^\dagger$) **unfolding** *UU'* **using** *A* **by** *simp*
have *U* $*$ *diag-matrix* (*map complex-of-real D*) = (*U* $*$ ($U^\dagger$)) $*$ *A* $*$ *U*
unfolding *e*[*symmetric*] **using** *U Ud A* **by** (*mat-assoc n*)
thus *A* $*$ *U* = *U* $*$ *diag-matrix* (*map complex-of-real D*) **unfolding** *UU'* **using**

A **by** *simp*
qed

The columns of U are eigenvectors: $Au_j = D_j u_j$.

lemma *spectral-decomp-col*:
assumes *A*: *A* ∈ *carrier-mat n n* **and** *sd*: *spectral-decomp A U D* **and** *j*: *j* < *n*
shows *A* ∗$_v$ *Matrix.col U j* = *complex-of-real* (*D* ! *j*) ·$_v$ *Matrix.col U j*
proof −
note *d* = *spectral-decompD*[*OF A sd*]
have *A* ∗$_v$ *Matrix.col U j* = *Matrix.col* (*A* ∗ *U*) *j* **by** (*rule col-mult2*[*OF A d*(*1*) *j*, *symmetric*])
also have ... = *Matrix.col* (*U* ∗ *diag-matrix* (*map complex-of-real D*)) *j* **using** *d*(*7*) **by** *simp*
also have ... = *complex-of-real* (*D* ! *j*) ·$_v$ *Matrix.col U j*
using *col-mult-diag*[*of U n map complex-of-real D j*] *d*(*1*,*3*) *j* **by** *simp*
finally show *?thesis* **.**
qed

3.4 The definition

B acts as g on the eigenvectors of A, written *B acts-as g on-eigenvectors-of A*, if $Av = \lambda v$ implies $Bv = g(\lambda)\, v$. The matrix $g(A)$ is the $n \times n$ matrix with this property; it exists and is unique (below).

definition *acts-on-eigenvectors* :: *complex Matrix.mat* ⇒ (*real* ⇒ *complex*) ⇒ *complex Matrix.mat* ⇒ *bool* **where**
acts-on-eigenvectors A g B ⟷ (∀ *v l*. *dim-vec v* = *dim-row A* ⟶
A ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v* ⟶ *B* ∗$_v$ *v* = *g l* ·$_v$ *v*)

definition *mat-fun* :: (*real* ⇒ *complex*) ⇒ *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
mat-fun g A = (*SOME B*. *B* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*) ∧ *B acts-as g on-eigenvectors-of A*)

The spectrum: the real eigenvalues (JNF's *eigenvalue*).

definition *herm-spectrum* :: *complex Matrix.mat* ⇒ *real set* **where**
herm-spectrum A = {*l*. *eigenvalue A* (*complex-of-real l*)}

lemma *herm-spectrum-iff*:
l ∈ *herm-spectrum A* ⟷ (∃ *v*. *dim-vec v* = *dim-row A* ∧ *v* ≠ *0*$_v$ (*dim-row A*) ∧ *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*)
unfolding *herm-spectrum-def mem-Collect-eq eigenvalue-def eigenvector-def carrier-vec-def* **by** *simp*

Each D_j of a spectral decomposition is an eigenvalue of A.

lemma *spectral-decomp-eig-in-spectrum*:
assumes *A*: *A* ∈ *carrier-mat n n* **and** *sd*: *spectral-decomp A U D* **and** *j*: *j* < *n*
shows *D* ! *j* ∈ *herm-spectrum A*

proof −
note d = *spectral-decompD*[*OF A sd*]
have *cd*: *dim-vec* (*Matrix.col U j*) = *n* **using** *d*(*1*) **by** *simp*
have *nz*: *Matrix.col U j* ≠ 0_v *n*
proof
assume *z*: *Matrix.col U j* = 0_v *n*
have ($U^\dagger$ ∗ *U*) $$ (*j*, *j*) = *Matrix.row* ($U^\dagger$) *j* · *Matrix.col U j*
by (*rule index-mult-mat*(*1*)) (*use d*(*1*) *j* **in** *simp-all*)
also have ... = *0* **unfolding** *z* **by** (*rule scalar-prod-right-zero*) (*use d*(*1*) *j* **in** ‹*simp add*: *dagger-def*›)
finally show *False* **using** *d*(*4*) *j* **by** *simp*
qed
show *?thesis* **unfolding** *herm-spectrum-iff*
using *spectral-decomp-col*[*OF A sd j*] *cd nz A* **by** (*intro exI*[*of - Matrix.col U j*]) *simp*
qed

lemma *acts-on-eigenvectorsD*:
B acts-as g on-eigenvectors-of A ⟹ *dim-vec v* = *dim-row A* ⟹ *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v* ⟹
B ∗$_v$ *v* = *g l* ·$_v$ *v*
unfolding *acts-on-eigenvectors-def* **by** *blast*

$U\,\mathrm{diag}(g \circ D)\,U^*$ acts correctly on eigenvectors: if $Av = \lambda v$ then $w = U^*v$ has $D_i w_i = \lambda w_i$, so $g(D_i)w_i = g(\lambda)w_i$.

lemma *spectral-decomp-acts-on-eigenvectors*:
assumes *A*: *A* ∈ *carrier-mat n n* **and** *sd*: *spectral-decomp A U D*
shows *U* ∗ *diag-matrix* (*map g D*) ∗ ($U^\dagger$) *acts-as g on-eigenvectors-of A*
unfolding *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix *v l* **assume** *v*: *dim-vec v* = *dim-row A* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
note d = *spectral-decompD*[*OF A sd*]
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*) (*use v A* **in** *simp*)
have *Uc*: $U^\dagger$ ∈ *carrier-mat n n* **by** (*rule dagger-carrier*[*OF d*(*1*)])
define *w* **where** *w* = $U^\dagger$ ∗$_v$ *v*
have *wc*: *w* ∈ *carrier-vec n* **unfolding** *w-def* **using** *Uc vc* **by** *simp*
have *Uw*: *U* ∗$_v$ *w* = *v*
unfolding *w-def* **using** *assoc-mult-mat-vec*[*OF d*(*1*) *Uc vc*, *symmetric*] *d*(*5*) *vc* **by** *simp*
have *diag-matrix* (*map complex-of-real D*) ∗$_v$ *w* = ($U^\dagger$ ∗ *A* ∗ *U*) ∗$_v$ *w*
using *sd* **unfolding** *spectral-decomp-def* **by** *simp*
also have ... = $U^\dagger$ ∗$_v$ (*A* ∗$_v$ (*U* ∗$_v$ *w*))
using *Uc A d*(*1*) *wc* **by** (*simp add*: *assoc-mult-mat-vec*[*of - n n - n*])
also have ... = *complex-of-real l* ·$_v$ *w*
unfolding *Uw ev* **unfolding** *w-def* **by** (*rule mult-mat-vec*[*OF Uc vc*])
finally have *Dw*: *diag-matrix* (*map complex-of-real D*) ∗$_v$ *w* = *complex-of-real l* ·$_v$ *w* **.**
have *wi*: *w* $ *i* = *0* ∨ *D* ! *i* = *l* **if** *i*: *i* < *n* **for** *i*

proof −
have (*diag-matrix* (*map complex-of-real D*) $*_v$ *w*) $ *i* = *complex-of-real* (*D* ! *i*) $*$ *w* $ *i*
using *diag-matrix-mult-vec*[*of w map complex-of-real D*] *wc d*(*3*) *i* **by** *simp*
hence *complex-of-real* (*D* ! *i*) $*$ *w* $ *i* = *complex-of-real l* $*$ *w* $ *i*
using *Dw i wc* **by** (*metis carrier-vecD index-smult-vec*(*1*))
thus *?thesis* **by** *auto*
qed
have *gw*: *diag-matrix* (*map g D*) $*_v$ *w* = *g l* $\cdot_v$ *w*
proof (*rule eq-vecI*)
fix *i* **assume** *i* < *dim-vec* (*g l* $\cdot_v$ *w*)
hence *i*: *i* < *n* **using** *wc* **by** *simp*
show (*diag-matrix* (*map g D*) $*_v$ *w*) $ *i* = (*g l* $\cdot_v$ *w*) $ *i*
using *diag-matrix-mult-vec*[*of w map g D*] *wc d*(*3*) *i wi*[*OF i*] **by** *auto*
qed (*use wc d*(*3*) **in** *simp*)
have *gc*: *diag-matrix* (*map g D*) ∈ *carrier-mat n n* **using** *d*(*3*) *diag-matrix-carrier* **by** (*metis length-map*)
have (*U* $*$ *diag-matrix* (*map g D*) $*$ (*U*†)) $*_v$ *v* = *U* $*_v$ (*diag-matrix* (*map g D*) $*_v$ *w*)
unfolding *w-def assoc-mult-mat-vec*[*OF mult-carrier-mat*[*OF d*(*1*) *gc*] *Uc vc*]
by (*rule assoc-mult-mat-vec*[*OF d*(*1*) *gc mult-mat-vec-carrier*[*OF Uc vc*]])
also have ... = *g l* $\cdot_v$ (*U* $*_v$ *w*) **unfolding** *gw* **using** *d*(*1*) *wc* **by** (*rule mult-mat-vec*)
also note *Uw*
finally show (*U* $*$ *diag-matrix* (*map g D*) $*$ (*U*†)) $*_v$ *v* = *g l* $\cdot_v$ *v* .
qed

Two matrices that agree on every eigenvector of a Hermitian A agree on its eigenbasis U, hence are equal.

lemma *eigvec-mat-eqI*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *B1*: *B1* ∈ *carrier-mat K n* **and** *B2*: *B2* ∈ *carrier-mat K n*
and *eq*: ⋀*v l*. *dim-vec v* = *n* ⟹ *A* $*_v$ *v* = *complex-of-real l* $\cdot_v$ *v* ⟹ *B1* $*_v$ *v* = *B2* $*_v$ *v*
shows *B1* = *B2*
proof −
obtain *U D* **where** *sd*: *spectral-decomp A U D* **using** *spectral-decomp-exists*[*OF A*] **by** *blast*
note *d* = *spectral-decompD*[*OF A*(*1*) *sd*]
have *BU*: *B1* $*$ *U* = *B2* $*$ *U*
proof (*rule Matrix.mat-col-eqI*)
fix *j* **assume** *j* < *dim-col* (*B2* $*$ *U*)
hence *j*: *j* < *n* **using** *d*(*1*) **by** *simp*
have *cU*: *dim-vec* (*Matrix.col U j*) = *n* **using** *d*(*1*) **by** *simp*
have *Matrix.col* (*B1* $*$ *U*) *j* = *B1* $*_v$ *Matrix.col U j* **by** (*rule col-mult2*[*OF B1 d*(*1*) *j*])
also have ... = *B2* $*_v$ *Matrix.col U j* **by** (*rule eq*[*OF cU spectral-decomp-col*[*OF A*(*1*) *sd j*]])
also have ... = *Matrix.col* (*B2* $*$ *U*) *j* **by** (*rule col-mult2*[*OF B2 d*(*1*) *j*,

symmetric])
  finally show *Matrix.col* (*B1* ∗ *U*) *j* = *Matrix.col* (*B2* ∗ *U*) *j* **.**
 qed (*use B1 B2 d*(*1*) **in** *simp-all*)
 have *B1* = (*B1* ∗ *U*) ∗ (*U*†)
  using *assoc-mult-mat*[*OF B1 d*(*1*) *dagger-carrier*[*OF d*(*1*)]] *d*(*5*) *B1* **by** *simp*
 also have ... = (*B2* ∗ *U*) ∗ (*U*†) **by** (*simp only*: *BU*)
 also have ... = *B2*
  using *assoc-mult-mat*[*OF B2 d*(*1*) *dagger-carrier*[*OF d*(*1*)]] *d*(*5*) *B2* **by** *simp*
 finally show *?thesis* **.**
qed

Hence $g(A)$ is well defined: an $n \times n$ matrix acting as g on the eigenvectors.

lemma
 assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
 shows *mat-fun-carrier* [*simp*]: *mat-fun g A* ∈ *carrier-mat n n*
  and *mat-fun-acts-on-eigenvectors*: *mat-fun g A acts-as g on-eigenvectors-of A*
proof −
 obtain *U D* **where** *sd*: *spectral-decomp A U D* **using** *spectral-decomp-exists*[*OF A*] **by** *blast*
 note *d* = *spectral-decompD*[*OF A*(*1*) *sd*]
 have *U* ∗ *diag-matrix* (*map g D*) ∗ (*U*†) ∈ *carrier-mat n n*
  by (*rule diag-conj-carrier*) (*use d*(*1*,*3*) **in** *simp-all*)
 hence *ex*: ∃ *B*. *B* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*) ∧ *B acts-as g on-eigenvectors-of A*
  using *spectral-decomp-acts-on-eigenvectors*[*OF A*(*1*) *sd*] *A*(*1*) **by** *auto*
 have *mat-fun g A* ∈ *carrier-mat* (*dim-row A*) (*dim-row A*) ∧ *mat-fun g A acts-as g on-eigenvectors-of A*
  unfolding *mat-fun-def* **by** (*rule someI-ex*[*OF ex*])
 thus *mat-fun g A* ∈ *carrier-mat n n mat-fun g A acts-as g on-eigenvectors-of A*
using *A*(*1*) **by** *auto*
qed

In particular $g(A)$ has the dimensions of A.

lemma *mat-fun-dims* [*simp*]:
 assumes *A* ∈ *carrier-mat n n hermitian A*
 shows *dim-row* (*mat-fun g A*) = *n dim-col* (*mat-fun g A*) = *n*
 using *mat-fun-carrier*[*OF assms*] **by** *auto*

$$Av = \lambda v \quad \Longrightarrow \quad g(A)\, v = g(\lambda)\, v.$$

theorem *mat-fun-eigen*:
 assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
  and *v*: *dim-vec v* = *n* **and** *ev*: *A* ∗*v v* = *complex-of-real l* ·*v v*
 shows *mat-fun g A* ∗*v v* = *g l* ·*v v*
 using *acts-on-eigenvectorsD*[*OF mat-fun-acts-on-eigenvectors*[*OF A*] - *ev*] *v A*(*1*)
by *simp*

Uniqueness: every $n \times n$ matrix acting as g on the eigenvectors is $g(A)$.

lemma *mat-fun-eqI*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *B*: *B* ∈ *carrier-mat n n B acts-as g on-eigenvectors-of A*
shows *mat-fun g A* = *B*
proof (*rule eigvec-mat-eqI*[*OF A mat-fun-carrier*[*OF A*] *B*(*1*)])
fix *v l* **assume** *v*: *dim-vec v* = *n* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
show *mat-fun g A* ∗$_v$ *v* = *B* ∗$_v$ *v*
using *acts-on-eigenvectorsD*[*OF mat-fun-acts-on-eigenvectors*[*OF A*] - *ev*] *acts-on-eigenvectorsD*[*OF B*(*2*) - *ev*] *v A*(*1*) **by** *simp*
qed

So $g(A)$ is the unique $n \times n$ matrix that acts as g on the eigenvectors of A.

corollary *mat-fun-unique*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows ∃!*B*. *B* ∈ *carrier-mat n n* ∧ *B acts-as g on-eigenvectors-of A*
using *mat-fun-carrier*[*OF A*] *mat-fun-acts-on-eigenvectors*[*OF A*] *mat-fun-eqI*[*OF A*] **by** *blast*

So for every spectral decomposition of A,

$$g(A) = U\ \mathrm{diag}\big(g(D_1), \ldots, g(D_n)\big)\, U^*.$$

theorem *mat-fun-decomp*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A* **and** *sd*: *spectral-decomp A U D*
shows *mat-fun g A* = *U* ∗ *diag-matrix* (*map g D*) ∗ (*U*†)
proof (*rule mat-fun-eqI*[*OF A*])
note *d* = *spectral-decompD*[*OF A*(*1*) *sd*]
show *U* ∗ *diag-matrix* (*map g D*) ∗ (*U*†) ∈ *carrier-mat n n*
by (*rule diag-conj-carrier*) (*use d*(*1*,*3*) **in** *simp-all*)
show *U* ∗ *diag-matrix* (*map g D*) ∗ (*U*†) *acts-as g on-eigenvectors-of A*
by (*rule spectral-decomp-acts-on-eigenvectors*[*OF A*(*1*) *sd*])
qed

Only the values of g on the spectrum matter.

lemma *mat-fun-cong*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *gh*: ⋀*l*. *l* ∈ *herm-spectrum A* ⟹ *g l* = *h l*
shows *mat-fun g A* = *mat-fun h A*
proof (*rule mat-fun-eqI*[*OF A mat-fun-carrier*[*OF A*]])
show *mat-fun h A acts-as g on-eigenvectors-of A* **unfolding** *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix *v l* **assume** *v*: *dim-vec v* = *dim-row A* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
have *hv*: *mat-fun h A* ∗$_v$ *v* = *h l* ·$_v$ *v* **using** *mat-fun-eigen*[*OF A* - *ev*] *v A*(*1*) **by** *simp*
show *mat-fun h A* ∗$_v$ *v* = *g l* ·$_v$ *v*
proof (*cases v* = *0*$_v$ (*dim-row A*))
case *True*
thus *?thesis* **using** *hv* **by** (*auto intro*!: *eq-vecI*)

```
    next
      case False
     hence l ∈ herm-spectrum A unfolding herm-spectrum-iff using v ev by blast
      thus ?thesis using hv gh by simp
    qed
  qed
qed
```

3.5 Algebra

The identity function gives A, a constant c gives cI:

```
lemma mat-fun-id: A ∈ carrier-mat n n ⟹ hermitian A ⟹ mat-fun complex-of-real A = A
  by (rule mat-fun-eqI) (auto simp: acts-on-eigenvectors-def)

lemma mat-fun-const:
  assumes A: A ∈ carrier-mat n n hermitian A
  shows mat-fun (λ-. c) A = c ·_m 1_m n
proof (rule mat-fun-eqI[OF A])
 show c ·_m 1_m n acts-as (λ-. c) on-eigenvectors-of A unfolding acts-on-eigenvectors-def
    using A(1) by (auto simp: smult-mat-mult-mat-vec-assoc[of 1_m n n n])
qed simp

lemma mat-fun-one: A ∈ carrier-mat n n ⟹ hermitian A ⟹ mat-fun (λ-. 1) A = 1_m n
  using mat-fun-const[of A n 1] by (auto intro!: eq-matI)
```

The calculus is multiplicative and respects adjoints:

$$g(A)\,h(A) = (gh)(A), \qquad g(A)^* = \overline{g}(A).$$

```
theorem mat-fun-mult:
  assumes A: A ∈ carrier-mat n n hermitian A
  shows mat-fun g A * mat-fun h A = mat-fun (λx. g x * h x) A
proof (rule mat-fun-eqI[OF A, symmetric])
 show mat-fun g A * mat-fun h A ∈ carrier-mat n n by (rule mult-carrier-mat[OF
mat-fun-carrier[OF A] mat-fun-carrier[OF A]])
  show mat-fun g A * mat-fun h A acts-as (λx. g x * h x) on-eigenvectors-of A
unfolding acts-on-eigenvectors-def
  proof (intro allI impI)
    fix v l assume v: dim-vec v = dim-row A and ev: A *_v v = complex-of-real l
·_v v
    have vn: dim-vec v = n using v A(1) by simp
    have vc: v ∈ carrier-vec n by (rule carrier-vecI[OF vn])
    have (mat-fun g A * mat-fun h A) *_v v = mat-fun g A *_v (mat-fun h A *_v v)
      by (rule assoc-mult-mat-vec[OF mat-fun-carrier[OF A] mat-fun-carrier[OF
A] vc])
    also have ... = mat-fun g A *_v (h l ·_v v) by (simp only: mat-fun-eigen[OF A
vn ev])
```

also have ... = $h\ l \cdot_v$ (*mat-fun g A* $*_v$ *v*) **by** (*rule mult-mat-vec*[*OF mat-fun-carrier*[*OF A*] *vc*])
also have ... = (*g l* $*$ *h l*) $\cdot_v$ *v* **by** (*simp add*: *mat-fun-eigen*[*OF A vn ev*] *smult-smult-assoc mult.commute*)
finally show (*mat-fun g A* $*$ *mat-fun h A*) $*_v$ *v* = (*g l* $*$ *h l*) $\cdot_v$ *v* **.**
qed
qed

theorem *mat-fun-dagger*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows (*mat-fun g A*)† = *mat-fun* (λ*x*. *cnj* (*g x*)) *A*
proof −
obtain *U D* **where** *sd*: *spectral-decomp A U D* **using** *spectral-decomp-exists*[*OF A*] **by** *blast*
note *d* = *spectral-decompD*[*OF A*(*1*) *sd*]
have *gc*: *diag-matrix* (*map g D*) ∈ *carrier-mat n n* **using** *d*(*3*) *diag-matrix-carrier* **by** (*metis length-map*)
have *Ud*: $U^\dagger$ ∈ *carrier-mat n n* **by** (*rule dagger-carrier*[*OF d*(*1*)])
have *cg*: *diag-matrix* (*map* (λ*x*. *cnj* (*g x*)) *D*) ∈ *carrier-mat n n*
using *diag-matrix-carrier*[*of map* (λ*x*. *cnj* (*g x*)) *D*] *d*(*3*) **by** *simp*
have *Ug*: *U* $*$ *diag-matrix* (*map g D*) ∈ *carrier-mat n n* **by** (*rule mult-carrier-mat*[*OF d*(*1*) *gc*])
have (*U* $*$ *diag-matrix* (*map g D*) $*$ ($U^\dagger$))† = (($U^\dagger$)†) $*$ ((*U* $*$ *diag-matrix* (*map g D*))†)
by (*rule dagger-of-prod*) (*use d*(*1*,*3*) **in** *simp*)
also have ... = *U* $*$ (((*diag-matrix* (*map g D*))†) $*$ ($U^\dagger$))
using *dagger-of-prod*[*of U diag-matrix* (*map g D*)] *d*(*1*,*3*) *gc* **by** (*simp add*: *dagger-of-dagger-is-id*)
also have ... = *U* $*$ (*diag-matrix* (*map* (λ*x*. *cnj* (*g x*)) *D*) $*$ ($U^\dagger$))
by (*simp add*: *diag-matrix-dagger o-def*)
also have ... = *U* $*$ *diag-matrix* (*map* (λ*x*. *cnj* (*g x*)) *D*) $*$ ($U^\dagger$)
by (*rule assoc-mult-mat*[*OF d*(*1*) *cg Ud*, *symmetric*])
finally show *?thesis* **using** *mat-fun-decomp*[*OF A sd*] **by** *simp*
qed

Hence $f(A)$ is Hermitian for real f, and $g(A)$ is unitary when $|g| = 1$ on the spectrum, since $g(A)^* g(A) = |g|^2(A) = I$.

lemma *mat-fun-hermitian*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *hermitian* (*mat-fun* (λ*x*. *complex-of-real* (*f x*)) *A*)
unfolding *hermitian-dagger* **by** (*simp add*: *mat-fun-dagger*[*OF A*])

theorem *mat-fun-unitary*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *g*: ⋀*l*. *l* ∈ *herm-spectrum A* ⟹ *cmod* (*g l*) = *1*
shows *unitary* (*mat-fun g A*)
proof −
have *c*: *mat-fun* (λ*x*. *cnj* (*g x*) $*$ *g x*) *A* = 1_m *n mat-fun* (λ*x*. *g x* $*$ *cnj* (*g x*)) *A* = 1_m *n*

proof −
have *e*: *cnj* (*g l*) ∗ *g l* = *1 g l* ∗ *cnj* (*g l*) = *1* **if** *l* ∈ *herm-spectrum A* **for** *l*
proof −
have *g l* ∗ *cnj* (*g l*) = *complex-of-real* ((*cmod* (*g l*))^2) **by** (*rule complex-norm-square*[*symmetric*])
thus *g l* ∗ *cnj* (*g l*) = *1 cnj* (*g l*) ∗ *g l* = *1* **using** *g*[*OF that*] **by** (*simp-all add*: *mult.commute*)
qed
show *mat-fun* (λ*x*. *cnj* (*g x*) ∗ *g x*) *A* = *1*$_m$ *n*
using *mat-fun-cong*[*OF A*, *of* λ*x*. *cnj* (*g x*) ∗ *g x* λ-. *1*] *e*(*1*) *mat-fun-one*[*OF A*] **by** *simp*
show *mat-fun* (λ*x*. *g x* ∗ *cnj* (*g x*)) *A* = *1*$_m$ *n*
using *mat-fun-cong*[*OF A*, *of* λ*x*. *g x* ∗ *cnj* (*g x*) λ-. *1*] *e*(*2*) *mat-fun-one*[*OF A*] **by** *simp*
qed
show *?thesis* **unfolding** *unitary-def*
using *c mat-fun-mult*[*OF A*] *mat-fun-dagger*[*OF A*] *carrier-matD*[*OF mat-fun-carrier*[*OF A*, *of g*]] **by** *simp*
qed

The calculus is additive in the function: $(g + h)(A) = g(A) + h(A)$.

lemma *mat-fun-add*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun* (λ*x*. *g x* + *h x*) *A* = *mat-fun g A* + *mat-fun h A*
proof (*rule mat-fun-eqI*[*OF A*])
show *mat-fun g A* + *mat-fun h A* ∈ *carrier-mat n n* **using** *A* **by** *simp*
show *mat-fun g A* + *mat-fun h A acts-as* (λ*x*. *g x* + *h x*) *on-eigenvectors-of A*
unfolding *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix *v l* **assume** *v*: *dim-vec v* = *dim-row A* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
have *vn*: *dim-vec v* = *n* **using** *v A*(*1*) **by** *simp*
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF vn*])
have (*mat-fun g A* + *mat-fun h A*) ∗$_v$ *v* = *mat-fun g A* ∗$_v$ *v* + *mat-fun h A* ∗$_v$ *v*
by (*rule add-mult-distrib-mat-vec*[*OF mat-fun-carrier*[*OF A*] *mat-fun-carrier*[*OF A*] *vc*])
also have ... = *g l* ·$_v$ *v* + *h l* ·$_v$ *v* **by** (*simp add*: *mat-fun-eigen*[*OF A vn ev*])
also have ... = (*g l* + *h l*) ·$_v$ *v* **by** (*simp add*: *add-smult-distrib-vec*)
finally show (*mat-fun g A* + *mat-fun h A*) ∗$_v$ *v* = (*g l* + *h l*) ·$_v$ *v* **.**
qed
qed

It is homogeneous: $(cg)(A) = c\,g(A)$.

lemma *mat-fun-smult*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun* (λ*x*. *c* ∗ *g x*) *A* = *c* ·$_m$ *mat-fun g A*
proof (*rule mat-fun-eqI*[*OF A*])
show *c* ·$_m$ *mat-fun g A* ∈ *carrier-mat n n* **using** *A* **by** *simp*

show $c \cdot_m$ *mat-fun g A acts-as* $(\lambda x.\ c * g\ x)$ *on-eigenvectors-of A* **unfolding** *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix *v l* **assume** *v*: *dim-vec v = dim-row A* **and** *ev*: $A *_v v = complex\text{-}of\text{-}real\ l \cdot_v v$
have *vn*: *dim-vec v = n* **using** *v A*(*1*) **by** *simp*
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF vn*])
have $(c \cdot_m mat\text{-}fun\ g\ A) *_v v = c \cdot_v (mat\text{-}fun\ g\ A *_v v)$
by (*rule smult-mat-mult-mat-vec-assoc*[*OF mat-fun-carrier*[*OF A*] *vc*])
thus $(c \cdot_m mat\text{-}fun\ g\ A) *_v v = (c * g\ l) \cdot_v v$ **by** (*simp add*: *mat-fun-eigen*[*OF A vn ev*] *smult-smult-assoc*)
qed
qed

Any two functions of A commute: $g(A)\,h(A) = h(A)\,g(A)$.

lemma *mat-fun-commute*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun g A* ∗ *mat-fun h A* = *mat-fun h A* ∗ *mat-fun g A*
by (*simp add*: *mat-fun-mult*[*OF A*] *mult.commute*)

Powers: $g(A)^k = (g^k)(A)$.

lemma *mat-fun-pow*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun g A* $\hat{}_m$ *k* = *mat-fun* $(\lambda x.\ g\ x\ \hat{}\ k)$ *A*
proof (*induction k*)
case *0* **show** *?case* **using** *mat-fun-one*[*OF A*] *mat-fun-carrier*[*OF A*, *of g*] **by** *simp*
next
case (*Suc k*)
have *mat-fun g A* $\hat{}_m$ *Suc k* = *mat-fun* $(\lambda x.\ g\ x\ \hat{}\ k)$ *A* ∗ *mat-fun g A* **using** *Suc* **by** *simp*
also have ... = *mat-fun* $(\lambda x.\ g\ x\ \hat{}\ Suc\ k)$ *A* **by** (*simp add*: *mat-fun-mult*[*OF A*] *mult.commute*)
finally show *?case* **.**
qed

Composition: for real f, $g(f(A)) = (g \circ f)(A)$, since an eigenvector of A with eigenvalue λ is one of $f(A)$ with eigenvalue $f(\lambda)$.

lemma *mat-fun-comp*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun g* (*mat-fun* $(\lambda x.\ complex\text{-}of\text{-}real\ (f\ x))$ *A*) = *mat-fun* $(\lambda x.\ g\ (f\ x))$ *A*
proof −
let *?B* = *mat-fun* $(\lambda x.\ complex\text{-}of\text{-}real\ (f\ x))$ *A*
have *B*: *?B* ∈ *carrier-mat n n hermitian ?B* **using** *A mat-fun-hermitian*[*OF A*, *of f*] **by** *simp-all*
show *?thesis*
proof (*rule mat-fun-eqI*[*OF A*, *symmetric*])
show *mat-fun g ?B* ∈ *carrier-mat n n* **using** *B* **by** *simp*

show *mat-fun g ?B acts-as* (λ*x*. *g* (*f x*)) *on-eigenvectors-of A* **unfolding** *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix *v l* **assume** *v*: *dim-vec v* = *dim-row A* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
have *vn*: *dim-vec v* = *n* **using** *v A*(*1*) **by** *simp*
have *?B* ∗$_v$ *v* = *complex-of-real* (*f l*) ·$_v$ *v* **by** (*rule mat-fun-eigen*[*OF A vn ev*])
thus *mat-fun g ?B* ∗$_v$ *v* = *g* (*f l*) ·$_v$ *v* **by** (*rule mat-fun-eigen*[*OF B vn*])
qed
qed
qed

For a real diagonal matrix the calculus acts entrywise:

$$g(\mathrm{diag}(D_1, \ldots, D_n)) = \mathrm{diag}(g(D_1), \ldots, g(D_n)).$$

lemma *mat-fun-diag*:
mat-fun g (*diag-matrix* (*map complex-of-real D*)) = *diag-matrix* (*map g D*)
proof −
let *?n* = *length D* **and** *?A* = *diag-matrix* (*map complex-of-real D*)
have *Ac*: *?A* ∈ *carrier-mat ?n ?n* **by** (*metis diag-matrix-carrier length-map*)
have *Ah*: *hermitian ?A* **unfolding** *hermitian-dagger diag-matrix-dagger* **by** (*simp add*: *comp-def*)
have *sd*: *spectral-decomp ?A* (*1*$_m$ *?n*) *D*
unfolding *spectral-decomp-def* **using** *Ac* **by** (*simp add*: *id-is-unitary*)
have *mat-fun g ?A* = (*1*$_m$ *?n* ∗ *diag-matrix* (*map g D*)) ∗ ((*1*$_m$ *?n*)†) **by** (*rule mat-fun-decomp*[*OF Ac Ah sd*])
also have ... = *diag-matrix* (*map g D*) **by** *simp*
finally show *?thesis* **.**
qed

