Matrices for ODEs

Jonathan Julián Huerta y Munive

May 14, 2024

Abstract

Our theories formalise various matrix properties that serve to establish existence, uniqueness and characterisation of the solution to affine systems of ordinary differential equations (ODEs). In particular, we formalise the operator and maximum norm of matrices. Then we use them to prove that square matrices form a Banach space, and in this setting, we show an instance of Picard-Lindelöf's theorem for affine systems of ODEs. Finally, we apply this formalisation by verifying three simple hybrid programs.

Contents

1 Introductory Remarks

Affine systems of ordinary differential equations (ODEs) are those whose associated vector fields are linear transformations. That is, if there is a matrix-valued function $A : \mathbb{R} \to M_{n\times n}(\mathbb{R})$ and vector function $B : \mathbb{R} \to \mathbb{R}^n$ such that the system of ODEs $x'\,t = f\,(t, x\,t)$ can be rewritten as $x'\,t = A{\cdot}(x\,t){+}B\,t$, then the system is affine. Similarly, the associated linear system of ODEs is $x'\,t = A \cdot (x\,t)$ for matrix-vector multiplication $\cdot$. Our theories formalise affine (hence linear) systems of ordinary differential equations. For this purpose, we extend the ODE libraries of [6] and linear algebra in HOL-Analysis. We add to them various results about invertibility of matrices, their diagonalisation, their operator and maximum norms, and properties relating them with vectors. We also define a new type of square matrices and prove that this is a Banach space. Then we obtain results about derivatives of matrix-vector multiplication and use them to prove Picard-Lindelöf's theorem as formalised in [3]. The Banach space instance allows us to characterise the general solution to affine systems of ODEs in terms of the matrix-exponential. Finally, we use the components of [3] to do three simple verification examples in the style of differential dynamic logic [7] as showcased in [1, 2, 5]. The paper [4] has a detailed overview of the various contributions that this formalisation adds to the verification components.

2 Mathematical Preliminaries

This section adds useful syntax, abbreviations and theorems to the Isabelle distribution.

theory *MTX-Preliminaries*

imports *Hybrid-Systems-VCs.HS-Preliminaries*

begin

2.1 Syntax

abbreviation e $k \equiv axis\ k\ 1$

syntax
-ivl-integral :: *real* ⇒ *real* ⇒ *'a* ⇒ *pttrn* ⇒ *bool* $((3\int\_^{-}\ (-)\partial/-)\ [0,\ 0,\ 10]\ 10)$

translations
$\int_a^b\ f\ \partial x \rightleftharpoons CONST\ ivl\text{-}integral\ a\ b\ (\lambda x.\ f)$

notation *matrix-inv* $(-^{-1}\ [90])$

abbreviation *entries* $(A::'a^{\wedge}'n^{\wedge}'m) \equiv \{A\ \$\ i\ \$\ j \mid i\ j.\ i \in UNIV \wedge j \in UNIV\}$

2.2 Topology and sets

lemmas *compact-imp-bdd-above* = *compact-imp-bounded*[*THEN bounded-imp-bdd-above*]

lemma *comp-cont-image-spec*: *continuous-on T f* ⟹ *compact T* ⟹ *compact* {*f t* |*t. t* ∈ *T*}
using *compact-continuous-image* **by** (*simp add*: *Setcompr-eq-image*)

lemmas *bdd-above-cont-comp-spec* = *compact-imp-bdd-above*[*OF comp-cont-image-spec*]

lemmas *bdd-above-norm-cont-comp* = *continuous-on-norm*[*THEN bdd-above-cont-comp-spec*]

lemma *open-cballE*: $t_0 \in T \Longrightarrow open\ T \Longrightarrow \exists\, e{>}0.\ cball\ t_0\ e \subseteq T$
using *open-contains-cball* **by** *blast*

lemma *open-ballE*: $t_0 \in T \Longrightarrow open\ T \Longrightarrow \exists\, e{>}0.\ ball\ t_0\ e \subseteq T$
using *open-contains-ball* **by** *blast*

lemma *funcset-UNIV*: $f \in A \to UNIV$
by *auto*

lemma *finite-image-of-finite*[*simp*]:
fixes *f*::*'a*::*finite* ⇒ *'b*
shows *finite* {*x*. ∃ *i*. *x* = *f i*}
using *finite-Atleast-Atmost-nat* **by** *force*

lemma *finite-image-of-finite2*:
fixes *f* :: *'a*::*finite* ⇒ *'b*::*finite* ⇒ *'c*
shows *finite* {*f x y* |*x y. P x y*}
proof−
have *finite* $(\bigcup x.\ \{f\ x\ y | y.\ P\ x\ y\})$
by *simp*

moreover have $\{f\ x\ y | x\ y.\ P\ x\ y\} = (\bigcup x.\ \{f\ x\ y | y.\ P\ x\ y\})$
by *auto*
ultimately show *?thesis*
by *simp*
qed

2.3 Functions

lemma *finite-sum-univ-singleton*: $(sum\ g\ UNIV) = sum\ g\ \{i::'a::finite\} + sum\ g\ (UNIV - \{i\})$
by (*metis add.commute finite-class.finite-UNIV sum.subset-diff top-greatest*)

lemma *suminfI*:
fixes $f :: nat \Rightarrow 'a::\{t2\text{-}space, comm\text{-}monoid\text{-}add\}$
shows $f\ sums\ k \Longrightarrow suminf\ f = k$
unfolding *sums-iff* **by** *simp*

lemma *suminf-eq-sum*:
fixes $f :: nat \Rightarrow ('a::real\text{-}normed\text{-}vector)$
assumes $\bigwedge n.\ n > m \Longrightarrow f\ n = 0$
shows $(\sum n.\ f\ n) = (\sum n \leq m.\ f\ n)$
using *assms* **by** (*meson atMost-iff finite-atMost not-le suminf-finite*)

lemma *suminf-multr*: $summable\ f \Longrightarrow (\sum n.\ f\ n * c) = (\sum n.\ f\ n) * c$ **for** $c::'a::real\text{-}normed\text{-}algebra$
by (*rule bounded-linear.suminf* [*OF bounded-linear-mult-left*, *symmetric*])

lemma *sum-if-then-else-simps*[*simp*]:
fixes $q :: ('a::semiring\text{-}0)$ **and** $i :: 'n::finite$
shows $(\sum j \in UNIV.\ f\ j * (if\ j = i\ then\ q\ else\ 0)) = f\ i * q$
and $(\sum j \in UNIV.\ f\ j * (if\ i = j\ then\ q\ else\ 0)) = f\ i * q$
and $(\sum j \in UNIV.\ (if\ i = j\ then\ q\ else\ 0) * f\ j) = q * f\ i$
and $(\sum j \in UNIV.\ (if\ j = i\ then\ q\ else\ 0) * f\ j) = q * f\ i$
by (*auto simp*: *finite-sum-univ-singleton*[*of - i*])

2.4 Suprema

lemma *le-max-image-of-finite*[*simp*]:
fixes $f::'a::finite \Rightarrow 'b::linorder$
shows $(f\ i) \leq Max\ \{x.\ \exists i.\ x = f\ i\}$
by (*rule Max.coboundedI*, *simp-all*) (*rule-tac x=i* **in** *exI*, *simp*)

lemma *cSup-eq*:
fixes $c::'a::conditionally\text{-}complete\text{-}lattice$
assumes $\forall x \in X.\ x \leq c$ **and** $\exists x \in X.\ c \leq x$
shows $Sup\ X = c$
by (*metis assms cSup-eq-maximum order-class.order.antisym*)

lemma *cSup-mem-eq*:
$c \in X \Longrightarrow \forall x \in X.\ x \leq c \Longrightarrow Sup\ X = c$ **for** $c::'a::conditionally\text{-}complete\text{-}lattice$

by (*rule cSup-eq*, *auto*)

lemma *cSup-finite-ex*:
finite $X \Longrightarrow X \neq \{\} \Longrightarrow \exists\, x{\in}X.\ Sup\ X = x$ **for** $X{::}'a{::}$*conditionally-complete-linorder set*
by (*metis* (*full-types*) *bdd-finite*(*1*) *cSup-upper finite-Sup-less-iff order-less-le*)

lemma *cMax-finite-ex*:
finite $X \Longrightarrow X \neq \{\} \Longrightarrow \exists\, x{\in}X.\ Max\ X = x$ **for** $X{::}'a{::}$*conditionally-complete-linorder set*
apply(*subst cSup-eq-Max*[*symmetric*])
using *cSup-finite-ex* **by** *auto*

lemma *finite-nat-minimal-witness*:
fixes $P :: ('a{::}$*finite*$) \Rightarrow nat \Rightarrow bool$
assumes $\forall\, i.\ \exists\, N{::}nat.\ \forall\, n \geq N.\ P\ i\ n$
shows $\exists\, N.\ \forall\, i.\ \forall\, n \geq N.\ P\ i\ n$
proof−
let *?bound* $i = (LEAST\ N.\ \forall\, n \geq N.\ P\ i\ n)$
let $?N = Max\ \{$*?bound* $i\ |i.\ i \in UNIV\}$
{fix $n{::}nat$ **and** $i{::}'a$
assume $n \geq ?N$
obtain M **where** $\forall\, n \geq M.\ P\ i\ n$
using *assms* **by** *blast*
hence *obs*: $\forall\ m \geq$ *?bound i*. $P\ i\ m$
using *LeastI*[*of* $\lambda N.\ \forall\, n \geq N.\ P\ i\ n$] **by** *blast*
have *finite* {*?bound i* $|i.\ i \in UNIV\}$
by *simp*
hence $?N \geq$ *?bound i*
using *Max-ge* **by** *blast*
hence $n \geq$ *?bound i*
using ‹$n \geq ?N$› **by** *linarith*
hence $P\ i\ n$
using *obs* **by** *blast***}**
thus $\exists\, N.\ \forall\, i\ n.\ N \leq n \longrightarrow P\ i\ n$
by *blast*
qed

2.5 Real numbers

named-theorems *field-power-simps simplification rules for powers to the nth*

declare *semiring-normalization-rules*(*18*) [*field-power-simps*]
and *semiring-normalization-rules*(*26*) [*field-power-simps*]
and *semiring-normalization-rules*(*27*) [*field-power-simps*]
and *semiring-normalization-rules*(*28*) [*field-power-simps*]
and *semiring-normalization-rules*(*29*) [*field-power-simps*]

WARNING: Adding $?x * ?x^{?q} = ?x^{Suc\ ?q}$ to our tactic makes its combination with simp to loop infinitely in some proofs.

lemma *sq-le-cancel*:
shows $(a::real) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a\hat{}2 \leq b * a \Longrightarrow a \leq b$
and $(a::real) \geq 0 \Longrightarrow b \geq 0 \Longrightarrow a\hat{}2 \leq a * b \Longrightarrow a \leq b$
apply (*metis less-eq-real-def mult.commute mult-le-cancel-left semiring-normalization-rules*(*29*))
by (*metis less-eq-real-def mult-le-cancel-left semiring-normalization-rules*(*29*))

lemma *frac-diff-eq1*: $a \neq b \Longrightarrow a \,/\, (a - b) - b \,/\, (a - b) = 1$ **for** *a*::*real*
by (*metis* (*no-types*) *ab-left-minus add.commute add-left-cancel*
diff-divide-distrib diff-minus-eq-add div-self)

lemma *exp-add*: $x * y - y * x = 0 \Longrightarrow exp\ (x + y) = exp\ x * exp\ y$
by (*rule exp-add-commuting*) (*simp add*: *ac-simps*)

lemmas *mult-exp-exp* = *exp-add*[*symmetric*]

2.6 Vectors and matrices

lemma *sum-axis*[*simp*]:
fixes $q :: ('a::semiring\text{-}0)$
shows $(\sum j{\in}UNIV.\ f\ j * axis\ i\ q\ \$\ j) = f\ i * q$
and $(\sum j{\in}UNIV.\ axis\ i\ q\ \$\ j * f\ j) = q * f\ i$
unfolding *axis-def* **by**(*auto simp*: *vec-eq-iff*)

lemma *sum-scalar-nth-axis*: $sum\ (\lambda i.\ (x\ \$\ i) *s\ e\ i)\ UNIV = x$ **for** $x :: ('a::semiring\text{-}1)\hat{}'n$
unfolding *vec-eq-iff axis-def* **by** *simp*

lemma *scalar-eq-scaleR*[*simp*]: $c *s\ x = c *_R\ x$
unfolding *vec-eq-iff* **by** *simp*

lemma *matrix-add-rdistrib*: $((B + C) **\ A) = (B **\ A) + (C **\ A)$
by (*vector matrix-matrix-mult-def sum.distrib*[*symmetric*] *field-simps*)

lemma *vec-mult-inner*: $(A *v\ v) \cdot w = v \cdot (transpose\ A *v\ w)$ **for** $A :: real\ \hat{}'n\hat{}'n$
unfolding *matrix-vector-mult-def transpose-def inner-vec-def*
apply(*simp add*: *sum-distrib-right sum-distrib-left*)
apply(*subst sum.swap*)
apply(*subgoal-tac* $\forall i\ j.\ A\ \$\ i\ \$\ j * v\ \$\ j * w\ \$\ i = v\ \$\ j * (A\ \$\ i\ \$\ j * w\ \$\ i)$)
by *presburger simp*

lemma *uminus-axis-eq*[*simp*]: $-\ axis\ i\ k = axis\ i\ (-k)$ **for** $k :: 'a::ring$
unfolding *axis-def* **by**(*simp add*: *vec-eq-iff*)

lemma *norm-axis-eq*[*simp*]: $\|axis\ i\ k\| = \|k\|$
proof(*simp add*: *axis-def norm-vec-def L2-set-def*)
let $?\delta_K = \lambda i\ j\ k.\ if\ i = j\ then\ k\ else\ 0$
have $(\sum j{\in}UNIV.\ (\|(?\delta_K\ j\ i\ k)\|)^2) = (\sum j{\in}\{i\}.\ (\|(?\delta_K\ j\ i\ k)\|)^2) + (\sum j{\in}(UNIV-\{i\}).$
$(\|(?\delta_K\ j\ i\ k)\|)^2)$
using *finite-sum-univ-singleton* **by** *blast*
also have ... $= (\|k\|)^2$

by *simp*
finally show *sqrt* $(\sum j{\in}UNIV.\ (norm\ (if\ j = i\ then\ k\ else\ 0))^2) = norm\ k$
by *simp*
qed

lemma *matrix-axis-0*:
fixes $A :: ('a{::}idom)^{\wedge\prime}n^{\wedge\prime}m$
assumes $k \neq 0$ **and** $h{:}\forall\, i.\ (A *v (axis\ i\ k)) = 0$
shows $A = 0$
proof−
{**fix** $i{::}'n$
have $0 = (\sum j{\in}UNIV.\ (axis\ i\ k)\ \$\ j *s column\ j\ A)$
using *h matrix-mult-sum*[*of A axis i k*] **by** *simp*
also have $... = k *s column\ i\ A$
by (*simp add*: *axis-def vector-scalar-mult-def column-def vec-eq-iff mult.commute*)
finally have $k *s column\ i\ A = 0$
unfolding *axis-def* **by** *simp*
hence $column\ i\ A = 0$
using *vector-mul-eq-0* ‹$k \neq 0$› **by** *blast*}
thus $A = 0$
unfolding *column-def vec-eq-iff* **by** *simp*
qed

lemma *scaleR-norm-sgn-eq*: $(\|x\|) *_R sgn\ x = x$
by (*metis divideR-right norm-eq-zero scale-eq-0-iff sgn-div-norm*)

lemma *vector-scaleR-commute*: $A *v c *_R x = c *_R (A *v x)$ **for** $x :: ('a{::}real\text{-}normed\text{-}algebra\text{-}1)^{\wedge\prime}n$
unfolding *scaleR-vec-def matrix-vector-mult-def* **by**(*auto simp*: *vec-eq-iff scaleR-right.sum*)

lemma *scaleR-vector-assoc*: $c *_R (A *v x) = (c *_R A) *v x$ **for** $x :: ('a{::}real\text{-}normed\text{-}algebra\text{-}1)^{\wedge\prime}n$
unfolding *matrix-vector-mult-def* **by**(*auto simp*: *vec-eq-iff scaleR-right.sum*)

lemma *mult-norm-matrix-sgn-eq*:
fixes $x :: ('a{::}real\text{-}normed\text{-}algebra\text{-}1)^{\wedge\prime}n$
shows $(\|A *v sgn\ x\|) * (\|x\|) = \|A *v x\|$
proof−
have $\|A *v x\| = \|A *v ((\|x\|) *_R sgn\ x)\|$
by(*simp add*: *scaleR-norm-sgn-eq*)
also have $... = (\|A *v sgn\ x\|) * (\|x\|)$
by(*simp add*: *vector-scaleR-commute*)
finally show *?thesis* **..**
qed