Powers: $G^{i+j} = G^i G^j$, and $Uu = zu$ implies $U^k u = z^k u$.

lemma *pow-mat-add*: *G* ∈ *carrier-mat m m* ⟹ *G* ^$_m$ (*i* + *j*) = *G* ^$_m$ *i* ∗ *G* ^$_m$ *j*
proof (*induction j*)
case *0*
show *?case* **using** *0* **by** *simp*
next
case (*Suc j*)
have *G* ^$_m$ (*i* + *Suc j*) = *G* ^$_m$ *i* ∗ *G* ^$_m$ *j* ∗ *G* **using** *Suc* **by** *simp*
also have ... = *G* ^$_m$ *i* ∗ (*G* ^$_m$ *j* ∗ *G*) **by** (*rule assoc-mult-mat*) (*use Suc.prems* **in** *auto*)
finally show *?case* **by** *simp*
qed

lemma *pow-eigen*:
fixes *U* :: *complex Matrix.mat*
assumes *U*: *U* ∈ *carrier-mat m m* **and** *u*: *dim-vec u* = *m* **and** *ev*: *U* ∗$_v$ *u* = *z* ·$_v$ *u*

shows $(U \mathbin{\hat{}}_m k) *_v u = z \mathbin{\hat{}} k \cdot_v u$
proof (*induction k*)
case *0*
have $u \in$ *carrier-vec m* **using** *u* **by** (*rule carrier-vecI*)
thus *?case* **using** *carrier-matD*(*1*)[*OF U*] **by** (*auto intro*!: *eq-vecI*)
next
case (*Suc k*)
have *uc*: $u \in$ *carrier-vec m* **using** *u* **by** (*rule carrier-vecI*)
have $(U \mathbin{\hat{}}_m$ *Suc k*$) *_v u = (U \mathbin{\hat{}}_m k) *_v (U *_v u)$
using *assoc-mult-mat-vec*[*OF pow-carrier-mat*[*OF U*] *U uc*, *of k*] **by** *simp*
also have $\ldots = z \cdot_v ((U \mathbin{\hat{}}_m k) *_v u)$ **unfolding** *ev* **by** (*rule mult-mat-vec*[*OF pow-carrier-mat*[*OF U*] *uc*])
also have $\ldots = z \mathbin{\hat{}}$ *Suc k* $\cdot_v u$ **unfolding** *Suc* **by** (*simp add*: *smult-smult-assoc*)
finally show *?case* **.**
qed

3.6 Time evolution

e^{itA}, the evolution under the Hamiltonian A.

definition *evolution* :: *complex Matrix.mat* ⇒ *real* ⇒ *complex Matrix.mat* **where**
evolution A t = *mat-fun* (λ*x*. *cis* (*t* ∗ *x*)) *A*

context
fixes *A* :: *complex Matrix.mat* **and** *n* :: *nat*
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
begin

e^{itA} is unitary, acts on an eigenvector by $e^{itA}v = e^{it\lambda}v$, and is a one-parameter group:

$$e^{i0A} = I, \qquad e^{i(s+t)A} = e^{isA}e^{itA}, \qquad \left(e^{itA}\right)^k = e^{iktA}.$$

lemma *evolution-carrier* [*simp*]: *evolution A t* ∈ *carrier-mat n n*
unfolding *evolution-def* **using** *A* **by** *simp*

lemma *evolution-dims* [*simp*]: *dim-row* (*evolution A t*) = *n dim-col* (*evolution A t*) = *n*
using *carrier-matD*[*OF evolution-carrier*] **by** *auto*

lemma *evolution-eigen*:
dim-vec v = *n* ⟹ *A *_v v* = *complex-of-real l* ·_v *v* ⟹ *evolution A t *_v v* = *cis* (*t* ∗ *l*) ·_v *v*
unfolding *evolution-def* **by** (*rule mat-fun-eigen*[*OF A*])

lemma *evolution-unitary*: *unitary* (*evolution A t*)
unfolding *evolution-def* **by** (*rule mat-fun-unitary*[*OF A*]) *simp*

lemma *evolution-zero*: *evolution A 0* = *1*$_m$ *n*
unfolding *evolution-def* **using** *mat-fun-one*[*OF A*] **by** *simp*

lemma *evolution-add*: *evolution A (s + t) = evolution A s ∗ evolution A t*
unfolding *evolution-def mat-fun-mult*[*OF A*] **by** (*simp add*: *cis-mult distrib-right*)

lemma *evolution-pow*: *evolution A t ^_m k = evolution A (real k ∗ t)*
proof (*induction k*)
case *0*
show *?case* **using** *evolution-zero carrier-matD*(*1*)[*OF evolution-carrier*] **by** *simp*
next
case (*Suc k*)
have *evolution A t ^_m Suc k = evolution A (real k ∗ t) ∗ evolution A t* **using** *Suc* **by** *simp*
also have *. . . = evolution A (real (Suc k) ∗ t)* **by** (*simp add*: *evolution-add*[*symmetric*] *distrib-right add.commute*)
finally show *?case* **.**
qed

end

3.7 The inverse

$A^{-1} = g(A)$ for $g(x) = 1/x$; since $1/0 = 0$ in HOL this is the pseudo-inverse in general.

definition *herm-inv* :: *complex Matrix.mat ⇒ complex Matrix.mat* **where**
herm-inv A = mat-fun (*λx. complex-of-real* (*inverse x*)) *A*

A^{-1} is an $n \times n$ matrix.

lemma *herm-inv-carrier* [*simp*]: *A ∈ carrier-mat n n ⟹ hermitian A ⟹ herm-inv A ∈ carrier-mat n n*
unfolding *herm-inv-def* **by** *simp*

lemma *herm-inv-dims* [*simp*]:
assumes *A ∈ carrier-mat n n hermitian A*
shows *dim-row* (*herm-inv A*) = *n dim-col* (*herm-inv A*) = *n*
using *herm-inv-carrier*[*OF assms*] **by** *auto*

If 0 is not an eigenvalue, $g(x) = 1/x$ gives the inverse: $A^{-1}A = AA^{-1} = I$.

theorem *herm-inv*:
assumes *A*: *A ∈ carrier-mat n n hermitian A* **and** *z*: *0 ∉ herm-spectrum A*
shows *herm-inv A ∗ A = 1_m n A ∗ herm-inv A = 1_m n*
proof −
have *mat-fun* (*λx. complex-of-real* (*inverse x*) ∗ *complex-of-real x*) *A = mat-fun* (*λ-. 1*) *A*
mat-fun (*λx. complex-of-real x ∗ complex-of-real* (*inverse x*)) *A = mat-fun* (*λ-. 1*) *A*
by (*rule mat-fun-cong*[*OF A*], *metis z of-real-eq-0-iff left-inverse right-inverse of-real-inverse*)+
thus *herm-inv A ∗ A = 1_m n A ∗ herm-inv A = 1_m n*

unfolding *herm-inv-def* **using** *mat-fun-mult*[*OF A*] *mat-fun-id*[*OF A*] *mat-fun-one*[*OF A*] **by** *metis+*
qed

So $x = A^{-1}b$ solves $Ax = b$; A^{-1} acts on eigenvectors by λ^{-1} and is Hermitian.

lemma *herm-inv-solves*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A* **and** *z*: *0* ∉ *herm-spectrum A* **and** *b*: *dim-vec b* = *n*
shows *A* ∗$_v$ (*herm-inv A* ∗$_v$ *b*) = *b*
proof −
have *Hc*: *herm-inv A* ∈ *carrier-mat n n* **unfolding** *herm-inv-def* **by** (*rule mat-fun-carrier*[*OF A*])
have *A* ∗$_v$ (*herm-inv A* ∗$_v$ *b*) = (*A* ∗ *herm-inv A*) ∗$_v$ *b* **by** (*rule assoc-mult-mat-vec*[*symmetric*, *OF A*(*1*) *Hc*]) (*use b* **in** *auto*)
also have *A* ∗ *herm-inv A* = *1*$_m$ *n* **by** (*rule herm-inv*(*2*)[*OF A z*])
finally show *?thesis* **using** *b* **by** (*simp add*: *one-mult-mat-vec*[*OF carrier-vecI*[*OF b*]])
qed

lemma *herm-inv-eigen*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *v*: *dim-vec v* = *n* **and** *ev*: *A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
shows *herm-inv A* ∗$_v$ *v* = *complex-of-real* (*inverse l*) ·$_v$ *v*
unfolding *herm-inv-def* **by** (*rule mat-fun-eigen*[*OF A v ev*])

lemma *herm-inv-hermitian*: *A* ∈ *carrier-mat n n* ⟹ *hermitian A* ⟹ *hermitian* (*herm-inv A*)
unfolding *herm-inv-def* **by** (*rule mat-fun-hermitian*)

lemma *mult-zero-vec-right*: *B* ∈ *carrier-mat K n* ⟹ *B* ∗$_v$ *0*$_v$ *n* = *0*$_v$ *K*
by (*rule eq-vecI*) (*auto simp*: *scalar-prod-def*)

For a unit vector b and invertible A the solution $A^{-1}b$ is not zero.

lemma *herm-inv-sol-pos*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A* **and** *z*: *0* ∉ *herm-spectrum A* **and** *b*: *dim-vec b* = *n* ‖*b*‖ = *1*
shows *0* < ‖*herm-inv A* ∗$_v$ *b*‖
proof (*rule vnorm-pos*)
have *Hc*: *herm-inv A* ∈ *carrier-mat n n* **unfolding** *herm-inv-def* **by** (*rule mat-fun-carrier*[*OF A*])
thus *dim-vec* (*herm-inv A* ∗$_v$ *b*) = *n* **by** *simp*
show *herm-inv A* ∗$_v$ *b* ≠ *0*$_v$ *n*
proof
assume *herm-inv A* ∗$_v$ *b* = *0*$_v$ *n*
hence *b* = *0*$_v$ *n* **using** *herm-inv-solves*[*OF A z b*(*1*)] *mult-zero-vec-right*[*OF A*(*1*)] **by** *simp*
thus *False* **using** *b*(*2*) *vnorm-zero-vec* **by** *simp*
qed

qed

4 Finite sums of matrices

For QHLProver's sums $\sum_{k<n} F_k$ of $K \times K$ matrices: entries are sums of entries, and a sum of multiples of one matrix is a multiple of it,

$$\Big(\sum_k F_k\Big)_{ij} = \sum_k (F_k)_{ij}, \qquad \sum_k f_k\, B = \Big(\sum_k f_k\Big) B.$$

lemma *matrix-sum-index*:
assumes *f*: $\bigwedge k.\ k < n \Longrightarrow f\ k \in$ *carrier-mat K K* **and** *ij*: $i < K\ j < K$
shows *matrix-sum K f n* $$ (*i*, *j*) = ($\sum$ *k*<*n*. *f k* $$ (*i*, *j*))
using *f*
proof (*induction n*)
case (*Suc n*)
have *c*: *matrix-sum K f n* ∈ *carrier-mat K K* **by** (*rule matrix-sum-dim*) (*use Suc.prems* **in** *auto*)
have *matrix-sum K f* (*Suc n*) $$ (*i*, *j*) = *f n* $$ (*i*, *j*) + *matrix-sum K f n* $$ (*i*, *j*)
using *c Suc.prems*[*of n*] *ij* **by** *simp*
thus *?case* **using** *Suc* **by** (*simp add*: *add-ac*)
qed (*use ij* **in** *simp*)

lemma *matrix-sum-scalar*:
assumes *B*: *B* ∈ *carrier-mat d d*
shows *matrix-sum d* (λ*k*. *f k* ·$_m$ *B*) *m* = ($\sum$ *k*<*m*. *f k*) ·$_m$ *B*
proof (*induction m*)
case *0* **show** *?case* **by** (*rule eq-matI*) (*use B* **in** *auto*)
next
case (*Suc m*)
show *?case* **unfolding** *matrix-sum.simps Suc.IH* **using** *B* **by** (*intro eq-matI*) (*auto simp*: *algebra-simps*)
qed

5 The Lie–Trotter product formula

For Hermitian $H_1, \ldots, H_m$ with $\|H_j v\| \le L\|v\|$, r rounds of $e^{itH_1/r} \cdots e^{itH_m/r}$ are within $2(mLt)^2/r$ of $e^{it(H_1+\cdots+H_m)}$ (*trotter*). Each factor is $I + i\delta H_j$ up to $(\delta L)^2$, from $|e^{ix} - 1 - ix| \le x^2$ and the functional calculus; the errors add along products and powers of unitaries.

5.1 Scalar bounds

Taylor remainders of first order:

$|\sin x - x| \le \frac{x^2}{2}, \qquad |\cos x - 1| \le \frac{x^2}{2}, \qquad |e^{ix} - 1 - ix| \le x^2, \qquad |e^{ix} - 1| \le |x|.$

lemma *sin-minus-x-bound*: $|sin\ (x::real) - x| \le x^2\ /\ 2$
proof (*cases* $|x| \le 3$)
case *True*
have *m*: $|sin\ x - (\sum m{<}3.\ sin\text{-}coeff\ m * x\ \hat{}\ m)| \le inverse\ (fact\ 3) * |x|\ \hat{}\ 3$
by (*rule Maclaurin-sin-bound*)
have *s*: $(\sum m{<}3.\ sin\text{-}coeff\ m * x\ \hat{}\ m) = x$
by (*simp add*: *sin-coeff-def eval-nat-numeral*)
have *f*: $inverse\ (fact\ 3) * |x|\ \hat{}\ 3 = |x|\ \hat{}\ 3\ /\ (6::real)$ **by** (*simp add*: *fact-numeral*)
have $|sin\ x - x| \le |x|\ \hat{}\ 3\ /\ 6$ **using** *m* **unfolding** *s f* **.**
also have $\ldots = |x| * x^2\ /\ 6$ **by** (*simp add*: *power2-eq-square power3-eq-cube*)
also have $\ldots \le 3 * x^2\ /\ 6$ **by** (*rule divide-right-mono*, *rule mult-right-mono*)
(*use True* **in** *simp-all*)
finally show *?thesis* **by** *simp*
next
case *False*
have $|sin\ x - x| \le 1 + |x|$ **using** *abs-sin-le-one*[*of x*] **by** *linarith*
also have $\ldots \le x^2\ /\ 2$
proof −
have $3 * |x| \le |x| * |x|$ **using** *False* **by** (*intro mult-right-mono*) *simp-all*
thus *?thesis* **using** *False* **by** (*simp add*: *power2-eq-square*)
qed
finally show *?thesis* **.**
qed

lemma *cos-minus-one-bound*: $|cos\ (x::real) - 1| \le x^2\ /\ 2$
proof −
have $cos\ x = 1 - 2 * sin\ (x\ /\ 2)\ \hat{}\ 2$ **using** *cos-double-sin*[*of x / 2*] **by** *simp*
hence $|cos\ x - 1| = 2 * sin\ (x\ /\ 2)\ \hat{}\ 2$ **by** *simp*
also have $\ldots \le 2 * (x\ /\ 2)\ \hat{}\ 2$
using *abs-le-square-iff*[*THEN iffD1*, *OF abs-sin-x-le-abs-x*[*of x / 2*]] **by** *simp*
also have $\ldots = x^2\ /\ 2$ **by** (*simp add*: *power2-eq-square*)
finally show *?thesis* **.**
qed

lemma *cis-remainder*: $cmod\ (cis\ (x::real) - 1 - \mathrm{i} * complex\text{-}of\text{-}real\ x) \le x^2$
proof −
have $cmod\ (cis\ x - 1 - \mathrm{i} * complex\text{-}of\text{-}real\ x) \le |Re\ (cis\ x - 1 - \mathrm{i} * complex\text{-}of\text{-}real\ x)| + |Im\ (cis\ x - 1 - \mathrm{i} * complex\text{-}of\text{-}real\ x)|$
by (*rule cmod-le*)
also have $\ldots = |cos\ x - 1| + |sin\ x - x|$ **by** *simp*
also have $\ldots \le x^2\ /\ 2 + x^2\ /\ 2$ **by** (*rule add-mono*[*OF cos-minus-one-bound sin-minus-x-bound*])
finally show *?thesis* **by** *simp*
qed

lemma *cis-minus-one*: *cmod* (*cis* (*x*::*real*) − *1*) ≤ |*x*|
proof −
have (*cmod* (*cis x* − *1*))² = (*cos x* − *1*)² + (*sin x*)² **by** (*simp add*: *cmod-def*)
also have ... = *2* − *2* ∗ *cos x* **using** *sin-cos-squared-add*[*of x*] **by** (*simp add*: *power2-eq-square algebra-simps*)
also have ... = *4* ∗ (*sin* (*x* / *2*))² **using** *cos-double-sin*[*of x* / *2*] **by** *simp*
also have ... ≤ *4* ∗ (*x* / *2*)²
using *abs-le-square-iff*[*THEN iffD1*, *OF abs-sin-x-le-abs-x*[*of x* / *2*]] **by** *simp*
also have ... = |*x*|² **by** (*simp add*: *power2-eq-square*)
finally show *?thesis* **by** (*simp add*: *abs-le-square-iff*[*symmetric*])
qed

5.2 Bounds from the spectrum

lemma *vnorm-minus-le*:
assumes *dim-vec a* = *dim-vec b*
shows ‖*a* − *b*‖ ≤ ‖*a*‖ + ‖*b*‖
proof −
have *a* − *b* = *a* + (−*1*) ·$_v$ *b* **using** *assms* **by** (*intro eq-vecI*) *simp-all*
thus *?thesis* **using** *vnorm-add-le*[*of a* (−*1*) ·$_v$ *b*] *assms cpx-vec-length-smult*[*of* −*1 b*] **by** *simp*
qed

If $\|Av\| \le L\|v\|$ for all v, every eigenvalue satisfies $|\lambda| \le L$; and if $|g| \le c$ on the spectrum, $\|g(A)v\| \le c\|v\|$.

lemma *herm-spectrum-bound*:
assumes *A*: *A* ∈ *carrier-mat n n* **and** *b*: ⋀*v*. *dim-vec v* = *n* ⟹ ‖*A* ∗$_v$ *v*‖ ≤ *L* ∗ ‖*v*‖
and *l*: *l* ∈ *herm-spectrum A*
shows |*l*| ≤ *L*
proof −
obtain *v* **where** *v*: *dim-vec v* = *n v* ≠ *0*$_v$ *n A* ∗$_v$ *v* = *complex-of-real l* ·$_v$ *v*
using *l A* **unfolding** *herm-spectrum-iff* **by** *auto*
have |*l*| ∗ ‖*v*‖ = ‖*A* ∗$_v$ *v*‖ **using** *v*(*3*) **by** (*simp add*: *cpx-vec-length-smult*)
also have ... ≤ *L* ∗ ‖*v*‖ **by** (*rule b*[*OF v*(*1*)])
finally show *?thesis* **using** *mult-right-le-imp-le vnorm-pos*[*OF v*(*1*,*2*)] **by** *blast*
qed

lemma *mat-fun-bound*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A* **and** *c*: *0* ≤ *c*
and *g*: ⋀*l*. *l* ∈ *herm-spectrum A* ⟹ *cmod* (*g l*) ≤ *c* **and** *v*: *dim-vec v* = *n*
shows ‖*mat-fun g A* ∗$_v$ *v*‖ ≤ *c* ∗ ‖*v*‖
proof −
obtain *U Dl* **where** *sd*: *spectral-decomp A U Dl* **using** *spectral-decomp-exists*[*OF A*] **by** *blast*
note *d* = *spectral-decompD*[*OF A*(*1*) *sd*]
have *Ud*: *U*† ∈ *carrier-mat n n* **by** (*rule dagger-carrier*[*OF d*(*1*)])
have *gD*: *diag-matrix* (*map g Dl*) ∈ *carrier-mat n n* **using** *diag-matrix-carrier*[*of*

map g Dl] *d*(*3*) **by** *simp*
define *w* **where** $w = U^\dagger *_v v$
have *w*: *dim-vec w* = *n* **using** *carrier-matD*[*OF d*(*1*)] **by** (*simp add*: *w-def*)
have *vc*: $v \in$ *carrier-vec n* **by** (*rule carrier-vecI*[*OF v*])
have *wc*: $w \in$ *carrier-vec n* **by** (*rule carrier-vecI*[*OF w*])
have *mat-fun g A* $*_v$ *v* = (*U* $*$ *diag-matrix* (*map g Dl*)) $*_v$ *w*
unfolding *mat-fun-decomp*[*OF A sd*] *w-def* **by** (*rule assoc-mult-mat-vec*[*OF mult-carrier-mat*[*OF d*(*1*) *gD*] *Ud vc*])
also have ... = *U* $*_v$ (*diag-matrix* (*map g Dl*) $*_v$ *w*) **by** (*rule assoc-mult-mat-vec*[*OF d*(*1*) *gD wc*])
finally have ‖*mat-fun g A* $*_v$ *v*‖ = ‖*diag-matrix* (*map g Dl*) $*_v$ *w*‖
using *unitary-vnorm*[*OF d*(*1*,*2*), *of diag-matrix* (*map g Dl*) $*_v$ *w*] *gD d*(*3*) **by** *simp*
also have ... ≤ *c* $*$ ‖*w*‖
proof −
have *dv*: *diag-matrix* (*map g Dl*) $*_v$ *w* = *Matrix.vec n* (λ*i*. *g* (*Dl* ! *i*) $*$ *w* $ *i*)
using *diag-matrix-mult-vec*[*of w map g Dl*] *w d*(*3*) **by** (*auto intro*!: *eq-vecI*)
have ‖*diag-matrix* (*map g Dl*) $*_v$ *w*‖^2 = (∑ *i*<*n*. (*cmod* (*g* (*Dl* ! *i*) $*$ *w* $ *i*))^2)
unfolding *dv vnorm-sq* **by** *simp*
also have ... ≤ (∑ *i*<*n*. *c*^{2} $*$ (*cmod* (*w* $ *i*))^2)
proof (*rule sum-mono*)
fix *i* **assume** *i* ∈ {..<*n*}
hence *cmod* (*g* (*Dl* ! *i*)) ≤ *c* **using** *g spectral-decomp-eig-in-spectrum*[*OF A*(*1*) *sd*] **by** *simp*
hence (*cmod* (*g* (*Dl* ! *i*)))^2 ≤ *c*^{2} **by** (*rule power-mono*) *simp*
thus (*cmod* (*g* (*Dl* ! *i*) $*$ *w* $ *i*))^2 ≤ *c*^{2} $*$ (*cmod* (*w* $ *i*))^2
by (*simp add*: *norm-mult power-mult-distrib mult-right-mono*)
qed
also have ... = (*c* $*$ ‖*w*‖)^2 **by** (*simp add*: *vnorm-sq w power-mult-distrib sum-distrib-left*)
finally show *?thesis* **by** (*rule power2-le-imp-le*) (*simp add*: *c vnorm-ge0*)
qed
also have ‖*w*‖ = ‖*v*‖ **unfolding** *w-def* **by** (*rule unitary-vnorm*[*OF Ud unitary-dagger*[*OF d*(*2*)] *v*])
finally show *?thesis* **.**
qed

The functions $e^{idx} - 1 - idx$ and $e^{idx} - 1$ applied to A give $e^{idA} - I - idA$ and $e^{idA} - I$.

lemma *mat-fun-evolution-rem*:
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
shows *mat-fun* (λ*x*. *cis* (*d* $*$ *x*) − *1* − i $*$ *complex-of-real* (*d* $*$ *x*)) *A* =
evolution A d − *1*$_m$ *n* − (i $*$ *complex-of-real d*) $\cdot_m$ *A* (**is** - = *?B*)
proof (*rule mat-fun-eqI*[*OF A*])
show *?B* ∈ *carrier-mat n n* **by** (*intro minus-carrier-mat smult-carrier-mat*) (*rule A*(*1*))
show *?B acts-as* (λ*x*. *cis* (*d* $*$ *x*) − *1* − i $*$ *complex-of-real* (*d* $*$ *x*)) *on-eigenvectors-of A* **unfolding** *acts-on-eigenvectors-def*
proof (*intro allI impI*)

fix v l **assume** v: *dim-vec* v = *dim-row* A **and** ev: $A *_v v$ = *complex-of-real* l $\cdot_v$ v
have vn: *dim-vec* $v = n$ **using** v $A(1)$ **by** *simp*
have vc: $v \in$ *carrier-vec* n **by** (*rule carrier-vecI*[*OF vn*])
have e: *evolution* A $d *_v v$ = *cis* $(d * l) \cdot_v v$ **by** (*rule evolution-eigen*[*OF A vn ev*])
have *E1*: *evolution* A $d - 1_m$ $n \in$ *carrier-mat* n n **by** (*rule minus-carrier-mat*) *simp*
have *m1*: (*evolution* A $d - 1_m$ n) $*_v$ v = *evolution* A $d *_v v - 1_m$ $n *_v v$
by (*rule minus-mult-distrib-mat-vec*[*OF evolution-carrier*[*OF A*] *one-carrier-mat vc*])
have *?B* $*_v$ v = (*evolution* A $d - 1_m$ n) $*_v$ v − ((i ∗ *complex-of-real* d) $\cdot_m$ A) $*_v$ v
by (*rule minus-mult-distrib-mat-vec*[*OF E1 smult-carrier-mat*[*OF A(1)*] *vc*])
also have ... = *evolution* A $d *_v v - 1_m$ $n *_v v$ − ((i ∗ *complex-of-real* d) $\cdot_m$ A) $*_v$ v
by (*simp only*: *m1*)
also have ... = *cis* $(d * l) \cdot_v v - v$ − (i ∗ *complex-of-real* d) $\cdot_v$ (*complex-of-real* l $\cdot_v$ v)
using e *ev smult-mat-mult-mat-vec-assoc*[*OF A(1) carrier-vecI*[*OF vn*]] *vc* **by** *simp*
also have ... = (*cis* $(d * l)$ − *1* − i ∗ *complex-of-real* $(d * l)$) $\cdot_v$ v
by (*intro eq-vecI*) (*simp-all add*: *algebra-simps*)
finally show *?B* $*_v$ v = (*cis* $(d * l)$ − *1* − i ∗ *complex-of-real* $(d * l)$) $\cdot_v$ v **.**
qed
qed

lemma *mat-fun-evolution-one*:
assumes A: $A \in$ *carrier-mat* n n *hermitian* A
shows *mat-fun* $(\lambda x.$ *cis* $(d * x) - 1)$ A = *evolution* A $d - 1_m$ n (**is** - = *?B*)
proof (*rule mat-fun-eqI*[*OF A*])
show *?B* $\in$ *carrier-mat* n n **by** (*rule minus-carrier-mat*) *simp*
show *?B acts-as* $(\lambda x.$ *cis* $(d * x) - 1)$ *on-eigenvectors-of* A **unfolding** *acts-on-eigenvectors-def*
proof (*intro allI impI*)
fix v l **assume** v: *dim-vec* v = *dim-row* A **and** ev: $A *_v v$ = *complex-of-real* l $\cdot_v$ v
have vn: *dim-vec* $v = n$ **using** v $A(1)$ **by** *simp*
have vc: $v \in$ *carrier-vec* n **by** (*rule carrier-vecI*[*OF vn*])
have e: *evolution* A $d *_v v$ = *cis* $(d * l) \cdot_v v$ **by** (*rule evolution-eigen*[*OF A vn ev*])
have *?B* $*_v$ v = *evolution* A $d *_v v - 1_m$ $n *_v v$
by (*rule minus-mult-distrib-mat-vec*[*OF evolution-carrier*[*OF A*] *one-carrier-mat vc*])
also have ... = (*cis* $(d * l) - 1) \cdot_v v$ **using** e *vc* **by** (*intro eq-vecI*, *simp-all add*: *algebra-simps*)
finally show *?B* $*_v$ v = (*cis* $(d * l) - 1) \cdot_v v$ **.**
qed
qed

5.3 Closeness of operators

U and V are η-close, written $U \approx_\eta V$, if $\|Uv - Vv\| \leq \eta\|v\|$ for every v: the operator norm of $U - V$ is at most η.

definition *op-close* :: *complex Matrix.mat* ⇒ *real* ⇒ *complex Matrix.mat* ⇒ *bool*
((- ≈-/ -) [51, 0, 51] 50) **where**
$U \approx_\eta V \longleftrightarrow (\forall v.$ *dim-vec* $v =$ *dim-col* $U \longrightarrow \|U *_v v - V *_v v\| \leq \eta * \|v\|)$

For $U \in M_{m,n}$ closeness is a bound on the vectors of length n.

lemma *op-close-iff*:
assumes $U \in$ *carrier-mat* m n
shows $U \approx_\eta V \longleftrightarrow (\forall v.$ *dim-vec* $v = n \longrightarrow \|U *_v v - V *_v v\| \leq \eta * \|v\|)$
using *assms* **by** (*simp add*: *op-close-def*)

lemma *op-closeI*:
assumes $U \in$ *carrier-mat* m n **and** $\bigwedge v.$ *dim-vec* $v = n \Longrightarrow \|U *_v v - V *_v v\| \leq \eta * \|v\|$
shows $U \approx_\eta V$
using *assms* **by** (*simp add*: *op-close-def*)

lemma *op-closeD*:
$U \approx_\eta V \Longrightarrow U \in$ *carrier-mat* m $n \Longrightarrow$ *dim-vec* $v = n \Longrightarrow \|U *_v v - V *_v v\| \leq \eta * \|v\|$
by (*simp add*: *op-close-def*)

Closeness is reflexive and symmetric, and passes to adjoints of unitaries, since $U^* - V^* = U^*(V - U)V^*$ and unitaries preserve norms.