2.7 Diagonalization

lemma *invertibleI*: $A ** B = mat\ 1 \Longrightarrow B ** A = mat\ 1 \Longrightarrow invertible\ A$
unfolding *invertible-def* **by** *auto*

lemma *invertibleD*[*simp*]:

assumes *invertible A*
shows A^{-1} ** A = *mat 1* **and** A ** A^{-1} = *mat 1*
using *assms* **unfolding** *matrix-inv-def invertible-def*
by (*simp-all add*: *verit-sko-ex'*)

lemma *matrix-inv-unique*:
assumes A ** B = *mat 1* **and** B ** A = *mat 1*
shows A^{-1} = B
by (*metis assms invertibleD*(*2*) *invertibleI matrix-mul-assoc matrix-mul-lid*)

lemma *invertible-matrix-inv*: *invertible A* ⟹ *invertible* (A^{-1})
using *invertibleD invertibleI* **by** *blast*

lemma *matrix-inv-idempotent*[*simp*]: *invertible A* ⟹ A^{-1-1} = A
using *invertibleD matrix-inv-unique* **by** *blast*

lemma *matrix-inv-matrix-mul*:
assumes *invertible A* **and** *invertible B*
shows (A ** B)$^{-1}$ = B^{-1} ** A^{-1}
proof(*rule matrix-inv-unique*)
have A ** B ** (B^{-1} ** A^{-1}) = A ** (B ** B^{-1}) ** A^{-1}
by (*simp add*: *matrix-mul-assoc*)
also have ... = *mat 1*
using *assms* **by** *simp*
finally show A ** B ** (B^{-1} ** A^{-1}) = *mat 1* **.**
next
have B^{-1} ** A^{-1} ** (A ** B) = B^{-1} ** (A^{-1} ** A) ** B
by (*simp add*: *matrix-mul-assoc*)
also have ... = *mat 1*
using *assms* **by** *simp*
finally show B^{-1} ** A^{-1} ** (A ** B) = *mat 1* **.**
qed

lemma *mat-inverse-simps*[*simp*]:
fixes *c* :: *'a*::*division-ring*
assumes *c* ≠ *0*
shows *mat* (*inverse c*) ** *mat c* = *mat 1*
and *mat c* ** *mat* (*inverse c*) = *mat 1*
unfolding *matrix-matrix-mult-def mat-def* **by** (*auto simp*: *vec-eq-iff assms*)

lemma *matrix-inv-mat*[*simp*]: *c* ≠ *0* ⟹ (*mat c*)$^{-1}$ = *mat* (*inverse c*) **for** *c* :: *'a*::*division-ring*
by (*simp add*: *matrix-inv-unique*)

lemma *invertible-mat*[*simp*]: *c* ≠ *0* ⟹ *invertible* (*mat c*) **for** *c* :: *'a*::*division-ring*
using *invertibleI mat-inverse-simps*(*1*) *mat-inverse-simps*(*2*) **by** *blast*

lemma *matrix-inv-mat-1*: (*mat* (*1*::*'a*::*division-ring*))$^{-1}$ = *mat 1*
by *simp*

lemma *invertible-mat-1*: *invertible* (*mat* (*1*::*'a*::*division-ring*))
by *simp*

definition *similar-matrix* :: (*'a*::*semiring-1*)^*'m*^*'m* ⇒ (*'a*::*semiring-1*)^*'n*^*'n* ⇒ *bool* (**infixr** ∼ *25*)
where *similar-matrix A B* ⟷ (∃ *P*. *invertible P* ∧ *A* = *P*$^{-1}$ ** *B* ** *P*)

lemma *similar-matrix-refl*[*simp*]: *A* ∼ *A* **for** *A* :: *'a*::*division-ring*^*'n*^*'n*
by (*unfold similar-matrix-def*, *rule-tac x*=*mat 1* **in** *exI*, *simp*)

lemma *similar-matrix-simm*: *A* ∼ *B* ⟹ *B* ∼ *A* **for** *A B* :: (*'a*::*semiring-1*)^*'n*^*'n*
apply(*unfold similar-matrix-def*, *clarsimp*)
apply(*rule-tac x*=*P*$^{-1}$ **in** *exI*, *simp add*: *invertible-matrix-inv*)
by (*metis invertible-def matrix-inv-unique matrix-mul-assoc matrix-mul-lid matrix-mul-rid*)

lemma *similar-matrix-trans*: *A* ∼ *B* ⟹ *B* ∼ *C* ⟹ *A* ∼ *C* **for** *A B C* :: (*'a*::*semiring-1*)^*'n*^*'n*
proof(*unfold similar-matrix-def*, *clarsimp*)
fix *P Q*
assume *A* = *P*$^{-1}$ ** (*Q*$^{-1}$ ** *C* ** *Q*) ** *P* **and** *B* = *Q*$^{-1}$ ** *C* ** *Q*
let *?R* = *Q* ** *P*
assume *inverts*: *invertible Q invertible P*
hence *?R*$^{-1}$ = *P*$^{-1}$ ** *Q*$^{-1}$
by (*rule matrix-inv-matrix-mul*)
also have *invertible ?R*
using *inverts invertible-mult* **by** *blast*
ultimately show ∃ *R*. *invertible R* ∧ *P*$^{-1}$ ** (*Q*$^{-1}$ ** *C* ** *Q*) ** *P* = *R*$^{-1}$ ** *C* ** *R*
by (*metis matrix-mul-assoc*)
qed

lemma *mat-vec-nth-simps*[*simp*]:
i = *j* ⟹ *mat c* $ *i* $ *j* = *c*
i ≠ *j* ⟹ *mat c* $ *i* $ *j* = *0*
by (*simp-all add*: *mat-def*)

definition *diag-mat f* = (χ *i j*. *if i* = *j then f i else 0*)

lemma *diag-mat-vec-nth-simps*[*simp*]:
i = *j* ⟹ *diag-mat f* $ *i* $ *j* = *f i*
i ≠ *j* ⟹ *diag-mat f* $ *i* $ *j* = *0*
unfolding *diag-mat-def* **by** *simp-all*

lemma *diag-mat-const-eq*[*simp*]: *diag-mat* (λ*i*. *c*) = *mat c*
unfolding *mat-def diag-mat-def* **by** *simp*

lemma *matrix-vector-mul-diag-mat*: *diag-mat f* ∗*v s* = (χ *i*. *f i* ∗ *s*$*i*)

unfolding *diag-mat-def matrix-vector-mult-def* **by** *simp*

lemma *matrix-vector-mul-diag-axis*[*simp*]: *diag-mat f* $*v$ (*axis i k*) = *axis i* (*f i* $*$ *k*)
by (*simp add*: *matrix-vector-mul-diag-mat axis-def fun-eq-iff*)

lemma *matrix-mul-diag-matl*: *diag-mat f* $**$ *A* = (χ *i j*. *f i* $*$ *A*$*i*$*j*)
unfolding *diag-mat-def matrix-matrix-mult-def* **by** *simp*

lemma *matrix-matrix-mul-diag-matr*: *A* $**$ *diag-mat f* = (χ *i j*. *A*$*i*$*j* $*$ *f j*)
unfolding *diag-mat-def matrix-matrix-mult-def* **apply**(*clarsimp simp*: *fun-eq-iff*)
subgoal for *i j*
by (*auto simp*: *finite-sum-univ-singleton*[*of - j*])
done

lemma *matrix-mul-diag-diag*: *diag-mat f* $**$ *diag-mat g* = *diag-mat* (λ*i*. *f i* $*$ *g i*)
unfolding *diag-mat-def matrix-matrix-mult-def vec-eq-iff* **by** *simp*

lemma *compow-matrix-mul-diag-mat-eq*: (($**$) (*diag-mat f*) ^^ *n*) (*mat 1*) = *diag-mat* (λ*i*. *f i*^*n*)
apply(*induct n*, *simp-all add*: *matrix-mul-diag-matl*)
by (*auto simp*: *vec-eq-iff diag-mat-def*)

lemma *compow-similar-diag-mat-eq*:
assumes *invertible P*
and *A* = P^{-1} $**$ (*diag-mat f*) $**$ *P*
shows (($**$) *A* ^^ *n*) (*mat 1*) = P^{-1} $**$ (*diag-mat* (λ*i*. *f i*^*n*)) $**$ *P*
proof(*induct n*, *simp-all add*: *assms*)
fix *n*::*nat*
have P^{-1} $**$ *diag-mat f* $**$ *P* $**$ (P^{-1} $**$ *diag-mat* (λ*i*. *f i* ^ *n*) $**$ *P*) =
P^{-1} $**$ *diag-mat f* $**$ *diag-mat* (λ*i*. *f i* ^ *n*) $**$ *P* (**is** *?lhs* = -)
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *invertibleD*(*2*) *matrix-mul-rid matrix-mul-assoc*)

also have ... = P^{-1} $**$ *diag-mat* (λ*i*. *f i* $*$ *f i* ^ *n*) $**$ *P* (**is** - = *?rhs*)
by (*metis* (*full-types*) *matrix-mul-assoc matrix-mul-diag-diag*)
finally show *?lhs* = *?rhs* **.**
qed

lemma *compow-similar-diag-mat*:
assumes *A* $\sim$ (*diag-mat f*)
shows (($**$) *A* ^^ *n*) (*mat 1*) $\sim$ *diag-mat* (λ*i*. *f i*^*n*)
proof(*unfold similar-matrix-def*)
obtain *P* **where** *invertible P* **and** *A* = P^{-1} $**$ (*diag-mat f*) $**$ *P*
using *assms* **unfolding** *similar-matrix-def* **by** *blast*
thus $\exists$ *P*. *invertible P* $\wedge$ (($**$) *A* ^^ *n*) (*mat 1*) = P^{-1} $**$ *diag-mat* (λ*i*. *f i* ^ *n*) $**$ *P*
using *compow-similar-diag-mat-eq* **by** *blast*
qed

no-notation *matrix-inv* ($-^{-1}$ [*90*])
and *similar-matrix* (**infixr** $\sim$ *25*)

end

3 Matrix norms

Here, we explore some properties about the operator and the maximum norms for matrices.

theory *MTX-Norms*
imports *MTX-Preliminaries*

begin

3.1 Matrix operator norm

abbreviation *op-norm* :: ('*a*::*real-normed-algebra-1*)$^{\frown}$'*n*$^{\frown}$'*m* $\Rightarrow$ *real* ((*1*$\|$-$\|_{op}$) [*65*] *61*)
where $\|A\|_{op} \equiv$ *onorm* ($\lambda x.$ *A* $*v$ *x*)

lemma *norm-matrix-bound*:
fixes *A* :: ('*a*::*real-normed-algebra-1*)$^{\frown}$'*n*$^{\frown}$'*m*
shows $\|x\| = 1 \Longrightarrow \|A *v x\| \leq \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v 1\|$
proof−
fix *x* :: ('*a*, '*n*) *vec* **assume** $\|x\| = 1$
hence *xi-le1*:$\bigwedge i.\ \|x\ \$\ i\| \leq 1$
by (*metis Finite-Cartesian-Product.norm-nth-le*)
{fix *j*::'*m*
have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \leq (\sum i{\in}UNIV.\ \|A\ \$\ j\ \$\ i * x\ \$\ i\|)$
using *norm-sum* **by** *blast*
also have ... $\leq (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * (\|x\ \$\ i\|))$
by (*simp add*: *norm-mult-ineq sum-mono*)
also have ... $\leq (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$
using *xi-le1* **by** (*simp add*: *sum-mono mult-left-le*)
finally have $\|(\sum i{\in}UNIV.\ A\ \$\ j\ \$\ i * x\ \$\ i)\| \leq (\sum i{\in}UNIV.\ (\|A\ \$\ j\ \$\ i\|) * 1)$ **by** *simp***}**
hence $\bigwedge j.\ \|(A *v x)\ \$\ j\| \leq ((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v 1)\ \$\ j$
unfolding *matrix-vector-mult-def* **by** *simp*
hence $(\sum j{\in}UNIV.\ (\|(A *v x)\ \$\ j\|)^2) \leq (\sum j{\in}UNIV.\ (\|((\chi\ i1\ i2.\ \|A\ \$\ i1\ \$\ i2\|) *v 1)\ \$\ j\|)^2)$
by (*metis* (*mono-tags*, *lifting*) *norm-ge-zero power2-abs power-mono real-norm-def sum-mono*)
thus $\|A *v x\| \leq \|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|) *v 1\|$
unfolding *norm-vec-def L2-set-def* **by** *simp*
qed

lemma *onorm-set-proptys*:

fixes $A :: ('a::real\text{-}normed\text{-}algebra\text{-}1)^{\wedge}n^{\wedge}m$
shows *bounded* (*range* ($\lambda x.$ ($\|A *v x\|$) / ($\|x\|$)))
and *bdd-above* (*range* ($\lambda x.$ ($\|A *v x\|$) / ($\|x\|$)))
and (*range* ($\lambda x.$ ($\|A *v x\|$) / ($\|x\|$))) $\neq$ {}
unfolding *bounded-def bdd-above-def image-def dist-real-def*
apply(*rule-tac x=0* **in** *exI*)
by (*rule-tac x*=$\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|)\ *v\ 1\|$ **in** *exI*, *clarsimp*,
subst mult-norm-matrix-sgn-eq[*symmetric*], *clarsimp*,
rule-tac x=sgn - **in** *norm-matrix-bound*, *simp add*: *norm-sgn*)+ *force*

lemma *op-norm-set-proptys*:
fixes $A :: ('a::real\text{-}normed\text{-}algebra\text{-}1)^{\wedge}n^{\wedge}m$
shows *bounded* {$\|A *v x\|$ | $x.\ \|x\| = 1$}
and *bdd-above* {$\|A *v x\|$ | $x.\ \|x\| = 1$}
and {$\|A *v x\|$ | $x.\ \|x\| = 1$} $\neq$ {}
unfolding *bounded-def bdd-above-def* **apply** *safe*
apply(*rule-tac x=0* **in** *exI*, *rule-tac x*=$\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|)\ *v\ 1\|$ **in** *exI*)
apply(*force simp*: *norm-matrix-bound dist-real-def*)
apply(*rule-tac x*=$\|(\chi\ i\ j.\ \|A\ \$\ i\ \$\ j\|)\ *v\ 1\|$ **in** *exI*, *force simp*: *norm-matrix-bound*)
using *ex-norm-eq-1* **by** *blast*

lemma *op-norm-def*: $\|A\|_{op} = Sup$ {$\|A *v x\|$ | $x.\ \|x\| = 1$}
apply(*rule antisym*[*OF onorm-le cSup-least*[*OF op-norm-set-proptys*(*3*)]])
apply(*case-tac x = 0*, *simp*)
apply(*subst mult-norm-matrix-sgn-eq*[*symmetric*], *simp*)
apply(*rule cSup-upper*[*OF - op-norm-set-proptys*(*2*)])
apply(*force simp*: *norm-sgn*)
unfolding *onorm-def*
apply(*rule cSup-upper*[*OF - onorm-set-proptys*(*2*)])
by (*simp add*: *image-def*, *clarsimp*) (*metis div-by-1*)