lemma *op-close-refl*:
assumes *e*: $0 \leq \eta$ **and** *U*: $U \in$ *carrier-mat* n n
shows $U \approx_\eta U$
proof −
have *z*: $U *_v v - U *_v v = 0_v$ n **if** *dim-vec* $v = n$ **for** v :: *complex Matrix.vec*
using *that U* **by** (*intro eq-vecI*) *simp-all*
show *?thesis*
proof (*rule op-closeI*[*OF U*])
fix v :: *complex Matrix.vec* **assume** *v*: *dim-vec* $v = n$
have $\|U *_v v - U *_v v\| = 0$ **unfolding** *z*[*OF v*] **by** (*simp add*: *cpx-vec-length-def*)
thus $\|U *_v v - U *_v v\| \leq \eta * \|v\|$ **using** *e vnorm-ge0*[*of v*] **by** *simp*
qed
qed

lemma *op-close-sym*:
assumes $U \approx_\eta V$ $U \in$ *carrier-mat* n n $V \in$ *carrier-mat* n n
shows $V \approx_\eta U$
proof (*rule op-closeI*[*OF assms*(*3*)])
fix v :: *complex Matrix.vec* **assume** *v*: *dim-vec* $v = n$
have $\|V *_v v - U *_v v\| = \|U *_v v - V *_v v\|$ **by** (*rule vnorm-minus-commute*) (*use assms* **in** *simp*)

thus $\|V *_v v - U *_v v\| \leq \eta * \|v\|$ **using** *op-closeD*[*OF assms*(*1*,*2*) *v*] **by** *simp*
qed

lemma *op-close-dagger*:
assumes *U*: $U \in$ *carrier-mat n n unitary U* **and** *V*: $V \in$ *carrier-mat n n unitary V*
and *c*: $U \approx_\eta V$
shows $U^\dagger \approx_\eta V^\dagger$
proof (*rule op-closeI*[*OF dagger-carrier*[*OF U*(*1*)]])
fix *v* :: *complex Matrix.vec* **assume** *v*: *dim-vec v* = *n*
have *vc*: $v \in$ *carrier-vec n* **using** *v* **by** (*rule carrier-vecI*)
have *Ud*: $U^\dagger \in$ *carrier-mat n n* **and** *Vd*: $V^\dagger \in$ *carrier-mat n n* **using** *U V* **by** (*simp-all add*: *dagger-carrier*)
have *UU*: $U * (U^\dagger) = 1_m\ n$ **and** *VV*: $V * (V^\dagger) = 1_m\ n$ **using** *U V* **unfolding** *unitary-def* **by** *auto*
let *?w* = $V^\dagger *_v v$
have *w*: *dim-vec ?w* = *n* **using** *V*(*1*) **by** *simp*
have *dd*: *dim-vec* ($U^\dagger *_v v - ?w$) = *n* **using** *U*(*1*) *V*(*1*) **by** *simp*
have *uv*: $U *_v (U^\dagger *_v v) = v$ **using** *assoc-mult-mat-vec*[*OF U*(*1*) *Ud vc*, *symmetric*] *UU vc* **by** *simp*
have *vw*: $V *_v ?w = v$ **using** *assoc-mult-mat-vec*[*OF V*(*1*) *Vd vc*, *symmetric*] *VV vc* **by** *simp*
have *e*: $U *_v (U^\dagger *_v v - ?w) = V *_v ?w - U *_v ?w$
proof −
have $U *_v (U^\dagger *_v v - ?w) = U *_v (U^\dagger *_v v) - U *_v ?w$
by (*rule mult-minus-distrib-mat-vec*[*OF U*(*1*)]) (*use Ud Vd vc* **in** *simp-all*)
thus *?thesis* **unfolding** *uv vw* **.**
qed
have $\|U^\dagger *_v v - ?w\| = \|U *_v (U^\dagger *_v v - ?w)\|$ **by** (*rule unitary-vnorm*[*OF U dd*, *symmetric*])
also have ... $= \|U *_v ?w - V *_v ?w\|$ **unfolding** *e* **by** (*rule vnorm-minus-commute*) (*use U V* **in** *simp*)
also have ... $\leq \eta * \|?w\|$ **by** (*rule op-closeD*[*OF c U*(*1*) *w*])
also have $\|?w\| = \|v\|$ **by** (*rule unitary-vnorm*[*OF Vd unitary-dagger*[*OF V*(*2*)] *v*])
finally show $\|U^\dagger *_v v - V^\dagger *_v v\| \leq \eta * \|v\|$ **.**
qed

5.4 First-order bounds

context
fixes *A* :: *complex Matrix.mat* **and** *n* :: *nat* **and** *L* :: *real*
assumes *A*: *A* ∈ *carrier-mat n n hermitian A*
and *Ab*: $\bigwedge v.$ *dim-vec v* = *n* $\Longrightarrow \|A *_v v\| \leq L * \|v\|$
begin

For $\|Av\| \leq L\|v\|$, the bounds above applied on the spectrum give

$$\|e^{idA}v - v - id\,Av\| \leq (dL)^2\|v\|, \qquad \|e^{idA}v - v\| \leq |d|L\|v\|.$$

lemma *evolution-first-order*:
assumes *v*: *dim-vec v* = *n*
shows ‖*evolution A d* ∗$_v$ *v* − *v* − (i ∗ *complex-of-real d*) ·$_v$ (*A* ∗$_v$ *v*)‖ ≤ (*d* ∗ *L*)^2 ∗ ‖*v*‖
proof −
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF v*])
have *E1*: *evolution A d* − *1*$_m$ *n* ∈ *carrier-mat n n* **by** (*rule minus-carrier-mat*) *simp*
have *m1*: (*evolution A d* − *1*$_m$ *n*) ∗$_v$ *v* = *evolution A d* ∗$_v$ *v* − *1*$_m$ *n* ∗$_v$ *v*
by (*rule minus-mult-distrib-mat-vec*[*OF evolution-carrier*[*OF A*] *one-carrier-mat vc*])
have *mat-fun* (λ*x*. *cis* (*d* ∗ *x*) − *1* − i ∗ *complex-of-real* (*d* ∗ *x*)) *A* ∗$_v$ *v* =
(*evolution A d* − *1*$_m$ *n* − (i ∗ *complex-of-real d*) ·$_m$ *A*) ∗$_v$ *v*
by (*simp only*: *mat-fun-evolution-rem*[*OF A*])
also have ... = (*evolution A d* − *1*$_m$ *n*) ∗$_v$ *v* − ((i ∗ *complex-of-real d*) ·$_m$ *A*) ∗$_v$ *v*
by (*rule minus-mult-distrib-mat-vec*[*OF E1 smult-carrier-mat*[*OF A*(*1*)] *vc*])
also have ... = *evolution A d* ∗$_v$ *v* − *v* − (i ∗ *complex-of-real d*) ·$_v$ (*A* ∗$_v$ *v*)
using *m1 smult-mat-mult-mat-vec-assoc*[*OF A*(*1*) *carrier-vecI*[*OF v*]] *vc* **by** *simp*
finally have *e*: *mat-fun* (λ*x*. *cis* (*d* ∗ *x*) − *1* − i ∗ *complex-of-real* (*d* ∗ *x*)) *A* ∗$_v$ *v* =
evolution A d ∗$_v$ *v* − *v* − (i ∗ *complex-of-real d*) ·$_v$ (*A* ∗$_v$ *v*) **.**
have ‖*mat-fun* (λ*x*. *cis* (*d* ∗ *x*) − *1* − i ∗ *complex-of-real* (*d* ∗ *x*)) *A* ∗$_v$ *v*‖ ≤ (*d* ∗ *L*)^2 ∗ ‖*v*‖
proof (*rule mat-fun-bound*[*OF A - - v*])
fix *l* **assume** *l0*: *l* ∈ *herm-spectrum A*
have *l*: |*l*| ≤ *L* **by** (*rule herm-spectrum-bound*[*OF A*(*1*) - *l0*]) (*rule Ab*)
have *l*^{2} ≤ *L*^{2} **using** *power-mono*[*OF l, of 2*] **by** *simp*
hence *d*^{2} ∗ *l*^{2} ≤ *d*^{2} ∗ *L*^{2} **by** (*rule mult-left-mono*) *simp*
hence (*d* ∗ *l*)^2 ≤ (*d* ∗ *L*)^2 **by** (*simp add*: *power-mult-distrib*)
thus *cmod* (*cis* (*d* ∗ *l*) − *1* − i ∗ *complex-of-real* (*d* ∗ *l*)) ≤ (*d* ∗ *L*)^2
using *cis-remainder*[*of d* ∗ *l*] **by** *linarith*
qed *simp*
thus *?thesis* **unfolding** *e* **.**
qed

lemma *evolution-step*:
assumes *v*: *dim-vec v* = *n*
shows ‖*evolution A d* ∗$_v$ *v* − *v*‖ ≤ |*d*| ∗ *L* ∗ ‖*v*‖
proof −
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF v*])
have *mat-fun* (λ*x*. *cis* (*d* ∗ *x*) − *1*) *A* ∗$_v$ *v* = (*evolution A d* − *1*$_m$ *n*) ∗$_v$ *v*
by (*simp only*: *mat-fun-evolution-one*[*OF A*])
also have ... = *evolution A d* ∗$_v$ *v* − *v*
using *minus-mult-distrib-mat-vec*[*OF evolution-carrier*[*OF A*] *one-carrier-mat vc*] *vc* **by** *simp*
finally have *e*: *mat-fun* (λ*x*. *cis* (*d* ∗ *x*) − *1*) *A* ∗$_v$ *v* = *evolution A d* ∗$_v$ *v* − *v* **.**
have *L0*: *0* ≤ *L* **if** *0* < *n*

```
  proof −
    have ‖A *v unit-vec n 0‖ ≤ L * ‖unit-vec n 0‖ by (rule Ab) simp
    thus ?thesis using that vnorm-ge0[of A *v unit-vec n 0] by simp
  qed
  show ?thesis
  proof (cases n = 0)
    case True
    hence evolution A d *v v − v = 0v 0 using v evolution-carrier[OF A, of d]
by (intro eq-vecI) simp-all
    thus ?thesis using vnorm-ge0[of v] True v by (simp add: cpx-vec-length-def)
  next
    case False
    have ‖mat-fun (λx. cis (d * x) − 1) A *v v‖ ≤ (|d| * L) * ‖v‖
    proof (rule mat-fun-bound[OF A - - v])
      show 0 ≤ |d| * L using L0 False by simp
      fix l assume l0: l ∈ herm-spectrum A
      have l: |l| ≤ L by (rule herm-spectrum-bound[OF A(1) - l0]) (rule Ab)
      have |d| * |l| ≤ |d| * L by (rule mult-left-mono[OF l]) simp
     thus cmod (cis (d * l) − 1) ≤ |d| * L using cis-minus-one[of d * l] by (simp
add: abs-mult)
    qed
    thus ?thesis unfolding e by simp
  qed
qed

end
```

5.5 Products of evolutions

evolution-product n d Hs is $e^{idH_1} \cdots e^{idH_m}$ and *hamiltonian-sum n Hs* is $H_1 + \cdots + H_m$.

```
fun evolution-product :: nat ⇒ real ⇒ complex Matrix.mat list ⇒ complex Ma-
trix.mat where
  evolution-product n d [] = 1m n
| evolution-product n d (H # Hs) = evolution H d * evolution-product n d Hs

fun hamiltonian-sum :: nat ⇒ complex Matrix.mat list ⇒ complex Matrix.mat
where
  hamiltonian-sum n [] = 0m n n
| hamiltonian-sum n (H # Hs) = H + hamiltonian-sum n Hs
```

herm-bounded n L H: H is a Hermitian $n \times n$ matrix with $\|Hv\| \leq L\|v\|$.

```
definition herm-bounded :: nat ⇒ real ⇒ complex Matrix.mat ⇒ bool where
  herm-bounded n L H ⟷ H ∈ carrier-mat n n ∧ hermitian H ∧
     (∀ v. dim-vec v = n ⟶ ‖H *v v‖ ≤ L * ‖v‖)

lemma herm-boundedD:
  assumes herm-bounded n L H
```

shows $H \in$ *carrier-mat n n hermitian H* $\bigwedge v$. *dim-vec v* $= n \Longrightarrow \|H *_v v\| \leq L * \|v\|$
using *assms* **unfolding** *herm-bounded-def* **by** *auto*

lemma *herm-bounded-herm*: *herm-bounded n L H* $\Longrightarrow$ *H* $\in$ *carrier-mat n n* $\wedge$ *hermitian H*
by (*simp add*: *herm-bounded-def*)

lemma *evolution-product-carrier*:
($\bigwedge H$. *H* $\in$ *set Hs* $\Longrightarrow$ *H* $\in$ *carrier-mat n n* $\wedge$ *hermitian H*) $\Longrightarrow$ *evolution-product n d Hs* $\in$ *carrier-mat n n*
proof (*induction Hs*)
case (*Cons H Hs*)
have *h*: *H* $\in$ *carrier-mat n n hermitian H* **using** *Cons.prems* **by** *auto*
show *?case* **using** *mult-carrier-mat*[*OF evolution-carrier*[*OF h*] *Cons.IH*] *Cons.prems* **by** *simp*
qed *simp*

lemma *hamiltonian-sum-carrier*: ($\bigwedge H$. *H* $\in$ *set Hs* $\Longrightarrow$ *H* $\in$ *carrier-mat n n*) $\Longrightarrow$ *hamiltonian-sum n Hs* $\in$ *carrier-mat n n*
by (*induction Hs*) *auto*

context
fixes *n* :: *nat* **and** *L* :: *real*
begin

For Hamiltonians bounded by L: the product of their evolutions is unitary, their sum is Hermitian with $\| \sum_j H_j v\| \leq mL\|v\|$, and

$$\|e^{idH_1} \cdots e^{idH_m} v - v\| \leq m\,|d|L\,\|v\|.$$

lemma *evolution-product-unitary*:
($\bigwedge H$. *H* $\in$ *set Hs* $\Longrightarrow$ *herm-bounded n L H*) $\Longrightarrow$ *unitary* (*evolution-product n d Hs*)
proof (*induction Hs*)
case *Nil*
show *?case* **by** (*simp add*: *id-is-unitary*)
next
case (*Cons H Hs*)
note *h* = *herm-boundedD*[*OF Cons.prems*[*of H*, *OF list.set-intros*(*1*)]]
have *unitary* (*evolution H d* $*$ *evolution-product n d Hs*)
by (*rule unitary-mult*[*OF evolution-unitary*[*OF h*(*1*,*2*)] *Cons.IH evolution-carrier*[*OF h*(*1*,*2*)] *evolution-product-carrier*])
(*use Cons.prems* **in** ‹*simp-all add*: *herm-bounded-def*›)
thus *?case* **by** *simp*
qed

lemma *hamiltonian-sum-hermitian*:
($\bigwedge H$. *H* $\in$ *set Hs* $\Longrightarrow$ *herm-bounded n L H*) $\Longrightarrow$ *hermitian* (*hamiltonian-sum n Hs*)

proof (*induction Hs*)
case *Nil*
show *?case* **unfolding** *hermitian-def* **by** (*rule eq-matI*) (*simp-all add*: *adjoint-eval*)
next
case (*Cons H Hs*)
note *h* = *herm-boundedD*[*OF Cons.prems*[*of H*, *OF list.set-intros*(*1*)]]
have *c*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n* **by** (*rule hamiltonian-sum-carrier*) (*use Cons.prems* **in** ‹*simp add*: *herm-bounded-def*›)
have *ih*: *hermitian* (*hamiltonian-sum n Hs*) **by** (*rule Cons.IH*) (*use Cons.prems* **in** *simp*)
have *hermitian* (*H* + *hamiltonian-sum n Hs*) **using** *adjoint-add*[*OF h*(*1*) *c*] *h*(*2*) *ih* **unfolding** *hermitian-def* **by** *simp*
thus *?case* **by** *simp*
qed

lemma *hamiltonian-sum-bound*:
(⋀*H*. *H* ∈ *set Hs* ⟹ *herm-bounded n L H*) ⟹ *dim-vec v* = *n* ⟹
‖*hamiltonian-sum n Hs* ∗$_v$ *v*‖ ≤ *real* (*length Hs*) ∗ *L* ∗ ‖*v*‖
proof (*induction Hs*)
case *Nil*
have *hamiltonian-sum n* [] ∗$_v$ *v* = 0$_v$ *n* **using** *carrier-vecI*[*OF Nil.prems*(*2*)] *Nil.prems*(*2*) **by** (*intro eq-vecI*) *simp-all*
thus *?case* **by** (*simp add*: *cpx-vec-length-def*)
next
case (*Cons H Hs*)
note *h* = *herm-boundedD*[*OF Cons.prems*(*1*)[*of H*, *OF list.set-intros*(*1*)]]
have *S*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n* **by** (*rule hamiltonian-sum-carrier*) (*use Cons.prems* **in** ‹*simp add*: *herm-bounded-def*›)
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF Cons.prems*(*2*)])
have *hamiltonian-sum n* (*H* # *Hs*) ∗$_v$ *v* = *H* ∗$_v$ *v* + *hamiltonian-sum n Hs* ∗$_v$ *v*
using *add-mult-distrib-mat-vec*[*OF h*(*1*) *S vc*] **by** *simp*
hence ‖*hamiltonian-sum n* (*H* # *Hs*) ∗$_v$ *v*‖ ≤ ‖*H* ∗$_v$ *v*‖ + ‖*hamiltonian-sum n Hs* ∗$_v$ *v*‖
using *vnorm-add-le*[*of H* ∗$_v$ *v hamiltonian-sum n Hs* ∗$_v$ *v*] *h*(*1*) *S* **by** *simp*
also have ... ≤ *L* ∗ ‖*v*‖ + *real* (*length Hs*) ∗ *L* ∗ ‖*v*‖
using *h*(*3*)[*OF Cons.prems*(*2*)] *Cons.IH Cons.prems* **by** (*intro add-mono*) *simp-all*
finally show *?case* **by** (*simp add*: *algebra-simps*)
qed

lemma *evolution-product-norm*:
(⋀*H*. *H* ∈ *set Hs* ⟹ *herm-bounded n L H*) ⟹ *dim-vec v* = *n* ⟹ ‖*evolution-product n d Hs* ∗$_v$ *v*‖ = ‖*v*‖
by (*rule unitary-vnorm*[*OF evolution-product-carrier*[*OF herm-bounded-herm*] *evolution-product-unitary*])

lemma *evolution-product-step*:
(⋀*H*. *H* ∈ *set Hs* ⟹ *herm-bounded n L H*) ⟹ *dim-vec v* = *n* ⟹
‖*evolution-product n d Hs* ∗$_v$ *v* − *v*‖ ≤ *real* (*length Hs*) ∗ (|*d*| ∗ *L*) ∗ ‖*v*‖

proof (*induction Hs arbitrary*: *v*)
case *Nil*
have *evolution-product n d* [] $*_v$ *v* − *v* = 0_v *n* **using** *carrier-vecI*[*OF Nil.prems*(*2*)] *Nil.prems*(*2*) **by** (*intro eq-vecI*) *simp-all*
thus *?case* **by** (*simp add*: *cpx-vec-length-def*)
next
case (*Cons H Hs v*)
note *h* = *herm-boundedD*[*OF Cons.prems*(*1*)[*of H*, *OF list.set-intros*(*1*)]]
have *hs*: ⋀*G*. *G* ∈ *set Hs* ⟹ *herm-bounded n L G* **using** *Cons.prems*(*1*) **by** *simp*
have *P*: *evolution-product n d Hs* ∈ *carrier-mat n n* **by** (*rule evolution-product-carrier*[*OF herm-bounded-herm*[*OF hs*]])
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF Cons.prems*(*2*)])
let *?w* = *evolution-product n d Hs* $*_v$ *v*
have *w*: *dim-vec ?w* = *n* **using** *P* **by** *simp*
have *e*: *evolution-product n d* (*H* # *Hs*) $*_v$ *v* = *evolution H d* $*_v$ *?w*
using *assoc-mult-mat-vec*[*OF evolution-carrier*[*OF h*(*1*,*2*)] *P vc*] **by** *simp*
have ‖*evolution H d* $*_v$ *?w* − *v*‖ ≤ ‖*evolution H d* $*_v$ *?w* − *?w*‖ + ‖*?w* − *v*‖
by (*rule vnorm-diff-le*[**where** *n* = *n*]) (*use w Cons.prems*(*2*) *evolution-carrier*[*OF h*(*1*,*2*), *of d*] **in** *simp-all*)
also have ... ≤ |*d*| ∗ *L* ∗ ‖*?w*‖ + *real* (*length Hs*) ∗ (|*d*| ∗ *L*) ∗ ‖*v*‖
by (*rule add-mono*[*OF evolution-step*[*OF h*(*1*,*2*) *h*(*3*) *w*] *Cons.IH*[*OF hs Cons.prems*(*2*)]])
also have ‖*?w*‖ = ‖*v*‖ **by** (*rule evolution-product-norm*[*OF hs Cons.prems*(*2*)])
finally show *?case* **unfolding** *e* **by** (*simp add*: *algebra-simps*)
qed

end

To first order the product of the evolutions is the evolution of the sum:

$$\left\| e^{idH_1} \cdots e^{idH_m} v - v - id \sum_j H_j v \right\| \leq m^2 (dL)^2 \|v\|.$$

lemma *evolution-product-first-order*:
fixes *n* :: *nat* **and** *L* :: *real*
shows (⋀*H*. *H* ∈ *set Hs* ⟹ *herm-bounded n L H*) ⟹ *dim-vec v* = *n* ⟹
‖*evolution-product n d Hs* $*_v$ *v* − *v* − (i ∗ *complex-of-real d*) $\cdot_v$ (*hamiltonian-sum n Hs* $*_v$ *v*)‖
≤ (*d* ∗ *L*)^2 ∗ (*real* (*length Hs*))^2 ∗ ‖*v*‖
proof (*induction Hs arbitrary*: *v*)
case *Nil*
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF Nil.prems*(*2*)])
have *evolution-product n d* [] $*_v$ *v* − *v* − (i ∗ *complex-of-real d*) $\cdot_v$ (*hamiltonian-sum n* [] $*_v$ *v*) = 0_v *n*
using *vc Nil.prems*(*2*) **by** (*intro eq-vecI*) *simp-all*
thus *?case* **by** (*simp add*: *cpx-vec-length-def*)
next
case (*Cons H Hs v*)
note *h* = *herm-boundedD*[*OF Cons.prems*(*1*)[*of H*, *OF list.set-intros*(*1*)]]

have *hs*: $\bigwedge G.\ G \in set\ Hs \Longrightarrow herm\text{-}bounded\ n\ L\ G$ **using** *Cons.prems*(*1*) **by** *simp*
have *v*: *dim-vec v* = *n* **by** (*rule Cons.prems*(*2*))
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF v*])
have *P*: *evolution-product n d Hs* ∈ *carrier-mat n n* **by** (*rule evolution-product-carrier*[*OF herm-bounded-herm*[*OF hs*]])
have *S*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n* **by** (*rule hamiltonian-sum-carrier*[*OF herm-boundedD*(*1*)[*OF hs*]])
have *E*: *evolution H d* ∈ *carrier-mat n n* **by** (*rule evolution-carrier*[*OF h*(*1*,*2*)])
define *c* **where** *c* = i ∗ *complex-of-real d*
define *k* **where** *k* = *real* (*length Hs*)
let *?w* = *evolution-product n d Hs* ∗$_v$ *v*
have *w*: *dim-vec ?w* = *n* **using** *P* **by** *simp*
have *wc*: *?w* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF w*])
have *e1*: *evolution n d* (*H* # *Hs*) ∗$_v$ *v* = *evolution H d* ∗$_v$ *?w*
using *assoc-mult-mat-vec*[*OF E P vc*] **by** *simp*
have *e2*: *hamiltonian-sum n* (*H* # *Hs*) ∗$_v$ *v* = *H* ∗$_v$ *v* + *hamiltonian-sum n Hs* ∗$_v$ *v*
using *add-mult-distrib-mat-vec*[*OF h*(*1*) *S vc*] **by** *simp*
have *e3*: *H* ∗$_v$ (*?w* − *v*) = *H* ∗$_v$ *?w* − *H* ∗$_v$ *v* **by** (*rule mult-minus-distrib-mat-vec*[*OF h*(*1*) *wc vc*])
have *split*: *evolution H d* ∗$_v$ *?w* − *v* − *c* ·$_v$ (*H* ∗$_v$ *v* + *hamiltonian-sum n Hs* ∗$_v$ *v*) =
(*evolution H d* ∗$_v$ *?w* − *?w* − *c* ·$_v$ (*H* ∗$_v$ *?w*)) + ((*?w* − *v* − *c* ·$_v$ (*hamiltonian-sum n Hs* ∗$_v$ *v*)) + *c* ·$_v$ (*H* ∗$_v$ (*?w* − *v*)))
unfolding *e3* **using** *E h*(*1*) *S w v* **by** (*intro eq-vecI*) (*simp-all add*: *algebra-simps*)
have *b1*: ‖*evolution H d* ∗$_v$ *?w* − *?w* − *c* ·$_v$ (*H* ∗$_v$ *?w*)‖ ≤ (*d* ∗ *L*)^2 ∗ ‖*v*‖
using *evolution-first-order*[*OF h*(*1*,*2*) *h*(*3*) *w*, *of d*] *evolution-product-norm*[*OF hs v*] **unfolding** *c-def* **by** *simp*
have *b2*: ‖*?w* − *v* − *c* ·$_v$ (*hamiltonian-sum n Hs* ∗$_v$ *v*)‖ ≤ (*d* ∗ *L*)^2 ∗ *k*^{2} ∗ ‖*v*‖
using *Cons.IH*[*OF hs v*] **unfolding** *c-def k-def* **.**
have *b3*: ‖*c* ·$_v$ (*H* ∗$_v$ (*?w* − *v*))‖ ≤ (*d* ∗ *L*)^2 ∗ *k* ∗ ‖*v*‖
proof −
have ‖*c* ·$_v$ (*H* ∗$_v$ (*?w* − *v*))‖ = |*d*| ∗ ‖*H* ∗$_v$ (*?w* − *v*)‖ **by** (*simp add*: *cpx-vec-length-smult c-def norm-mult*)
also have ... ≤ |*d*| ∗ (*L* ∗ ‖*?w* − *v*‖) **by** (*rule mult-left-mono*[*OF h*(*3*)]) (*use w v* **in** *simp-all*)
also have ... ≤ |*d*| ∗ (*L* ∗ (*k* ∗ (|*d*| ∗ *L*) ∗ ‖*v*‖))
proof (*cases n* = *0*)
case *True*
hence ‖*?w* − *v*‖ = *0* ‖*v*‖ = *0* **using** *w v* **by** (*simp-all add*: *cpx-vec-length-def*)
thus *?thesis* **by** *simp*
next
case *False*
have *L0*: *0* ≤ *L*
proof −
have ‖*H* ∗$_v$ *unit-vec n 0*‖ ≤ *L* ∗ ‖*unit-vec n 0*‖ **by** (*rule h*(*3*)) *simp*
thus *?thesis* **using** *False vnorm-ge0*[*of H* ∗$_v$ *unit-vec n 0*] **by** *simp*

qed
show *?thesis*
using *evolution-product-step*[**where** $d = d$, *OF hs v*] *L0* **unfolding** *k-def*
by (*intro mult-left-mono*) *simp-all*
qed
also have $\ldots = (d * L)^2 * k * \|v\|$ **by** (*simp add*: *power2-eq-square*)
finally show *?thesis* **.**
qed
have *nn*: $0 \leq (d * L)^2 * \|v\|$ **using** *vnorm-ge0*[*of v*] **by** *simp*
have $\|evolution\ H\ d *_v\ ?w - v - c \cdot_v (H *_v v + hamiltonian\text{-}sum\ n\ Hs *_v v)\|$
$\leq \|evolution\ H\ d *_v\ ?w - ?w - c \cdot_v (H *_v\ ?w)\| + (\|?w - v - c \cdot_v$
(*hamiltonian-sum n Hs* $*_v$ *v*)$\| + \|c \cdot_v (H *_v (?w - v))\|)$
proof −
let *?X* = *evolution H d* $*_v$ *?w* − *?w* − *c* $\cdot_v$ (*H* $*_v$ *?w*)
let *?Y* = *?w* − *v* − *c* $\cdot_v$ (*hamiltonian-sum n Hs* $*_v$ *v*) **and** *?Z* = *c* $\cdot_v$ (*H* $*_v$
(*?w* − *v*))
have *dX*: *dim-vec ?X* = *n* **and** *dY*: *dim-vec ?Y* = *n* **and** *dZ*: *dim-vec ?Z* = *n*
using *E h*(*1*) *S w v* **by** *simp-all*
have $\|?X + (?Y + ?Z)\| \leq \|?X\| + \|?Y + ?Z\|$ **by** (*rule vnorm-add-le*) (*use*
dX dY **in** *simp*)
also have $\|?Y + ?Z\| \leq \|?Y\| + \|?Z\|$ **by** (*rule vnorm-add-le*) (*use dY dZ* **in**
simp)
finally show *?thesis* **unfolding** *split* **by** *simp*
qed
also have $\ldots \leq (d * L)^2 * \|v\| + ((d * L)^2 * k^2 * \|v\| + (d * L)^2 * k * \|v\|)$
using *b1 b2 b3* **by** *linarith*
also have $\ldots = (d * L)^2 * \|v\| * (1 + k^2 + k)$ **by** (*simp add*: *algebra-simps*)
also have $\ldots \leq (d * L)^2 * \|v\| * (k + 1)^2$
by (*rule mult-left-mono*[*OF - nn*]) (*simp add*: *power2-eq-square algebra-simps*
k-def)
finally show *?case* **unfolding** *e1 e2 c-def k-def* **by** (*simp add*: *algebra-simps*)
qed

One Trotter step: $\|e^{idH_1} \cdots e^{idH_m} - e^{id(H_1+\cdots+H_m)}\| \leq 2m^2(dL)^2$.

lemma *trotter-step*:
assumes *Hs*: $\bigwedge H.\ H \in set\ Hs \Longrightarrow herm\text{-}bounded\ n\ L\ H$
shows *evolution-product n d Hs* $\approx_{2 * (d * L)^2 * (real\ (length\ Hs))^2}$ *evolution* (*hamiltonian-sum*
n Hs) *d*
proof (*rule op-closeI*[*OF evolution-product-carrier*[*OF herm-bounded-herm*[*OF Hs*]]])
fix *v* :: *complex Matrix.vec* **assume** *v*: *dim-vec v* = *n*
have *P*: *evolution-product n d Hs* ∈ *carrier-mat n n* **by** (*rule evolution-product-carrier*[*OF*
herm-bounded-herm[*OF Hs*]])
have *S*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n hermitian* (*hamiltonian-sum n*
Hs)
using *hamiltonian-sum-carrier*[*OF herm-boundedD*(*1*)[*OF Hs*]] *hamiltonian-sum-hermitian*[*OF*
Hs] **by** *simp-all*
have *E*: *evolution* (*hamiltonian-sum n Hs*) *d* ∈ *carrier-mat n n* **by** (*rule evolu-*
tion-carrier[*OF S*])
let *?c* = i ∗ *complex-of-real d* **and** *?m* = *real* (*length Hs*)

let *?a* = *evolution-product n d Hs* $*_v$ *v* − *v* − *?c* $\cdot_v$ (*hamiltonian-sum n Hs* $*_v$ *v*)
let *?b* = *evolution* (*hamiltonian-sum n Hs*) *d* $*_v$ *v* − *v* − *?c* $\cdot_v$ (*hamiltonian-sum n Hs* $*_v$ *v*)
have *a*: $\|?a\| \leq (d * L)^2 * ?m^2 * \|v\|$ **by** (*rule evolution-product-first-order*[*OF Hs v*])
have *b*: $\|?b\| \leq (d * (?m * L))^2 * \|v\|$
by (*rule evolution-first-order*[*OF S - v*]) (*rule hamiltonian-sum-bound*[*OF Hs*])
have *evolution-product n d Hs* $*_v$ *v* − *evolution* (*hamiltonian-sum n Hs*) *d* $*_v$ *v* = *?a* − *?b*
using *P E S*(*1*) *v* **by** (*intro eq-vecI*) *simp-all*
hence $\|$*evolution-product n d Hs* $*_v$ *v* − *evolution* (*hamiltonian-sum n Hs*) *d* $*_v$ *v*$\| \leq \|?a\| + \|?b\|$
using *vnorm-minus-le*[*of ?a ?b*] *P E S*(*1*) *v* **by** *simp*
also have ... $\leq (d * L)^2 * ?m^2 * \|v\| + (d * (?m * L))^2 * \|v\|$ **using** *a b* **by** *linarith*
also have ... = *2* $* (d * L)^2 * ?m^2 * \|v\|$ **by** (*simp add*: *power-mult-distrib algebra-simps*)
finally show $\|$*evolution-product n d Hs* $*_v$ *v* − *evolution* (*hamiltonian-sum n Hs*) *d* $*_v$ *v*$\| \leq$ *2* $* (d * L)^2 * ?m^2 * \|v\|$ **.**
qed