lemma *norm-matrix-le-op-norm*: $\|x\| = 1 \Longrightarrow \|A *v x\| \leq \|A\|_{op}$
apply(*unfold onorm-def*, *rule cSup-upper*[*OF - onorm-set-proptys*(*2*)])
unfolding *image-def* **by** (*clarsimp*, *rule-tac x=x* **in** *exI*) *simp*

lemma *op-norm-ge-0*: $0 \leq \|A\|_{op}$
using *ex-norm-eq-1 norm-ge-zero norm-matrix-le-op-norm basic-trans-rules*(*23*)
by *blast*

lemma *norm-sgn-le-op-norm*: $\|A *v sgn\ x\| \leq \|A\|_{op}$
by (*cases x=0*, *simp-all add*: *norm-sgn norm-matrix-le-op-norm op-norm-ge-0*)

lemma *norm-matrix-le-mult-op-norm*: $\|A *v x\| \leq (\|A\|_{op}) * (\|x\|)$
proof−
have $\|A *v x\| = (\|A *v sgn\ x\|) * (\|x\|)$
by(*simp add*: *mult-norm-matrix-sgn-eq*)
also have ... $\leq (\|A\|_{op}) * (\|x\|)$
using *norm-sgn-le-op-norm*[*of A*] **by** (*simp add*: *mult-mono'*)
finally show *?thesis* **by** *simp*

qed

lemma *blin-matrix-vector-mult*: *bounded-linear* ((∗v) *A*) **for** *A* :: (*'a*::*real-normed-algebra-1*)^*'n*^*'m*
by (*unfold-locales*) (*auto intro*: *norm-matrix-le-mult-op-norm simp*:
mult.commute matrix-vector-right-distrib vector-scaleR-commute)

lemma *op-norm-eq-0*: (‖*A*‖$_{op}$ = *0*) = (*A* = *0*) **for** *A* :: (*'a*::*real-normed-field*)^*'n*^*'m*
unfolding *onorm-eq-0*[*OF blin-matrix-vector-mult*] **using** *matrix-axis-0*[*of 1 A*]
by *fastforce*

lemma *op-norm0*: ‖(*0*::(*'a*::*real-normed-field*)^*'n*^*'m*)‖$_{op}$ = *0*
using *op-norm-eq-0*[*of 0*] **by** *simp*

lemma *op-norm-triangle*: ‖*A* + *B*‖$_{op}$ ≤ (‖*A*‖$_{op}$) + (‖*B*‖$_{op}$)
using *onorm-triangle*[*OF blin-matrix-vector-mult*[*of A*] *blin-matrix-vector-mult*[*of B*]]
matrix-vector-mult-add-rdistrib[*symmetric*, *of A - B*] **by** *simp*

lemma *op-norm-scaleR*: ‖*c* ∗$_R$ *A*‖$_{op}$ = |*c*| ∗ (‖*A*‖$_{op}$)
unfolding *onorm-scaleR*[*OF blin-matrix-vector-mult*, *symmetric*] *scaleR-vector-assoc*
..

lemma *op-norm-matrix-matrix-mult-le*: ‖*A* ∗∗ *B*‖$_{op}$ ≤ (‖*A*‖$_{op}$) ∗ (‖*B*‖$_{op}$)
proof(*rule onorm-le*)
have *0* ≤ (‖*A*‖$_{op}$)
by(*rule onorm-pos-le*[*OF blin-matrix-vector-mult*])
fix *x* **have** ‖*A* ∗∗ *B* ∗*v x*‖ = ‖*A* ∗*v* (*B* ∗*v x*)‖
by (*simp add*: *matrix-vector-mul-assoc*)
also have ... ≤ (‖*A*‖$_{op}$) ∗ (‖*B* ∗*v x*‖)
by (*simp add*: *norm-matrix-le-mult-op-norm*[*of - B* ∗*v x*])
also have ... ≤ (‖*A*‖$_{op}$) ∗ ((‖*B*‖$_{op}$) ∗ (‖*x*‖))
using *norm-matrix-le-mult-op-norm*[*of B x*] ‹*0* ≤ (‖*A*‖$_{op}$)› *mult-left-mono* **by**
blast
finally show ‖*A* ∗∗ *B* ∗*v x*‖ ≤ (‖*A*‖$_{op}$) ∗ (‖*B*‖$_{op}$) ∗ (‖*x*‖)
by *simp*
qed

lemma *norm-matrix-vec-mult-le-transpose*:
‖*x*‖ = *1* ⟹ (‖*A* ∗*v x*‖) ≤ *sqrt* (‖*transpose A* ∗∗ *A*‖$_{op}$) ∗ (‖*x*‖) **for** *A* :: *real*^*'n*^*'n*
proof−
assume ‖*x*‖ = *1*
have (‖*A* ∗*v x*‖)^2 = (*A* ∗*v x*) · (*A* ∗*v x*)
using *dot-square-norm*[*of* (*A* ∗*v x*)] **by** *simp*
also have ... = *x* · (*transpose A* ∗*v* (*A* ∗*v x*))
using *vec-mult-inner* **by** *blast*
also have ... ≤ (‖*x*‖) ∗ (‖*transpose A* ∗*v* (*A* ∗*v x*)‖)
using *norm-cauchy-schwarz* **by** *blast*
also have ... ≤ (‖*transpose A* ∗∗ *A*‖$_{op}$) ∗ (‖*x*‖)^*2*
apply(*subst matrix-vector-mul-assoc*)

using *norm-matrix-le-mult-op-norm*[*of transpose A ** A x*]
by (*simp add*: ‹$\|x\| = 1$›)
finally have $((\|A *v x\|))^2 \leq (\|transpose\ A ** A\|_{op}) * (\|x\|)^2$
by *linarith*
thus $(\|A *v x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
by (*simp add*: ‹$\|x\| = 1$› *real-le-rsqrt*)
qed

lemma *op-norm-le-sum-column*: $\|A\|_{op} \leq (\sum i{\in}UNIV.\ \|column\ i\ A\|)$ **for** $A :: real^{\wedge}{}'n^{\wedge}{}'m$
proof(*unfold op-norm-def*, *rule cSup-least*[*OF op-norm-set-proptys*(*3*)], *clarsimp*)
fix $x :: real^{\wedge}{}'n$ **assume** *x-def*:$\|x\| = 1$
hence *x-hyp*:$\bigwedge i.\ \|x\ \$\ i\| \leq 1$
by (*simp add*: *norm-bound-component-le-cart*)
have $(\|A *v x\|) = \|(\sum i{\in}UNIV.\ x\ \$\ i *s column\ i\ A)\|$
by(*subst matrix-mult-sum*[*of A*], *simp*)
also have ... $\leq (\sum i{\in}UNIV.\ \|x\ \$\ i *s column\ i\ A\|)$
by (*simp add*: *sum-norm-le*)
also have ... $= (\sum i{\in}UNIV.\ (\|x\ \$\ i\|) * (\|column\ i\ A\|))$
by (*simp add*: *mult-norm-matrix-sgn-eq*)
also have ... $\leq (\sum i{\in}UNIV.\ \|column\ i\ A\|)$
using *x-hyp* **by** (*simp add*: *mult-left-le-one-le sum-mono*)
finally show $\|A *v x\| \leq (\sum i{\in}UNIV.\ \|column\ i\ A\|)$ **.**
qed

lemma *op-norm-le-transpose*: $\|A\|_{op} \leq \|transpose\ A\|_{op}$ **for** $A :: real^{\wedge}{}'n^{\wedge}{}'n$
proof−
have *obs*:$\forall x.\ \|x\| = 1 \longrightarrow (\|A *v x\|) \leq sqrt\ ((\|transpose\ A ** A\|_{op})) * (\|x\|)$
using *norm-matrix-vec-mult-le-transpose* **by** *blast*
have $(\|A\|_{op}) \leq sqrt\ ((\|transpose\ A ** A\|_{op}))$
using *obs* **apply**(*unfold op-norm-def*)
by (*rule cSup-least*[*OF op-norm-set-proptys*(*3*)]) *clarsimp*
hence $((\|A\|_{op}))^2 \leq (\|transpose\ A ** A\|_{op})$
using *power-mono*[*of* $(\|A\|_{op})$ - *2*] *op-norm-ge-0*
by (*metis not-le real-less-lsqrt*)
also have ... $\leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$
using *op-norm-matrix-matrix-mult-le* **by** *blast*
finally have $((\|A\|_{op}))^2 \leq (\|transpose\ A\|_{op}) * (\|A\|_{op})$
by *linarith*
thus $(\|A\|_{op}) \leq (\|transpose\ A\|_{op})$
using *sq-le-cancel*[*of* $(\|A\|_{op})$] *op-norm-ge-0* **by** *metis*
qed

3.2 Matrix maximum norm

abbreviation *max-norm* :: $real^{\wedge}{}'n^{\wedge}{}'m \Rightarrow real$ ((*1*‖-‖$_{max}$) [*65*] *61*)
where $\|A\|_{max} \equiv Max\ (abs\ `\ (entries\ A))$

lemma *max-norm-def*: $\|A\|_{max} = Max\ \{|A\ \$\ i\ \$\ j||i\ j.\ i{\in}UNIV \wedge j{\in}UNIV\}$

by (*simp add*: *image-def*, *rule arg-cong*[*of* - - *Max*], *blast*)

lemma *max-norm-set-proptys*: *finite* {|*A* $ *i* $ *j*| |*i j*. *i* ∈ *UNIV* ∧ *j* ∈ *UNIV*} (**is** *finite ?X*)
proof−
have ⋀*i*. *finite* {|*A* $ *i* $ *j*| | *j*. *j* ∈ *UNIV*}
using *finite-Atleast-Atmost-nat* **by** *fastforce*
hence *finite* (⋃ *i*∈*UNIV*. {|*A* $ *i* $ *j*| | *j*. *j* ∈ *UNIV*}) (**is** *finite ?Y*)
using *finite-class.finite-UNIV* **by** *blast*
also have *?X* ⊆ *?Y*
by *auto*
ultimately show *?thesis*
using *finite-subset* **by** *blast*
qed

lemma *max-norm-ge-0*: *0* ≤ $\|A\|_{max}$
unfolding *max-norm-def*
apply(*rule order.trans*[*OF abs-ge-zero*[*of A* $ - $ -] *Max-ge*])
using *max-norm-set-proptys* **by** *auto*

lemma *op-norm-le-max-norm*:
fixes *A* :: *real*^('*n*::*finite*)^('*m*::*finite*)
shows $\|A\|_{op}$ ≤ *real CARD*('*m*) ∗ *real CARD*('*n*) ∗ ($\|A\|_{max}$)
apply(*rule onorm-le-matrix-component*)
unfolding *max-norm-def* **by**(*rule Max-ge*[*OF max-norm-set-proptys*]) *force*

lemma *sqrt-Sup-power2-eq-Sup-abs*:
finite A ⟹ *A* ≠ {} ⟹ *sqrt* (*Sup* {(*f i*)^2 |*i*. *i* ∈ *A*}) = *Sup* {|*f i*| |*i*. *i* ∈ *A*}
proof(*rule sym*)
assume *assms*: *finite A A* ≠ {}
then obtain *i* **where** *i-def*: *i* ∈ *A* ∧ *Sup* {(*f i*)^2|*i*. *i* ∈ *A*} = (*f i*)^2
using *cSup-finite-ex*[*of* {(*f i*)^2|*i*. *i* ∈ *A*}] **by** *auto*
hence *lhs*: *sqrt* (*Sup* {(*f i*)^2 |*i*. *i* ∈ *A*}) = |*f i*|
by *simp*
have *finite* {(*f i*)^2|*i*. *i* ∈ *A*}
using *assms* **by** *simp*
hence ∀ *j*∈*A*. (*f j*)^2 ≤ (*f i*)^2
using *i-def cSup-upper*[*of* - {(*f i*)^2 |*i*. *i* ∈ *A*}] **by** *force*
hence ∀ *j*∈*A*. |*f j*| ≤ |*f i*|
using *abs-le-square-iff* **by** *blast*
also have |*f i*| ∈ {|*f i*| |*i*. *i* ∈ *A*}
using *i-def* **by** *auto*
ultimately show *Sup* {|*f i*| |*i*. *i* ∈ *A*} = *sqrt* (*Sup* {(*f i*)^2 |*i*. *i* ∈ *A*})
using *cSup-mem-eq*[*of* |*f i*| {|*f i*| |*i*. *i* ∈ *A*}] *lhs* **by** *auto*
qed

lemma *sqrt-Max-power2-eq-max-abs*:
finite A ⟹ *A* ≠ {} ⟹ *sqrt* (*Max* {(*f i*)^2|*i*. *i* ∈ *A*}) = *Max* {|*f i*| |*i*. *i* ∈ *A*}
apply(*subst cSup-eq-Max*[*symmetric*], *simp-all*)+

using *sqrt-Sup-power2-eq-Sup-abs* **.**

lemma *op-norm-diag-mat-eq*: $\|diag\text{-}mat\ f\|_{op} = Max\ \{|f\ i|\ |i.\ i \in UNIV\}$ (**is** - = *Max ?A*)
proof(*unfold op-norm-def*)
have *obs*: $\bigwedge x\ i.\ (f\ i)^2 * (x\ \$\ i)^2 \leq Max\ \{(f\ i)^2 | i.\ i \in UNIV\} * (x\ \$\ i)^2$
apply(*rule mult-right-mono*[*OF - zero-le-power2*])
using *le-max-image-of-finite*[*of* $\lambda i.\ (f\ i)\hat{}2$] **by** *simp*
{fix *r* **assume** $r \in \{\|diag\text{-}mat\ f * v\ x\|\ |x.\ \|x\| = 1\}$
then obtain *x* **where** *x-def*: $\|diag\text{-}mat\ f * v\ x\| = r \wedge \|x\| = 1$
by *blast*
hence $r^2 = (\sum i \in UNIV.\ (f\ i)^2 * (x\ \$\ i)^2)$
unfolding *norm-vec-def L2-set-def matrix-vector-mul-diag-mat*
apply (*simp add*: *power-mult-distrib*)
by (*metis* (*no-types*, *lifting*) *x-def norm-ge-zero real-sqrt-ge-0-iff real-sqrt-pow2*)
also have $... \leq (Max\ \{(f\ i)^2 | i.\ i \in UNIV\}) * (\sum i \in UNIV.\ (x\ \$\ i)^2)$
using *obs*[*of - x*] **by** (*simp add*: *sum-mono sum-distrib-left*)
also have $... = Max\ \{(f\ i)^2 | i.\ i \in UNIV\}$
using *x-def* **by** (*simp add*: *norm-vec-def L2-set-def*)
finally have $r \leq sqrt\ (Max\ \{(f\ i)^2 | i.\ i \in UNIV\})$
using *x-def real-le-rsqrt* **by** *blast*
hence $r \leq Max\ ?A$
by (*subst* (*asm*) *sqrt-Max-power2-eq-max-abs*[*of UNIV f*], *simp-all*)**}**
hence *1*: $\forall x \in \{\|diag\text{-}mat\ f * v\ x\|\ |x.\ \|x\| = 1\}.\ x \leq Max\ ?A$
unfolding *diag-mat-def* **by** *blast*
obtain *i* **where** *i-def*: $Max\ ?A = \|diag\text{-}mat\ f * v\ e\ i\|$
using *cMax-finite-ex*[*of ?A*] **by** *force*
hence *2*: $\exists x \in \{\|diag\text{-}mat\ f * v\ x\|\ |x.\ \|x\| = 1\}.\ Max\ ?A \leq x$
by (*metis* (*mono-tags*, *lifting*) *abs-1 mem-Collect-eq norm-axis-eq order-refl real-norm-def*)
show $Sup\ \{\|diag\text{-}mat\ f * v\ x\|\ |x.\ \|x\| = 1\} = Max\ ?A$
by (*rule cSup-eq*[*OF 1 2*])
qed

lemma *op-max-norms-eq-at-diag*: $\|diag\text{-}mat\ f\|_{op} = \|diag\text{-}mat\ f\|_{max}$
proof(*rule antisym*)
have $\{|f\ i|\ |i.\ i \in UNIV\} \subseteq \{|diag\text{-}mat\ f\ \$\ i\ \$\ j|\ |i\ j.\ i \in UNIV \wedge j \in UNIV\}$
by (*smt Collect-mono diag-mat-vec-nth-simps*(*1*))
thus $\|diag\text{-}mat\ f\|_{op} \leq \|diag\text{-}mat\ f\|_{max}$
unfolding *op-norm-diag-mat-eq max-norm-def*
by (*rule Max.subset-imp*) (*blast*, *simp only*: *finite-image-of-finite2*)
next
have $Sup\ \{|diag\text{-}mat\ f\ \$\ i\ \$\ j|\ |i\ j.\ i \in UNIV \wedge j \in UNIV\} \leq Sup\ \{|f\ i|\ |i.\ i \in UNIV\}$
apply(*rule cSup-least*, *blast*, *clarify*, *case-tac* $i = j$, *simp*)
by (*rule cSup-upper*, *blast*, *simp-all*) (*rule cSup-upper2*, *auto*)
thus $\|diag\text{-}mat\ f\|_{max} \leq \|diag\text{-}mat\ f\|_{op}$
unfolding *op-norm-diag-mat-eq max-norm-def*
apply (*subst cSup-eq-Max*[*symmetric*], *simp only*: *finite-image-of-finite2*, *blast*)

by (*subst cSup-eq-Max*[*symmetric*], *simp*, *blast*)
qed

end

4 Square Matrices

The general solution for affine systems of ODEs involves the exponential function. Unfortunately, this operation is only available in Isabelle for the type class "banach". Hence, we define a type of square matrices and prove that it is an instance of this class.

theory *SQ-MTX*
imports *MTX-Norms*

begin

4.1 Definition

typedef *'m sq-mtx* = *UNIV*::(*real*^'*m*^'*m*) *set*
morphisms *to-vec to-mtx* **by** *simp*

declare *to-mtx-inverse* [*simp*]
and *to-vec-inverse* [*simp*]

setup-lifting *type-definition-sq-mtx*

lift-definition *sq-mtx-ith* :: *'m sq-mtx* ⇒ *'m* ⇒ (*real*^'*m*) (**infixl** $$ *90*) **is** ($) .

lift-definition *sq-mtx-vec-mult* :: *'m sq-mtx* ⇒ (*real*^'*m*) ⇒ (*real*^'*m*) (**infixl** $*_V$ *90*) **is** (∗*v*) .

lift-definition *vec-sq-mtx-prod* :: (*real*^'*m*) ⇒ *'m sq-mtx* ⇒ (*real*^'*m*) **is** (*v*∗) .

lift-definition *sq-mtx-diag* :: ((*'m*::*finite*) ⇒ *real*) ⇒ (*'m*::*finite*) *sq-mtx* (**binder** diag *10*)
is *diag-mat* .

lift-definition *sq-mtx-transpose* :: (*'m*::*finite*) *sq-mtx* ⇒ *'m sq-mtx* (-†) **is** *transpose* .

lift-definition *sq-mtx-inv* :: (*'m*::*finite*) *sq-mtx* ⇒ *'m sq-mtx* (-$^{-1}$ [*90*]) **is** *matrix-inv* .

lift-definition *sq-mtx-row* :: *'m* ⇒ (*'m*::*finite*) *sq-mtx* ⇒ *real*^'*m* (row) **is** *row* .

lift-definition *sq-mtx-col* :: *'m* ⇒ (*'m*::*finite*) *sq-mtx* ⇒ *real*^'*m* (col) **is** *column* .

lemma *to-vec-eq-ith*: (*to-vec A*) $ *i* = *A* $$ *i*
by *transfer simp*

lemma *to-mtx-ith*[*simp*]:
(*to-mtx A*) $$ *i1* = *A* $ *i1*
(*to-mtx A*) $$ *i1* $ *i2* = *A* $ *i1* $ *i2*
by (*transfer*, *simp*)+

lemma *to-mtx-vec-lambda-ith*[*simp*]: *to-mtx* (χ *i j*. *x i j*) $$ *i1* $ *i2* = *x i1 i2*
by (*simp add*: *sq-mtx-ith-def*)

lemma *sq-mtx-eq-iff*:
shows *A* = *B* = (∀ *i j*. *A* $$ *i* $ *j* = *B* $$ *i* $ *j*)
and *A* = *B* = (∀ *i*. *A* $$ *i* = *B* $$ *i*)
by (*transfer*, *simp add*: *vec-eq-iff*)+

lemma *sq-mtx-diag-simps*[*simp*]:
i = *j* ⟹ *sq-mtx-diag f* $$ *i* $ *j* = *f i*
i ≠ *j* ⟹ *sq-mtx-diag f* $$ *i* $ *j* = *0*
sq-mtx-diag f $$ *i* = *axis i* (*f i*)
unfolding *sq-mtx-diag-def* **by** (*simp-all add*: *axis-def vec-eq-iff*)

lemma *sq-mtx-diag-vec-mult*: (diag *i*. *f i*) $*_V$ *s* = (χ *i*. *f i* ∗ *s*$*i*)
by (*simp add*: *matrix-vector-mul-diag-mat sq-mtx-diag.abs-eq sq-mtx-vec-mult.abs-eq*)

lemma *sq-mtx-vec-mult-diag-axis*: (diag *i*. *f i*) $*_V$ (*axis i k*) = *axis i* (*f i* ∗ *k*)
unfolding *sq-mtx-diag-vec-mult axis-def* **by** *auto*

lemma *sq-mtx-vec-mult-eq*: *m* $*_V$ *x* = (χ *i*. *sum* (λ*j*. (*m* $$ *i* $ *j*) ∗ (*x* $ *j*)) *UNIV*)
by (*transfer*, *simp add*: *matrix-vector-mult-def*)

lemma *sq-mtx-transpose-transpose*[*simp*]: $(A^\dagger)^\dagger = A$
by (*transfer*, *simp*)

lemma *transpose-mult-vec-canon-row*[*simp*]: $(A^\dagger)$ $*_V$ (e *i*) = row *i A*
by *transfer* (*simp add*: *row-def transpose-def axis-def matrix-vector-mult-def*)

lemma *row-ith*[*simp*]: row *i A* = *A* $$ *i*
by *transfer* (*simp add*: *row-def*)

lemma *mtx-vec-mult-canon*: *A* $*_V$ (e *i*) = col *i A*
by (*transfer*, *simp add*: *matrix-vector-mult-basis*)

4.2 Ring of square matrices

instantiation *sq-mtx* :: (*finite*) *ring*
begin

lift-definition *plus-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (+) **.**

lift-definition *zero-sq-mtx* :: *'a sq-mtx* **is** *0* **.**

lift-definition *uminus-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* **is** *uminus* **.**

lift-definition *minus-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (−) **.**

lift-definition *times-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** (∗∗) **.**

declare *plus-sq-mtx.rep-eq* [*simp*]
and *minus-sq-mtx.rep-eq* [*simp*]

instance apply *intro-classes*
by(*transfer*, *simp add*: *algebra-simps matrix-mul-assoc matrix-add-rdistrib matrix-add-ldistrib*)+

end

lemma *sq-mtx-zero-ith*[*simp*]: *0* $$ *i* = *0*
by (*transfer*, *simp*)

lemma *sq-mtx-zero-nth*[*simp*]: *0* $$ *i* $ *j* = *0*
by *transfer simp*

lemma *sq-mtx-plus-eq*: *A* + *B* = *to-mtx* (χ *i j*. *A*$$*i*$*j* + *B*$$*i*$*j*)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-plus-ith*[*simp*]:(*A* + *B*) $$ *i* = *A* $$ *i* + *B* $$ *i*
unfolding *sq-mtx-plus-eq* **by** (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-uminus-eq*: − *A* = *to-mtx* (χ *i j*. − *A*$$*i*$*j*)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-minus-eq*: *A* − *B* = *to-mtx* (χ *i j*. *A*$$*i*$*j* − *B*$$*i*$*j*)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-minus-ith*[*simp*]:(*A* − *B*) $$ *i* = *A* $$ *i* − *B* $$ *i*
unfolding *sq-mtx-minus-eq* **by** (*simp add*: *vec-eq-iff*)

lemma *sq-mtx-times-eq*: *A* ∗ *B* = *to-mtx* (χ *i j*. *sum* (λ*k*. *A*$$*i*$*k* ∗ *B*$$*k*$*j*) *UNIV*)
by *transfer* (*simp add*: *matrix-matrix-mult-def*)

lemma *sq-mtx-plus-diag-diag*[*simp*]: *sq-mtx-diag f* + *sq-mtx-diag g* = (diag *i*. *f i* + *g i*)
by (*subst sq-mtx-eq-iff*) (*simp add*: *axis-def*)

lemma *sq-mtx-minus-diag-diag*[*simp*]: *sq-mtx-diag f* − *sq-mtx-diag g* = (diag *i*. *f i* − *g i*)

by (*subst sq-mtx-eq-iff*) (*simp add*: *axis-def*)

lemma *sum-sq-mtx-diag*[*simp*]: $(\sum n{<}m.\ \textit{sq-mtx-diag}\ (g\ n)) = (\text{diag}\ i.\ \sum n{<}m.\ (g\ n\ i))$ **for** *m*::*nat*
by (*induct m*, *simp*, *subst sq-mtx-eq-iff*, *simp-all*)

lemma *sq-mtx-mult-diag-diag*[*simp*]: *sq-mtx-diag f* $*$ *sq-mtx-diag g* $=$ (diag *i*. *f i* $*$ *g i*)
by (*simp add*: *matrix-mul-diag-diag sq-mtx-diag.abs-eq times-sq-mtx.abs-eq*)

lemma *sq-mtx-mult-diagl*: (diag *i*. *f i*) $*$ *A* $=$ *to-mtx* (χ *i j*. *f i* $*$ *A* $$ *i* $ *j*)
by *transfer* (*simp add*: *matrix-mul-diag-matl*)

lemma *sq-mtx-mult-diagr*: *A* $*$ (diag *i*. *f i*) $=$ *to-mtx* (χ *i j*. *A* $$ *i* $ *j* $*$ *f j*)
by *transfer* (*simp add*: *matrix-matrix-mul-diag-matr*)

lemma *mtx-vec-mult-0l*[*simp*]: *0* $*_V$ *x* $=$ *0*
by (*simp add*: *sq-mtx-vec-mult.abs-eq zero-sq-mtx-def*)

lemma *mtx-vec-mult-0r*[*simp*]: *A* $*_V$ *0* $=$ *0*
by (*transfer*, *simp*)

lemma *mtx-vec-mult-add-rdistr*: (*A* $+$ *B*) $*_V$ *x* $=$ *A* $*_V$ *x* $+$ *B* $*_V$ *x*
unfolding *plus-sq-mtx-def*
apply(*transfer*)
by (*simp add*: *matrix-vector-mult-add-rdistrib*)

lemma *mtx-vec-mult-add-rdistl*: *A* $*_V$ (*x* $+$ *y*) $=$ *A* $*_V$ *x* $+$ *A* $*_V$ *y*
unfolding *plus-sq-mtx-def*
apply *transfer*
by (*simp add*: *matrix-vector-right-distrib*)

lemma *mtx-vec-mult-minus-rdistrib*: (*A* $-$ *B*) $*_V$ *x* $=$ *A* $*_V$ *x* $-$ *B* $*_V$ *x*
unfolding *minus-sq-mtx-def* **by**(*transfer*, *simp add*: *matrix-vector-mult-diff-rdistrib*)

lemma *mtx-vec-mult-minus-ldistrib*: *A* $*_V$ (*x* $-$ *y*) $=$ *A* $*_V$ *x* $-$ *A* $*_V$ *y*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel diff-add-cancel*
matrix-vector-right-distrib sq-mtx-vec-mult.rep-eq)

lemma *sq-mtx-times-vec-assoc*: (*A* $*$ *B*) $*_V$ *x* $=$ *A* $*_V$ (*B* $*_V$ *x*)
by (*transfer*, *simp add*: *matrix-vector-mul-assoc*)

lemma *sq-mtx-vec-mult-sum-cols*: *A* $*_V$ *x* $=$ *sum* ($\lambda i.$ *x* $ *i* $*_R$ col *i A*) *UNIV*
by(*transfer*) (*simp add*: *matrix-mult-sum scalar-mult-eq-scaleR*)

4.3 Real normed vector space of square matrices

instantiation *sq-mtx* :: (*finite*) *real-normed-vector*
begin

definition *norm-sq-mtx* :: *'a sq-mtx* ⇒ *real* **where** $\|A\| = \|to\text{-}vec\ A\|_{op}$

lift-definition *scaleR-sq-mtx* :: *real* ⇒ *'a sq-mtx* ⇒ *'a sq-mtx* **is** *scaleR* **.**

definition *sgn-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx*
where *sgn-sq-mtx A* = (*inverse* (‖*A*‖)) $*_R$ *A*

definition *dist-sq-mtx* :: *'a sq-mtx* ⇒ *'a sq-mtx* ⇒ *real*
where *dist-sq-mtx A B* = ‖*A* − *B*‖

definition *uniformity-sq-mtx* :: (*'a sq-mtx* × *'a sq-mtx*) *filter*
where *uniformity-sq-mtx* = (*INF e*∈{*0*<..}. *principal* {(*x*, *y*). *dist x y* < *e*})

definition *open-sq-mtx* :: *'a sq-mtx set* ⇒ *bool*
where *open-sq-mtx U* = (∀ *x*∈*U*. ∀ *F* (*x'*, *y*) *in uniformity*. *x'* = *x* ⟶ *y* ∈ *U*)

instance apply *intro-classes*
unfolding *sgn-sq-mtx-def open-sq-mtx-def dist-sq-mtx-def uniformity-sq-mtx-def*
prefer *10*
apply(*transfer*, *simp add*: *norm-sq-mtx-def op-norm-triangle*)
prefer *9*
apply(*simp-all add*: *norm-sq-mtx-def zero-sq-mtx-def op-norm-eq-0*)
by (*transfer*, *simp add*: *norm-sq-mtx-def op-norm-scaleR algebra-simps*)+

end

lemma *sq-mtx-scaleR-eq*: *c* $*_R$ *A* = *to-mtx* (χ *i j*. *c* $*_R$ *A* $$ *i* $ *j*)
by *transfer* (*simp add*: *vec-eq-iff*)

lemma *scaleR-to-mtx-ith*[*simp*]: *c* $*_R$ (*to-mtx A*) $$ *i1* $ *i2* = *c* ∗ *A* $ *i1* $ *i2*
by *transfer* (*simp add*: *scaleR-vec-def*)

lemma *sq-mtx-scaleR-ith*[*simp*]: (*c* $*_R$ *A*) $$ *i* = (*c* $*_R$ (*A* $$ *i*))
by (*unfold scaleR-sq-mtx-def*, *transfer*, *simp*)

lemma *scaleR-sq-mtx-diag*: *c* $*_R$ *sq-mtx-diag f* = (diag *i*. *c* ∗ *f i*)
by (*subst sq-mtx-eq-iff*, *simp add*: *axis-def*)