Errors of unitaries add up along powers.

lemma *op-close-pow*:
assumes *U*: *U* ∈ *carrier-mat n n unitary U* **and** *V*: *V* ∈ *carrier-mat n n unitary V*
and *c*: *U* $\approx_\eta$ *V*
shows *U* $\hat{}_m$ *k* $\approx_{real\ k * \eta}$ *V* $\hat{}_m$ *k*
proof (*induction k*)
case *0*
show *?case*
proof (*rule op-closeI*[*OF pow-carrier-mat*[*OF U*(*1*)]])
fix *v* :: *complex Matrix.vec* **assume** *v*: *dim-vec v* = *n*
have *U* $\hat{}_m$ *0* $*_v$ *v* − *V* $\hat{}_m$ *0* $*_v$ *v* = 0_v *n* **using** *v carrier-matD*[*OF U*(*1*)] *carrier-matD*[*OF V*(*1*)] *carrier-vecI*[*OF v*] **by** (*intro eq-vecI*) *simp-all*
thus $\|$*U* $\hat{}_m$ *0* $*_v$ *v* − *V* $\hat{}_m$ *0* $*_v$ *v*$\| \leq$ *real 0* $*$ *η* $* \|v\|$ **by** (*simp add*: *vnorm-zero-vec*)
qed
next
case (*Suc k*)
show *?case*
proof (*rule op-closeI*[*OF pow-carrier-mat*[*OF U*(*1*)]])
fix *v* :: *complex Matrix.vec* **assume** *v*: *dim-vec v* = *n*
have *vc*: *v* ∈ *carrier-vec n* **by** (*rule carrier-vecI*[*OF v*])
have *Uk*: *U* $\hat{}_m$ *k* ∈ *carrier-mat n n* **and** *Vk*: *V* $\hat{}_m$ *k* ∈ *carrier-mat n n*
using *pow-carrier-mat U*(*1*) *V*(*1*) **by** *blast+*
have *uVk*: *unitary* (*V* $\hat{}_m$ *k*) **by** (*rule unitary-pow*[*OF V*])
let *?x* = *U* $*_v$ *v* **and** *?y* = *V* $*_v$ *v*
have *x*: *dim-vec ?x* = *n* **and** *y*: *dim-vec ?y* = *n* **using** *U*(*1*) *V*(*1*) **by** *simp-all*
have *e1*: *U* $\hat{}_m$ *Suc k* $*_v$ *v* = *U* $\hat{}_m$ *k* $*_v$ *?x* **using** *assoc-mult-mat-vec*[*OF Uk*

$U(1)$ *vc*] **by** *simp*
have *e2*: $V \mathbin{\hat{}}_m Suc\ k *_v v = V \mathbin{\hat{}}_m k *_v ?y$ **using** *assoc-mult-mat-vec*[*OF Vk* $V(1)$ *vc*] **by** *simp*
have $\|U \mathbin{\hat{}}_m k *_v ?x - V \mathbin{\hat{}}_m k *_v ?y\| \leq \|U \mathbin{\hat{}}_m k *_v ?x - V \mathbin{\hat{}}_m k *_v ?x\| + \|V \mathbin{\hat{}}_m k *_v ?x - V \mathbin{\hat{}}_m k *_v ?y\|$
by (*rule vnorm-diff-le*[**where** $n = n$]) (*use Uk Vk carrier-matD*[*OF* $U(1)$] *carrier-matD*[*OF* $V(1)$] **in** *simp-all*)
also have $V \mathbin{\hat{}}_m k *_v ?x - V \mathbin{\hat{}}_m k *_v ?y = V \mathbin{\hat{}}_m k *_v (?x - ?y)$
by (*rule mult-minus-distrib-mat-vec*[*OF Vk*, *symmetric*]) (*use x y* **in** ‹*simp-all add*: *carrier-vecI*›)
also have $\|V \mathbin{\hat{}}_m k *_v (?x - ?y)\| = \|?x - ?y\|$ **by** (*rule unitary-vnorm*[*OF Vk uVk*]) (*use y* **in** *simp*)
also have $\|U \mathbin{\hat{}}_m k *_v ?x - V \mathbin{\hat{}}_m k *_v ?x\| \leq real\ k * \eta * \|?x\|$ **by** (*rule op-closeD*[*OF Suc pow-carrier-mat*[*OF* $U(1)$] *x*])
also have $\|?x\| = \|v\|$ **by** (*rule unitary-vnorm*[*OF U v*])
also have $\|?x - ?y\| \leq \eta * \|v\|$ **by** (*rule op-closeD*[*OF c* $U(1)$ *v*])
finally show $\|U \mathbin{\hat{}}_m Suc\ k *_v v - V \mathbin{\hat{}}_m Suc\ k *_v v\| \leq real\ (Suc\ k) * \eta * \|v\|$
unfolding *e1 e2* **by** (*simp add*: *algebra-simps*)
qed
qed

The Lie–Trotter product formula with explicit error: $\|(e^{itH_1/r} \cdots e^{itH_m/r})^r - e^{it(H_1+\cdots+H_m)}\| \leq 2(mLt)^2/r$.

theorem *trotter*:
assumes *Hs*: $\bigwedge H.\ H \in set\ Hs \Longrightarrow herm\text{-}bounded\ n\ L\ H$ **and** *r*: $0 < r$
shows *evolution-product n* (*t* / *real r*) *Hs* $\mathbin{\hat{}}_m r \approx_{2 * (real\ (length\ Hs) * L * t)^2 / real\ r}$ *evolution* (*hamiltonian-sum n Hs*) *t*
proof −
have *S*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n hermitian* (*hamiltonian-sum n Hs*)
using *hamiltonian-sum-carrier*[*OF herm-boundedD*(1)[*OF Hs*]] *hamiltonian-sum-hermitian*[*OF Hs*] **by** *simp-all*
have *c*: *evolution-product n* (*t* / *real r*) *Hs* $\mathbin{\hat{}}_m r \approx_{real\ r * (2 * (t / real\ r * L)^2 * (real\ (length\ Hs))^2)}$ *evolution* (*hamiltonian-sum n Hs*) (*t* / *real r*) $\mathbin{\hat{}}_m r$
by (*rule op-close-pow*[*OF evolution-product-carrier*[*OF herm-bounded-herm*[*OF Hs*]] *evolution-product-unitary*[*OF Hs*] *evolution-carrier*[*OF S*] *evolution-unitary*[*OF S*] *trotter-step*[*OF Hs*]])
have *p*: *evolution* (*hamiltonian-sum n Hs*) (*t* / *real r*) $\mathbin{\hat{}}_m r$ = *evolution* (*hamiltonian-sum n Hs*) *t*
using *evolution-pow*[*OF S*, *of t* / *real r r*] *r* **by** *simp*
have *a*: *real r* $* (2 * (t / real\ r * L)^2 * (real\ (length\ Hs))^2) = 2 * (real\ (length\ Hs) * L * t)^2 / real\ r$
using *r* **by** (*simp add*: *power2-eq-square field-simps*)
show *?thesis* **using** *c* **unfolding** *p a* **.**
qed

6 Order conditions for product formulas

Taylor coefficients of products $e^{tY_1} \cdots e^{tY_m}$ as convolutions of matrix sequences. A palindromic product has odd error order (*sym-order*), and Suzuki's five-fold recursion raises the order by two (*scaled-copies*), so S_{2k+2} agrees with e^{itH} up to degree $2k+2$ (*suzuki-order*).

A sequence $f_0, f_1, \ldots$ of $n \times n$ matrices stands for the power series $\sum_p f_p t^p$, and the product of two series has the coefficients of the Cauchy product

$$(f * g)_p = \sum_{q \le p} f_q \, g_{p-q}.$$

definition *cauchy-prod* :: *nat* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *nat* ⇒ *complex Matrix.mat* **where**
  cauchy-prod n f g p = *matrix-sum n* (λ*q*. *f q* ∗ *g* (*p* − *q*)) (*Suc p*)

All terms of the sequence are $n \times n$.

definition *seq-carrier* :: *nat* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *bool* **where**
  seq-carrier n f ⟷ (∀ *p*. *f p* ∈ *carrier-mat n n*)

lemma *seq-carrier-in* [*simp*]: *seq-carrier n f* ⟹ *f q* ∈ *carrier-mat n n*
  by (*simp add*: *seq-carrier-def*)

lemma *seq-carrier-row* [*simp*]: *seq-carrier n f* ⟹ *dim-row* (*f q*) = *n*
  by (*rule carrier-matD*(*1*)[*OF seq-carrier-in*])

lemma *seq-carrier-col* [*simp*]: *seq-carrier n f* ⟹ *dim-col* (*f q*) = *n*
  by (*rule carrier-matD*(*2*)[*OF seq-carrier-in*])

lemma *seq-carrier-mult* [*simp*]: *seq-carrier n f* ⟹ *seq-carrier n g* ⟹ *f q* ∗ *g r* ∈ *carrier-mat n n*
  by (*rule mult-carrier-mat*[*OF seq-carrier-in seq-carrier-in*])

lemma *cauchy-prod-carrier*: *seq-carrier n f* ⟹ *seq-carrier n g* ⟹ *seq-carrier n* (*cauchy-prod n f g*)
  unfolding *seq-carrier-def cauchy-prod-def* **by** (*intro allI matrix-sum-dim*, *rule mult-carrier-mat*) *auto*

lemma *cauchy-prod-index*:
  assumes *seq-carrier n f seq-carrier n g i* < *n j* < *n*
  shows *cauchy-prod n f g p* $$ (*i*, *j*) = (∑ *q*≤*p*. ∑ *c*<*n*. *f q* $$ (*i*, *c*) ∗ *g* (*p* − *q*) $$ (*c*, *j*))
proof −
  have *cauchy-prod n f g p* $$ (*i*, *j*) = (∑ *q*<*Suc p*. (*f q* ∗ *g* (*p* − *q*)) $$ (*i*, *j*))
    unfolding *cauchy-prod-def* **by** (*rule matrix-sum-index*) (*use assms* **in** *simp-all*)
  also have ... = (∑ *q*≤*p*. ∑ *c*<*n*. *f q* $$ (*i*, *c*) ∗ *g* (*p* − *q*) $$ (*c*, *j*))
      using *assms* **unfolding** *lessThan-Suc-atMost*
    by (*intro sum.cong refl*) (*auto simp*: *scalar-prod-def atLeast0LessThan*)

finally show *?thesis* **.**
qed

lemma *triple-reindex*:
fixes $F :: nat \Rightarrow nat \Rightarrow nat \Rightarrow 'a::comm\text{-}monoid\text{-}add$
shows $(\sum q{\le}p.\ \sum r{\le}q.\ F\ r\ (q - r)\ (p - q)) = (\sum r{\le}p.\ \sum s{\le}p - r.\ F\ r\ s\ (p - r - s))$
proof −
have $(\sum q{\le}p.\ \sum r{\le}q.\ F\ r\ (q - r)\ (p - q)) = (\sum (q, r){\in}Sigma\ \{..p\}\ (\lambda q.\ \{..q\}).\ F\ r\ (q - r)\ (p - q))$
by (*rule sum.Sigma*) *auto*
also have $\ldots = (\sum (r, s){\in}Sigma\ \{..p\}\ (\lambda r.\ \{..p - r\}).\ F\ r\ s\ (p - r - s))$
by (*rule sum.reindex-bij-witness*[**where** $i = \lambda(r, s).\ (r + s, r)$ **and** $j = \lambda(q, r).\ (r, q - r)$]) *auto*
also have $\ldots = (\sum r{\le}p.\ \sum s{\le}p - r.\ F\ r\ s\ (p - r - s))$
by (*rule sum.Sigma*[*symmetric*]) *auto*
finally show *?thesis* **.**
qed

The Cauchy product is associative:

lemma *cauchy-prod-assoc*:
assumes *f*: *seq-carrier n f* **and** *g*: *seq-carrier n g* **and** *h*: *seq-carrier n h*
shows *cauchy-prod n* (*cauchy-prod n f g*) *h p* = *cauchy-prod n f* (*cauchy-prod n g h*) *p*
proof (*rule eq-matI*)
fix *i j* **assume** *i*: $i <$ *dim-row* (*cauchy-prod n f* (*cauchy-prod n g h*) *p*) **and** *j*: $j <$ *dim-col* (*cauchy-prod n f* (*cauchy-prod n g h*) *p*)
have *ij*: $i < n\ j < n$ **using** *i j cauchy-prod-carrier*[*OF f cauchy-prod-carrier*[*OF g h*]] **by** *simp-all*
have *fg*: *seq-carrier n* (*cauchy-prod n f g*) **and** *gh*: *seq-carrier n* (*cauchy-prod n g h*) **using** *cauchy-prod-carrier f g h* **by** *auto*
have *sw1*: $(\sum c{<}n.\ \sum r{\le}q.\ \sum e{<}n.\ T\ c\ r\ e) = (\sum r{\le}q.\ \sum e{<}n.\ \sum c{<}n.\ T\ c\ r\ e)$ **for** $T :: nat \Rightarrow nat \Rightarrow nat \Rightarrow complex$ **and** *q*
by (*subst sum.swap*) (*rule sum.cong*[*OF refl*], *rule sum.swap*)
have *sw2*: $(\sum e{<}n.\ \sum s{\le}m.\ \sum c{<}n.\ T\ e\ s\ c) = (\sum s{\le}m.\ \sum e{<}n.\ \sum c{<}n.\ T\ e\ s\ c)$ **for** $T :: nat \Rightarrow nat \Rightarrow nat \Rightarrow complex$ **and** *m*
by (*rule sum.swap*)
have *cauchy-prod n* (*cauchy-prod n f g*) *h p* $$ (*i*, *j*)
$= (\sum q{\le}p.\ \sum c{<}n.\ (\sum r{\le}q.\ \sum e{<}n.\ f\ r\ \$\$\ (i,\ e) * g\ (q - r)\ \$\$\ (e,\ c)) * h\ (p - q)\ \$\$\ (c,\ j))$
unfolding *cauchy-prod-index*[*OF fg h ij*] **using** *cauchy-prod-index*[*OF f g ij*(*1*)]
by (*intro sum.cong refl*) *simp*
also have $\ldots = (\sum q{\le}p.\ \sum r{\le}q.\ \sum e{<}n.\ \sum c{<}n.\ f\ r\ \$\$\ (i,\ e) * g\ (q - r)\ \$\$\ (e,\ c) * h\ (p - q)\ \$\$\ (c,\ j))$
by (*simp only*: *sum-distrib-right sw1*)
also have $\ldots = (\sum r{\le}p.\ \sum s{\le}p - r.\ \sum e{<}n.\ \sum c{<}n.\ f\ r\ \$\$\ (i,\ e) * g\ s\ \$\$\ (e,\ c) * h\ (p - r - s)\ \$\$\ (c,\ j))$
by (*rule triple-reindex*)
also have $\ldots = (\sum r{\le}p.\ \sum e{<}n.\ f\ r\ \$\$\ (i,\ e) * (\sum s{\le}p - r.\ \sum c{<}n.\ g\ s\ \$\$\ (e,$

c) ∗ *h* (*p* − *r* − *s*) $$ (*c*, *j*)))
unfolding *sum-distrib-left sw2* **by** (*simp add*: *mult-ac*)
also have ... = *cauchy-prod n f* (*cauchy-prod n g h*) *p* $$ (*i*, *j*)
unfolding *cauchy-prod-index*[*OF f gh ij*] **using** *cauchy-prod-index*[*OF g h - ij*(*2*)] **by** (*intro sum.cong refl*) *simp*
finally show *cauchy-prod n* (*cauchy-prod n f g*) *h p* $$ (*i*, *j*) = *cauchy-prod n f* (*cauchy-prod n g h*) *p* $$ (*i*, *j*) .
qed (*use cauchy-prod-carrier*[*OF cauchy-prod-carrier*[*OF f g*] *h*] *cauchy-prod-carrier*[*OF f cauchy-prod-carrier*[*OF g h*]] **in** ‹*simp-all add*: *seq-carrier-def*›)

The unit series $1, 0, 0, \ldots$; scaling $f_p \mapsto u^p f_p$, which substitutes ut for t; and the coefficients $X^p/p!$ of e^{tX}.

definition *seq-one* :: *nat* ⇒ *nat* ⇒ *complex Matrix.mat* **where**
seq-one n p = (*if p* = *0 then 1*$_m$ *n else 0*$_m$ *n n*)

definition *seq-scale* :: *complex* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *nat* ⇒ *complex Matrix.mat* **where**
seq-scale u f p = (*u* ^ *p*) ·$_m$ *f p*

definition *exp-coeff* :: *complex Matrix.mat* ⇒ *nat* ⇒ *complex Matrix.mat* **where**
exp-coeff X p = *complex-of-real* (*1* / *fact p*) ·$_m$ *X* ^$_m$ *p*

lemma *seq-one-carrier* [*simp*]: *seq-carrier n* (*seq-one n*)
by (*simp add*: *seq-carrier-def seq-one-def*)

lemma *seq-scale-carrier* [*simp*]: *seq-carrier n f* ⟹ *seq-carrier n* (*seq-scale u f*)
by (*simp add*: *seq-carrier-def seq-scale-def*)

lemma *exp-coeff-carrier* [*simp*]: *X* ∈ *carrier-mat n n* ⟹ *seq-carrier n* (*exp-coeff X*)
by (*simp add*: *seq-carrier-def exp-coeff-def*)

The unit is neutral and scaling commutes with products:

lemma *cauchy-prod-unit-left*: *seq-carrier n f* ⟹ *cauchy-prod n* (*seq-one n*) *f p* = *f p*
proof (*induction p*)
case *0*
thus *?case* **by** (*simp add*: *cauchy-prod-def seq-one-def*)
next
case (*Suc p*)
have *matrix-sum n* (λ*q*. *seq-one n q* ∗ *f* (*Suc p* − *q*)) (*Suc p*) = *matrix-sum n* (λ*q*. *if q* = *0 then f* (*Suc p*) *else 0*$_m$ *n n*) (*Suc p*)
by (*rule matrix-sum-cong*) (*use Suc.prems* **in** ‹*auto simp*: *seq-one-def*›)
also have ... = *f* (*Suc p*)
proof −
have *z*: *matrix-sum n* (λ*q*. *if q* = *0 then f* (*Suc p*) *else 0*$_m$ *n n*) *m* = *f* (*Suc p*) **if** *0* < *m* **for** *m*
using *that* **by** (*induction m*) (*use Suc.prems* **in** ‹*auto simp*: *gr0-conv-Suc*›)
show *?thesis* **by** (*rule z*) *simp*

qed
finally show *?case* **using** *Suc.prems* **by** (*simp add*: *cauchy-prod-def seq-one-def*)
qed

lemma *cauchy-prod-unit-right*: *seq-carrier n f* ⟹ *cauchy-prod n f* (*seq-one n*) *p* = *f p*
proof (*rule eq-matI*)
fix *i j* **assume** *f*: *seq-carrier n f* **and** *i*: *i* < *dim-row* (*f p*) **and** *j*: *j* < *dim-col* (*f p*)
have *ij*: *i* < *n j* < *n* **using** *i j f* **by** *simp-all*
have *cauchy-prod n f* (*seq-one n*) *p* $$ (*i*, *j*) = (∑ *q*≤*p*. ∑ *c*<*n*. *f q* $$ (*i*, *c*) ∗ *seq-one n* (*p* − *q*) $$ (*c*, *j*))
by (*rule cauchy-prod-index*[*OF f seq-one-carrier ij*])
also have ... = (∑ *q*≤*p*. *if q* = *p then f p* $$ (*i*, *j*) *else 0*)
proof (*rule sum.cong*[*OF refl*])
fix *q* **assume** *q*: *q* ∈ {..*p*}
show (∑ *c*<*n*. *f q* $$ (*i*, *c*) ∗ *seq-one n* (*p* − *q*) $$ (*c*, *j*)) = (*if q* = *p then f p* $$ (*i*, *j*) *else 0*)
proof (*cases q* = *p*)
case *True*
have (∑ *c*<*n*. *f p* $$ (*i*, *c*) ∗ (*if c* = *j then 1 else 0*)) = (∑ *c*<*n*. *if c* = *j then f p* $$ (*i*, *c*) *else 0*)
by (*intro sum.cong*) *auto*
also have ... = *f p* $$ (*i*, *j*) **using** *ij* **by** *simp*
finally have (∑ *c*<*n*. *f p* $$ (*i*, *c*) ∗ (*if c* = *j then 1 else 0*)) = *f p* $$ (*i*, *j*) .
thus *?thesis* **using** *True ij* **by** (*simp add*: *seq-one-def one-mat-def*)
qed (*use q ij* **in** ‹*auto simp*: *seq-one-def*›)
qed
also have ... = *f p* $$ (*i*, *j*) **by** *simp*
finally show *cauchy-prod n f* (*seq-one n*) *p* $$ (*i*, *j*) = *f p* $$ (*i*, *j*) .
qed (*simp-all add*: *cauchy-prod-carrier*[*THEN seq-carrier-row*] *cauchy-prod-carrier*[*THEN seq-carrier-col*])

lemma *seq-scale-cauchy-prod*: *seq-carrier n f* ⟹ *seq-carrier n g* ⟹ *cauchy-prod n* (*seq-scale u f*) (*seq-scale u g*) *p* = *seq-scale u* (*cauchy-prod n f g*) *p*
proof (*rule eq-matI*)
fix *i j* **assume** *f*: *seq-carrier n f* **and** *g*: *seq-carrier n g* **and** *i*: *i* < *dim-row* (*seq-scale u* (*cauchy-prod n f g*) *p*)
and *j*: *j* < *dim-col* (*seq-scale u* (*cauchy-prod n f g*) *p*)
have *ij*: *i* < *n j* < *n* **using** *i j cauchy-prod-carrier*[*OF f g*] **by** (*simp-all add*: *seq-scale-def*)
have *cauchy-prod n* (*seq-scale u f*) (*seq-scale u g*) *p* $$ (*i*, *j*) = (∑ *q*≤*p*. ∑ *c*<*n*. (*u* ^ *q* ∗ *f q* $$ (*i*, *c*)) ∗ (*u* ^ (*p* − *q*) ∗ *g* (*p* − *q*) $$ (*c*, *j*)))
unfolding *cauchy-prod-index*[*OF seq-scale-carrier*[*OF f*] *seq-scale-carrier*[*OF g*] *ij*] **using** *ij f g* **by** (*intro sum.cong refl*) (*simp add*: *seq-scale-def*)
also have ... = (∑ *q*≤*p*. ∑ *c*<*n*. *u* ^ *p* ∗ (*f q* $$ (*i*, *c*) ∗ *g* (*p* − *q*) $$ (*c*, *j*)))
by (*intro sum.cong refl*) (*simp add*: *mult-ac power-add*[*symmetric*])
also have ... = *seq-scale u* (*cauchy-prod n f g*) *p* $$ (*i*, *j*)
unfolding *seq-scale-def* **using** *ij cauchy-prod-index*[*OF f g ij*] *cauchy-prod-carrier*[*OF*

f g] **by** (*simp add*: *sum-distrib-left*)
finally show *cauchy-prod n* (*seq-scale u f*) (*seq-scale u g*) *p* $$ (*i*, *j*) = *seq-scale u* (*cauchy-prod n f g*) *p* $$ (*i*, *j*) **.**
qed (*simp-all add*: *seq-scale-def cauchy-prod-carrier*[*THEN seq-carrier-row*] *cauchy-prod-carrier*[*THEN seq-carrier-col*])

lemma *smult-pow-mat*: *X* ∈ *carrier-mat n n* ⟹ (*u* ·$_m$ *X*) ^$_m$ *p* = (*u* ^ *p*) ·$_m$ (*X* ^$_m$ *p*)
for *X* :: *complex Matrix.mat*
proof (*induction p*)
case *0*
thus *?case* **by** *simp* (*intro eq-matI*, *auto*)
next
case (*Suc p*)
have *Xp*: *X* ^$_m$ *p* ∈ *carrier-mat n n* **using** *Suc.prems* **by** *simp*
have (*u* ·$_m$ *X*) ^$_m$ *Suc p* = (*u* ^ *p* ·$_m$ *X* ^$_m$ *p*) ∗ (*u* ·$_m$ *X*) **using** *Suc* **by** *simp*
also have ... = *u* ^ *p* ·$_m$ (*X* ^$_m$ *p* ∗ (*u* ·$_m$ *X*))
by (*rule mult-smult-assoc-mat*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*]) (*use Xp Suc.prems* **in** *simp-all*)
also have *X* ^$_m$ *p* ∗ (*u* ·$_m$ *X*) = *u* ·$_m$ (*X* ^$_m$ *p* ∗ *X*)
by (*rule mult-smult-distrib*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*]) (*use Xp Suc.prems* **in** *simp-all*)
also have *u* ^ *p* ·$_m$ (*u* ·$_m$ (*X* ^$_m$ *p* ∗ *X*)) = (*u* ^ *Suc p*) ·$_m$ (*X* ^$_m$ *p* ∗ *X*)
by (*rule eq-matI*) (*auto simp*: *mult-ac*)
finally show *?case* **by** *simp*
qed

lemma *smult-mult-smult*:
fixes *A B* :: *complex Matrix.mat*
assumes *A* ∈ *carrier-mat n n B* ∈ *carrier-mat n n*
shows (*c* ·$_m$ *A*) ∗ (*d* ·$_m$ *B*) = (*c* ∗ *d*) ·$_m$ (*A* ∗ *B*)
proof −
have (*c* ·$_m$ *A*) ∗ (*d* ·$_m$ *B*) = *c* ·$_m$ (*A* ∗ (*d* ·$_m$ *B*))
by (*rule mult-smult-assoc-mat*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*]) (*use assms* **in** *simp-all*)
also have *A* ∗ (*d* ·$_m$ *B*) = *d* ·$_m$ (*A* ∗ *B*)
by (*rule mult-smult-distrib*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*]) (*use assms* **in** *simp-all*)
also have *c* ·$_m$ (*d* ·$_m$ (*A* ∗ *B*)) = (*c* ∗ *d*) ·$_m$ (*A* ∗ *B*) **by** (*rule eq-matI*) *auto*
finally show *?thesis* **.**
qed

lemma *exp-coeff-smult*: *X* ∈ *carrier-mat n n* ⟹ *exp-coeff* (*u* ·$_m$ *X*) *p* = *seq-scale u* (*exp-coeff X*) *p*
by (*simp add*: *exp-coeff-def seq-scale-def smult-pow-mat*) (*rule eq-matI*, *auto simp*: *mult-ac*)

lemma *exp-coeff-zero*: *exp-coeff* (*0*$_m$ *n n*) *p* = *seq-one n p*
by (*cases p*) (*simp-all add*: *exp-coeff-def seq-one-def one-smult-mat*)

$e^{taX}e^{tbX} = e^{t(a+b)X}$, coefficient by coefficient:

$$\sum_{q\le p} \frac{(aX)^q}{q!}\,\frac{(bX)^{p-q}}{(p-q)!} = \frac{((a+b)X)^p}{p!}.$$

lemma *exp-coeff-add*:
assumes X: $X \in$ *carrier-mat n n*
shows *cauchy-prod n* (*exp-coeff* (*a* $\cdot_m$ *X*)) (*exp-coeff* (*b* $\cdot_m$ *X*)) *p* = *exp-coeff* ((*a* + *b*) $\cdot_m$ *X*) *p*
proof −
have *Xq*: $\bigwedge q$. *X* $\hat{}_m$ *q* ∈ *carrier-mat n n* **using** *X* **by** *simp*
have *tm*: *exp-coeff* (*a* $\cdot_m$ *X*) *q* ∗ *exp-coeff* (*b* $\cdot_m$ *X*) (*p* − *q*)
= (*a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*))) $\cdot_m$ *X* $\hat{}_m$ *p* **if** *q* ≤ *p* **for** *q*
proof −
have *X* $\hat{}_m$ *q* ∗ *X* $\hat{}_m$ (*p* − *q*) = *X* $\hat{}_m$ *p* **using** *that X* **by** (*simp add*: *pow-mat-add*[*OF X*, *symmetric*])
hence (*1* / *fact q* ∗ *a* ^ *q*) $\cdot_m$ *X* $\hat{}_m$ *q* ∗ ((*1* / *fact* (*p* − *q*) ∗ *b* ^ (*p* − *q*)) $\cdot_m$ *X* $\hat{}_m$ (*p* − *q*))
= (*a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*))) $\cdot_m$ *X* $\hat{}_m$ *p*
by (*simp add*: *smult-mult-smult*[*OF Xq Xq*] *field-simps*)
moreover have *exp-coeff* (*a* $\cdot_m$ *X*) *q* = (*1* / *fact q* ∗ *a* ^ *q*) $\cdot_m$ *X* $\hat{}_m$ *q*
exp-coeff (*b* $\cdot_m$ *X*) (*p* − *q*) = (*1* / *fact* (*p* − *q*) ∗ *b* ^ (*p* − *q*)) $\cdot_m$ *X* $\hat{}_m$ (*p* − *q*)
by (*simp-all add*: *exp-coeff-def smult-pow-mat*[*OF X*]) (*rule eq-matI*, *auto*)+
ultimately show *?thesis* **by** *simp*
qed
have *cauchy-prod n* (*exp-coeff* (*a* $\cdot_m$ *X*)) (*exp-coeff* (*b* $\cdot_m$ *X*)) *p*
= *matrix-sum n* (λ*q*. (*a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*))) $\cdot_m$ *X* $\hat{}_m$ *p*) (*Suc p*)
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*simp add*: *tm*)
also have ... = (∑ *q*<*Suc p*. *a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*))) $\cdot_m$ *X* $\hat{}_m$ *p*
by (*rule matrix-sum-scalar*[*OF Xq*])
also have (∑ *q*<*Suc p*. *a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*))) = (*a* + *b*) ^ *p* / *fact p*
proof −
have (*a* + *b*) ^ *p* = (∑ *q*≤*p*. *of-nat* (*p choose q*) ∗ *a* ^ *q* ∗ *b* ^ (*p* − *q*)) **by** (*rule binomial-ring*)
also have ... = *fact p* ∗ (∑ *q*≤*p*. *a* ^ *q* ∗ *b* ^ (*p* − *q*) / (*fact q* ∗ *fact* (*p* − *q*)))
unfolding *sum-distrib-left* **by** (*intro sum.cong refl*) (*simp add*: *binomial-fact field-simps*)
finally show *?thesis* **by** (*simp add*: *lessThan-Suc-atMost field-simps*)
qed
also have ((*a* + *b*) ^ *p* / *fact p*) $\cdot_m$ *X* $\hat{}_m$ *p* = *exp-coeff* ((*a* + *b*) $\cdot_m$ *X*) *p*
by (*simp add*: *exp-coeff-def smult-pow-mat*[*OF X*]) (*rule eq-matI*, *auto*)
finally show *?thesis* **.**
qed

The coefficients of $e^{tY_1}\cdots e^{tY_m}$.

fun *prod-coeff* :: *nat* ⇒ *complex Matrix.mat list* ⇒ *nat* ⇒ *complex Matrix.mat*
where
prod-coeff n [] = *seq-one n*
| *prod-coeff n* (*Y* # *Ys*) = *cauchy-prod n* (*exp-coeff Y*) (*prod-coeff n Ys*)

lemma *prod-coeff-carrier*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ⟹ *seq-carrier n* (*prod-coeff n Ys*)
by (*induction Ys*, *auto intro*: *cauchy-prod-carrier*)