lemma *scaleR-mtx-vec-assoc*: (*c* $*_R$ *A*) $*_V$ *x* = *c* $*_R$ (*A* $*_V$ *x*)
unfolding *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply** *simp*
by (*simp add*: *scaleR-matrix-vector-assoc*)

lemma *mtx-vec-scaleR-commute*: *A* $*_V$ (*c* $*_R$ *x*) = *c* $*_R$ (*A* $*_V$ *x*)
unfolding *scaleR-sq-mtx-def sq-mtx-vec-mult-def* **apply**(*simp*, *transfer*)
by (*simp add*: *vector-scaleR-commute*)

lemma *mtx-times-scaleR-commute*: *A* ∗ (*c* $*_R$ *B*) = *c* $*_R$ (*A* ∗ *B*) **for** *A*::(*'n*::*finite*) *sq-mtx*

unfolding *sq-mtx-scaleR-eq sq-mtx-times-eq*
apply(*simp add*: *to-mtx-inject*)
apply(*simp add*: *vec-eq-iff fun-eq-iff*)
by (*simp add*: *semiring-normalization-rules*(*19*) *vector-space-over-itself*.*scale-sum-right*)

lemma *le-mtx-norm*: $m \in \{\|A *_V x\| \ |x.\ \|x\| = 1\} \Longrightarrow m \leq \|A\|$
using *cSup-upper*[*of* - $\{\|(to\text{-}vec\ A) *v x\| \mid x.\ \|x\| = 1\}$]
by (*simp add*: *op-norm-set-proptys*(*2*) *op-norm-def norm-sq-mtx-def sq-mtx-vec-mult.rep-eq*)

lemma *norm-vec-mult-le*: $\|A *_V x\| \leq (\|A\|) * (\|x\|)$
by (*simp add*: *norm-matrix-le-mult-op-norm norm-sq-mtx-def sq-mtx-vec-mult.rep-eq*)

lemma *bounded-bilinear-sq-mtx-vec-mult*: *bounded-bilinear* ($\lambda A\ s.\ A *_V s$)
apply (*rule bounded-bilinear.intro*, *simp-all add*: *mtx-vec-mult-add-rdistr*
mtx-vec-mult-add-rdistl scaleR-mtx-vec-assoc mtx-vec-scaleR-commute)
by (*rule-tac x=1* **in** *exI*, *auto intro*!: *norm-vec-mult-le*)

lemma *norm-sq-mtx-def2*: $\|A\| = Sup\ \{\|A *_V x\| \ |x.\ \|x\| = 1\}$
unfolding *norm-sq-mtx-def op-norm-def sq-mtx-vec-mult-def* **by** *simp*

lemma *norm-sq-mtx-def3*: $\|A\| = (SUP\ x.\ (\|A *_V x\|) / (\|x\|))$
unfolding *norm-sq-mtx-def onorm-def sq-mtx-vec-mult-def* **by** *simp*

lemma *norm-sq-mtx-diag*: $\|sq\text{-}mtx\text{-}diag\ f\| = Max\ \{|f\ i| \ |i.\ i \in UNIV\}$
unfolding *norm-sq-mtx-def* **apply** *transfer*
by (*rule op-norm-diag-mat-eq*)

lemma *sq-mtx-norm-le-sum-col*: $\|A\| \leq (\sum i \in UNIV.\ \|\text{col}\ i\ A\|)$
using *op-norm-le-sum-column*[*of to-vec A*]
apply(*simp add*: *norm-sq-mtx-def*)
by(*transfer*, *simp add*: *op-norm-le-sum-column*)

lemma *norm-le-transpose*: $\|A\| \leq \|A^\dagger\|$
unfolding *norm-sq-mtx-def* **by** *transfer* (*rule op-norm-le-transpose*)

lemma *norm-eq-norm-transpose*[*simp*]: $\|A^\dagger\| = \|A\|$
using *norm-le-transpose*[*of A*] **and** *norm-le-transpose*[*of* $A^\dagger$] **by** *simp*

lemma *norm-column-le-norm*: $\|A\ \$\$\ i\| \leq \|A\|$
using *norm-vec-mult-le*[*of* $A^\dagger$ e *i*] **by** *simp*

4.4 Real normed algebra of square matrices

instantiation *sq-mtx* :: (*finite*) *real-normed-algebra-1*
begin

lift-definition *one-sq-mtx* :: *'a sq-mtx* **is** *to-mtx* (*mat 1*) .

lemma *sq-mtx-one-idty*: $1 * A = A\ A * 1 = A$ **for** A :: *'a sq-mtx*

by(*transfer*, *transfer*, *unfold mat-def matrix-matrix-mult-def*, *simp add*: *vec-eq-iff*)+

lemma *sq-mtx-norm-1*: $\|(1::'a\ sq\text{-}mtx)\| = 1$
unfolding *one-sq-mtx-def norm-sq-mtx-def*
apply(*simp add*: *op-norm-def*)
apply(*subst cSup-eq*[*of* - *1*])
using *ex-norm-eq-1* **by** *auto*

lemma *sq-mtx-norm-times*: $\|A * B\| \leq (\|A\|) * (\|B\|)$ **for** $A :: 'a\ sq\text{-}mtx$
unfolding *norm-sq-mtx-def times-sq-mtx-def* **by**(*simp add*: *op-norm-matrix-matrix-mult-le*)

instance
apply *intro-classes*
apply(*simp-all add*: *sq-mtx-one-idty sq-mtx-norm-1 sq-mtx-norm-times*)
apply(*simp-all add*: *to-mtx-inject vec-eq-iff one-sq-mtx-def zero-sq-mtx-def mat-def*)
by(*transfer*, *simp add*: *scalar-matrix-assoc matrix-scalar-ac*)+

end

lemma *sq-mtx-one-ith-simps*[*simp*]: *1* $$ *i* $ *i* = *1 i* $\neq$ *j* $\Longrightarrow$ *1* $$ *i* $ *j* = *0*
unfolding *one-sq-mtx-def mat-def* **by** *simp-all*

lemma *of-nat-eq-sq-mtx-diag*[*simp*]: *of-nat m* = (diag *i*. *m*)
by (*induct m*) (*simp*, *subst sq-mtx-eq-iff*, *simp add*: *axis-def*)+

lemma *mtx-vec-mult-1*[*simp*]: *1* $*_V$ *s* = *s*
by (*auto simp*: *sq-mtx-vec-mult-def one-sq-mtx-def*
mat-def vec-eq-iff matrix-vector-mult-def)

lemma *sq-mtx-diag-one*[*simp*]: (diag *i*. *1*) = *1*
by (*subst sq-mtx-eq-iff*, *simp add*: *one-sq-mtx-def mat-def axis-def*)

abbreviation *mtx-invertible A* $\equiv$ *invertible* (*to-vec A*)

lemma *mtx-invertible-def*: *mtx-invertible A* $\longleftrightarrow$ ($\exists$ *A'*. *A'* $*$ *A* = *1* $\wedge$ *A* $*$ *A'* = *1*)
apply (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def invertible-def*, *clarsimp*, *safe*)
apply(*rule-tac x*=*to-mtx A'* **in** *exI*, *simp*)
by (*rule-tac x*=*to-vec A'* **in** *exI*, *simp add*: *to-mtx-inject*)

lemma *mtx-invertibleI*:
assumes *A* $*$ *B* = *1* **and** *B* $*$ *A* = *1*
shows *mtx-invertible A*
using *assms* **unfolding** *mtx-invertible-def* **by** *auto*

lemma *mtx-invertibleD*[*simp*]:
assumes *mtx-invertible A*
shows A^{-1} $*$ *A* = *1* **and** *A* $*$ A^{-1} = *1*
apply (*unfold sq-mtx-inv-def times-sq-mtx-def one-sq-mtx-def*)

using *assms* **by** *simp-all*

lemma *mtx-invertible-inv*[*simp*]: *mtx-invertible* $A \Longrightarrow$ *mtx-invertible* (A^{-1})
using *mtx-invertibleD mtx-invertibleI* **by** *blast*

lemma *mtx-invertible-one*[*simp*]: *mtx-invertible 1*
by (*simp add*: *one-sq-mtx.rep-eq*)

lemma *sq-mtx-inv-unique*:
assumes $A * B = 1$ **and** $B * A = 1$
shows $A^{-1} = B$
by (*metis* (*no-types*, *lifting*) *assms mtx-invertibleD*(*2*)
mtx-invertibleI mult.assoc sq-mtx-one-idty(*1*))

lemma *sq-mtx-inv-idempotent*[*simp*]: *mtx-invertible* $A \Longrightarrow A^{-1-1} = A$
using *mtx-invertibleD sq-mtx-inv-unique* **by** *blast*

lemma *sq-mtx-inv-mult*:
assumes *mtx-invertible A* **and** *mtx-invertible B*
shows $(A * B)^{-1} = B^{-1} * A^{-1}$
by (*simp add*: *assms matrix-inv-matrix-mul sq-mtx-inv-def times-sq-mtx-def*)

lemma *sq-mtx-inv-one*[*simp*]: $1^{-1} = 1$
by (*simp add*: *sq-mtx-inv-unique*)

definition *similar-sq-mtx* :: ('*n*::*finite*) *sq-mtx* ⇒ '*n sq-mtx* ⇒ *bool* (**infixr** $\sim$ *25*)
where $(A \sim B) \longleftrightarrow (\exists\ P.$ *mtx-invertible* $P \wedge A = P^{-1} * B * P)$

lemma *similar-sq-mtx-matrix*: $(A \sim B) =$ *similar-matrix* (*to-vec A*) (*to-vec B*)
apply(*unfold similar-matrix-def similar-sq-mtx-def*, *safe*)
apply (*metis sq-mtx-inv.rep-eq times-sq-mtx.rep-eq*)
by (*metis UNIV-I sq-mtx-inv.abs-eq times-sq-mtx.abs-eq to-mtx-inverse to-vec-inverse*)

lemma *similar-sq-mtx-refl*[*simp*]: $A \sim A$
by (*unfold similar-sq-mtx-def*, *rule-tac x=1* **in** *exI*, *simp*)

lemma *similar-sq-mtx-simm*: $A \sim B \Longrightarrow B \sim A$
apply(*unfold similar-sq-mtx-def*, *clarsimp*)
apply(*rule-tac x=*P^{-1} **in** *exI*, *simp add*: *mult.assoc*)
by (*metis mtx-invertibleD*(*2*) *mult.assoc mult.left-neutral*)

lemma *similar-sq-mtx-trans*: $A \sim B \Longrightarrow B \sim C \Longrightarrow A \sim C$
unfolding *similar-sq-mtx-matrix* **using** *similar-matrix-trans* **by** *blast*

lemma *power-sq-mtx-diag*: (*sq-mtx-diag f*)^*n* = (diag *i*. *f i*^*n*)
by (*induct n*, *simp-all*)

lemma *power-similiar-sq-mtx-diag-eq*:
assumes *mtx-invertible P*

and $A = P^{-1} * (sq\text{-}mtx\text{-}diag\ f) * P$
shows $A \hat{} n = P^{-1} * (\text{diag}\ i.\ f\ i \hat{} n) * P$
proof(*induct n*, *simp-all add*: *assms*)
fix *n*::*nat*
have $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (\text{diag}\ i.\ f\ i \hat{}\ n) * P) =$
$P^{-1} * sq\text{-}mtx\text{-}diag\ f * (\text{diag}\ i.\ f\ i \hat{}\ n) * P$
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *mtx-invertibleD*(*2*) *mult.assoc mult.right-neutral*)
also have $... = P^{-1} * (\text{diag}\ i.\ f\ i * f\ i \hat{}\ n) * P$
by (*simp add*: *mult.assoc*)
finally show $P^{-1} * sq\text{-}mtx\text{-}diag\ f * P * (P^{-1} * (\text{diag}\ i.\ f\ i \hat{}\ n) * P) =$
$P^{-1} * (\text{diag}\ i.\ f\ i * f\ i \hat{}\ n) * P$ **.**
qed

lemma *power-similar-sq-mtx-diag*:
assumes $A \sim (sq\text{-}mtx\text{-}diag\ f)$
shows $A \hat{} n \sim (\text{diag}\ i.\ f\ i \hat{} n)$
using *assms power-similiar-sq-mtx-diag-eq*
unfolding *similar-sq-mtx-def* **by** *blast*

4.5 Banach space of square matrices

lemma *Cauchy-cols*:
fixes $X :: nat \Rightarrow ('a::finite)\ sq\text{-}mtx$
assumes *Cauchy X*
shows *Cauchy* $(\lambda n.\ \text{col}\ i\ (X\ n))$
proof(*unfold Cauchy-def dist-norm*, *clarsimp*)
fix ε::*real* **assume** $\varepsilon > 0$
then obtain M **where** $M\text{-}def: \forall\, m \geq M.\ \forall\, n \geq M.\ \|X\ m - X\ n\| < \varepsilon$
using ‹*Cauchy X*› **unfolding** *Cauchy-def* **by**(*simp add*: *dist-sq-mtx-def*) *metis*
{fix $m\ n$ **assume** $m \geq M$ **and** $n \geq M$
hence $\varepsilon > \|X\ m - X\ n\|$
using *M-def* **by** *blast*
moreover have $\|X\ m - X\ n\| \geq \|(X\ m - X\ n) *_V \text{e}\ i\|$
by(*rule le-mtx-norm*[*of* - $X\ m - X\ n$], *force*)
moreover have $\|(X\ m - X\ n) *_V \text{e}\ i\| = \|X\ m *_V \text{e}\ i - X\ n *_V \text{e}\ i\|$
by (*simp add*: *mtx-vec-mult-minus-rdistrib*)
moreover have $... = \|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\|$
by (*simp add*: *mtx-vec-mult-minus-rdistrib mtx-vec-mult-canon*)
ultimately have $\|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\| < \varepsilon$
by *linarith***}**
thus $\exists\, M.\ \forall\, m \geq M.\ \forall\, n \geq M.\ \|\text{col}\ i\ (X\ m) - \text{col}\ i\ (X\ n)\| < \varepsilon$
by *blast*
qed

lemma *col-convergence*:
assumes $\forall\, i.\ (\lambda n.\ \text{col}\ i\ (X\ n)) \longrightarrow L\ \$\ i$
shows $X \longrightarrow to\text{-}mtx\ (transpose\ L)$
proof(*unfold LIMSEQ-def dist-norm*, *clarsimp*)
let *?L* = *to-mtx* (*transpose L*)

let *?a* = *CARD*('*a*) **fix** *ε*::*real* **assume** *ε* > *0*
hence *ε* / *?a* > *0* **by** *simp*
hence ∀ *i*. ∃ *N*. ∀ *n*≥*N*. ‖col *i* (*X n*) − *L* $ *i*‖ < *ε*/*?a*
using *assms* **unfolding** *LIMSEQ-def dist-norm convergent-def* **by** *blast*
then obtain *N* **where** ∀ *i*. ∀ *n*≥*N*. ‖col *i* (*X n*) − *L* $ *i*‖ < *ε*/*?a*
using *finite-nat-minimal-witness*[*of* λ *i n*. ‖col *i* (*X n*) − *L* $ *i*‖ < *ε*/*?a*] **by** *blast*
also have ⋀*i n*. (col *i* (*X n*) − *L* $ *i*) = (col *i* (*X n* − *?L*))
unfolding *minus-sq-mtx-def* **by**(*transfer*, *simp add*: *transpose-def vec-eq-iff column-def*)
ultimately have *N-def*:∀ *i*. ∀ *n*≥*N*. ‖col *i* (*X n* − *?L*)‖ < *ε*/*?a*
by *auto*
have ∀ *n*≥*N*. ‖*X n* − *?L*‖ < *ε*
proof(*rule allI*, *rule impI*)
fix *n*::*nat* **assume** *N* ≤ *n*
hence ∀ *i*. ‖col *i* (*X n* − *?L*)‖ < *ε*/*?a*
using *N-def* **by** *blast*
hence (∑ *i*∈*UNIV*. ‖col *i* (*X n* − *?L*)‖) < (∑ (*i*::'*a*)∈*UNIV*. *ε*/*?a*)
using *sum-strict-mono*[*of* - λ*i*. ‖col *i* (*X n* − *?L*)‖] **by** *force*
moreover have ‖*X n* − *?L*‖ ≤ (∑ *i*∈*UNIV*. ‖col *i* (*X n* − *?L*)‖)
using *sq-mtx-norm-le-sum-col* **by** *blast*
moreover have (∑ (*i*::'*a*)∈*UNIV*. *ε*/*?a*) = *ε*
by *force*
ultimately show ‖*X n* − *?L*‖ < *ε*
by *linarith*
qed
thus ∃ *no*. ∀ *n*≥*no*. ‖*X n* − *?L*‖ < *ε*
by *blast*
qed