Concatenating factor lists multiplies the series, scaling all factors scales the series, and $e^{-tY_m} \cdots e^{-tY_1}$ inverts $e^{tY_1} \cdots e^{tY_m}$:

lemma *prod-coeff-append*:
assumes ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ∀ *Y* ∈ *set Zs*. *Y* ∈ *carrier-mat n n*
shows *prod-coeff n* (*Ys* @ *Zs*) = *cauchy-prod n* (*prod-coeff n Ys*) (*prod-coeff n Zs*)
using *assms*(*1*)
proof (*induction Ys*)
case *Nil*
show *?case* **using** *prod-coeff-carrier*[*OF assms*(*2*)] **by** (*simp add*: *fun-eq-iff cauchy-prod-unit-left*)
next
case (*Cons Y Ys*)
have *c*: *seq-carrier n* (*exp-coeff Y*) *seq-carrier n* (*prod-coeff n Ys*) *seq-carrier n* (*prod-coeff n Zs*)
using *Cons.prems prod-coeff-carrier*[*OF assms*(*2*)] *prod-coeff-carrier*[*of Ys n*]
by *auto*
have *ih*: *prod-coeff n* (*Ys* @ *Zs*) = *cauchy-prod n* (*prod-coeff n Ys*) (*prod-coeff n Zs*) **using** *Cons* **by** *simp*
show *?case* **unfolding** *append-Cons prod-coeff.simps ih* **using** *cauchy-prod-assoc*[*OF c*] **by** (*simp add*: *fun-eq-iff*)
qed

lemma *seq-scale-seq-one*: *seq-scale u* (*seq-one n*) = *seq-one n*
by (*auto simp*: *fun-eq-iff seq-scale-def seq-one-def one-smult-mat*)

lemma *prod-coeff-scale*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ⟹ *prod-coeff n* (*map* (($\cdot_m$) *u*) *Ys*) = *seq-scale u* (*prod-coeff n Ys*)
proof (*induction Ys*)
case *Nil*
thus *?case* **by** (*simp add*: *seq-scale-seq-one*)
next
case (*Cons Y Ys*)
have *Y*: *Y* ∈ *carrier-mat n n* **and** *F*: *seq-carrier n* (*prod-coeff n Ys*) **using** *Cons.prems prod-coeff-carrier*[*of Ys n*] **by** *auto*
have *t*: *exp-coeff* (*u* $\cdot_m$ *Y*) = *seq-scale u* (*exp-coeff Y*) **by** (*simp add*: *fun-eq-iff exp-coeff-smult*[*OF Y*])
have *ih*: *prod-coeff n* (*map* (($\cdot_m$) *u*) *Ys*) = *seq-scale u* (*prod-coeff n Ys*) **using** *Cons* **by** *simp*
show *?case* **unfolding** *list.map prod-coeff.simps t ih* **using** *seq-scale-cauchy-prod*[*OF exp-coeff-carrier*[*OF Y*] *F*] **by** (*simp add*: *fun-eq-iff*)

qed

lemma *zero-smult-mat*: $A \in$ *carrier-mat n n* $\Longrightarrow$ $(0::complex) \cdot_m A = 0_m\ n\ n$
by (*rule eq-matI*) *auto*

lemma *exp-coeff-inverse*:
assumes *Y*: $Y \in$ *carrier-mat n n*
shows *cauchy-prod n* (*exp-coeff* $((-1) \cdot_m Y)$) (*exp-coeff Y*) = *seq-one n*
proof −
have *cauchy-prod n* (*exp-coeff* $((-1) \cdot_m Y)$) (*exp-coeff* $(1 \cdot_m Y)$) *p* = *exp-coeff* $((-1 + 1) \cdot_m Y)$ *p* **for** *p*
by (*rule exp-coeff-add*[*OF Y*])
thus *?thesis* **using** *zero-smult-mat*[*OF Y*] **by** (*simp add*: *fun-eq-iff one-smult-mat exp-coeff-zero*)
qed

lemma *cauchy-prod-assoc-fun*: *seq-carrier n f* $\Longrightarrow$ *seq-carrier n g* $\Longrightarrow$ *seq-carrier n h* $\Longrightarrow$ *cauchy-prod n* (*cauchy-prod n f g*) *h* = *cauchy-prod n f* (*cauchy-prod n g h*)
by (*simp add*: *fun-eq-iff cauchy-prod-assoc*)

lemma *cauchy-prod-unit-right-fun*: *seq-carrier n f* $\Longrightarrow$ *cauchy-prod n f* (*seq-one n*) = *f*
by (*simp add*: *fun-eq-iff cauchy-prod-unit-right*)

lemma *cauchy-prod-unit-left-fun*: *seq-carrier n f* $\Longrightarrow$ *cauchy-prod n* (*seq-one n*) *f* = *f*
by (*simp add*: *fun-eq-iff cauchy-prod-unit-left*)

lemma *prod-coeff-inverse*:
$\forall\, Y \in$ *set Ys*. *Y* $\in$ *carrier-mat n n* $\Longrightarrow$ *cauchy-prod n* (*prod-coeff n* (*rev* (*map* $((\cdot_m)\ (-1))$ *Ys*))) (*prod-coeff n Ys*) = *seq-one n*
proof (*induction Ys*)
case *Nil*
thus *?case* **by** (*simp add*: *fun-eq-iff cauchy-prod-unit-left*)
next
case (*Cons Y Ys*)
have *Y*: *Y* $\in$ *carrier-mat n n* **and** *Ys*: $\forall\, Z \in$ *set Ys*. *Z* $\in$ *carrier-mat n n* **using** *Cons.prems* **by** *auto*
let *?A* = *rev* (*map* $((\cdot_m)\ (-1))$ *Ys*)
have *A*: $\forall\, Z \in$ *set ?A*. *Z* $\in$ *carrier-mat n n* **using** *Ys* **by** *auto*
have *nY*: $(-1) \cdot_m Y \in$ *carrier-mat n n* **using** *Y* **by** *simp*
have *c*: *seq-carrier n* (*prod-coeff n ?A*) *seq-carrier n* (*exp-coeff* $((-1) \cdot_m Y)$) *seq-carrier n* (*exp-coeff Y*) *seq-carrier n* (*prod-coeff n Ys*)
using *prod-coeff-carrier*[*OF A*] *prod-coeff-carrier*[*OF Ys*] *Y nY* **by** *auto*
have *prod-coeff n* (*rev* (*map* $((\cdot_m)\ (-1))$ (*Y* # *Ys*))) = *cauchy-prod n* (*prod-coeff n ?A*) (*exp-coeff* $((-1) \cdot_m Y)$)
using *prod-coeff-append*[*OF A*, *of* $[(-1) \cdot_m Y]$] *nY* **by** (*simp add*: *cauchy-prod-unit-right-fun*)
hence *cauchy-prod n* (*prod-coeff n* (*rev* (*map* $((\cdot_m)\ (-1))$ (*Y* # *Ys*)))) (*prod-coeff*

n (*Y* # *Ys*))
= *cauchy-prod n* (*cauchy-prod n* (*prod-coeff n ?A*) (*exp-coeff* ((−*1*) ·$_m$ *Y*))) (*cauchy-prod n* (*exp-coeff Y*) (*prod-coeff n Ys*)) **by** *simp*
also have ... = *cauchy-prod n* (*prod-coeff n ?A*) (*cauchy-prod n* (*cauchy-prod n* (*exp-coeff* ((−*1*) ·$_m$ *Y*)) (*exp-coeff Y*)) (*prod-coeff n Ys*))
by (*simp only*: *cauchy-prod-assoc-fun*[*OF c*(*1*,*2*) *cauchy-prod-carrier*[*OF c*(*3*,*4*)]] *cauchy-prod-assoc-fun*[*OF c*(*2*,*3*,*4*), *symmetric*])
also have ... = *cauchy-prod n* (*prod-coeff n ?A*) (*prod-coeff n Ys*)
by (*simp only*: *exp-coeff-inverse*[*OF Y*] *cauchy-prod-unit-left-fun*[*OF c*(*4*)])
finally show *?case* **using** *Cons.IH*[*OF Ys*] **by** *simp*
qed

f vanishes below degree q: $f_p = 0$ for $p < q$.

definition *vanishes-below* :: *nat* ⇒ *nat* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *bool*
where
vanishes-below n q f ⟷ (∀ *p*<*q*. *f p* = *0*$_m$ *n n*)

lemma *cauchy-prod-add-left*:
seq-carrier n f ⟹ *seq-carrier n g* ⟹ *seq-carrier n h* ⟹ *cauchy-prod n* (λ*p*. *f p* + *g p*) *h p* = *cauchy-prod n f h p* + *cauchy-prod n g h p*
proof −
assume *a*: *seq-carrier n f seq-carrier n g seq-carrier n h*
have *cauchy-prod n* (λ*p*. *f p* + *g p*) *h p* = *matrix-sum n* (λ*q*. *f q* ∗ *h* (*p* − *q*) + *g q* ∗ *h* (*p* − *q*)) (*Suc p*)
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*rule add-mult-distrib-mat*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*], *use a* **in** *simp-all*)
also have ... = *cauchy-prod n f h p* + *cauchy-prod n g h p*
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-add-distrib*) (*use a* **in** *simp-all*)
finally show *?thesis* **.**
qed

lemma *cauchy-prod-add-right*:
seq-carrier n f ⟹ *seq-carrier n g* ⟹ *seq-carrier n h* ⟹ *cauchy-prod n h* (λ*p*. *f p* + *g p*) *p* = *cauchy-prod n h f p* + *cauchy-prod n h g p*
proof −
assume *a*: *seq-carrier n f seq-carrier n g seq-carrier n h*
have *cauchy-prod n h* (λ*p*. *f p* + *g p*) *p* = *matrix-sum n* (λ*q*. *h q* ∗ *f* (*p* − *q*) + *h q* ∗ *g* (*p* − *q*)) (*Suc p*)
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*rule mult-add-distrib-mat*[**where** *nr* = *n* **and** *n* = *n* **and** *nc* = *n*], *use a* **in** *simp-all*)
also have ... = *cauchy-prod n h f p* + *cauchy-prod n h g p*
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-add-distrib*) (*use a* **in** *simp-all*)
finally show *?thesis* **.**
qed

lemma *matrix-sum-zero*: *matrix-sum n* (λ*r*. *0*$_m$ *n n*) *m* = *0*$_m$ *n n*
by (*induction m*) *simp-all*

lemma *matrix-sum-single*:

assumes $\bigwedge q.\ q < m \Longrightarrow F\ q \in$ *carrier-mat n n* **and** $\bigwedge q.\ q < m \Longrightarrow q \neq k \Longrightarrow F\ q = 0_m\ n\ n$ **and** $k < m$
shows *matrix-sum n F m* $= F\ k$
using *assms*
proof (*induction m*)
case (*Suc m*)
show *?case*
proof (*cases k* $= m$)
case *True*
have *matrix-sum n F m* $= 0_m\ n\ n$
proof −
have *matrix-sum n F m* $=$ *matrix-sum n* $(\lambda q.\ 0_m\ n\ n)\ m$ **by** (*rule matrix-sum-cong*) (*use Suc.prems True* **in** *auto*)
also have $\ldots = 0_m\ n\ n$ **by** (*induction m*) *simp-all*
finally show *?thesis* **.**
qed
thus *?thesis* **using** *True Suc.prems*(*1*)[*of m*] **by** *simp*
next
case *False*
thus *?thesis* **using** *Suc.IH*[*OF Suc.prems*(*1*,*2*)] *Suc.prems* **by** (*simp add*: *less-Suc-eq matrix-sum-dim*)
qed
qed *simp*

If d vanishes below q, then $(d * g)_q = d_q g_0$, $(g * d)_q = g_0 d_q$, both products vanish below q, and the product of two such sequences vanishes at $q > 0$ too:

lemma *cauchy-prod-low-left*:
assumes *d*: *seq-carrier n d vanishes-below n q d* **and** *g*: *seq-carrier n g*
shows *cauchy-prod n d g q* $=$ *d q* $*$ *g 0*
proof −
have *matrix-sum n* $(\lambda r.\ d\ r * g\ (q - r))$ $(Suc\ q) = d\ q * g\ (q - q)$
by (*rule matrix-sum-single*) (*use d g* **in** ‹*auto simp*: *vanishes-below-def*›)
thus *?thesis* **by** (*simp add*: *cauchy-prod-def*)
qed

lemma *cauchy-prod-low-right*:
assumes *d*: *seq-carrier n d vanishes-below n q d* **and** *g*: *seq-carrier n g*
shows *cauchy-prod n g d q* $=$ *g 0* $*$ *d q*
proof −
have *matrix-sum n* $(\lambda r.\ g\ r * d\ (q - r))$ $(Suc\ q) = g\ 0 * d\ (q - 0)$
by (*rule matrix-sum-single*, *use d g* **in** ‹*auto simp*: *vanishes-below-def*›)
thus *?thesis* **by** (*simp add*: *cauchy-prod-def*)
qed

lemma *cauchy-prod-low-below*:
assumes *d*: *seq-carrier n d vanishes-below n q d* **and** *g*: *seq-carrier n g* **and** *p*: $p < q$
shows *cauchy-prod n d g p* $= 0_m\ n\ n$ *cauchy-prod n g d p* $= 0_m\ n\ n$

proof −
have *cauchy-prod n d g p* = *matrix-sum n* (λ*r*. 0_m *n n*) (*Suc p*)
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*use d g p* **in** ‹*auto simp*: *vanishes-below-def*›)
thus *cauchy-prod n d g p* = 0_m *n n* **by** (*simp only*: *matrix-sum-zero*)
have *cauchy-prod n g d p* = *matrix-sum n* (λ*r*. 0_m *n n*) (*Suc p*)
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*use d g p* **in** ‹*auto simp*: *vanishes-below-def*›)
thus *cauchy-prod n g d p* = 0_m *n n* **by** (*simp only*: *matrix-sum-zero*)
qed

lemma *cauchy-prod-low-both*:
assumes *seq-carrier n d vanishes-below n q d seq-carrier n e vanishes-below n q e 0* < *q*
shows *cauchy-prod n d e q* = 0_m *n n*
proof −
have *cauchy-prod n d e q* = *matrix-sum n* (λ*r*. 0_m *n n*) (*Suc q*)
unfolding *cauchy-prod-def*
proof (*rule matrix-sum-cong*)
fix *r* **assume** *r* < *Suc q*
hence *r* < *q* ∨ *q* − *r* < *q* **using** *assms*(*5*) **by** *linarith*
thus *d r* ∗ *e* (*q* − *r*) = 0_m *n n* **using** *assms* **by** (*auto simp*: *vanishes-below-def*)
qed
thus *?thesis* **by** (*simp only*: *matrix-sum-zero*)
qed

A palindromic formula has odd error order: agreement with e^{tX} below an even degree q extends to degree q.

lemma *sym-order*:
assumes *Ys*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* **and** *X*: *X* ∈ *carrier-mat n n*
and *pal*: *rev Ys* = *Ys* **and** *ord*: ∀ *p*<*q*. *prod-coeff n Ys p* = *exp-coeff X p* **and** *q*: *even q 0* < *q*
shows *prod-coeff n Ys q* = *exp-coeff X q*
proof −
let *?F* = *prod-coeff n Ys* **and** *?E* = *exp-coeff X*
define *D* **where** *D* = (λ*p*. *?F p* − *?E p*)
have *F*: *seq-carrier n ?F* **and** *E*: *seq-carrier n ?E* **using** *prod-coeff-carrier*[*OF Ys*] *X* **by** *auto*
have *Dc*: *seq-carrier n D* **unfolding** *seq-carrier-def D-def* **using** *seq-carrier-in*[*OF E*] **by** (*simp add*: *minus-carrier-mat*)
have *Dl*: *vanishes-below n q D* **using** *ord seq-carrier-in*[*OF E*] **by** (*auto simp*: *vanishes-below-def D-def minus-r-inv-mat*)
have *FE*: *?F* = (λ*p*. *?E p* + *D p*)
using *F E* **by** (*auto simp*: *fun-eq-iff D-def intro*!: *eq-matI*)
let *?mE* = *seq-scale* (−*1*) *?E* **and** *?mD* = *seq-scale* (−*1*) *D*
have *mEc*: *seq-carrier n ?mE* **and** *mDc*: *seq-carrier n ?mD* **and** *mDl*: *vanishes-below n q ?mD*
using *E Dc Dl* **by** (*auto simp*: *vanishes-below-def seq-scale-def*)
have *mF*: *seq-scale* (−*1*) *?F* = (λ*p*. *?mE p* + *?mD p*)

using *E Dc* **unfolding** *FE* **by** (*auto simp*: *fun-eq-iff seq-scale-def add-smult-distrib-left-mat*[**where** *nr* = *n* **and** *nc* = *n*])
have *rv*: *rev* (*map* ((·$_m$) (−*1*)) *Ys*) = *map* ((·$_m$) (−*1*)) *Ys* **using** *pal* **by** (*metis rev-map*)
have *inv*: *cauchy-prod n* (*seq-scale* (−*1*) *?F*) *?F* = *seq-one n*
using *prod-coeff-inverse*[*OF Ys*] **unfolding** *rv prod-coeff-scale*[*OF Ys*] **.**
have *invE*: *cauchy-prod n ?mE ?E q* = *0*$_m$ *n n*
proof −
have *?mE* = *exp-coeff* ((−*1*) ·$_m$ *X*) **by** (*simp add*: *fun-eq-iff exp-coeff-smult*[*OF X*])
thus *?thesis* **using** *exp-coeff-inverse*[*OF X*] *q*(*2*) **by** (*simp add*: *seq-one-def*)
qed
have *E0*: *?E 0* = *1*$_m$ *n* **and** *mE0*: *?mE 0* = *1*$_m$ *n* **using** *X* **by** (*simp-all add*: *exp-coeff-def seq-scale-def one-smult-mat*)
have *0*$_m$ *n n* = *cauchy-prod n* (λ*p*. *?mE p* + *?mD p*) (λ*p*. *?E p* + *D p*) *q*
using *inv q*(*2*) **unfolding** *mF FE*[*symmetric*] **by** (*simp add*: *seq-one-def*)
also have ... = *cauchy-prod n ?mE ?E q* + *cauchy-prod n ?mE D q* + (*cauchy-prod n ?mD ?E q* + *cauchy-prod n ?mD D q*)
using *mEc mDc E Dc* **by** (*simp add*: *cauchy-prod-add-left cauchy-prod-add-right seq-carrier-def*)
also have ... = *D q* + (−*1*) ^ *q* ·$_m$ *D q*
using *invE cauchy-prod-low-right*[*OF Dc Dl mEc*] *cauchy-prod-low-left*[*OF mDc mDl E*] *cauchy-prod-low-both*[*OF mDc mDl Dc Dl q*(*2*)] *E0 mE0 Dc*
by (*simp add*: *seq-scale-def*)
also have ... = *2* ·$_m$ *D q* **using** *q*(*1*) *Dc* **by** (*auto intro*!: *eq-matI*)
finally have *z*: *0*$_m$ *n n* = *2* ·$_m$ *D q* **.**
show *?thesis*
proof (*rule eq-matI*)
fix *i j* **assume** *i*: *i* < *dim-row* (*exp-coeff X q*) **and** *j*: *j* < *dim-col* (*exp-coeff X q*)
have *ij*: *i* < *n j* < *n* **using** *i j E* **by** *simp-all*
have *0* = *2* ∗ *D q* $$ (*i*, *j*) **using** *arg-cong*[*OF z*, *of* λ*M*. *M* $$ (*i*, *j*)] *Dc ij* **by** *simp*
hence *D q* $$ (*i*, *j*) = *0* **by** *simp*
thus *prod-coeff n Ys q* $$ (*i*, *j*) = *exp-coeff X q* $$ (*i*, *j*) **using** *F E ij* **by** (*simp add*: *D-def*)
qed (*use F E* **in** *simp-all*)
qed

Perturbations of degree q add: if $A_0 = B_0 = I$ and D_a, D_b vanish below q, then up to degree q

$$(A + D_a) * (B + D_b) = A * B + D_a + D_b.$$

lemma *pert2*:
assumes *A*: *seq-carrier n A A 0* = *1*$_m$ *n* **and** *B*: *seq-carrier n B B 0* = *1*$_m$ *n*
and *Da*: *seq-carrier n Da vanishes-below n q Da* **and** *Db*: *seq-carrier n Db vanishes-below n q Db* **and** *q*: *0* < *q* **and** *p*: *p* ≤ *q*
shows *cauchy-prod n* (λ*p*. *A p* + *Da p*) (λ*p*. *B p* + *Db p*) *p* = *cauchy-prod n A B p* + *Da p* + *Db p*

proof −
have *e*: *cauchy-prod n* (λ*p*. *A p* + *Da p*) (λ*p*. *B p* + *Db p*) *p*
= *cauchy-prod n A B p* + *cauchy-prod n A Db p* + (*cauchy-prod n Da B p* + *cauchy-prod n Da Db p*)
using *A B Da Db* **by** (*simp add*: *cauchy-prod-add-left cauchy-prod-add-right seq-carrier-def*)
have *AB*: *cauchy-prod n A B p* ∈ *carrier-mat n n* **using** *cauchy-prod-carrier*[*OF A*(*1*) *B*(*1*)] **by** *simp*
show *?thesis*
proof (*cases p* < *q*)
case *True*
thus *?thesis* **unfolding** *e* **using** *cauchy-prod-low-below*[*OF Db A*(*1*) *True*] *cauchy-prod-low-below*[*OF Da B*(*1*) *True*]
cauchy-prod-low-below[*OF Da Db*(*1*) *True*] *Da Db AB* **by** (*simp add*: *vanishes-below-def*)
next
case *False*
hence *pq*: *p* = *q* **using** *p* **by** *simp*
have *cauchy-prod n A Db q* = *Db q cauchy-prod n Da B q* = *Da q cauchy-prod n Da Db q* = 0_m *n n*
using *cauchy-prod-low-right*[*OF Db A*(*1*)] *cauchy-prod-low-left*[*OF Da B*(*1*)] *cauchy-prod-low-both*[*OF Da Db q*] *A B Da Db* **by** *simp-all*
thus *?thesis* **unfolding** *e* **unfolding** *pq* **using** *Da Db cauchy-prod-carrier*[*OF A*(*1*) *B*(*1*)] **by** (*auto intro*!: *eq-matI*)
qed
qed

lemma *cauchy-prod-congr*:
assumes ∀ *p*≤*q*. *f p* = *f*′ *p* ∀ *p*≤*q*. *g p* = *g*′ *p p* ≤ *q*
shows *cauchy-prod n f g p* = *cauchy-prod n f*′ *g*′ *p*
unfolding *cauchy-prod-def* **by** (*rule matrix-sum-cong*) (*use assms* **in** *auto*)

The key step of Suzuki's recursion. If the formula Φ (with factors Y) agrees with e^{tX} below degree q, then the formula made of copies of Φ with the factors scaled by $c_1, \ldots, c_k$ satisfies, up to degree q,

$$\mathrm{coeff}_p = \frac{\big((\sum_j c_j)X\big)^p}{p!} + \Big(\sum_j c_j^p\Big)\big(\mathrm{coeff}_p(\Phi) - X^p/p!\big),$$

so the error at degree q vanishes when $\sum_j c_j^q = 0$.

lemma *scaled-copies*:
assumes *Ys*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* **and** *X*: *X* ∈ *carrier-mat n n*
and *ord*: ∀ *p*<*q*. *prod-coeff n Ys p* = *exp-coeff X p* **and** *q*: *0* < *q*
shows ∀ *p*≤*q*. *prod-coeff n* (*concat* (*map* (λ*c*. *map* ((·$_m$) *c*) *Ys*) *cs*)) *p*
= *exp-coeff* (*sum-list cs* ·$_m$ *X*) *p* + (∑ *c*←*cs*. *c* ^ *p*) ·$_m$ (*prod-coeff n Ys p* − *exp-coeff X p*)
proof (*induction cs*)
case *Nil*
show *?case*

proof (*intro allI impI*)
fix p **assume** $p \leq q$
have *exp-coeff* $(0 \cdot_m X)$ p = *seq-one n p* **using** *zero-smult-mat*[*OF X*] *exp-coeff-zero* **by** *simp*
show *prod-coeff n* (*concat* (*map* (λ*c*. *map* ((·$_m$) *c*) *Ys*) [])) *p*
= *exp-coeff* (*sum-list* [] ·$_m$ *X*) *p* + (∑ *c*←[]. *c* ⁀ *p*) ·$_m$ (*prod-coeff n Ys p* − *exp-coeff X p*)
proof −
have *prod-coeff n Ys p* − *exp-coeff X p* ∈ *carrier-mat n n* **using** *X* **by** (*simp add*: *minus-carrier-mat*)
hence *z*: (*0*::*complex*) ·$_m$ (*prod-coeff n Ys p* − *exp-coeff X p*) = *0*$_m$ *n n* **by** (*rule zero-smult-mat*)
show *?thesis* **using** ‹*exp-coeff* (*0* ·$_m$ *X*) *p* = *seq-one n p*› **by** (*simp add*: *z seq-one-def*)
qed
qed
next
case (*Cons c cs*)
let *?F* = *prod-coeff n Ys* **and** *?E* = *exp-coeff X*
define *D* **where** *D* = (λ*p*. *?F p* − *?E p*)
let *?R* = *concat* (*map* (λ*c*. *map* ((·$_m$) *c*) *Ys*) *cs*)
have *F*: *seq-carrier n ?F* **and** *E*: *seq-carrier n ?E* **using** *prod-coeff-carrier*[*OF Ys*] *X* **by** *auto*
have *Dc*: *seq-carrier n D* **unfolding** *seq-carrier-def D-def* **using** *seq-carrier-in*[*OF E*] **by** (*simp add*: *minus-carrier-mat*)
have *Dl*: *vanishes-below n q D* **using** *ord seq-carrier-in*[*OF E*] **by** (*auto simp*: *vanishes-below-def D-def*)
have *cY*: ∀ *Y* ∈ *set* (*map* ((·$_m$) *c*) *Ys*). *Y* ∈ *carrier-mat n n* **and** *cR*: ∀ *Y* ∈ *set ?R*. *Y* ∈ *carrier-mat n n*
using *Ys* **by** *auto*
define *Ac* **where** *Ac* = *exp-coeff* (*c* ·$_m$ *X*)
define *Dc′* **where** *Dc′* = *seq-scale c D*
define *Ar* **where** *Ar* = *exp-coeff* (*sum-list cs* ·$_m$ *X*)
define *Dr* **where** *Dr* = (λ*p*. (∑ *c*←*cs*. *c* ⁀ *p*) ·$_m$ *D p*)
have *Acc*: *seq-carrier n Ac Ac 0* = *1*$_m$ *n* **and** *Arc*: *seq-carrier n Ar Ar 0* = *1*$_m$ *n*
using *X* **by** (*simp-all add*: *Ac-def Ar-def exp-coeff-def one-smult-mat*)
have *Dcc*: *seq-carrier n Dc′ vanishes-below n q Dc′* **using** *Dc Dl* **by** (*auto simp*: *Dc′-def seq-scale-def vanishes-below-def*)
have *Drc*: *seq-carrier n Dr vanishes-below n q Dr* **using** *Dc Dl* **by** (*auto simp*: *Dr-def seq-carrier-def vanishes-below-def*)
have *sF*: *seq-scale c ?F* = (λ*p*. *Ac p* + *Dc′ p*)
proof −
have *?F* = (λ*p*. *?E p* + *D p*) **using** *F E* **by** (*auto simp*: *fun-eq-iff D-def intro*!: *eq-matI*)
hence *seq-scale c ?F* = (λ*p*. *seq-scale c ?E p* + *seq-scale c D p*)
using *E Dc* **by** (*auto simp*: *fun-eq-iff seq-scale-def add-smult-distrib-left-mat*[**where** *nr* = *n* **and** *nc* = *n*])
thus *?thesis* **by** (*simp add*: *Ac-def Dc′-def fun-eq-iff exp-coeff-smult*[*OF X*])

qed
have *IH*: $\forall\, p{\le}q.$ *prod-coeff n ?R p* = *Ar p* + *Dr p* **using** *Cons.IH* **by** (*simp add*: *Ar-def Dr-def D-def*)
show *?case*
proof (*intro allI impI*)
fix *p* **assume** *p*: $p \le q$
have *prod-coeff n* (*concat* (*map* ($\lambda c.$ *map* ($(\cdot_m)$ *c*) *Ys*) (*c* # *cs*))) *p* = *cauchy-prod n* (*seq-scale c ?F*) (*prod-coeff n ?R*) *p*
using *prod-coeff-append*[*OF cY cR*] *prod-coeff-scale*[*OF Ys*] **by** *simp*
also have ... = *cauchy-prod n* ($\lambda p.$ *Ac p* + *Dc′ p*) ($\lambda p.$ *Ar p* + *Dr p*) *p*
unfolding *sF* **by** (*rule cauchy-prod-congr*) (*use IH p* **in** *auto*)
also have ... = *cauchy-prod n Ac Ar p* + *Dc′ p* + *Dr p* **by** (*rule pert2*[*OF Acc Arc Dcc Drc q p*])
also have *cauchy-prod n Ac Ar p* = *exp-coeff* (*sum-list* (*c* # *cs*) $\cdot_m$ *X*) *p*
unfolding *Ac-def Ar-def* **using** *exp-coeff-add*[*OF X*, *of c sum-list cs p*] **by** *simp*
finally have *r*: *prod-coeff n* (*concat* (*map* ($\lambda c.$ *map* ($(\cdot_m)$ *c*) *Ys*) (*c* # *cs*))) *p*
= *exp-coeff* (*sum-list* (*c* # *cs*) $\cdot_m$ *X*) *p* + *Dc′ p* + *Dr p* **.**
have *d*: *Dc′ p* + *Dr p* = ($\sum$ *c*←*c* # *cs*. *c* ^ *p*) $\cdot_m$ *D p*
using *Dc* **by** (*simp add*: *Dc′-def Dr-def seq-scale-def add-smult-distrib-right-mat*[*OF seq-carrier-in*[*OF Dc*], *symmetric*])
have *exp-coeff* (*sum-list* (*c* # *cs*) $\cdot_m$ *X*) *p* + *Dc′ p* + *Dr p* = *exp-coeff* (*sum-list* (*c* # *cs*) $\cdot_m$ *X*) *p* + (*Dc′ p* + *Dr p*)
by (*rule assoc-add-mat*[**where** *nr* = *n* **and** *nc* = *n*]) (*use X Dcc Drc* **in** *simp-all*)
thus *prod-coeff n* (*concat* (*map* ($\lambda c.$ *map* ($(\cdot_m)$ *c*) *Ys*) (*c* # *cs*))) *p*
= *exp-coeff* (*sum-list* (*c* # *cs*) $\cdot_m$ *X*) *p* + ($\sum$ *c*←*c* # *cs*. *c* ^ *p*) $\cdot_m$ (*prod-coeff n Ys p* − *exp-coeff X p*)
using *r d* **by** (*simp add*: *D-def*)
qed
qed

The constant term of a product is I and its linear term the sum of the factors; the sum of the factors is additive, symmetric under reversal, and linear under scaling.

lemma *prod-coeff-zero*: $\forall$ *Y* $\in$ *set Ys*. *Y* $\in$ *carrier-mat n n* $\Longrightarrow$ *prod-coeff n Ys 0* = 1_m *n*
by (*induction Ys*) (*auto simp*: *cauchy-prod-def seq-one-def exp-coeff-def one-smult-mat*)

lemma *prod-coeff-one*: $\forall$ *Y* $\in$ *set Ys*. *Y* $\in$ *carrier-mat n n* $\Longrightarrow$ *prod-coeff n Ys 1* = *hamiltonian-sum n Ys*
proof (*induction Ys*)
case *Nil*
thus *?case* **by** (*simp add*: *seq-one-def*)
next
case (*Cons Y Ys*)
have *Y*: *Y* $\in$ *carrier-mat n n* **and** *Ys*: $\forall$ *Z* $\in$ *set Ys*. *Z* $\in$ *carrier-mat n n* **using** *Cons.prems* **by** *auto*
have *F1*: *prod-coeff n Ys 1* $\in$ *carrier-mat n n* **and** *H*: *hamiltonian-sum n Ys* $\in$

carrier-mat n n
using *prod-coeff-carrier*[*OF Ys*] *hamiltonian-sum-carrier*[*of Ys n*] *Ys* **by** *auto*
have *prod-coeff n* (*Y # Ys*) *1* = *exp-coeff Y 1* ∗ *prod-coeff n Ys 0* + *exp-coeff Y 0* ∗ *prod-coeff n Ys 1*
using *seq-carrier-mult*[*OF exp-coeff-carrier*[*OF Y*] *prod-coeff-carrier*[*OF Ys*], *of 0 1*] **by** (*simp add*: *cauchy-prod-def*)
also have ... = *Y* + *prod-coeff n Ys 1* **using** *Y F1 prod-coeff-zero*[*OF Ys*] **by** (*simp add*: *exp-coeff-def one-smult-mat*)
finally show *?case* **using** *Cons.IH*[*OF Ys*] **by** *simp*
qed

lemma *hamiltonian-sum-append*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ⟹ ∀ *Y* ∈ *set Zs*. *Y* ∈ *carrier-mat n n* ⟹ *hamiltonian-sum n* (*Ys* @ *Zs*) = *hamiltonian-sum n Ys* + *hamiltonian-sum n Zs*
proof (*induction Ys*)
case *Nil*
thus *?case* **using** *hamiltonian-sum-carrier*[*of Zs n*] **by** *simp*
next
case (*Cons Y Ys*)
have *c*: *Y* ∈ *carrier-mat n n hamiltonian-sum n Ys* ∈ *carrier-mat n n hamiltonian-sum n Zs* ∈ *carrier-mat n n*
using *Cons.prems hamiltonian-sum-carrier*[*of Ys n*] *hamiltonian-sum-carrier*[*of Zs n*] **by** *auto*
thus *?case* **using** *Cons* **by** (*simp add*: *assoc-add-mat*[*OF c*])
qed

lemma *hamiltonian-sum-rev*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ⟹ *hamiltonian-sum n* (*rev Ys*) = *hamiltonian-sum n Ys*
proof (*induction Ys*)
case (*Cons Y Ys*)
have *c*: *Y* ∈ *carrier-mat n n hamiltonian-sum n Ys* ∈ *carrier-mat n n* **using** *Cons.prems hamiltonian-sum-carrier*[*of Ys n*] **by** *auto*
have *hamiltonian-sum n* (*rev* (*Y # Ys*)) = *hamiltonian-sum n* (*rev Ys*) + (*Y* + 0_m *n n*) **using** *Cons.prems* **by** (*simp add*: *hamiltonian-sum-append*)
thus *?case* **using** *Cons c* **by** (*simp add*: *comm-add-mat*[*OF c*(*2*) *c*(*1*)])
qed *simp*

lemma *hamiltonian-sum-scale*: ∀ *Y* ∈ *set Ys*. *Y* ∈ *carrier-mat n n* ⟹ *hamiltonian-sum n* (*map* ((·$_m$) *c*) *Ys*) = *c* ·$_m$ *hamiltonian-sum n Ys*
proof (*induction Ys*)
case *Nil*
show *?case* **by** (*rule eq-matI*) *auto*
next
case (*Cons Y Ys*)
thus *?case* **using** *hamiltonian-sum-carrier*[*of Ys n*] **by** (*simp add*: *add-smult-distrib-left-mat*[**where** *nr* = *n* **and** *nc* = *n*])
qed

A product formula is a list of factors (a, H) standing for e^{iatH}; its coefficients

are those of the matrices *factor-gen* $(a,\ H) = iaH$. The symmetric Strang formula $e^{itH_1/2} \cdots e^{itH_m/2} e^{itH_m/2} \cdots e^{itH_1/2}$ and Suzuki's recursion $S_{2k+2} = S_{2k}(u)^2\, S_{2k}(1-4u)\, S_{2k}(u)^2$ with $4u^q + (1-4u)^q = 0$, $q = 2k+3$.

definition *factor-gen* :: *real* × *complex Matrix.mat* ⇒ *complex Matrix.mat* **where**
factor-gen x = (i ∗ *complex-of-real* (*fst x*)) ·$_m$ *snd x*

definition *strang* :: *complex Matrix.mat list* ⇒ (*real* × *complex Matrix.mat*) *list* **where**
strang Hs = *map* (λ*H*. (*1* / *2*, *H*)) *Hs* @ *rev* (*map* (λ*H*. (*1* / *2*, *H*)) *Hs*)

definition *suzuki-weight* :: *nat* ⇒ *real* **where**
suzuki-weight q = *1* / (*4* − *root q 4*)

definition *suzuki-weights* :: *nat* ⇒ *real list* **where**
suzuki-weights q = (*let u* = *suzuki-weight q in* [*u*, *u*, *1* − *4* ∗ *u*, *u*, *u*])

fun *suzuki* :: *nat* ⇒ *complex Matrix.mat list* ⇒ (*real* × *complex Matrix.mat*) *list* **where**
suzuki 0 Hs = *strang Hs*
| *suzuki* (*Suc k*) *Hs* = *concat* (*map* (λ*c*. *map* (λ*x*. (*c* ∗ *fst x*, *snd x*)) (*suzuki k Hs*)) (*suzuki-weights* (*2* ∗ *k* + *3*)))

Suzuki's weights: for odd $q \geq 3$, $u = 1/(4-4^{1/q})$ satisfies $4u^q + (1-4u)^q = 0$, and the weights $u, u, 1-4u, u, u$ add up to 1.

lemma *suzuki-weight-cond*:
assumes *q*: *odd q 3* ≤ *q*
shows *4* ∗ *suzuki-weight q* ^ *q* + (*1* − *4* ∗ *suzuki-weight q*) ^ *q* = *0*
proof −
define *s* **where** *s* = *root q 4*
have *sq*: *s* ^ *q* = *4* **using** *q* **by** (*simp add*: *s-def*)
have *s4*: *4* − *s* ≠ *0*
proof
assume *4* − *s* = *0*
hence *s* ^ *q* = *4* ^ *q* **by** *simp*
moreover have (*4*::*real*) ^ *q* > *4* **using** *power-strict-increasing*[*of 1 q 4*::*real*] *q* **by** *simp*
ultimately show *False* **using** *sq* **by** *simp*
qed
have *1* − *4* ∗ *suzuki-weight q* = − *s* / (*4* − *s*) **using** *s4* **by** (*simp add*: *suzuki-weight-def s-def*[*symmetric*] *field-simps*)
hence *4* ∗ *suzuki-weight q* ^ *q* + (*1* − *4* ∗ *suzuki-weight q*) ^ *q* = (*4* − *s* ^ *q*) / (*4* − *s*) ^ *q*
using *q*(*1*) *s4* **by** (*simp add*: *suzuki-weight-def s-def*[*symmetric*] *power-divide power-minus-odd field-simps*)
thus *?thesis* **using** *sq* **by** *simp*
qed

Scaling a factor scales its generator; the formulas consist of factors of the given Hamiltonians, are $n \times n$, and are palindromes:

lemma *factor-gen-scale*: *factor-gen* $(c * a, H)$ = *complex-of-real* $c \cdot_m$ *factor-gen* (a, H)
by (*rule eq-matI*) (*auto simp*: *factor-gen-def*)

lemma *suzuki-snd*: $x \in$ *set* (*suzuki k Hs*) $\Longrightarrow$ *snd* $x \in$ *set Hs*
by (*induction k arbitrary*: *x*) (*auto simp*: *strang-def*)

lemma *suzuki-carrier*: $\forall H \in$ *set Hs*. $H \in$ *carrier-mat n n* $\Longrightarrow \forall Y \in$ *set* (*map factor-gen* (*suzuki k Hs*)). $Y \in$ *carrier-mat n n*
using *suzuki-snd* **by** (*fastforce simp*: *factor-gen-def*)

lemma *suzuki-palindrome*: *rev* (*suzuki k Hs*) = *suzuki k Hs*
by (*induction k*) (*simp-all add*: *strang-def suzuki-weights-def Let-def rev-map*)

The Strang formula agrees with e^{itH}, $H = \sum_j H_j$, up to degree 2:

lemma *strang-order*:
assumes *Hs*: $\forall H \in$ *set Hs*. $H \in$ *carrier-mat n n*
shows $\forall p \leq 2$. *prod-coeff n* (*map factor-gen* (*strang Hs*)) *p* = *exp-coeff* (i $\cdot_m$ *hamiltonian-sum n Hs*) *p*
proof −
let *?Z* = *map* ((·$_m$) (i / 2)) *Hs*
have *st*: *map factor-gen* (*strang Hs*) = *?Z* @ *rev ?Z*
by (*simp add*: *strang-def factor-gen-def rev-map comp-def*)
have *Z*: $\forall Y \in$ *set ?Z*. $Y \in$ *carrier-mat n n* **using** *Hs* **by** *auto*
have *St*: $\forall Y \in$ *set* (*map factor-gen* (*strang Hs*)). $Y \in$ *carrier-mat n n* **unfolding** *st* **using** *Z* **by** *auto*
have *pal*: *rev* (*map factor-gen* (*strang Hs*)) = *map factor-gen* (*strang Hs*) **unfolding** *st* **by** *simp*
have *H*: *hamiltonian-sum n Hs* ∈ *carrier-mat n n* **using** *hamiltonian-sum-carrier*[*of Hs n*] *Hs* **by** *auto*
have *X*: i $\cdot_m$ *hamiltonian-sum n Hs* ∈ *carrier-mat n n* **using** *H* **by** *simp*
have *h2*: *hamiltonian-sum n* (*map factor-gen* (*strang Hs*)) = i $\cdot_m$ *hamiltonian-sum n Hs*
proof −
have *hamiltonian-sum n* (*map factor-gen* (*strang Hs*)) = (i / 2) $\cdot_m$ *hamiltonian-sum n Hs* + (i / 2) $\cdot_m$ *hamiltonian-sum n Hs*
unfolding *st hamiltonian-sum-append*[*OF Z*] **by** (*simp add*: *Z hamiltonian-sum-rev hamiltonian-sum-scale*[*OF Hs*] *hamiltonian-sum-append*)
also have ... = i $\cdot_m$ *hamiltonian-sum n Hs* **using** *H* **by** (*auto intro*!: *eq-matI*)
finally show *?thesis* **.**
qed
have *ord1*: $\forall p < 2$. *prod-coeff n* (*map factor-gen* (*strang Hs*)) *p* = *exp-coeff* (i $\cdot_m$ *hamiltonian-sum n Hs*) *p*
proof (*intro allI impI*)
fix *p* :: *nat* **assume** *p* < 2
hence *p* = 0 ∨ *p* = 1 **by** *auto*
thus *prod-coeff n* (*map factor-gen* (*strang Hs*)) *p* = *exp-coeff* (i $\cdot_m$ *hamiltonian-sum n Hs*) *p*
using *prod-coeff-zero*[*OF St*] *prod-coeff-one*[*OF St*] *h2 X* **by** (*auto simp*:

exp-coeff-def one-smult-mat)
qed
have *prod-coeff n* (*map factor-gen* (*strang Hs*)) *2* = *exp-coeff* (i ·$_m$ *hamiltonian-sum n Hs*) *2* **by** (*rule sym-order*[*OF St X pal ord1*]) *simp-all*
thus *?thesis* **using** *ord1* **by** (*auto simp*: *le-less*)
qed

By induction on k, using the palindrome property (odd order) and *scaled-copies* with $\sum_j c_j = 1$ and $\sum_j c_j^{2k+3} = 0$:

$$\mathrm{coeff}_p(S_{2k+2}) = \frac{(iH)^p}{p!} \qquad (p \le 2k+2).$$

theorem *suzuki-order*:
assumes *Hs*: ∀ *H* ∈ *set Hs*. *H* ∈ *carrier-mat n n*
shows ∀ *p*≤*2* ∗ *k* + *2*. *prod-coeff n* (*map factor-gen* (*suzuki k Hs*)) *p* = *exp-coeff* (i ·$_m$ *hamiltonian-sum n Hs*) *p*
proof (*induction k*)
case *0*
thus *?case* **using** *strang-order*[*OF Hs*] **by** *simp*
next
case (*Suc k*)
let *?S* = *map factor-gen* (*suzuki k Hs*) **and** *?q* = *2* ∗ *k* + *3* **and** *?X* = i ·$_m$ *hamiltonian-sum n Hs*
let *?cs* = *map complex-of-real* (*suzuki-weights ?q*)
have *S*: ∀ *Y* ∈ *set ?S*. *Y* ∈ *carrier-mat n n* **by** (*rule suzuki-carrier*[*OF Hs*])
have *X*: *?X* ∈ *carrier-mat n n* **using** *hamiltonian-sum-carrier*[*of Hs n*] *Hs* **by** *auto*
have *ordS*: ∀ *p*<*?q*. *prod-coeff n ?S p* = *exp-coeff ?X p* **using** *Suc.IH* **by** *auto*
have *mt*: *map factor-gen* (*suzuki* (*Suc k*) *Hs*) = *concat* (*map* (λ*c*. *map* ((·$_m$) *c*) *?S*) *?cs*)
by (*simp add*: *map-concat comp-def factor-gen-scale*)
define *u* **where** *u* = *complex-of-real* (*suzuki-weight ?q*)
have *cs*: *?cs* = [*u*, *u*, *1* − *4* ∗ *u*, *u*, *u*] **by** (*simp add*: *suzuki-weights-def u-def Let-def*)
have *sum1*: *sum-list ?cs* = *1* **by** (*simp add*: *cs*)
have *sumq*: (∑ *c*←*?cs*. *c* ^ *?q*) = *0*
proof −
have (∑ *c*←*?cs*. *c* ^ *?q*) = *complex-of-real* (*4* ∗ *suzuki-weight ?q* ^ *?q* + (*1* − *4* ∗ *suzuki-weight ?q*) ^ *?q*)
by (*simp add*: *cs u-def*)
thus *?thesis* **using** *suzuki-weight-cond*[*of ?q*] **by** *simp*
qed
have *sc*: ∀ *p*≤*?q*. *prod-coeff n* (*map factor-gen* (*suzuki* (*Suc k*) *Hs*)) *p*
= *exp-coeff* (*sum-list ?cs* ·$_m$ *?X*) *p* + (∑ *c*←*?cs*. *c* ^ *p*) ·$_m$ (*prod-coeff n ?S p* − *exp-coeff ?X p*)
unfolding *mt* **by** (*rule scaled-copies*[*OF S X ordS*]) *simp*
have *ordq*: ∀ *p*≤*?q*. *prod-coeff n* (*map factor-gen* (*suzuki* (*Suc k*) *Hs*)) *p* = *exp-coeff ?X p*

proof (*intro allI impI*)
fix *p* **assume** *p*: $p \leq ?q$
have *z*: $(\sum c{\leftarrow}?cs.\ c\ ^\frown p) \cdot_m (\mathit{prod\text{-}coeff}\ n\ ?S\ p - \mathit{exp\text{-}coeff}\ ?X\ p) = 0_m\ n\ n$
proof (*cases* $p < ?q$)
case *True*
hence $\mathit{prod\text{-}coeff}\ n\ ?S\ p - \mathit{exp\text{-}coeff}\ ?X\ p = 0_m\ n\ n$ **using** *ordS seq-carrier-in*[*OF exp-coeff-carrier*[*OF X*]] **by** *simp*
thus *?thesis* **by** (*auto intro*!: *eq-matI*)
next
case *False*
hence $p = ?q$ **using** *p* **by** *simp*
have $\mathit{prod\text{-}coeff}\ n\ ?S\ p - \mathit{exp\text{-}coeff}\ ?X\ p \in \mathit{carrier\text{-}mat}\ n\ n$ **using** *X* **by** (*simp add*: *minus-carrier-mat*)
thus *?thesis* **using** *sumq* ‹$p = ?q$› **by** (*simp add*: *zero-smult-mat*)
qed
have *e*: $\mathit{prod\text{-}coeff}\ n\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs}))\ p$
$= \mathit{exp\text{-}coeff}\ (\mathit{sum\text{-}list}\ ?cs \cdot_m ?X)\ p + (\sum c{\leftarrow}?cs.\ c\ ^\frown p) \cdot_m (\mathit{prod\text{-}coeff}\ n\ ?S\ p - \mathit{exp\text{-}coeff}\ ?X\ p)$
using *sc p* **by** *blast*
show $\mathit{prod\text{-}coeff}\ n\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs}))\ p = \mathit{exp\text{-}coeff}\ ?X\ p$
unfolding *e z sum1* **using** *seq-carrier-in*[*OF exp-coeff-carrier*[*OF X*]] *X* **by** (*simp add*: *one-smult-mat*)
qed
have S': $\forall Y \in \mathit{set}\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs}))\ .\ Y \in \mathit{carrier\text{-}mat}\ n\ n$ **by** (*rule suzuki-carrier*[*OF Hs*])
have *pal*′: $\mathit{rev}\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs})) = \mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs})$
using *suzuki-palindrome*[*of Suc k Hs*] **by** (*metis rev-map*)
have *top*: $\mathit{prod\text{-}coeff}\ n\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs}))\ (?q + 1) = \mathit{exp\text{-}coeff}\ ?X\ (?q + 1)$
by (*rule sym-order*[*OF S*′ *X pal*′]) (*use ordq* **in** *auto*)
show *?case*
proof (*intro allI impI*)
fix *p* **assume** $p \leq 2 * \mathit{Suc}\ k + 2$
hence $p \leq ?q \vee p = ?q + 1$ **by** *arith*
thus $\mathit{prod\text{-}coeff}\ n\ (\mathit{map}\ \mathit{factor\text{-}gen}\ (\mathit{suzuki}\ (\mathit{Suc}\ k)\ \mathit{Hs}))\ p = \mathit{exp\text{-}coeff}\ ?X\ p$ **using** *ordq top* **by** *blast*
qed
qed

7 Higher-order product formulas

Suzuki's formula of order $2k{+}2$ over r steps is within $r(2|F|{+}1)(\Lambda|t|/r)^{2k+3}e^{\Lambda|t|/r}/(2k{+}3)!$ of e^{itH} (*suzuki-error*), so $r \sim |t|^{1+1/(2k+2)}$ steps suffice. The proof compares both sides with their Taylor polynomials of degree $2k+2$, which agree by *suzuki-order*.

Two facts about factorials and the exponential series: $i!\,k! \leq (i+k)!$, and

$$\left|e^z - \sum_{q \leq P} \frac{z^q}{q!}\right| \leq \frac{|z|^{P+1} e^{|z|}}{(P+1)!}.$$

lemma *fact-mult-le-fact-add*: *fact i* $*$ *fact k* $\leq$ (*fact* (*i* + *k*) :: *real*)
proof (*induction k*)
case (*Suc k*)
have *fact i* $*$ *fact* (*Suc k*) = (*fact i* $*$ *fact k*) $*$ *real* (*Suc k*) **by** (*simp add*: *algebra-simps*)
also have ... $\leq$ *fact* (*i* + *k*) $*$ *real* (*Suc* (*i* + *k*)) **by** (*rule mult-mono*) (*use Suc.IH* **in** *auto*)
also have ... = *fact* (*i* + *Suc k*) **by** (*simp add*: *algebra-simps*)
finally show *?case* **.**
qed *simp*

lemma *exp-tail*:
fixes *z* :: *complex*
shows *cmod* (*exp z* − (∑ *q*≤*P*. *z* ^ *q* / *fact q*)) ≤ *cmod z* ^ *Suc P* $*$ *exp* (*cmod z*) / *fact* (*Suc P*)
proof −
let *?f* = λ*n*. *z* ^ *n* /$_R$ *fact n* **and** *?x* = *cmod z*
have *s*: *?f sums exp z* **by** (*rule exp-converges*)
have *t*: (λ*i*. *?f* (*i* + *Suc P*)) *sums* (*exp z* − (∑ *i*<*Suc P*. *?f i*)) **by** (*rule sums-split-initial-segment*[*OF s*])
have *e*: *exp z* − (∑ *q*≤*P*. *z* ^ *q* / *fact q*) = (∑ *i*. *?f* (*i* + *Suc P*))
using *sums-unique*[*OF t*] **by** (*simp add*: *lessThan-Suc-atMost scaleR-conv-of-real field-simps*)
have *sn*: *summable* (λ*i*. *norm* (*?f* (*i* + *Suc P*)))
by (*rule summable-ignore-initial-segment*[*OF summable-norm-exp*])
have *sx*: *summable* (λ*i*. *?x* ^ *i* / *fact i*) **using** *summable-exp*[*of ?x*] **by** (*simp add*: *field-simps*)
have *cmod* (∑ *i*. *?f* (*i* + *Suc P*)) ≤ (∑ *i*. *norm* (*?f* (*i* + *Suc P*))) **by** (*rule summable-norm*[*OF sn*])
also have ... ≤ (∑ *i*. *?x* ^ *Suc P* / *fact* (*Suc P*) $*$ (*?x* ^ *i* / *fact i*))
proof (*intro suminf-le sn summable-mult*[*OF sx*])
fix *i*
have *norm* (*?f* (*i* + *Suc P*)) = *?x* ^ (*i* + *Suc P*) / *fact* (*i* + *Suc P*) **by** (*simp add*: *norm-power norm-mult field-simps del*: *fact-Suc power-Suc add-Suc-right*)
also have ... ≤ *?x* ^ (*i* + *Suc P*) / (*fact i* $*$ *fact* (*Suc P*))
by (*rule divide-left-mono*) (*use fact-mult-le-fact-add*[*of i Suc P*] **in** ‹*simp-all del*: *fact-Suc add-Suc-right*›)
also have ... = *?x* ^ *Suc P* / *fact* (*Suc P*) $*$ (*?x* ^ *i* / *fact i*) **by** (*simp add*: *power-add field-simps*)
finally show *norm* (*?f* (*i* + *Suc P*)) ≤ *?x* ^ *Suc P* / *fact* (*Suc P*) $*$ (*?x* ^ *i* / *fact i*) **.**
qed
also have ... = *?x* ^ *Suc P* / *fact* (*Suc P*) $*$ *exp ?x*
proof −

have *ex*: $(\sum i.\ ?x \mathbin{\hat{}} i \mathbin{/} \mathit{fact}\ i) = \mathit{exp}\ ?x$
using *sums-unique*[*OF exp-converges*[*of ?x*]] **by** (*simp add*: *field-simps*)
show *?thesis* **using** *suminf-mult*[*OF sx*, *of ?x* $\hat{}$ *Suc P* / *fact* (*Suc P*)] *ex* **by** *simp*
qed
finally show *?thesis* **using** *e* **by** (*simp add*: *field-simps*)
qed

A sum of matrices that act on v by scalars acts by the sum of the scalars.

lemma *matrix-sum-eigen*:
fixes *F* :: *nat* ⇒ *complex Matrix.mat*
assumes *F*: $\bigwedge q.\ q < m \Longrightarrow F\ q \in \mathit{carrier\text{-}mat}\ n\ n$ **and** *v*: *dim-vec v* = *n*
and *ev*: $\bigwedge q.\ q < m \Longrightarrow F\ q *_v\ v = c\ q \cdot_v v$
shows *matrix-sum n F m* $*_v$ *v* = $(\sum q{<}m.\ c\ q) \cdot_v v$
using *F ev*
proof (*induction m*)
case *0*
show *?case* **using** *v* **by** (*auto intro*!: *eq-vecI*)
next
case (*Suc m*)
have *S*: *matrix-sum n F m* ∈ *carrier-mat n n* **by** (*rule matrix-sum-dim*) (*use Suc.prems* **in** *auto*)
have *matrix-sum n F* (*Suc m*) $*_v$ *v* = *F m* $*_v$ *v* + *matrix-sum n F m* $*_v$ *v*
using *add-mult-distrib-mat-vec*[*OF Suc.prems*(*1*)[*of m*] *S carrier-vecI*[*OF v*]] **by** *simp*
also have ... = *c m* $\cdot_v$ *v* + $(\sum q{<}m.\ c\ q) \cdot_v v$ **using** *Suc* **by** *simp*
also have ... = $(\sum q{<}\mathit{Suc}\ m.\ c\ q) \cdot_v v$ **by** (*auto intro*!: *eq-vecI simp*: *algebra-simps*)
finally show *?case* **.**
qed

The Taylor polynomial $\sum_{q \le P} (isH)^q/q!$ of e^{isH}. On an eigenvector it is the Taylor polynomial of $e^{is\lambda}$, so it is within $(|s|L)^{P+1} e^{|s|L}/(P+1)!$ of e^{isH} when $\|Hv\| \le L\|v\|$ (*evolution-taylor*).

definition *exp-taylor* :: *nat* ⇒ *complex Matrix.mat* ⇒ *real* ⇒ *nat* ⇒ *complex Matrix.mat* **where**
exp-taylor n H s P = *matrix-sum n* (λ*q*. *exp-coeff* ((i ∗ *complex-of-real s*) $\cdot_m$ *H*) *q*) (*Suc P*)

lemma *exp-taylor-carrier*: *H* ∈ *carrier-mat n n* ⟹ *exp-taylor n H s P* ∈ *carrier-mat n n*
unfolding *exp-taylor-def* **by** (*rule matrix-sum-dim*) (*simp add*: *exp-coeff-def*)

lemma *exp-taylor-eigen*:
assumes *H*: *H* ∈ *carrier-mat n n* **and** *v*: *dim-vec v* = *n* **and** *ev*: *H* $*_v$ *v* = *complex-of-real l* $\cdot_v$ *v*
shows *exp-taylor n H s P* $*_v$ *v* = $(\sum q{\le}P.$ (i ∗ *complex-of-real* (*s* ∗ *l*)) $\mathbin{\hat{}}$ *q* / *fact q*) $\cdot_v$ *v*
proof −

```
  have exp-coeff ((i * complex-of-real s) ·_m H) q *_v v = ((i * complex-of-real (s *
l)) ^ q / fact q) ·_v v for q
  proof −
    let ?c = complex-of-real (1 / fact q) * (i * complex-of-real s) ^ q
    have t: exp-coeff ((i * complex-of-real s) ·_m H) q = ?c ·_m H ^_m q
      using H by (simp add: exp-coeff-def smult-pow-mat) (rule eq-matI, auto)
    have Hq: H ^_m q ∈ carrier-mat n n using H by simp
    have exp-coeff ((i * complex-of-real s) ·_m H) q *_v v = ?c ·_v ((H ^_m q) *_v v)
      unfolding t by (rule smult-mat-mult-mat-vec-assoc[OF Hq carrier-vecI[OF
v]])
    also have ... = ?c ·_v ((complex-of-real l ^ q) ·_v v) unfolding pow-eigen[OF
H v ev] ..
    also have ... = ((i * complex-of-real (s * l)) ^ q / fact q) ·_v v
      by (simp add: smult-smult-assoc power-mult-distrib field-simps)
    finally show ?thesis .
  qed
  thus ?thesis unfolding exp-taylor-def lessThan-Suc-atMost[symmetric]
    by (intro matrix-sum-eigen[OF - v]) (use H in ‹simp-all add: exp-coeff-def›)
qed

lemma evolution-taylor:
  assumes H: herm-bounded n L H and L: 0 ≤ L
  shows evolution H s ≈(|s| * L) ^ Suc P * exp (|s| * L) / fact (Suc P) exp-taylor n
H s P
proof −
  have Hc: H ∈ carrier-mat n n hermitian H using herm-boundedD[OF H] by
simp-all
  let ?g = λx. cis (s * x) − (∑ q≤P. (i * complex-of-real (s * x)) ^ q / fact q)
  let ?B = evolution H s − exp-taylor n H s P
  have mf: mat-fun ?g H = ?B
  proof (rule mat-fun-eqI[OF Hc])
   show ?B ∈ carrier-mat n n by (rule minus-carrier-mat[OF exp-taylor-carrier[OF
Hc(1)]])
    show ?B acts-as ?g on-eigenvectors-of H unfolding acts-on-eigenvectors-def
    proof (intro allI impI)
      fix v l assume v: dim-vec v = dim-row H and ev: H *_v v = complex-of-real
l ·_v v
      have vn: dim-vec v = n using v Hc(1) by simp
      have ?B *_v v = evolution H s *_v v − exp-taylor n H s P *_v v
      by (rule minus-mult-distrib-mat-vec[OF evolution-carrier[OF Hc] exp-taylor-carrier[OF
Hc(1)] carrier-vecI[OF vn]])
      also have ... = ?g l ·_v v
        using evolution-eigen[OF Hc vn ev] exp-taylor-eigen[OF Hc(1) vn ev] by
(auto intro!: eq-vecI simp: algebra-simps)
      finally show ?B *_v v = ?g l ·_v v .
    qed
  qed
  have c0: 0 ≤ (|s| * L) ^ Suc P * exp (|s| * L) / fact (Suc P) using L by simp
  show ?thesis
```

proof (*rule op-closeI*[*OF evolution-carrier*[*OF Hc*]])
fix v :: *complex Matrix.vec* **assume** v: *dim-vec* $v = n$
have *evolution* H s $*_v$ v $-$ *exp-taylor* n H s P $*_v$ v $=$ *?B* $*_v$ v
by (*rule minus-mult-distrib-mat-vec*[*OF evolution-carrier*[*OF Hc*] *exp-taylor-carrier*[*OF Hc*(*1*)] *carrier-vecI*[*OF v*], *symmetric*])
also have ... $=$ *mat-fun ?g H* $*_v$ v **unfolding** *mf* **..**
finally have *e*: *evolution* H s $*_v$ v $-$ *exp-taylor* n H s P $*_v$ v $=$ *mat-fun ?g H* $*_v$ v **.**
have $\|$*mat-fun ?g H* $*_v$ $v\| \leq (|s| * L)$ ^ *Suc P* $*$ *exp* $(|s| * L)$ / *fact* (*Suc P*) $* \|v\|$
proof (*rule mat-fun-bound*[*OF Hc c0 - v*])
fix l **assume** l: $l \in$ *herm-spectrum H*
have *lL*: $|l| \leq L$ **by** (*rule herm-spectrum-bound*[*OF Hc*(*1*) *herm-boundedD*(*3*)[*OF H*] l])
have *cmod* (*?g l*) $=$ *cmod* (*exp* (i $*$ *complex-of-real* $(s * l)$) $-$ ($\sum q \leq P$. (i $*$ *complex-of-real* $(s * l)$) ^ q / *fact* q))
by (*simp add*: *cis-conv-exp*)
also have ... $\leq |s * l|$ ^ *Suc P* $*$ *exp* $|s * l|$ / *fact* (*Suc P*)
proof $-$
have *z*: *cmod* (i $*$ *complex-of-real* $(s * l)$) $= |s * l|$ **by** (*simp add*: *norm-mult abs-mult*)
show *?thesis* **using** *exp-tail*[*of* i $*$ *complex-of-real* $(s * l)$ P] **unfolding** z **.**
qed
also have ... $\leq (|s| * L)$ ^ *Suc P* $*$ *exp* $(|s| * L)$ / *fact* (*Suc P*)
proof $-$
have *sl*: $|s * l| \leq |s| * L$ **using** *lL* **by** (*simp add*: *abs-mult mult-left-mono*)
show *?thesis* **by** (*intro divide-right-mono mult-mono power-mono sl*) (*simp-all add*: *sl L*)
qed
finally show *cmod* (*?g l*) $\leq (|s| * L)$ ^ *Suc P* $*$ *exp* $(|s| * L)$ / *fact* (*Suc P*) **.**
qed
thus $\|$*evolution* H s $*_v$ v $-$ *exp-taylor* n H s P $*_v$ $v\| \leq (|s| * L)$ ^ *Suc P* $*$ *exp* $(|s| * L)$ / *fact* (*Suc P*) $* \|v\|$
unfolding e **.**
qed
qed

lemma *square-reindex*:
fixes G :: *nat* $\Rightarrow$ *nat* $\Rightarrow$ *'a*::*comm-monoid-add*
shows ($\sum q \leq P$. $\sum r \leq Q$. G q r) $=$ ($\sum p \leq P + Q$. $\sum q \leq p$. *if* $q \leq P \wedge p - q \leq Q$ *then* G q $(p - q)$ *else* 0)
proof $-$
have ($\sum q \leq P$. $\sum r \leq Q$. G q r) $=$ ($\sum (q, r) \in \{..P\} \times \{..Q\}$. G q r) **by** (*simp add*: *sum.cartesian-product*)
also have ... $=$ ($\sum (p, q) \in \{(p, q).\ p \leq P + Q \wedge q \leq p \wedge q \leq P \wedge p - q \leq Q\}$. G q $(p - q)$)
by (*rule sum.reindex-bij-witness*[**where** $i = \lambda(p, q).\ (q, p - q)$ **and** $j = \lambda(q, r).\ (q + r, q)$]) *auto*
also have ... $=$ ($\sum (p, q) \in$ *Sigma* $\{..P + Q\}$ $(\lambda p.\ \{..p\})$. *if* $q \leq P \wedge p - q \leq Q$

then G q (p − q) else 0)
by (*rule sum.mono-neutral-cong-left*) (*auto split*: *if-splits*)
also have ... = (∑ *p*≤*P* + *Q*. ∑ *q*≤*p*. *if q* ≤ *P* ∧ *p* − *q* ≤ *Q then G q* (*p* − *q*) *else 0*)
by (*rule sum.Sigma*[*symmetric*]) *auto*
finally show *?thesis* **.**
qed