instance *sq-mtx* :: (*finite*) *banach*
proof(*standard*)
fix *X* :: *nat* ⇒ '*a sq-mtx*
assume *Cauchy X*
hence ⋀*i*. *Cauchy* (λ*n*. col *i* (*X n*))
using *Cauchy-cols* **by** *blast*
hence *obs*: ∀ *i*. ∃! *L*. (λ*n*. col *i* (*X n*)) ⟶ *L*
using *Cauchy-convergent convergent-def LIMSEQ-unique* **by** *fastforce*
define *L* **where** *L* = (χ *i*. *lim* (λ*n*. col *i* (*X n*)))
hence ∀ *i*. (λ*n*. col *i* (*X n*)) ⟶ *L* $ *i*
using *obs theI-unique*[*of* λ*L*. (λ*n*. col - (*X n*)) ⟶ *L L* $ -] **by** (*simp add*: *lim-def*)
thus *convergent X*
using *col-convergence* **unfolding** *convergent-def* **by** *blast*
qed

lemma *exp-similiar-sq-mtx-diag-eq*:
assumes *mtx-invertible P*
and *A* = *P*⁻¹ ∗ (diag *i*. *f i*) ∗ *P*

shows *exp A* = P^{-1} * *exp* (diag *i*. *f i*) * *P*
proof(*unfold exp-def power-similiar-sq-mtx-diag-eq*[*OF assms*])
have ($\sum$ *n*. P^{-1} * (diag *i*. *f i* ^ *n*) * *P* /$_R$ *fact n*) =
($\sum$ *n*. P^{-1} * ((diag *i*. *f i* ^ *n*) /$_R$ *fact n*) * *P*)
by *simp*
also have ... = ($\sum$ *n*. P^{-1} * ((diag *i*. *f i* ^ *n*) /$_R$ *fact n*)) * *P*
apply(*subst suminf-multr*[*OF bounded-linear.summable*[*OF bounded-linear-mult-right*]])
unfolding *power-sq-mtx-diag*[*symmetric*] **by** (*simp-all add*: *summable-exp-generic*)
also have ... = P^{-1} * ($\sum$ *n*. (diag *i*. *f i* ^ *n*) /$_R$ *fact n*) * *P*
apply(*subst suminf-mult*[*of* - P^{-1}])
unfolding *power-sq-mtx-diag*[*symmetric*]
by (*simp-all add*: *summable-exp-generic*)
finally show ($\sum$ *n*. P^{-1} * (diag *i*. *f i* ^ *n*) * *P* /$_R$ *fact n*) =
P^{-1} * ($\sum$ *n*. *sq-mtx-diag f* ^ *n* /$_R$ *fact n*) * *P*
unfolding *power-sq-mtx-diag* **by** *simp*
qed

lemma *exp-similiar-sq-mtx-diag*:
assumes *A* ∼ *sq-mtx-diag f*
shows *exp A* ∼ *exp* (*sq-mtx-diag f*)
using *assms exp-similiar-sq-mtx-diag-eq*
unfolding *similar-sq-mtx-def* **by** *blast*

lemma *suminf-sq-mtx-diag*:
assumes ∀ *i*. (λ*n*. *f n i*) *sums* (*suminf* (λ*n*. *f n i*))
shows ($\sum$ *n*. (diag *i*. *f n i*)) = (diag *i*. $\sum$ *n*. *f n i*)
proof(*rule suminfI*, *unfold sums-def LIMSEQ-iff*, *clarsimp simp*: *norm-sq-mtx-diag*)
let *?g* = λ*n i*. |($\sum$ *n*<*n*. *f n i*) − ($\sum$ *n*. *f n i*)|
fix *r*::*real* **assume** *r* > *0*
have ∀ *i*. ∃ *no*. ∀ *n*≥*no*. *?g n i* < *r*
using *assms* ‹*r* > *0*› **unfolding** *sums-def LIMSEQ-iff* **by** *clarsimp*
then obtain *N* **where** *key*: ∀ *i*. ∀ *n*≥*N*. *?g n i* < *r*
using *finite-nat-minimal-witness*[*of* λ*i n*. *?g n i* < *r*] **by** *blast*
{fix *n*::*nat*
assume *n* ≥ *N*
obtain *i* **where** *i-def*: *Max* {*x*. ∃ *i*. *x* = *?g n i*} = *?g n i*
using *cMax-finite-ex*[*of* {*x*. ∃ *i*. *x* = *?g n i*}] **by** *auto*
hence *?g n i* < *r*
using *key* ‹*n* ≥ *N*› **by** *blast*
hence *Max* {*x*. ∃ *i*. *x* = *?g n i*} < *r*
unfolding *i-def*[*symmetric*] **.}**
thus ∃ *N*. ∀ *n*≥*N*. *Max* {*x*. ∃ *i*. *x* = *?g n i*} < *r*
by *blast*
qed

lemma *exp-sq-mtx-diag*: *exp* (*sq-mtx-diag f*) = (diag *i*. *exp* (*f i*))
apply(*unfold exp-def*, *simp add*: *power-sq-mtx-diag scaleR-sq-mtx-diag*)
apply(*rule suminf-sq-mtx-diag*)
using *exp-converges*[*of f* -]

unfolding *sums-def LIMSEQ-iff exp-def* **by** *force*

lemma *exp-scaleR-diagonal1*:
assumes *mtx-invertible P* **and** $A = P^{-1} * (\text{diag } i.\ f\ i) * P$
shows $exp\ (t *_R A) = P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$
proof−
have $exp\ (t *_R A) = exp\ (P^{-1} * (t *_R sq\text{-}mtx\text{-}diag\ f) * P)$
using *assms* **by** *simp*
also have $... = P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$
by (*metis assms*(*1*) *exp-similiar-sq-mtx-diag-eq exp-sq-mtx-diag scaleR-sq-mtx-diag*)
finally show $exp\ (t *_R A) = P^{-1} * (\text{diag } i.\ exp\ (t * f\ i)) * P$ **.**
qed

lemma *exp-scaleR-diagonal2*:
assumes *mtx-invertible P* **and** $A = P * (\text{diag } i.\ f\ i) * P^{-1}$
shows $exp\ (t *_R A) = P * (\text{diag } i.\ exp\ (t * f\ i)) * P^{-1}$
apply(*subst sq-mtx-inv-idempotent*[*OF assms*(*1*), *symmetric*])
apply(*rule exp-scaleR-diagonal1*)
by (*simp-all add*: *assms*)

4.6 Examples

definition *mtx A* = *to-mtx* (*vector* (*map vector A*))

lemma *vector-nth-eq*: (*vector A*) $ *i* = *foldr* (λ*x f n*. (*f* (*n* + *1*))(*n* := *x*)) *A* (λ*n x*. *0*) *1 i*
unfolding *vector-def* **by** *simp*

lemma *mtx-ith-eq*[*simp*]: *mtx A* $$ *i* $ *j* = *foldr* (λ*x f n*. (*f* (*n* + *1*))(*n* := *x*))
(*map* (λ*l*. *vec-lambda* (*foldr* (λ*x f n*. (*f* (*n* + *1*))(*n* := *x*)) *l* (λ*n x*. *0*) *1*)) *A*) (λ*n x*. *0*) *1 i* $ *j*
unfolding *mtx-def vector-def* **by** (*simp add*: *vector-nth-eq*)

4.6.1 2x2 matrices

lemma *mtx2-eq-iff*: (*mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) :: *2 sq-mtx*) = *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) ⟷ *a1* = *a2* ∧ *b1* = *b2* ∧ *c1* = *c2* ∧ *d1* = *d2*
apply(*simp add*: *sq-mtx-eq-iff*, *safe*)
using *exhaust-2* **by** *force*+

lemma *mtx2-to-mtx*: *mtx*
([*a*, *b*] #
[*c*, *d*] # []) =
to-mtx (χ *i j*::*2*. *if i*=*1* ∧ *j*=*1 then a*
else (*if i*=*1* ∧ *j*=*2 then b*
else (*if i*=*2* ∧ *j*=*1 then c*
else d)))

apply(*subst sq-mtx-eq-iff*)
using *exhaust-2* **by** *force*

abbreviation *diag2* :: *real* ⇒ *real* ⇒ *2 sq-mtx*
where *diag2* ι_1 ι_2 ≡ *mtx*
([ι_1, *0*] #
[*0*, ι_2] # [])

lemma *diag2-eq*: *diag2* (*ι 1*) (*ι 2*) = (diag *i*. *ι i*)
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-2* **by** (*force simp*: *axis-def*)

lemma *one-mtx2*: (*1*::*2 sq-mtx*) = *diag2 1 1*
apply(*subst sq-mtx-eq-iff*)
using *exhaust-2* **by** *force*

lemma *zero-mtx2*: (*0*::*2 sq-mtx*) = *diag2 0 0*
by (*simp add*: *sq-mtx-eq-iff*)

lemma *scaleR-mtx2*: *k* $*_R$ *mtx*
([*a*, *b*] #
[*c*, *d*] # []) = *mtx*
([*k*∗*a*, *k*∗*b*] #
[*k*∗*c*, *k*∗*d*] # [])
by (*simp add*: *sq-mtx-eq-iff*)

lemma *uminus-mtx2*: −*mtx*
([*a*, *b*] #
[*c*, *d*] # []) = (*mtx*
([−*a*, −*b*] #
[−*c*, −*d*] # [])::*2 sq-mtx*)
by (*simp add*: *sq-mtx-uminus-eq sq-mtx-eq-iff*)

lemma *plus-mtx2*: *mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) + *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) = ((*mtx*
([*a1*+*a2*, *b1*+*b2*] #
[*c1*+*c2*, *d1*+*d2*] # []))::*2 sq-mtx*)
by (*simp add*: *sq-mtx-eq-iff*)

lemma *minus-mtx2*: *mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) − *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) = ((*mtx*
([*a1*−*a2*, *b1*−*b2*] #
[*c1*−*c2*, *d1*−*d2*] # []))::*2 sq-mtx*)

by (*simp add*: *sq-mtx-eq-iff*)

lemma *times-mtx2*: *mtx*
([*a1*, *b1*] #
[*c1*, *d1*] # []) ∗ *mtx*
([*a2*, *b2*] #
[*c2*, *d2*] # []) = ((*mtx*
([*a1*∗*a2*+*b1*∗*c2*, *a1*∗*b2*+*b1*∗*d2*] #
[*c1*∗*a2*+*d1*∗*c2*, *c1*∗*b2*+*d1*∗*d2*] # []))::*2 sq-mtx*)
unfolding *sq-mtx-times-eq UNIV-2*
by (*simp add*: *sq-mtx-eq-iff*)

4.6.2 3x3 matrices

lemma *mtx3-to-mtx*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) =
to-mtx (χ *i j*::*3*. *if i*=*1* ∧ *j*=*1 then* a_{11}
else (*if i*=*1* ∧ *j*=*2 then* a_{12}
else (*if i*=*1* ∧ *j*=*3 then* a_{13}
else (*if i*=*2* ∧ *j*=*1 then* a_{21}
else (*if i*=*2* ∧ *j*=*2 then* a_{22}
else (*if i*=*2* ∧ *j*=*3 then* a_{23}
else (*if i*=*3* ∧ *j*=*1 then* a_{31}
else (*if i*=*3* ∧ *j*=*2 then* a_{32}
else a_{33}))))))))
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-3* **by** *force*

abbreviation *diag3* :: *real* ⇒ *real* ⇒ *real* ⇒ *3 sq-mtx*
where *diag3* ι_1 ι_2 ι_3 ≡ *mtx*
([ι_1, *0*, *0*] #
[*0*, ι_2, *0*] #
[*0*, *0*, ι_3] # [])

lemma *diag3-eq*: *diag3* (ι *1*) (ι *2*) (ι *3*) = (diag *i*. ι *i*)
apply(*simp add*: *sq-mtx-eq-iff*)
using *exhaust-3* **by** (*force simp*: *axis-def*)

lemma *one-mtx3*: (*1*::*3 sq-mtx*) = *diag3 1 1 1*
apply(*subst sq-mtx-eq-iff*)
using *exhaust-3* **by** *force*

lemma *zero-mtx3*: (*0*::*3 sq-mtx*) = *diag3 0 0 0*
by (*simp add*: *sq-mtx-eq-iff*)

lemma *scaleR-mtx3*: *k* ∗$_R$ *mtx*
([a_{11}, a_{12}, a_{13}] #

$[a_{21},\ a_{22},\ a_{23}]$ #
$[a_{31},\ a_{32},\ a_{33}]$ # []) = *mtx*
([k*a_{11}, k*a_{12}, k*a_{13}] #
[k*a_{21}, k*a_{22}, k*a_{23}] #
[k*a_{31}, k*a_{32}, k*a_{33}] # [])
by (*simp add*: *sq-mtx-eq-iff*)

lemma *plus-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) + *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([a_{11}+b_{11}, a_{12}+b_{12}, a_{13}+b_{13}] #
[a_{21}+b_{21}, a_{22}+b_{22}, a_{23}+b_{23}] #
[a_{31}+b_{31}, a_{32}+b_{32}, a_{33}+b_{33}] # [])::*3 sq-mtx*)
by (*subst sq-mtx-eq-iff*) *simp*

lemma *minus-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) − *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([a_{11}−b_{11}, a_{12}−b_{12}, a_{13}−b_{13}] #
[a_{21}−b_{21}, a_{22}−b_{22}, a_{23}−b_{23}] #
[a_{31}−b_{31}, a_{32}−b_{32}, a_{33}−b_{33}] # [])::*3 sq-mtx*)
by (*simp add*: *sq-mtx-eq-iff*)

lemma *times-mtx3*: *mtx*
([a_{11}, a_{12}, a_{13}] #
[a_{21}, a_{22}, a_{23}] #
[a_{31}, a_{32}, a_{33}] # []) * *mtx*
([b_{11}, b_{12}, b_{13}] #
[b_{21}, b_{22}, b_{23}] #
[b_{31}, b_{32}, b_{33}] # []) = (*mtx*
([a_{11}*b_{11}+a_{12}*b_{21}+a_{13}*b_{31}, a_{11}*b_{12}+a_{12}*b_{22}+a_{13}*b_{32}, a_{11}*b_{13}+a_{12}*b_{23}+a_{13}*b_{33}]
#
[a_{21}*b_{11}+a_{22}*b_{21}+a_{23}*b_{31}, a_{21}*b_{12}+a_{22}*b_{22}+a_{23}*b_{32}, a_{21}*b_{13}+a_{22}*b_{23}+a_{23}*b_{33}]
#
[a_{31}*b_{11}+a_{32}*b_{21}+a_{33}*b_{31}, a_{31}*b_{12}+a_{32}*b_{22}+a_{33}*b_{32}, a_{31}*b_{13}+a_{32}*b_{23}+a_{33}*b_{33}]
[])::*3 sq-mtx*)
unfolding *sq-mtx-times-eq*
unfolding *UNIV-3* **by** (*simp add*: *sq-mtx-eq-iff*)

end

5 Affine systems of ODEs

Affine systems of ordinary differential equations (ODEs) are those whose vector fields are linear operators. Broadly speaking, if there are functions A and B such that the system of ODEs $X'\,t = f\,(X\,t)$ turns into $X'\,t = (A\,t)\cdot(X\,t) + (B\,t)$, then it is affine. The end goal of this section is to prove that every affine system of ODEs has a unique solution, and to obtain a characterization of said solution.