Evaluating a series up to degree P, $\sum_{p\le P} t^p f_p$, and truncating it at degree P. Evaluation turns Cauchy products of truncations into products of matrices:

$$\Big(\sum_{p\le P} t^p f_p\Big)\Big(\sum_{q\le Q} t^q g_q\Big) = \sum_{p\le P+Q} t^p\,(f^{\le P} * g^{\le Q})_p.$$

definition *poly-eval* :: *nat* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *nat* ⇒ *real* ⇒ *complex Matrix.mat* **where**
poly-eval n f P t = *matrix-sum n* (λ*p*. (*complex-of-real t* ^ *p*) ·$_m$ *f p*) (*Suc P*)

definition *truncate-seq* :: *nat* ⇒ *nat* ⇒ (*nat* ⇒ *complex Matrix.mat*) ⇒ *nat* ⇒ *complex Matrix.mat* **where**
truncate-seq n P f p = (*if p* ≤ *P then f p else* 0$_m$ *n n*)

lemma *truncate-seq-carrier* [*simp*]: *seq-carrier n f* ⟹ *seq-carrier n* (*truncate-seq n P f*)
by (*simp add*: *seq-carrier-def truncate-seq-def*)

lemma *poly-eval-carrier*: *seq-carrier n f* ⟹ *poly-eval n f P t* ∈ *carrier-mat n n*
unfolding *poly-eval-def* **by** (*rule matrix-sum-dim*) *simp*

lemma *poly-eval-index*:
assumes *seq-carrier n f i* < *n j* < *n*
shows *poly-eval n f P t* $$ (*i*, *j*) = (∑ *p*≤*P*. *complex-of-real t* ^ *p* ∗ *f p* $$ (*i*, *j*))
unfolding *poly-eval-def lessThan-Suc-atMost*[*symmetric*] **using** *assms* **by** (*subst matrix-sum-index*) *simp-all*

lemma *poly-eval-mult*:
assumes *f*: *seq-carrier n f* **and** *g*: *seq-carrier n g*
shows *poly-eval n f P t* ∗ *poly-eval n g Q t* = *poly-eval n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) (*P* + *Q*) *t*
proof (*rule eq-matI*)
let *?t* = *complex-of-real t*
have *c*: *seq-carrier n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) **using** *f g* **by** (*simp add*: *cauchy-prod-carrier*)
fix *i j* **assume** *i*: *i* < *dim-row* (*poly-eval n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) (*P* + *Q*) *t*)
and *j*: *j* < *dim-col* (*poly-eval n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) (*P* + *Q*) *t*)
have *ij*: *i* < *n j* < *n* **using** *i j carrier-matD*[*OF poly-eval-carrier*[*OF c*]] **by** *auto*
have *sw*: (∑ *k*<*n*. ∑ *q*≤*P*. ∑ *r*≤*Q*. *T k q r*) = (∑ *q*≤*P*. ∑ *r*≤*Q*. ∑ *k*<*n*. *T k q r*) **for** *T* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *complex*

by (*subst sum.swap*) (*rule sum.cong*[*OF refl*], *rule sum.swap*)
have (*poly-eval n f P t* ∗ *poly-eval n g Q t*) $$ (*i*, *j*) = (∑ *k*<*n*. *poly-eval n f P t* $$ (*i*, *k*) ∗ *poly-eval n g Q t* $$ (*k*, *j*))
using *ij carrier-matD*[*OF poly-eval-carrier*[*OF f*, **where** *P* = *P* **and** *t* = *t*]] *carrier-matD*[*OF poly-eval-carrier*[*OF g*, **where** *P* = *Q* **and** *t* = *t*]]
by (*simp add*: *scalar-prod-def atLeast0LessThan*)
also have ... = (∑ *k*<*n*. (∑ *q*≤*P*. *?t* ^ *q* ∗ *f q* $$ (*i*, *k*)) ∗ (∑ *r*≤*Q*. *?t* ^ *r* ∗ *g r* $$ (*k*, *j*)))
by (*intro sum.cong refl*) (*simp add*: *poly-eval-index*[*OF f ij*(*1*)] *poly-eval-index*[*OF g - ij*(*2*)])
also have ... = (∑ *q*≤*P*. ∑ *r*≤*Q*. ∑ *k*<*n*. *?t* ^ (*q* + *r*) ∗ (*f q* $$ (*i*, *k*) ∗ *g r* $$ (*k*, *j*)))
by (*simp only*: *sum-product sw*) (*simp add*: *power-add mult-ac*)
also have ... = (∑ *p*≤*P* + *Q*. ∑ *q*≤*p*. *if q* ≤ *P* ∧ *p* − *q* ≤ *Q*
then ∑ *k*<*n*. *?t* ^ (*q* + (*p* − *q*)) ∗ (*f q* $$ (*i*, *k*) ∗ *g* (*p* − *q*) $$ (*k*, *j*)) *else 0*)
by (*rule square-reindex*)
also have ... = (∑ *p*≤*P* + *Q*. *?t* ^ *p* ∗ (∑ *q*≤*p*. ∑ *k*<*n*. *truncate-seq n P f q* $$ (*i*, *k*) ∗ *truncate-seq n Q g* (*p* − *q*) $$ (*k*, *j*)))
unfolding *sum-distrib-left*
by (*intro sum.cong refl*) (*use ij* **in** ‹*auto simp*: *truncate-seq-def sum-distrib-left mult-ac*›)
also have ... = *poly-eval n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) (*P* + *Q*) *t* $$ (*i*, *j*)
using *poly-eval-index*[*OF c ij*] *cauchy-prod-index*[*OF truncate-seq-carrier*[*OF f*] *truncate-seq-carrier*[*OF g*] *ij*] **by** *simp*
finally show (*poly-eval n f P t* ∗ *poly-eval n g Q t*) $$ (*i*, *j*) = *poly-eval n* (*cauchy-prod n* (*truncate-seq n P f*) (*truncate-seq n Q g*)) (*P* + *Q*) *t* $$ (*i*, *j*) .
qed (*use carrier-matD*[*OF poly-eval-carrier*[*OF f*, **where** *P* = *P* **and** *t* = *t*]] *carrier-matD*[*OF poly-eval-carrier*[*OF g*, **where** *P* = *Q* **and** *t* = *t*]]
carrier-matD[*OF poly-eval-carrier*[*OF cauchy-prod-carrier*[*OF truncate-seq-carrier*[*OF f*] *truncate-seq-carrier*[*OF g*]], **where** *P* = *P* + *Q* **and** *t* = *t*]] **in** *simp-all*)

op-bound n c M: $\|Mv\| \le c\|v\|$ for all v. Bounds multiply over products, scale, add over sums, and give $\|X^p/p!\| \le a^p/p!$ and $\|(f * g)_p\| \le \sum_{q \le p} c_q\, c'_{p-q}$.

definition *op-bound* :: *nat* ⇒ *real* ⇒ *complex Matrix.mat* ⇒ *bool* **where**
op-bound n c M ⟷ (∀ *v*. *dim-vec v* = *n* ⟶ ‖*M* ∗*v v*‖ ≤ *c* ∗ ‖*v*‖)

lemma *op-boundD*: *op-bound n c M* ⟹ *dim-vec v* = *n* ⟹ ‖*M* ∗*v v*‖ ≤ *c* ∗ ‖*v*‖
by (*simp add*: *op-bound-def*)

lemma *op-bound-zero*: *op-bound n 0* (*0m n n*)
proof −
have *0m n n* ∗*v v* = *0v n* **if** *dim-vec v* = *n* **for** *v* :: *complex Matrix.vec*
using *that* **by** (*intro eq-vecI*) (*auto simp*: *scalar-prod-def*)
thus *?thesis* **by** (*simp add*: *op-bound-def vnorm-zero-vec*)
qed

lemma *op-bound-mult*:

assumes A: $A \in$ *carrier-mat n n op-bound n a A* **and** B: $B \in$ *carrier-mat n n op-bound n b B* **and** a: $0 \leq a$
shows *op-bound n* $(a * b)$ $(A * B)$
unfolding *op-bound-def*
proof (*intro allI impI*)
fix v :: *complex Matrix.vec* **assume** v: *dim-vec* $v = n$
have $\|(A * B) *_v v\| = \|A *_v (B *_v v)\|$ **using** *assoc-mult-mat-vec*[*OF A*(*1*) *B*(*1*) *carrier-vecI*[*OF v*]] **by** *simp*
also have $\ldots \leq a * \|B *_v v\|$ **by** (*rule op-boundD*[*OF A*(*2*)]) (*use B*(*1*) **in** *simp*)
also have $\ldots \leq a * (b * \|v\|)$ **by** (*rule mult-left-mono*[*OF op-boundD*[*OF B*(*2*) *v*] *a*])
finally show $\|(A * B) *_v v\| \leq a * b * \|v\|$ **by** (*simp add*: *mult-ac*)
qed

lemma *op-bound-smult*: $A \in$ *carrier-mat n n* $\Longrightarrow$ *op-bound n a A* $\Longrightarrow$ *op-bound n* (*cmod c* $*$ *a*) $(c \cdot_m A)$
unfolding *op-bound-def*
proof (*intro allI impI*)
fix v :: *complex Matrix.vec* **assume** A: $A \in$ *carrier-mat n n* **and** b: $\forall v.$ *dim-vec* $v = n \longrightarrow \|A *_v v\| \leq a * \|v\|$
and v: *dim-vec* $v = n$
have $\|(c \cdot_m A) *_v v\| =$ *cmod c* $*$ $\|A *_v v\|$
using *smult-mat-mult-mat-vec-assoc*[*OF A carrier-vecI*[*OF v*]] **by** (*simp add*: *cpx-vec-length-smult*)
also have $\ldots \leq$ *cmod c* $*$ $(a * \|v\|)$ **using** *b v* **by** (*simp add*: *mult-left-mono*)
finally show $\|(c \cdot_m A) *_v v\| \leq$ *cmod c* $*$ *a* $*$ $\|v\|$ **by** (*simp add*: *mult-ac*)
qed

lemma *op-bound-matrix-sum*:
assumes $\bigwedge q.\ q < m \Longrightarrow F\ q \in$ *carrier-mat n n* $\bigwedge q.\ q < m \Longrightarrow$ *op-bound n* $(c\ q)$ $(F\ q)$
shows *op-bound n* $(\sum q{<}m.\ c\ q)$ (*matrix-sum n F m*)
using *assms*
proof (*induction m*)
case *0*
show *?case* **using** *op-bound-zero* **by** *simp*
next
case (*Suc m*)
have *S*: *matrix-sum n F m* $\in$ *carrier-mat n n* **by** (*rule matrix-sum-dim*) (*use Suc.prems* **in** *auto*)
have *IH*: *op-bound n* $(\sum q{<}m.\ c\ q)$ (*matrix-sum n F m*) **by** (*rule Suc.IH*) (*use Suc.prems* **in** *auto*)
show *?case* **unfolding** *op-bound-def*
proof (*intro allI impI*)
fix v :: *complex Matrix.vec* **assume** v: *dim-vec* $v = n$
have *matrix-sum n F* (*Suc m*) $*_v$ v = $F\ m *_v v$ + *matrix-sum n F m* $*_v$ v
using *add-mult-distrib-mat-vec*[*OF Suc.prems*(*1*)[*of m*] *S carrier-vecI*[*OF v*]] **by** *simp*
hence $\|$*matrix-sum n F* (*Suc m*) $*_v$ $v\| \leq \|F\ m *_v v\| + \|$*matrix-sum n F m*

$*_v$ $v\|$
using *vnorm-add-le*[*of F m* $*_v$ *v matrix-sum n F m* $*_v$ *v*] *Suc.prems*(*1*)[*of m*] *S* **by** *simp*
also have ... $\leq$ *c m* $*$ $\|v\|$ + ($\sum$ *q*<*m*. *c q*) $*$ $\|v\|$
using *op-boundD*[*OF Suc.prems*(*2*)[*of m*] *v*] *op-boundD*[*OF IH v*] **by** *simp*
finally show $\|$*matrix-sum n F* (*Suc m*) $*_v$ *v*$\|$ $\leq$ ($\sum$ *q*<*Suc m*. *c q*) $*$ $\|v\|$ **by** (*simp add*: *algebra-simps*)
qed
qed

lemma *op-bound-pow*:
assumes *A*: *A* $\in$ *carrier-mat n n op-bound n a A* **and** *a*: *0* $\leq$ *a*
shows *op-bound n* (*a* ^ *q*) (*A* ^$_m$ *q*)
proof (*induction q*)
case *0*
show *?case* **using** *A*(*1*) **by** (*simp add*: *op-bound-def one-mult-mat-vec*[*OF carrier-vecI*])
next
case (*Suc q*)
have *op-bound n* (*a* ^ *q* $*$ *a*) (*A* ^$_m$ *q* $*$ *A*) **by** (*rule op-bound-mult*[*OF - Suc A*]) (*use A a* **in** *simp-all*)
thus *?case* **by** (*simp add*: *mult.commute*)
qed

lemma *exp-coeff-op-bound*:
assumes *Y*: *Y* $\in$ *carrier-mat n n op-bound n a Y* **and** *a*: *0* $\leq$ *a*
shows *op-bound n* (*a* ^ *q* / *fact q*) (*exp-coeff Y q*)
proof −
have *cmod* (*complex-of-real* (*1* / *fact q*)) $*$ *a* ^ *q* = *a* ^ *q* / *fact q* **by** (*simp only*: *norm-of-real*) *simp*
thus *?thesis* **using** *op-bound-smult*[*OF pow-carrier-mat*[*OF Y*(*1*)] *op-bound-pow*[*OF Y a*], *of complex-of-real* (*1* / *fact q*) *q*]
unfolding *exp-coeff-def* **by** *simp*
qed

$$\sum_{q \leq p} \frac{a^q \, b^{p-q}}{q! \, (p-q)!} = \frac{(a+b)^p}{p!}.$$

lemma *binomial-exp-coef*:
fixes *a b* :: *'a*::*field-char-0*
shows ($\sum$ *q*$\leq$*p*. *a* ^ *q* $*$ *b* ^ (*p* − *q*) / (*fact q* $*$ *fact* (*p* − *q*))) = (*a* + *b*) ^ *p* / *fact p*
proof −
have (*a* + *b*) ^ *p* = ($\sum$ *q*$\leq$*p*. *of-nat* (*p choose q*) $*$ *a* ^ *q* $*$ *b* ^ (*p* − *q*)) **by** (*rule binomial-ring*)
also have ... = *fact p* $*$ ($\sum$ *q*$\leq$*p*. *a* ^ *q* $*$ *b* ^ (*p* − *q*) / (*fact q* $*$ *fact* (*p* − *q*)))
unfolding *sum-distrib-left* **by** (*intro sum.cong refl*) (*simp add*: *binomial-fact field-simps*)

finally show *?thesis* **by** (*simp add*: *field-simps*)
qed

lemma *op-bound-mono*: *op-bound n c M* $\Longrightarrow$ $c \leq c' \Longrightarrow$ *op-bound n c' M*
unfolding *op-bound-def*
proof (*intro allI impI*)
fix *v* :: *complex Matrix.vec* **assume** *b*: $\forall v.$ *dim-vec v* $= n \longrightarrow \|M *_v v\| \leq c * \|v\|$ **and** *cc*: $c \leq c'$ **and** *v*: *dim-vec v* $= n$
have $\|M *_v v\| \leq c * \|v\|$ **using** *b v* **by** *simp*
also have $\ldots \leq c' * \|v\|$ **by** (*rule mult-right-mono*[*OF cc vnorm-ge0*])
finally show $\|M *_v v\| \leq c' * \|v\|$ **.**
qed

lemma *cauchy-prod-op-bound*:
assumes *f*: *seq-carrier n f* **and** *g*: *seq-carrier n g* **and** *bf*: $\bigwedge q.$ *op-bound n* (*cf q*) (*f q*) **and** *bg*: $\bigwedge q.$ *op-bound n* (*cg q*) (*g q*)
and *cf*: $\bigwedge q.$ $0 \leq$ *cf q*
shows *op-bound n* ($\sum q{\leq}p.$ *cf q* $*$ *cg* (*p* $-$ *q*)) (*cauchy-prod n f g p*)
unfolding *cauchy-prod-def lessThan-Suc-atMost*[*symmetric*]
by (*rule op-bound-matrix-sum*) (*use f g* **in** ‹*simp-all add*: *op-bound-mult*[*OF seq-carrier-in*[*OF f*] *bf seq-carrier-in*[*OF g*] *bg cf*]›)

The total weight $\Lambda = L \sum_{(a,H)\in F} |a|$ of a formula. The p-th coefficient of the formula is bounded by $\Lambda^p/p!$ (*prod-coeff-op-bound*).

definition *total-weight* :: *real* $\Rightarrow$ (*real* $\times$ *complex Matrix.mat*) *list* $\Rightarrow$ *real* **where**
total-weight L F = ($\sum x{\leftarrow}F.$ |*fst x*|) $*$ *L*

lemma *total-weight-nonneg*: $0 \leq L \Longrightarrow 0 \leq$ *total-weight L F*
unfolding *total-weight-def* **by** (*intro mult-nonneg-nonneg sum-list-nonneg*) *auto*

lemma *factor-gen-carrier*: *herm-bounded n L* (*snd x*) $\Longrightarrow$ *factor-gen x* $\in$ *carrier-mat n n*
using *herm-boundedD*(*1*) **by** (*simp add*: *factor-gen-def*)

lemma *factor-gen-op-bound*:
assumes *herm-bounded n L* (*snd x*)
shows *op-bound n* (|*fst x*| $*$ *L*) (*factor-gen x*)
proof $-$
have *op-bound n L* (*snd x*) **using** *herm-boundedD*(*3*)[*OF assms*] **by** (*simp add*: *op-bound-def*)
from *op-bound-smult*[*OF herm-boundedD*(*1*)[*OF assms*] *this*, *of* i $*$ *complex-of-real* (*fst x*)]
show *?thesis* **by** (*simp add*: *factor-gen-def norm-mult*)
qed

lemma *prod-coeff-op-bound*:
assumes *F*: $\forall x \in$ *set F. herm-bounded n L* (*snd x*) **and** *L*: $0 \leq L$
shows *op-bound n* (*total-weight L F* ^ *p* / *fact p*) (*prod-coeff n* (*map factor-gen F*) *p*)

using *F*
proof (*induction F arbitrary*: *p*)
case *Nil*
show *?case* **using** *op-bound-zero* **by** (*cases p*) (*simp-all add*: *total-weight-def seq-one-def op-bound-def one-mult-mat-vec*[*OF carrier-vecI*])
next
case (*Cons x G*)
have *Y*: *factor-gen x* ∈ *carrier-mat n n op-bound n* (|*fst x*| ∗ *L*) (*factor-gen x*)
using *Cons.prems factor-gen-carrier*[*of n L x*] *factor-gen-op-bound*[*of n L x*]
by *auto*
have *G*: ∀ *Z* ∈ *set* (*map factor-gen G*). *Z* ∈ *carrier-mat n n* **using** *Cons.prems factor-gen-carrier* **by** *auto*
have *a0*: *0* ≤ |*fst x*| ∗ *L* **using** *L* **by** *simp*
have *op-bound n* (∑ *q*≤*p*. (|*fst x*| ∗ *L*) ^ *q* / *fact q* ∗ (*total-weight L G* ^ (*p* − *q*) / *fact* (*p* − *q*))) (*cauchy-prod n* (*exp-coeff* (*factor-gen x*)) (*prod-coeff n* (*map factor-gen G*)) *p*)
by (*rule cauchy-prod-op-bound*[*OF exp-coeff-carrier*[*OF Y*(*1*)] *prod-coeff-carrier*[*OF G*] *exp-coeff-op-bound*[*OF Y a0*] *Cons.IH*]) (*use Cons.prems a0* **in** *simp-all*)
moreover have (∑ *q*≤*p*. (|*fst x*| ∗ *L*) ^ *q* / *fact q* ∗ (*total-weight L G* ^ (*p* − *q*) / *fact* (*p* − *q*))) = *total-weight L* (*x* # *G*) ^ *p* / *fact p*
using *binomial-exp-coef*[**where** *a* = |*fst x*| ∗ *L* **and** *b* = *total-weight L G* **and** *p* = *p*] **by** (*simp add*: *total-weight-def algebra-simps*)
ultimately show *?case* **by** *simp*
qed

The operator of a formula, $\prod_{(a,H)\in F} e^{iatH}$; it is unitary.

fun *formula-product* :: *nat* ⇒ *real* ⇒ (*real* × *complex Matrix.mat*) *list* ⇒ *complex Matrix.mat* **where**
formula-product n t [] = 1_m *n*
| *formula-product n t* (*x* # *F*) = *evolution* (*snd x*) (*fst x* ∗ *t*) ∗ *formula-product n t F*

lemma *formula-product-carrier-unitary*:
assumes ∀ *x* ∈ *set F*. *herm-bounded n L* (*snd x*)
shows *formula-product n t F* ∈ *carrier-mat n n* ∧ *Quantum.unitary* (*formula-product n t F*)
using *assms*
proof (*induction F*)
case *Nil*
show *?case* **by** (*simp add*: *id-is-unitary*)
next
case (*Cons x F*)
note *h* = *herm-boundedD*[*of n L snd x*]
have *hx*: *snd x* ∈ *carrier-mat n n hermitian* (*snd x*) **using** *Cons.prems h* **by** *auto*
have *IH*: *formula-product n t F* ∈ *carrier-mat n n Quantum.unitary* (*formula-product n t F*) **using** *Cons* **by** *auto*
have *Quantum.unitary* (*evolution* (*snd x*) (*fst x* ∗ *t*) ∗ *formula-product n t F*)
by (*rule unitary-mult*[*OF evolution-unitary*[*OF hx*] *IH*(*2*) *evolution-carrier*[*OF hx*] *IH*(*1*)])

thus *?case* **using** *mult-carrier-mat*[*OF evolution-carrier*[*OF hx*] *IH*(*1*)] **by** *simp*
qed

lemma *exp-taylor-poly-eval*:
assumes *H*: $H \in$ *carrier-mat n n*
shows *exp-taylor n H* $(a * t)$ *P* = *poly-eval n* (*exp-coeff* (*factor-gen* (a, H))) *P t*
unfolding *exp-taylor-def poly-eval-def factor-gen-def fst-conv snd-conv*
proof (*rule matrix-sum-cong*)
fix *q* **assume** $q <$ *Suc P*
have (i $*$ *complex-of-real* $(a * t)$) $\cdot_m$ *H* = *complex-of-real t* $\cdot_m$ ((i $*$ *complex-of-real a*) $\cdot_m$ *H*)
by (*rule eq-matI*) *auto*
thus *exp-coeff* ((i $*$ *complex-of-real* $(a * t)$) $\cdot_m$ *H*) *q* = *complex-of-real t* $\hat{}$ *q* $\cdot_m$ *exp-coeff* ((i $*$ *complex-of-real a*) $\cdot_m$ *H*) *q*
using *exp-coeff-smult*[*of* (i $*$ *complex-of-real a*) $\cdot_m$ *H n complex-of-real t q*] *H*
by (*simp add*: *seq-scale-def*)
qed

lemma *poly-eval-cong*: $(\bigwedge p.\ p \leq P \Longrightarrow f\ p = g\ p) \Longrightarrow$ *poly-eval n f P t* = *poly-eval n g P t*
unfolding *poly-eval-def* **by** (*rule matrix-sum-cong*) *simp*

Splitting a polynomial at degree P, and the tail bound $\sum_{p<Q} z^{K+p}/(K+p)! \leq z^K e^z/K!$ for $z \geq 0$.

lemma *poly-eval-split*:
assumes *f*: *seq-carrier n f*
shows *poly-eval n f* $(P + Q)$ *t* = *poly-eval n f P t* + *matrix-sum n* $(\lambda p.$ (*complex-of-real t* $\hat{}$ (*Suc P* + *p*)) $\cdot_m$ *f* (*Suc P* + *p*)) *Q*
proof (*induction Q*)
case *0*
show *?case* **using** *poly-eval-carrier*[*OF f*, **where** *P* = *P* **and** *t* = *t*] **by** *simp*
next
case (*Suc Q*)
let *?h* = *matrix-sum n* $(\lambda p.$ (*complex-of-real t* $\hat{}$ (*Suc P* + *p*)) $\cdot_m$ *f* (*Suc P* + *p*)) *Q*
have *c*: (*complex-of-real t* $\hat{}$ (*Suc P* + *Q*)) $\cdot_m$ *f* (*Suc P* + *Q*) $\in$ *carrier-mat n n*
poly-eval n f P t $\in$ *carrier-mat n n*
?h $\in$ *carrier-mat n n* **using** *f poly-eval-carrier*[*OF f*] **by** (*auto intro*!: *matrix-sum-dim*)
have *poly-eval n f* $(P +$ *Suc Q*$)$ *t* = (*complex-of-real t* $\hat{}$ (*Suc P* + *Q*)) $\cdot_m$ *f* (*Suc P* + *Q*) + *poly-eval n f* $(P + Q)$ *t*
by (*simp add*: *poly-eval-def*)
also have ... = (*complex-of-real t* $\hat{}$ (*Suc P* + *Q*)) $\cdot_m$ *f* (*Suc P* + *Q*) + (*poly-eval n f P t* + *?h*) **unfolding** *Suc* **..**
also have ... = ((*complex-of-real t* $\hat{}$ (*Suc P* + *Q*)) $\cdot_m$ *f* (*Suc P* + *Q*) + *poly-eval n f P t*) + *?h*
by (*rule assoc-add-mat*[*OF c*, *symmetric*])
also have (*complex-of-real t* $\hat{}$ (*Suc P* + *Q*)) $\cdot_m$ *f* (*Suc P* + *Q*) + *poly-eval n f P t*

```
    = poly-eval n f P t + (complex-of-real t ^ (Suc P + Q)) ·_m f (Suc P + Q)
   by (rule comm-add-mat[OF c(1,2)])
 also have (poly-eval n f P t + (complex-of-real t ^ (Suc P + Q)) ·_m f (Suc P +
Q)) + ?h
    = poly-eval n f P t + ((complex-of-real t ^ (Suc P + Q)) ·_m f (Suc P + Q)
+ ?h)
   by (rule assoc-add-mat[OF c(2) c(1) c(3)])
 finally show ?case by simp
qed

lemma partial-tail:
 fixes z :: real
 assumes z: 0 ≤ z
 shows (∑ p<Q. z ^ (K + p) / fact (K + p)) ≤ z ^ K * exp z / fact K
proof −
 have (∑ p<Q. z ^ (K + p) / fact (K + p)) ≤ (∑ p<Q. z ^ K / fact K * (z ^
p / fact p))
 proof (rule sum-mono)
  fix p
  have z ^ (K + p) / fact (K + p) ≤ z ^ (K + p) / (fact K * fact p)
   by (rule divide-left-mono) (use z fact-mult-le-fact-add[of K p] in simp-all)
  thus z ^ (K + p) / fact (K + p) ≤ z ^ K / fact K * (z ^ p / fact p) by (simp
add: power-add field-simps)
 qed
 also have ... = z ^ K / fact K * (∑ p<Q. z ^ p / fact p) by (simp add:
sum-distrib-left)
 also have (∑ p<Q. z ^ p / fact p) ≤ exp z
 proof −
  have s: summable (λp. z ^ p / fact p) using summable-exp[of z] by (simp add:
field-simps)
  have (∑ p<Q. z ^ p / fact p) ≤ (∑ p. z ^ p / fact p) by (rule sum-le-suminf[OF
s]) (use z in simp-all)
   also have ... = exp z using sums-unique[OF exp-converges[of z]] by (simp
add: field-simps)
  finally show ?thesis .
 qed
 hence z ^ K / fact K * (∑ p<Q. z ^ p / fact p) ≤ z ^ K / fact K * exp z by
(rule mult-left-mono) (use z in simp)
 finally show ?thesis by simp
qed

lemma poly-eval-op-bound:
 assumes f: seq-carrier n f and b: ⋀p. op-bound n (c p) (f p)
 shows op-bound n (∑ p≤P. |t| ^ p * c p) (poly-eval n f P t)
 unfolding poly-eval-def lessThan-Suc-atMost[symmetric]
proof (rule op-bound-matrix-sum)
 fix p
 show (complex-of-real t ^ p) ·_m f p ∈ carrier-mat n n using f by simp
 have cmod (complex-of-real t ^ p) = |t| ^ p by (simp add: norm-power)
```

thus *op-bound n* (|*t*| ^ *p* ∗ *c p*) ((*complex-of-real t* ^ *p*) ·$_m$ *f p*)
using *op-bound-smult*[*OF seq-carrier-in*[*OF f*] *b*, *of complex-of-real t* ^ *p p*] **by** *simp*
qed

lemma *poly-eval-seq-one*: *poly-eval n* (*seq-one n*) *P t* = *1*$_m$ *n*
proof −
have *poly-eval n* (*seq-one n*) *P t* = (*complex-of-real t* ^ *0*) ·$_m$ *seq-one n 0*
unfolding *poly-eval-def* **by** (*rule matrix-sum-single*) (*auto simp*: *seq-one-def*)
thus *?thesis* **by** (*simp add*: *seq-one-def one-smult-mat*)
qed

A formula is close to its own Taylor polynomial: with $|F|$ factors and weight Λ,

$$\Big\| \prod_{(a,H)\in F} e^{iatH} - \sum_{p\le P} t^p \operatorname{coeff}_p \Big\| \le 2|F| \, \frac{(\Lambda|t|)^{P+1} e^{\Lambda|t|}}{(P+1)!}.$$

theorem *formula-product-approx*:
assumes *F*: ∀ *x* ∈ *set F*. *herm-bounded n L* (*snd x*) **and** *L*: *0* ≤ *L*
shows *formula-product n t F* ≈$_{2 * real\ (length\ F) * (total\text{-}weight\ L\ F * |t|)\ ^\frown\ Suc\ P * exp\ (total\text{-}weight\ L\ F * |t|)}$ *poly-eval n* (*prod-coeff n* (*map factor-gen F*)) *P t*
using *F*
proof (*induction F*)
case *Nil*
show *?case*
proof (*rule op-closeI*[**where** *m* = *n* **and** *n* = *n*])
show *formula-product n t* [] ∈ *carrier-mat n n* **by** *simp*
next
fix *v* :: *complex Matrix.vec* **assume** *v*: *dim-vec v* = *n*
have *formula-product n t* [] ∗$_v$ *v* − *poly-eval n* (*prod-coeff n* (*map factor-gen* [])) *P t* ∗$_v$ *v* = *0*$_v$ *n*
using *v* **by** (*intro eq-vecI*) (*simp-all add*: *poly-eval-seq-one*)
thus ‖*formula-product n t* [] ∗$_v$ *v* − *poly-eval n* (*prod-coeff n* (*map factor-gen* [])) *P t* ∗$_v$ *v*‖
≤ *2* ∗ *real* (*length* []) ∗ (*total-weight L* [] ∗ |*t*|) ^ *Suc P* ∗ *exp* (*total-weight L* [] ∗ |*t*|) / *fact* (*Suc P*) ∗ ‖*v*‖
by (*simp add*: *vnorm-zero-vec*)
qed
next
case (*Cons x G*)
let *?a* = *fst x* **and** *?H* = *snd x* **and** *?Y* = *factor-gen x* **and** *?fG* = *prod-coeff n* (*map factor-gen G*)
let *?x0* = |*fst x*| ∗ *L* **and** *?y0* = *total-weight L G* **and** *?P'* = *Suc P*
let *?z* = (*?x0* + *?y0*) ∗ |*t*|
have *hx*: *herm-bounded n L ?H* **and** *hG*: ∀ *y* ∈ *set G*. *herm-bounded n L* (*snd y*) **using** *Cons.prems* **by** *auto*
have *Hc*: *?H* ∈ *carrier-mat n n hermitian ?H* **using** *herm-boundedD*[*OF hx*] **by** *simp-all*

have G: $\forall Z \in set\ (map\ factor\text{-}gen\ G).\ Z \in carrier\text{-}mat\ n\ n$ **using** hG *factor-gen-carrier* **by** *auto*
have Yc: $?Y \in carrier\text{-}mat\ n\ n$ **and** Yb: *op-bound n ?x0 ?Y* **using** *factor-gen-carrier*[*OF hx*] *factor-gen-op-bound*[*OF hx*] **by** *simp-all*
have $x0$: $0 \leq ?x0$ **and** $y0$: $0 \leq ?y0$ **using** L *total-weight-nonneg* **by** *simp-all*
have fGc: *seq-carrier n ?fG* **by** (*rule prod-coeff-carrier*[*OF G*])
have tYc: *seq-carrier n* (*exp-coeff ?Y*) **by** (*rule exp-coeff-carrier*[*OF Yc*])
define E **where** $E = evolution\ ?H\ (?a * t)$
define T **where** $T = poly\text{-}eval\ n\ (exp\text{-}coeff\ ?Y)\ P\ t$
define SG **where** $SG = poly\text{-}eval\ n\ ?fG\ P\ t$
define W **where** $W = formula\text{-}product\ n\ t\ G$
define C **where** $C = cauchy\text{-}prod\ n\ (truncate\text{-}seq\ n\ P\ (exp\text{-}coeff\ ?Y))\ (truncate\text{-}seq\ n\ P\ ?fG)$
define hi **where** $hi = matrix\text{-}sum\ n\ (\lambda p.\ (complex\text{-}of\text{-}real\ t\ \hat{}\ (Suc\ P + p)) \cdot_m C\ (Suc\ P + p))\ P$
have Cc: *seq-carrier n C* **unfolding** *C-def* **by** (*simp add*: *cauchy-prod-carrier tYc fGc*)
have Ec: $E \in carrier\text{-}mat\ n\ n$ **unfolding** *E-def* **by** (*rule evolution-carrier*[*OF Hc*])
have Tc: $T \in carrier\text{-}mat\ n\ n$ **unfolding** *T-def* **by** (*rule poly-eval-carrier*[*OF tYc*])
have SGc: *SG* $\in$ *carrier-mat n n* **unfolding** *SG-def* **by** (*rule poly-eval-carrier*[*OF fGc*])
have Wcu: $W \in carrier\text{-}mat\ n\ n$ *Quantum.unitary W* **using** *formula-product-carrier-unitary*[*OF hG*] **by** (*simp-all add*: *W-def*)
have hic: *hi* $\in$ *carrier-mat n n* **unfolding** *hi-def* **by** (*rule matrix-sum-dim*) (*simp add*: *Cc*)
have fx: *prod-coeff n* (*map factor-gen* (*x # G*)) = *cauchy-prod n* (*exp-coeff ?Y*) *?fG* **by** *simp*
have *lamx*: *total-weight L* (*x # G*) $*$ $|t|$ = *?z* **by** (*simp add*: *total-weight-def algebra-simps*)
have TE: $T = exp\text{-}taylor\ n\ ?H\ (?a * t)\ P$ **unfolding** *T-def* **using** *exp-taylor-poly-eval*[*OF Hc*(*1*), *of ?a t P*] **by** (*simp add*: *factor-gen-def*)
have TSG: $T * SG = poly\text{-}eval\ n\ (cauchy\text{-}prod\ n\ (exp\text{-}coeff\ ?Y)\ ?fG)\ P\ t + hi$
proof −
have $T * SG = poly\text{-}eval\ n\ C\ (P + P)\ t$ **unfolding** *T-def SG-def C-def* **by** (*rule poly-eval-mult*[*OF tYc fGc*])
also have ... = *poly-eval n C P t* + *hi* **unfolding** *hi-def* **by** (*rule poly-eval-split*[*OF Cc*])
also have *poly-eval n C P t* = *poly-eval n* (*cauchy-prod n* (*exp-coeff ?Y*) *?fG*) *P t*
unfolding *C-def* **by** (*rule poly-eval-cong*, *rule cauchy-prod-congr*) (*auto simp*: *truncate-seq-def*)
finally show *?thesis* **.**
qed
— the three error terms
have rY: $E \approx_{(?x0 * |t|)\ \hat{}\ ?P' * exp\ (?x0 * |t|)\ /\ fact\ ?P'}\ T$
proof −
have e: $|?a * t| * L = ?x0 * |t|$ **by** (*simp add*: *abs-mult mult-ac*)

show *?thesis* **using** *evolution-taylor*[*OF hx L*, *of ?a* ∗ *t P*] **unfolding** *E-def TE e* **.**
qed
have *cT*: *op-bound n* (*exp* (*?x0* ∗ |*t*|)) *T*
proof −
have *op-bound n* (∑ *p*≤*P*. |*t*| ^ *p* ∗ (*?x0* ^ *p* / *fact p*)) *T*
unfolding *T-def* **by** (*rule poly-eval-op-bound*[*OF tYc exp-coeff-op-bound*[*OF Yc Yb x0*]])
moreover have (∑ *p*≤*P*. |*t*| ^ *p* ∗ (*?x0* ^ *p* / *fact p*)) ≤ *exp* (*?x0* ∗ |*t*|)
using *partial-tail*[**where** *z* = *?x0* ∗ |*t*| **and** *Q* = *Suc P* **and** *K* = *0*, *OF mult-nonneg-nonneg*[*OF x0 abs-ge-zero*]]
by (*simp add*: *lessThan-Suc-atMost power-mult-distrib mult-ac*)
ultimately show *?thesis* **by** (*rule op-bound-mono*)
qed
have *Cb*: *op-bound n* ((*?x0* + *?y0*) ^ *p* / *fact p*) (*C p*) **for** *p*
proof −
have *tb*: *op-bound n* (*?x0* ^ *q* / *fact q*) (*truncate-seq n P* (*exp-coeff ?Y*) *q*) **for** *q*
using *exp-coeff-op-bound*[*OF Yc Yb x0*, *of q*] *op-bound-mono*[*OF op-bound-zero*, *of ?x0* ^ *q* / *fact q n*] *x0*
by (*simp add*: *truncate-seq-def*)
have *gb*: *op-bound n* (*?y0* ^ *q* / *fact q*) (*truncate-seq n P ?fG q*) **for** *q*
using *prod-coeff-op-bound*[*OF hG L*, *of q*] *op-bound-mono*[*OF op-bound-zero*, *of ?y0* ^ *q* / *fact q n*] *y0*
by (*simp add*: *truncate-seq-def*)
have *op-bound n* (∑ *q*≤*p*. *?x0* ^ *q* / *fact q* ∗ (*?y0* ^ (*p* − *q*) / *fact* (*p* − *q*))) (*C p*)
unfolding *C-def* **by** (*rule cauchy-prod-op-bound*[*OF truncate-seq-carrier*[*OF tYc*] *truncate-seq-carrier*[*OF fGc*] *tb gb*]) (*use x0* **in** *simp*)
moreover have (∑ *q*≤*p*. *?x0* ^ *q* / *fact q* ∗ (*?y0* ^ (*p* − *q*) / *fact* (*p* − *q*))) = (*?x0* + *?y0*) ^ *p* / *fact p*
using *binomial-exp-coef*[**where** *a* = *?x0* **and** *b* = *?y0* **and** *p* = *p*] **by** *simp*
ultimately show *?thesis* **by** *simp*
qed
have *hb*: *op-bound n* (*?z* ^ *?P'* ∗ *exp ?z* / *fact ?P'*) *hi*
proof −
have *op-bound n* (∑ *p*<*P*. |*t*| ^ (*Suc P* + *p*) ∗ ((*?x0* + *?y0*) ^ (*Suc P* + *p*) / *fact* (*Suc P* + *p*))) *hi*
proof (*unfold hi-def*, *rule op-bound-matrix-sum*)
fix *p*
show (*complex-of-real t* ^ (*Suc P* + *p*)) ·$_m$ *C* (*Suc P* + *p*) ∈ *carrier-mat n n* **using** *Cc* **by** *simp*
have *cmod* (*complex-of-real t* ^ (*Suc P* + *p*)) = |*t*| ^ (*Suc P* + *p*) **by** (*simp only*: *norm-power norm-of-real*)
thus *op-bound n* (|*t*| ^ (*Suc P* + *p*) ∗ ((*?x0* + *?y0*) ^ (*Suc P* + *p*) / *fact* (*Suc P* + *p*))) ((*complex-of-real t* ^ (*Suc P* + *p*)) ·$_m$ *C* (*Suc P* + *p*))
using *op-bound-smult*[*OF seq-carrier-in*[*OF Cc*] *Cb*, *of complex-of-real t* ^ (*Suc P* + *p*) *Suc P* + *p*] **by** *simp*

```
    qed
    moreover have (∑p<P. |t| ^ (Suc P + p) * ((?x0 + ?y0) ^ (Suc P + p) /
fact (Suc P + p)))
        ≤ ?z ^ ?P' * exp ?z / fact ?P'
    using partial-tail[where z = ?z and Q = P and K = Suc P, OF mult-nonneg-nonneg[OF
add-nonneg-nonneg[OF x0 y0] abs-ge-zero]]
      by (simp add: power-mult-distrib mult-ac)
    ultimately show ?thesis by (rule op-bound-mono)
  qed
  let ?rG = 2 * real (length G) * (?y0 * |t|) ^ ?P' * exp (?y0 * |t|) / fact ?P'
  have IH: W ≈?rG SG using Cons.IH[OF hG] by (simp add: W-def SG-def)
  show ?case unfolding lamx
  proof (rule op-closeI[where m = n and n = n])
    show formula-product n t (x # G) ∈ carrier-mat n n
      using formula-product-carrier-unitary[OF Cons.prems] by simp
  next
    fix v :: complex Matrix.vec assume v: dim-vec v = n
    define w where w = W *v v
    have wn: dim-vec w = n and nw: ‖w‖ = ‖v‖ using Wcu v unitary-vnorm[OF
Wcu v] by (simp-all add: w-def)
    have sp: formula-product n t (x # G) *v v = E *v w
    unfolding w-def E-def W-def using assoc-mult-mat-vec[OF evolution-carrier[OF
Hc] - carrier-vecI[OF v]] Wcu(1)
      by (simp add: W-def)
     have pf: poly-eval n (prod-coeff n (map factor-gen (x # G))) P t *v v =
poly-eval n (cauchy-prod n (exp-coeff ?Y) ?fG) P t *v v by simp
    let ?pf = poly-eval n (cauchy-prod n (exp-coeff ?Y) ?fG) P t
   have pfc: ?pf ∈ carrier-mat n n by (rule poly-eval-carrier) (simp add: cauchy-prod-carrier
tYc fGc)
    have e3: T *v (SG *v v) − ?pf *v v = hi *v v
    proof −
      have T *v (SG *v v) = (?pf + hi) *v v using assoc-mult-mat-vec[OF Tc
SGc carrier-vecI[OF v]] TSG by simp
     also have ... = ?pf *v v + hi *v v by (rule add-mult-distrib-mat-vec[OF pfc
hic carrier-vecI[OF v]])
      finally have tsg: T *v (SG *v v) = ?pf *v v + hi *v v .
       have g: dim-vec a = n ⟹ dim-vec b = n ⟹ (a + b) − a = b for a b ::
complex Matrix.vec
        by (intro eq-vecI) auto
      show ?thesis unfolding tsg by (rule g) (use pfc hic v in simp-all)
    qed
    have e2: T *v w − T *v (SG *v v) = T *v (w − SG *v v)
    proof −
      have SG *v v ∈ carrier-vec n by (rule carrier-vecI) (use SGc in simp)
     thus ?thesis by (rule mult-minus-distrib-mat-vec[OF Tc carrier-vecI[OF wn],
symmetric])
    qed
    have b1: ‖E *v w − T *v w‖ ≤ (?x0 * |t|) ^ ?P' * exp (?x0 * |t|) / fact ?P'
* ‖v‖
```

using *op-closeD*[*OF rY Ec wn*] *nw* **by** *simp*
have *b2*: $\|T *_v (w - SG *_v v)\| \leq exp\ (?x0 * |t|) * (?rG * \|v\|)$
proof −
have $\|T *_v (w - SG *_v v)\| \leq exp\ (?x0 * |t|) * \|w - SG *_v v\|$
by (*rule op-boundD*[*OF cT*]) (*use wn SGc* **in** *simp*)
also have $\|w - SG *_v v\| \leq ?rG * \|v\|$ **using** *op-closeD*[*OF IH Wcu*(*1*) *v*]
by (*simp add*: *w-def*)
finally show *?thesis* **by** *simp*
qed
have *b3*: $\|hi *_v v\| \leq ?z \hat{} ?P' * exp\ ?z\ /\ fact\ ?P' * \|v\|$ **by** (*rule op-boundD*[*OF hb v*])
have $\|E *_v w - ?pf *_v v\| \leq \|E *_v w - T *_v w\| + \|T *_v w - ?pf *_v v\|$
by (*rule vnorm-diff-le*[**where** $n = n$]) (*use Ec Tc pfc wn v* **in** *simp-all*)
also have $\|T *_v w - ?pf *_v v\| \leq \|T *_v w - T *_v (SG *_v v)\| + \|T *_v (SG *_v v) - ?pf *_v v\|$
by (*rule vnorm-diff-le*[**where** $n = n$]) (*use Tc SGc pfc wn v* **in** *simp-all*)
finally have *tri*: $\|E *_v w - ?pf *_v v\| \leq (?x0 * |t|) \hat{} ?P' * exp\ (?x0 * |t|)\ /\ fact\ ?P' * \|v\|$
$+ exp\ (?x0 * |t|) * (?rG * \|v\|) + ?z \hat{} ?P' * exp\ ?z\ /\ fact\ ?P' * \|v\|$
using *b1 b2 b3* **unfolding** *e2 e3* **by** *linarith*
have *z0*: $0 \leq ?z\ ?x0 * |t| \leq ?z\ ?y0 * |t| \leq ?z$ **using** *x0 y0* **by** (*simp-all add*: *distrib-right*)
have *k1*: $(?x0 * |t|) \hat{} ?P' * exp\ (?x0 * |t|) \leq ?z \hat{} ?P' * exp\ ?z$
by (*intro mult-mono power-mono*) (*use z0 x0 y0* **in** *simp-all*)
have *k2*: $exp\ (?x0 * |t|) * ((?y0 * |t|) \hat{} ?P' * exp\ (?y0 * |t|)) \leq ?z \hat{} ?P' * exp\ ?z$
proof −
have $exp\ (?x0 * |t|) * exp\ (?y0 * |t|) = exp\ ?z$ **by** (*simp add*: *exp-add*[*symmetric*] *distrib-right*)
moreover have $(?y0 * |t|) \hat{} ?P' \leq ?z \hat{} ?P'$ **by** (*rule power-mono*) (*use z0 y0* **in** *simp-all*)
ultimately show *?thesis* **by** (*metis mult.left-commute mult-right-mono exp-ge-zero*)
qed
have $(?x0 * |t|) \hat{} ?P' * exp\ (?x0 * |t|)\ /\ fact\ ?P' * \|v\|$
$+ exp\ (?x0 * |t|) * (?rG * \|v\|) + ?z \hat{} ?P' * exp\ ?z\ /\ fact\ ?P' * \|v\|$
$\leq 2 * real\ (length\ (x\ \#\ G)) * ?z \hat{} ?P' * exp\ ?z\ /\ fact\ ?P' * \|v\|$
proof −
have $(?x0 * |t|) \hat{} ?P' * exp\ (?x0 * |t|) + exp\ (?x0 * |t|) * (2 * real\ (length\ G) * (?y0 * |t|) \hat{} ?P' * exp\ (?y0 * |t|))$
$+ ?z \hat{} ?P' * exp\ ?z \leq 2 * real\ (length\ (x\ \#\ G)) * (?z \hat{} ?P' * exp\ ?z)$
using *k1 mult-left-mono*[*OF k2*, *of* $2 * real\ (length\ G)$] **by** (*simp add*: *algebra-simps*)
hence $((?x0 * |t|) \hat{} ?P' * exp\ (?x0 * |t|) + exp\ (?x0 * |t|) * (2 * real\ (length\ G) * (?y0 * |t|) \hat{} ?P' * exp\ (?y0 * |t|))$
$+ ?z \hat{} ?P' * exp\ ?z)\ /\ fact\ ?P' * \|v\| \leq 2 * real\ (length\ (x\ \#\ G)) * ((?z \hat{} ?P' * exp\ ?z)\ /\ fact\ ?P' * \|v\|$
by (*intro mult-right-mono divide-right-mono*) (*simp-all add*: *vnorm-ge0*)
thus *?thesis* **by** (*simp add*: *algebra-simps add-divide-distrib*)

qed
with *tri* **show** $\|$*formula-product n t* $(x \# G) *_v v -$ *poly-eval n* (*prod-coeff n* (*map factor-gen* $(x \# G))) P\ t *_v v\|$
$\leq 2 * real\ (length\ (x \# G)) * ?z \hat{}\ ?P' * exp\ ?z\ /\ fact\ ?P' * \|v\|$
unfolding *sp pf* **by** *linarith*
qed
qed

Closeness is transitive, the errors adding up, and monotone in the error.

lemma *op-close-trans*:
assumes $U \approx_a V\ V \approx_b W\ U \in$ *carrier-mat n n* $V \in$ *carrier-mat n n* $W \in$ *carrier-mat n n*
shows $U \approx_{a+b} W$
proof (*rule op-closeI*[*OF assms*(*3*)])
fix $v ::$ *complex Matrix.vec* **assume** *v*: *dim-vec v* $= n$
have $\|U *_v v - W *_v v\| \leq \|U *_v v - V *_v v\| + \|V *_v v - W *_v v\|$
by (*rule vnorm-diff-le*[**where** $n = n$]) (*use assms* **in** *simp-all*)
also have $\ldots \leq a * \|v\| + b * \|v\|$ **using** *op-closeD*[*OF assms*(*1*,*3*) *v*] *op-closeD*[*OF assms*(*2*,*4*) *v*] **by** *simp*
finally show $\|U *_v v - W *_v v\| \leq (a + b) * \|v\|$ **by** (*simp add*: *algebra-simps*)
qed

lemma *op-close-mono*:
assumes $U \approx_a V\ a \leq b$
shows $U \approx_b V$
unfolding *op-close-def*
proof (*intro allI impI*)
fix $v ::$ *complex Matrix.vec* **assume** *dim-vec v* $=$ *dim-col U*
thus $\|U *_v v - V *_v v\| \leq b * \|v\|$
using *assms*(*1*)[*unfolded op-close-def*] *mult-right-mono*[*OF assms*(*2*) *vnorm-ge0*[*of v*]] *order-trans* **by** *blast*
qed

For Suzuki's formula: the sum of m Hamiltonians bounded by L is bounded by mL, the formula has $2 \cdot 5^k m$ factors, its weights add up to m, and $\Lambda \geq mL$.

lemma *hamiltonian-sum-herm-bounded*:
$\forall H \in$ *set Hs. herm-bounded n L H* $\Longrightarrow$ *herm-bounded n* (*real* (*length Hs*) $* L$) (*hamiltonian-sum n Hs*)
unfolding *herm-bounded-def*[*of n real* (*length Hs*) $* L$]
using *hamiltonian-sum-carrier*[*of Hs n*] *hamiltonian-sum-hermitian*[*of Hs n L*] *hamiltonian-sum-bound*[*of Hs n L*] *herm-boundedD*(*1*) **by** *blast*

lemma *length-suzuki*: *length* (*suzuki k Hs*) $= 2 * 5 \hat{}\ k *$ *length Hs*
by (*induction k*) (*simp-all add*: *strang-def suzuki-weights-def Let-def length-concat comp-def*)

lemma *sum-suzuki*: $(\sum x \leftarrow$*suzuki k Hs. fst x*$) =$ *real* (*length Hs*)
proof (*induction k*)
case *0*

show *?case* **by** (*simp add*: *strang-def comp-def sum-list-triv rev-map*)
next
case (*Suc k*)
thus *?case* **by** (*simp add*: *suzuki-weights-def Let-def comp-def sum-list-const-mult*) (*simp add*: *algebra-simps*)
qed

lemma *total-weight-suzuki*: $0 \leq L \Longrightarrow$ *real* (*length Hs*) $*$ *L* $\leq$ *total-weight L* (*suzuki k Hs*)
unfolding *total-weight-def sum-suzuki*[*of k Hs*, *symmetric*]
by (*intro mult-right-mono sum-list-mono*) *simp-all*

One Suzuki step: $\|S_{2k+2}(\delta) - e^{i\delta H}\| \leq (2|F|+1)(\Lambda|\delta|)^{2k+3} e^{\Lambda|\delta|}/(2k+3)!$.

lemma *suzuki-step*:
fixes *k* :: *nat* **and** *d* :: *real*
assumes *Hs*: $\forall$ *H* $\in$ *set Hs*. *herm-bounded n L H* **and** *L*: $0 \leq L$
defines *F* $\equiv$ *suzuki k Hs* **and** *q* $\equiv$ $2 * k + 3$
shows *formula-product n d F* $\approx$(2 $*$ *real* (*length F*) + 1) $*$ (*total-weight L F* $*$ |*d*|) ^ *q* $*$ *exp* (*total-weight L F* $*$ | *evolution* (*hamiltonian-sum n Hs*) *d*
proof −
have *F*: $\forall$ *x* $\in$ *set F*. *herm-bounded n L* (*snd x*) **using** *Hs suzuki-snd* **unfolding** *F-def* **by** *blast*
have *C*: $\forall$ *H* $\in$ *set Hs*. *H* $\in$ *carrier-mat n n* **using** *Hs herm-boundedD*(*1*) **by** *blast*
have *Hb*: *herm-bounded n* (*real* (*length Hs*) $*$ *L*) (*hamiltonian-sum n Hs*) **by** (*rule hamiltonian-sum-herm-bounded*[*OF Hs*])
note *Hh* = *herm-boundedD*[*OF Hb*]
have *mL*: $0 \leq$ *real* (*length Hs*) $*$ *L* **using** *L* **by** *simp*
let *?X* = i $\cdot_m$ *hamiltonian-sum n Hs* **and** *?f* = *prod-coeff n* (*map factor-gen F*)
have *q*: *q* = *Suc* (*2* $*$ *k* + *2*) **by** (*simp add*: *q-def*)
have *pe*: *poly-eval n ?f* (*2* $*$ *k* + *2*) *d* = *exp-taylor n* (*hamiltonian-sum n Hs*) *d* (*2* $*$ *k* + *2*)
proof −
have *poly-eval n ?f* (*2* $*$ *k* + *2*) *d* = *poly-eval n* (*exp-coeff* (*factor-gen* (*1*, *hamiltonian-sum n Hs*))) (*2* $*$ *k* + *2*) *d*
by (*rule poly-eval-cong*) (*use suzuki-order*[*OF C*, *of k*] **in** ‹*simp add*: *F-def factor-gen-def*›)
also have ... = *exp-taylor n* (*hamiltonian-sum n Hs*) *d* (*2* $*$ *k* + *2*) **using** *exp-taylor-poly-eval*[*OF Hh*(*1*), *of 1 d*] **by** *simp*
finally show *?thesis* **.**
qed
have *a*: *formula-product n d F* $\approx$*2* $*$ *real* (*length F*) $*$ (*total-weight L F* $*$ |*d*|) ^ *q* $*$ *exp* (*total-weight L F* $*$ |*d*|) / *exp-taylor n* (*hamiltonian-sum n Hs*) *d* (*2* $*$ *k* + *2*)
using *formula-product-approx*[*OF F L*, *of d 2* $*$ *k* + *2*] **unfolding** *pe q* **.**
have *b0*: *exp-taylor n* (*hamiltonian-sum n Hs*) *d* (*2* $*$ *k* + *2*) $\approx$(|*d*| $*$ (*real* (*length Hs*) $*$ *L*)) ^ *q* $*$ *exp* (|*d*| $*$ (*real* *evolution* (*hamiltonian-sum n Hs*) *d*
unfolding *q* **by** (*rule op-close-sym*[*OF evolution-taylor*[*OF Hb mL*] *evolution-carrier*[*OF Hh*(*1*,*2*)] *exp-taylor-carrier*[*OF Hh*(*1*)]])
have *le*: |*d*| $*$ (*real* (*length Hs*) $*$ *L*) $\leq$ *total-weight L F* $*$ |*d*|

```
    using mult-left-mono[OF total-weight-suzuki[OF L, of Hs k] abs-ge-zero[of d]]
unfolding F-def by (simp add: mult-ac)
 have b: exp-taylor n (hamiltonian-sum n Hs) d (2 * k + 2) ≈(total-weight L F * |d|) ^ q * exp (total-weight L F
evolution (hamiltonian-sum n Hs) d
  proof (rule op-close-mono[OF b0])
    have x0: 0 ≤ |d| * (real (length Hs) * L) using mL by simp
    have (|d| * (real (length Hs) * L)) ^ q * exp (|d| * (real (length Hs) * L))
       ≤ (total-weight L F * |d|) ^ q * exp (total-weight L F * |d|)
      by (rule mult-mono[OF power-mono[OF le x0]]) (use le order-trans[OF x0 le]
in simp-all)
    thus (|d| * (real (length Hs) * L)) ^ q * exp (|d| * (real (length Hs) * L)) /
fact q
       ≤ (total-weight L F * |d|) ^ q * exp (total-weight L F * |d|) / fact q
      by (rule divide-right-mono) simp
  qed
  show ?thesis
    using op-close-trans[OF a b conjunct1[OF formula-product-carrier-unitary[OF
F]] exp-taylor-carrier[OF Hh(1)] evolution-carrier[OF Hh(1,2)]]
    by (simp add: algebra-simps add-divide-distrib)
qed
```

Suzuki's formula of order $2k+2$ over r steps: $\|S_{2k+2}(t/r)^r - e^{itH}\| \leq r(2|F|+1)(\Lambda|t|/r)^{2k+3}e^{\Lambda|t|/r}/(2k+3)!$, with $|F| = 2 \cdot 5^k m$ factors and $\Lambda \geq mL$.

```
theorem suzuki-error:
  fixes k :: nat and t :: real
  assumes Hs: ∀ H ∈ set Hs. herm-bounded n L H and L: 0 ≤ L and r: 0 < r
  defines F ≡ suzuki k Hs and q ≡ 2 * k + 3
 shows formula-product n (t / real r) F ^m r ≈real r * ((2 * real (length F) + 1) * (total-weight L F * |t| / real r
evolution (hamiltonian-sum n Hs) t
proof −
  have Fh: ∀ x ∈ set F. herm-bounded n L (snd x) using Hs suzuki-snd unfolding
F-def by blast
  note Hh = herm-boundedD[OF hamiltonian-sum-herm-bounded[OF Hs]]
  note U = formula-product-carrier-unitary[OF Fh, of t / real r]
  have d: total-weight L F * |t / real r| = total-weight L F * |t| / real r using r
by simp
 have c: formula-product n (t / real r) F ^m r ≈real r * ((2 * real (length F) + 1) * (total-weight L F * |t / real r
evolution (hamiltonian-sum n Hs) (t / real r) ^m r
   by (rule op-close-pow[OF conjunct1[OF U] conjunct2[OF U] evolution-carrier[OF
Hh(1,2)]
          evolution-unitary[OF Hh(1,2)]]) (use suzuki-step[OF Hs L, where k = k
and d = t / real r] in ‹simp add: F-def q-def›)
 have p: evolution (hamiltonian-sum n Hs) (t / real r) ^m r = evolution (hamiltonian-sum
n Hs) t
    using evolution-pow[OF Hh(1,2), of t / real r r] r by simp
  show ?thesis using c unfolding p d .
qed
```

References

[1] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders. Efficient quantum algorithms for simulating sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, 2007.

[2] A. Bordg, H. Lachnitt, and Y. He. Isabelle marries Dirac: a library for quantum computation and quantum information. Archive of Formal Proofs, https://isa-afp.org/entries/Isabelle_Marries_Dirac.html, Nov. 2020.

[3] J. Liu, B. Zhan, S. Wang, S. Ying, T. Liu, Y. Li, M. Ying, and N. Zhan. Quantum Hoare logic. Archive of Formal Proofs, https://isa-afp.org/entries/QHLProver.html, Mar. 2019.

[4] S. Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[5] M. Suzuki. General theory of fractal path integrals with applications to many-body theories and statistical physics. *Journal of Mathematical Physics*, 32(2):400–407, 1991.

[6] R. Thiemann and A. Yamada. Matrices, Jordan normal forms, and spectral radius theory. Archive of Formal Proofs, https://isa-afp.org/entries/Jordan_Normal_Form.html, Aug. 2015.

[7] H. F. Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.