theory *MTX-Flows*
imports
SQ-MTX
Hybrid-Systems-VCs.HS-ODEs

begin

5.1 Existence and uniqueness for affine systems

definition *matrix-continuous-on* :: *real set* ⇒ (*real* ⇒ ('*a*::*real-normed-algebra-1*)^'*n*^'*m*) ⇒ *bool*
where *matrix-continuous-on T A* = (∀ *t* ∈ *T*. ∀ *ε* > *0*. ∃ *δ* > *0*. ∀ *τ*∈*T*. |*τ* − *t*| < *δ* ⟶ ‖*A τ* − *A t*‖$_{op}$ ≤ *ε*)

lemma *continuous-on-matrix-vector-multl*:
assumes *matrix-continuous-on T A*
shows *continuous-on T* (λ*t*. *A t* ∗*v s*)
proof(*rule continuous-onI*, *simp add*: *dist-norm*)
fix *e t*::*real* **assume** *0* < *e* **and** *t* ∈ *T*
let *?ε* = *e*/(‖(*if s* = *0 then 1 else s*)‖)
have *?ε* > *0*
using ‹*0* < *e*› **by** *simp*
then obtain *δ* **where** *dHyp*: *δ* > *0* ∧ (∀ *τ*∈*T*. |*τ* − *t*| < *δ* ⟶ ‖*A τ* − *A t*‖$_{op}$ ≤ *?ε*)
using *assms* ‹*t* ∈ *T*› **unfolding** *dist-norm matrix-continuous-on-def* **by** *fast-force*
{fix *τ* **assume** *τ* ∈ *T* **and** |*τ* − *t*| < *δ*
have *obs*: *?ε* ∗ (‖*s*‖) = (*if s* = *0 then 0 else e*)
by *auto*
have ‖*A τ* ∗*v s* − *A t* ∗*v s*‖ = ‖(*A τ* − *A t*) ∗*v s*‖
by (*simp add*: *matrix-vector-mult-diff-rdistrib*)
also have ... ≤ (‖*A τ* − *A t*‖$_{op}$) ∗ (‖*s*‖)
using *norm-matrix-le-mult-op-norm* **by** *blast*
also have ... ≤ *?ε* ∗ (‖*s*‖)
using *dHyp* ‹*τ* ∈ *T*› ‹|*τ* − *t*| < *δ*› *mult-right-mono norm-ge-zero* **by** *blast*
finally have ‖*A τ* ∗*v s* − *A t* ∗*v s*‖ ≤ *e*
by (*subst* (*asm*) *obs*) (*metis* (*mono-tags*) ‹*0* < *e*› *less-eq-real-def order-trans*)**}**
thus ∃ *d*>*0*. ∀ *τ*∈*T*. |*τ* − *t*| < *d* ⟶ ‖*A τ* ∗*v s* − *A t* ∗*v s*‖ ≤ *e*
using *dHyp* **by** *blast*
qed

lemma *lipschitz-cond-affine*:
fixes $A :: real \Rightarrow 'a::real\text{-}normed\text{-}algebra\text{-}1\hat{}'n\hat{}'m$ **and** $T::real\ set$
defines $L \equiv Sup\ \{\|A\ t\|_{op}\ |t.\ t \in T\}$
assumes $t \in T$ **and** *bdd-above* $\{\|A\ t\|_{op}\ |t.\ t \in T\}$
shows $\|A\ t *v\ x - A\ t *v\ y\| \leq L * (\|x - y\|)$
proof−
have *obs*: $\|A\ t\|_{op} \leq Sup\ \{\|A\ t\|_{op}\ |t.\ t \in T\}$
apply(*rule cSup-upper*)
using *continuous-on-subset assms* **by** (*auto simp*: *dist-norm*)
have $\|A\ t *v\ x - A\ t *v\ y\| = \|A\ t *v\ (x - y)\|$
by (*simp add*: *matrix-vector-mult-diff-distrib*)
also have $... \leq (\|A\ t\|_{op}) * (\|x - y\|)$
using *norm-matrix-le-mult-op-norm* **by** *blast*
also have $... \leq Sup\ \{\|A\ t\|_{op}\ |t.\ t \in T\} * (\|x - y\|)$
using *obs mult-right-mono norm-ge-zero* **by** *blast*
finally show $\|A\ t *v\ x - A\ t *v\ y\| \leq L * (\|x - y\|)$
unfolding *assms* **.**
qed

lemma *local-lipschitz-affine*:
fixes $A :: real \Rightarrow 'a::real\text{-}normed\text{-}algebra\text{-}1\hat{}'n\hat{}'m$
assumes *open* T **and** *open* S
and *Ahyp*: $\bigwedge\tau\ \varepsilon.\ \varepsilon > 0 \Longrightarrow \tau \in T \Longrightarrow cball\ \tau\ \varepsilon \subseteq T \Longrightarrow bdd\text{-}above\ \{\|A\ t\|_{op}$
$|t.\ t \in cball\ \tau\ \varepsilon\}$
shows *local-lipschitz* $T\ S\ (\lambda t\ s.\ A\ t *v\ s + B\ t)$
proof(*unfold local-lipschitz-def lipschitz-on-def*, *clarsimp*)
fix $s\ t$ **assume** $s \in S$ **and** $t \in T$
then obtain *e1 e2* **where** *cball t e1* $\subseteq T$ **and** *cball s e2* $\subseteq S$ **and** *min e1 e2* $>$
0
using *open-cballE*[*OF* - ‹*open T*›] *open-cballE*[*OF* - ‹*open S*›] **by** *force*
hence *obs*: *cball t* (*min e1 e2*) $\subseteq T$
by *auto*
let *?L* = *Sup* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in cball\ t\ (min\ e1\ e2)\}$
have $\|A\ t\|_{op} \in \{\|A\ \tau\|_{op}\ |\tau.\ \tau \in cball\ t\ (min\ e1\ e2)\}$
using ‹*min e1 e2* > 0› **by** *auto*
moreover have *bdd*: *bdd-above* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in cball\ t\ (min\ e1\ e2)\}$
by (*rule Ahyp*, *simp only*: ‹*min e1 e2* > 0›, *simp-all add*: ‹$t \in T$› *obs*)
moreover have *Sup* $\{\|A\ \tau\|_{op}\ |\tau.\ \tau \in cball\ t\ (min\ e1\ e2)\} \geq 0$
apply(*rule order.trans*[*OF op-norm-ge-0*[*of A t*]])
by (*rule cSup-upper*[*OF calculation*])
moreover have $\forall\, x{\in}cball\ s\ (min\ e1\ e2) \cap S.\ \forall\, y{\in}cball\ s\ (min\ e1\ e2) \cap S.$
$\forall\, \tau{\in}cball\ t\ (min\ e1\ e2) \cap T.\ dist\ (A\ \tau *v\ x)\ (A\ \tau *v\ y) \leq ?L * dist\ x\ y$
apply(*clarify*, *simp only*: *dist-norm*, *rule lipschitz-cond-affine*)
using ‹*min e1 e2* > 0› *bdd* **by** *auto*
ultimately show $\exists\, e{>}0.\ \exists\, L.\ \forall\, t{\in}cball\ t\ e \cap T.\ 0 \leq L \wedge$
$(\forall\, x{\in}cball\ s\ e \cap S.\ \forall\, y{\in}cball\ s\ e \cap S.\ dist\ (A\ t *v\ x)\ (A\ t *v\ y) \leq L * dist\ x\ y)$
using ‹*min e1 e2* > 0› **by** *blast*
qed

lemma *picard-lindeloef-affine*:
fixes A :: *real* $\Rightarrow$ *'a*::{*banach*,*real-normed-algebra-1*,*heine-borel*}$\hat{}'n\hat{}'n$
assumes *Ahyp*: *matrix-continuous-on T A*
and $\bigwedge\tau\ \varepsilon.\ \tau \in T \Longrightarrow \varepsilon > 0 \Longrightarrow$ *bdd-above* $\{\|A\ t\|_{op}\ |t.\ dist\ \tau\ t \leq \varepsilon\}$
and *Bhyp*: *continuous-on T B* **and** *open S*
and $t_0 \in T$ **and** *Thyp*: *open T is-interval T*
shows *picard-lindeloef* $(\lambda\ t\ s.\ A\ t *v\ s + B\ t)\ T\ S\ t_0$
apply (*unfold-locales*, *simp-all add*: *assms*, *clarsimp*)
apply (*rule continuous-on-add*[*OF continuous-on-matrix-vector-multl*[*OF Ahyp*] *Bhyp*])
by (*rule local-lipschitz-affine*) (*simp-all add*: *assms*)

lemma *picard-lindeloef-autonomous-affine*:
fixes A :: *'a*::{*banach*,*real-normed-field*,*heine-borel*}$\hat{}'n\hat{}'n$
shows *picard-lindeloef* $(\lambda\ t\ s.\ A *v\ s + B)$ *UNIV UNIV* t_0
using *picard-lindeloef-affine*[*of* - $\lambda t.\ A\ \lambda t.\ B$]
unfolding *matrix-continuous-on-def* **by** (*simp only*: *diff-self op-norm0*, *auto*)

lemma *picard-lindeloef-autonomous-linear*:
fixes A :: *'a*::{*banach*,*real-normed-field*,*heine-borel*}$\hat{}'n\hat{}'n$
shows *picard-lindeloef* $(\lambda\ t.\ (*v)\ A)$ *UNIV UNIV* t_0
using *picard-lindeloef-autonomous-affine*[*of A 0*] **by** *force*

lemmas *unique-sol-autonomous-affine* = *picard-lindeloef.ivp-unique-solution*[*OF picard-lindeloef-autonomous-affine UNIV-I - subset-UNIV*]

lemmas *unique-sol-autonomous-linear* = *picard-lindeloef.ivp-unique-solution*[*OF picard-lindeloef-autonomous-linear UNIV-I - subset-UNIV*]

5.2 Flow for affine systems

5.2.1 Derivative rules for square matrices

declare *has-derivative-component* [*simp del*]

lemma *has-derivative-exp-scaleRl*[*derivative-intros*]:
fixes *f*::*real* $\Rightarrow$ *real*
assumes $D\ f \mapsto f'$ *at t within T*
shows $D\ (\lambda t.\ exp\ (f\ t *_R A)) \mapsto (\lambda h.\ f'\ h *_R (exp\ (f\ t *_R A) * A))$ *at t within T*
proof −
have *bounded-linear f'*
using *assms* **by** *auto*
then obtain *m* **where** *obs*: $f' = (\lambda h.\ h * m)$
using *real-bounded-linear* **by** *blast*
thus *?thesis*
using *vector-diff-chain-within*[*OF - exp-scaleR-has-vector-derivative-right*] *assms obs* **by** (*auto simp*: *has-vector-derivative-def comp-def*)
qed

lemma *vderiv-on-exp-scaleRlI*[*poly-derivatives*]:
assumes $D\ f = f'\ on\ T$ **and** $g' = (\lambda x.\ f'\ x *_R\ exp\ (f\ x *_R\ A) * A)$
shows $D\ (\lambda x.\ exp\ (f\ x *_R\ A)) = g'\ on\ T$
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarsimp*
by (*rule has-derivative-exp-scaleRl*, *auto simp*: *fun-eq-iff*)

lemma *has-derivative-mtx-ith*[*derivative-intros*]:
fixes *t*::*real* **and** T :: *real set*
defines $t_0 \equiv netlimit\ (at\ t\ within\ T)$
assumes $D\ A \mapsto (\lambda h.\ h *_R\ A'\ t)\ at\ t\ within\ T$
shows $D\ (\lambda t.\ A\ t\ \$\$\ i) \mapsto (\lambda h.\ h *_R\ A'\ t\ \$\$\ i)\ at\ t\ within\ T$
using *assms* **unfolding** *has-derivative-def* **apply** *safe*
apply(*force simp*: *bounded-linear-def bounded-linear-axioms-def*)
apply(*rule-tac* $F{=}\lambda\tau.\ (A\ \tau - A\ t_0 - (\tau - t_0) *_R\ A'\ t) /_R\ (\|\tau - t_0\|)$ **in** *tendsto-zero-norm-bound*)
by (*clarsimp*, *rule mult-left-mono*, *metis* (*no-types*, *lifting*) *norm-column-le-norm*

sq-mtx-minus-ith sq-mtx-scaleR-ith) *simp-all*

lemmas *has-derivative-mtx-vec-mult*[*derivative-intros*] =
bounded-bilinear.FDERIV[*OF bounded-bilinear-sq-mtx-vec-mult*]

lemma *vderiv-on-mtx-vec-multI*[*poly-derivatives*]:
assumes $D\ u = u'\ on\ T$ **and** $D\ A = A'\ on\ T$
and $g = (\lambda t.\ A\ t *_V\ u'\ t + A'\ t *_V\ u\ t\ )$
shows $D\ (\lambda t.\ A\ t *_V\ u\ t) = g\ on\ T$
using *assms* **unfolding** *has-vderiv-on-def has-vector-derivative-def* **apply** *clarify*
apply(*erule-tac* $x{=}x$ **in** *ballE*, *simp-all*)+
apply(*rule derivative-eq-intros*)
by (*auto simp*: *fun-eq-iff mtx-vec-scaleR-commute pth-6 scaleR-mtx-vec-assoc*)

lemmas *has-vderiv-on-ivl-integral* = *ivl-integral-has-vderiv-on*[*OF vderiv-on-continuous-on*]

declare *has-vderiv-on-ivl-integral* [*poly-derivatives*]

lemma *has-derivative-mtx-vec-multl*[*derivative-intros*]:
assumes $\bigwedge i\ j.\ D\ (\lambda t.\ (A\ t)\ \$\$\ i\ \$\ j) \mapsto (\lambda\tau.\ \tau *_R\ (A'\ t)\ \$\$\ i\ \$\ j)\ (at\ t\ within\ T)$
shows $D\ (\lambda t.\ A\ t *_V\ x) \mapsto (\lambda\tau.\ \tau *_R\ (A'\ t) *_V\ x)\ at\ t\ within\ T$
unfolding *sq-mtx-vec-mult-sum-cols*
apply(*rule-tac* $f'1{=}\lambda i\ \tau.\ \tau *_R\ (x\ \$\ i *_R \text{col}\ i\ (A'\ t))$ **in** *derivative-eq-intros*(*10*))
apply(*simp-all add*: *scaleR-right.sum*)
apply(*rule-tac* $g'1{=}\lambda\tau.\ \tau *_R \text{col}\ i\ (A'\ t)$ **in** *derivative-eq-intros*(*4*), *simp-all add*: *mult.commute*)
using *assms* **unfolding** *sq-mtx-col-def column-def*
by (*transfer*, *simp add*: *has-derivative-component*)

declare *has-derivative-component* [*simp*]

lemma *continuous-on-mtx-vec-multr*: *continuous-on S* $((*_V)\ A)$
by *transfer* (*simp add*: *matrix-vector-mult-linear-continuous-on*)

Isabelle automatically generates derivative rules from this subsubsection

thm *derivative-eq-intros*(*140*−)

5.2.2 Existence and uniqueness with square matrices

Finally, we can use the *exp* operation to characterize the general solutions for affine systems of ODEs. We show that they satisfy the *local-flow* locale.

lemma *continuous-on-sq-mtx-vec-multl*:
fixes $A :: real \Rightarrow ('n::finite)\ sq\text{-}mtx$
assumes *continuous-on T A*
shows *continuous-on T* $(\lambda t.\ A\ t *_V s)$
proof−
have *matrix-continuous-on T* $(\lambda t.\ to\text{-}vec\ (A\ t))$
using *assms* **by** (*force simp*: *continuous-on-iff dist-norm norm-sq-mtx-def matrix-continuous-on-def*)
hence *continuous-on T* $(\lambda t.\ to\text{-}vec\ (A\ t) *v s)$
by (*rule continuous-on-matrix-vector-multl*)
thus *?thesis*
by *transfer*
qed

lemmas *continuous-on-affine* = *continuous-on-add*[*OF continuous-on-sq-mtx-vec-multl*]

lemma *local-lipschitz-sq-mtx-affine*:
fixes $A :: real \Rightarrow ('n::finite)\ sq\text{-}mtx$
assumes *continuous-on T A open T open S*
shows *local-lipschitz T S* $(\lambda t\ s.\ A\ t *_V s + B\ t)$
proof−
have *obs*: $\bigwedge \tau\ \varepsilon.\ 0 < \varepsilon \Longrightarrow \tau \in T \Longrightarrow cball\ \tau\ \varepsilon \subseteq T \Longrightarrow bdd\text{-}above\ \{\|A\ t\|\ |t.\ t \in cball\ \tau\ \varepsilon\}$
by (*rule bdd-above-norm-cont-comp*, *rule continuous-on-subset*[*OF assms*(*1*)], *simp-all*)
hence $\bigwedge \tau\ \varepsilon.\ 0 < \varepsilon \Longrightarrow \tau \in T \Longrightarrow cball\ \tau\ \varepsilon \subseteq T \Longrightarrow bdd\text{-}above\ \{\|to\text{-}vec\ (A\ t)\|_{op}\ |t.\ t \in cball\ \tau\ \varepsilon\}$
by (*simp add*: *norm-sq-mtx-def*)
hence *local-lipschitz T S* $(\lambda t\ s.\ to\text{-}vec\ (A\ t) *v s + B\ t)$
using *local-lipschitz-affine*[*OF assms*(*2*,*3*), *of* $\lambda t.\ to\text{-}vec\ (A\ t)$] **by** *force*
thus *?thesis*
by *transfer*
qed

lemma *picard-lindeloef-sq-mtx-affine*:
assumes *continuous-on T A* **and** *continuous-on T B*
and $t_0 \in T$ *is-interval T open T* **and** *open S*

shows *picard-lindeloef* $(\lambda t\ s.\ A\ t *_V s + B\ t)\ T\ S\ t_0$
apply(*unfold-locales*, *simp-all add*: *assms*, *clarsimp*)
using *continuous-on-affine assms* **apply** *blast*
by (*rule local-lipschitz-sq-mtx-affine*, *simp-all add*: *assms*)

lemmas *sq-mtx-unique-sol-autonomous-affine* = *picard-lindeloef.ivp-unique-solution*[*OF*

picard-lindeloef-sq-mtx-affine[*OF*
continuous-on-const
continuous-on-const
UNIV-I is-interval-univ
open-UNIV open-UNIV]
UNIV-I - subset-UNIV]

lemma *has-vderiv-on-sq-mtx-linear*:
$D\ (\lambda t.\ exp\ ((t - t_0) *_R A) *_V s) = (\lambda t.\ A *_V (exp\ ((t - t_0) *_R A) *_V s))\ on\ \{t_0--t\}$
by (*rule poly-derivatives*)+ (*auto simp*: *exp-times-scaleR-commute sq-mtx-times-vec-assoc*)

lemma *has-vderiv-on-sq-mtx-affine*:
fixes t_0::*real* **and** $A :: ('a::finite)\ sq\text{-}mtx$
defines $lSol\ c\ t \equiv exp\ ((c * (t - t_0)) *_R A)$
shows $D\ (\lambda t.\ lSol\ 1\ t *_V s + lSol\ 1\ t *_V (\int_{t0}^{t} (lSol\ (-1)\ \tau *_V B)\ \partial\tau)) =$
$(\lambda t.\ A *_V (lSol\ 1\ t *_V s + lSol\ 1\ t *_V (\int_{t0}^{t} (lSol\ (-1)\ \tau *_V B)\ \partial\tau)) + B)\ on\ \{t_0--t\}$
unfolding *assms* **apply**(*simp only*: *mult.left-neutral mult-minus1*)
apply(*rule poly-derivatives*, (*force*)*?*, (*force*)*?*, (*force*)*?*, (*force*)*?*)+
by (*simp add*: *mtx-vec-mult-add-rdistl sq-mtx-times-vec-assoc*[*symmetric*]
exp-minus-inverse exp-times-scaleR-commute mult-exp-exp scale-left-distrib[*symmetric*])

lemma *autonomous-linear-sol-is-exp*:
assumes $D\ X = (\lambda t.\ A *_V X\ t)\ on\ \{t_0--t\}$ **and** $X\ t_0 = s$
shows $X\ t = exp\ ((t - t_0) *_R A) *_V s$
apply(*rule sq-mtx-unique-sol-autonomous-affine*[*of* $\lambda s.\ \{t_0--t\}$ - $t\ X\ A\ 0$])
using *assms* **apply**(*simp-all add*: *ivp-sols-def*)
using *has-vderiv-on-sq-mtx-linear* **by** *force*+

lemma *autonomous-affine-sol-is-exp-plus-int*:
assumes $D\ X = (\lambda t.\ A *_V X\ t + B)\ on\ \{t_0--t\}$ **and** $X\ t_0 = s$
shows $X\ t = exp\ ((t - t_0) *_R A) *_V s + exp\ ((t - t_0) *_R A) *_V (\int_{t0}^{t} (exp\ (-(\tau - t_0) *_R A) *_V B) \partial\tau)$
apply(*rule sq-mtx-unique-sol-autonomous-affine*[*of* $\lambda s.\ \{t_0--t\}$ - $t\ X\ A\ B$])
using *assms* **apply**(*simp-all add*: *ivp-sols-def*)
using *has-vderiv-on-sq-mtx-affine* **by** *force*+

lemma *local-flow-sq-mtx-linear*: *local-flow* $((*_V)\ A)$ *UNIV UNIV* $(\lambda t\ s.\ exp\ (t *_R A) *_V s)$
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-sq-mtx-affine*[*of* - $\lambda t.\ A\ \lambda t.\ 0$] **apply** *force*

using *has-vderiv-on-sq-mtx-linear*[*of 0*] **by** *auto*

lemma *local-flow-sq-mtx-affine*: *local-flow* (λ*s*. *A* $*_V$ *s* + *B*) *UNIV UNIV*
(λ*t s*. *exp* (*t* $*_R$ *A*) $*_V$ *s* + *exp* (*t* $*_R$ *A*) $*_V$ (∫ $_0{}^t$(*exp* (− *τ* $*_R$ *A*) $*_V$ *B*)∂*τ*))
unfolding *local-flow-def local-flow-axioms-def* **apply** *safe*
using *picard-lindeloef-sq-mtx-affine*[*of* - λ*t*. *A* λ*t*. *B*] **apply** *force*
using *has-vderiv-on-sq-mtx-affine*[*of 0 A*] **by** *auto*

end

6 Verification examples

theory *MTX-Examples*
imports
MTX-Flows
Hybrid-Systems-VCs.HS-VC-Spartan

begin

6.1 Examples

abbreviation *hoareT* :: ('*a* ⇒ *bool*) ⇒ ('*a* ⇒ '*a set*) ⇒ ('*a* ⇒ *bool*) ⇒ *bool*
(*PRE- HP - POST* - [*85*,*85*]*85*) **where** *PRE P HP X POST Q* ≡ (*P* ≤ |*X*]*Q*)

6.1.1 Verification by uniqueness.

abbreviation *mtx-circ* :: *2 sq-mtx* (*A*)
where *A* ≡ *mtx*
([*0*, *1*] #
[−*1*, *0*] # [])

abbreviation *mtx-circ-flow* :: *real* ⇒ *real^2* ⇒ *real^2* (*φ*)
where *φ t s* ≡ (*χ i*. *if i* = *1 then s$1* ∗ *cos t* + *s$2* ∗ *sin t else* − *s$1* ∗ *sin t* + *s$2* ∗ *cos t*)

lemma *mtx-circ-flow-eq*: *exp* (*t* $*_R$ *A*) $*_V$ *s* = *φ t s*
apply(*rule local-flow.eq-solution*[*OF local-flow-sq-mtx-linear*, *symmetric*, *of* - λ*s*. *UNIV*], *simp-all*)
apply(*rule ivp-solsI*, *simp-all add*: *sq-mtx-vec-mult-eq vec-eq-iff*)
unfolding *UNIV-2* **using** *exhaust-2*
by (*force intro*!: *poly-derivatives simp*: *matrix-vector-mult-def*)+

lemma *mtx-circ*:
PRE(λ*s*. *r*^{2} = (*s* $ *1*)^2 + (*s* $ *2*)^2)
HP x´=($*_V$) *A* & *G*
POST (λ*s*. *r*^{2} = (*s* $ *1*)^2 + (*s* $ *2*)^2)
apply(*subst local-flow.fbox-g-ode-subset*[*OF local-flow-sq-mtx-linear*])
unfolding *mtx-circ-flow-eq* **by** *auto*

no-notation *mtx-circ* (*A*)
and *mtx-circ-flow* (φ)

6.1.2 Flow of diagonalisable matrix.

abbreviation *mtx-hOsc* :: *real* ⇒ *real* ⇒ *2 sq-mtx* (*A*)
where *A a b* ≡ *mtx*
([*0*, *1*] #
[*a*, *b*] # [])

abbreviation *mtx-chB-hOsc* :: *real* ⇒ *real* ⇒ *2 sq-mtx* (*P*)
where *P a b* ≡ *mtx*
([*a*, *b*] #
[*1*, *1*] # [])

lemma *inv-mtx-chB-hOsc*:
a ≠ *b* ⟹ (*P a b*)$^{-1}$ = (*1*/(*a* − *b*)) $*_R$ *mtx*
([*1*, −*b*] #
[−*1*, *a*] # [])
apply(*rule sq-mtx-inv-unique*, *unfold scaleR-mtx2 times-mtx2*)
by (*simp add*: *diff-divide-distrib*[*symmetric*] *one-mtx2*)+

lemma *invertible-mtx-chB-hOsc*: *a* ≠ *b* ⟹ *mtx-invertible* (*P a b*)
apply(*rule mtx-invertibleI*[*of* - (*P a b*)$^{-1}$])
apply(*unfold inv-mtx-chB-hOsc scaleR-mtx2 times-mtx2 one-mtx2*)
by (*subst sq-mtx-eq-iff*, *simp add*: *vector-def frac-diff-eq1*)+

lemma *mtx-hOsc-diagonalizable*:
fixes *a b* :: *real*
defines ι_1 ≡ (*b* − *sqrt* (*b*^*2*+*4*∗*a*))/*2* **and** ι_2 ≡ (*b* + *sqrt* (*b*^*2*+*4*∗*a*))/*2*
assumes b^2 + *a* ∗ *4* > *0* **and** *a* ≠ *0*
shows *A a b* = *P* (−ι_2/*a*) (−ι_1/*a*) ∗ (diag *i*. *if i* = *1 then* ι_1 *else* ι_2) ∗ (*P* (−ι_2/*a*) (−ι_1/*a*))$^{-1}$
unfolding *assms* **apply**(*subst inv-mtx-chB-hOsc*)
using *assms*(*3*,*4*) **apply**(*simp-all add*: *diag2-eq*[*symmetric*])
unfolding *sq-mtx-times-eq sq-mtx-scaleR-eq UNIV-2* **apply**(*subst sq-mtx-eq-iff*)
using *exhaust-2 assms* **by** (*auto simp*: *field-simps*, *auto simp*: *field-power-simps*)

lemma *mtx-hOsc-solution-eq*:
fixes *a b* :: *real*
defines ι_1 ≡ (*b* − *sqrt* (b^2+*4*∗*a*))/*2* **and** ι_2 ≡ (*b* + *sqrt* (b^2+*4*∗*a*))/*2*
defines Φ *t* ≡ *mtx* (
[ι_2∗*exp*(*t*∗ι_1) − ι_1∗*exp*(*t*∗ι_2), *exp*(*t*∗ι_2)−*exp*(*t*∗ι_1)]#
[*a*∗*exp*(*t*∗ι_2) − *a*∗*exp*(*t*∗ι_1), ι_2∗*exp*(*t*∗ι_2)−ι_1∗*exp*(*t*∗ι_1)]#[])
assumes b^2 + *a* ∗ *4* > *0* **and** *a* ≠ *0*
shows *P* (−ι_2/*a*) (−ι_1/*a*) ∗ (diag *i*. *exp* (*t* ∗ (*if i*=*1 then* ι_1 *else* ι_2))) ∗ (*P* (−ι_2/*a*) (−ι_1/*a*))$^{-1}$
= (*1*/*sqrt* (b^2 + *a* ∗ *4*)) $*_R$ (Φ *t*)

unfolding *assms* **apply**(*subst inv-mtx-chB-hOsc*)
using *assms* **apply**(*simp-all add*: *mtx-times-scaleR-commute*, *subst sq-mtx-eq-iff*)
unfolding *UNIV-2 sq-mtx-times-eq sq-mtx-scaleR-eq sq-mtx-uminus-eq* **apply**(*simp-all add*: *axis-def*)
by (*auto simp*: *field-simps*, *auto simp*: *field-power-simps*)+

lemma *local-flow-mtx-hOsc*:
fixes *a b*
defines $\iota_1 \equiv (b - sqrt\ (b\hat{}2+4*a))/2$ **and** $\iota_2 \equiv (b + sqrt\ (b\hat{}2+4*a))/2$
defines $\Phi\ t \equiv mtx$ (
$[\iota_2*exp(t*\iota_1) - \iota_1*exp(t*\iota_2),\quad exp(t*\iota_2)-exp(t*\iota_1)]\#$
$[a*exp(t*\iota_2) - a*exp(t*\iota_1),\ \iota_2*exp(t*\iota_2)-\iota_1*exp(t*\iota_1)]\#[]$)
assumes $b^2 + a * 4 > 0$ **and** $a \neq 0$
shows *local-flow* $((*_V)\ (A\ a\ b))$ *UNIV UNIV* $(\lambda t.\ (*_V)\ ((1/sqrt\ (b^2 + a * 4))\ *_R\ \Phi\ t))$
unfolding *assms* **using** *local-flow-sq-mtx-linear*[*of A a b*] *assms*
apply(*subst* (*asm*) *exp-scaleR-diagonal2*[*OF invertible-mtx-chB-hOsc mtx-hOsc-diagonalizable*])
apply(*simp*, *simp*, *simp*)
by (*subst* (*asm*) *mtx-hOsc-solution-eq*) *simp-all*

lemma *overdamped-door-arith*:
assumes $b^2 + a * 4 > 0$ **and** $a < 0$ **and** $b \leq 0$ **and** $t \geq 0$ **and** $s1 > 0$
shows $0 \leq ((b + sqrt\ (b^2 + 4 * a)) * exp\ (t * (b - sqrt\ (b^2 + 4 * a)) / 2) / 2\ -\ (b - sqrt\ (b^2 + 4 * a)) * exp\ (t * (b + sqrt\ (b^2 + 4 * a)) / 2) / 2) * s1 / sqrt\ (b^2 + a * 4)$
proof(*subst diff-divide-distrib*[*symmetric*], *simp*)
have *f0*: $s1 / (2 * sqrt\ (b^2 + a * 4)) > 0$ (**is** $s1/?c3 > 0$)
using *assms*(*1*,*5*) **by** *simp*
have *f1*: $(b - sqrt\ (b^2 + 4 * a)) < (b + sqrt\ (b^2 + 4 * a))$ (**is** $?c2 < ?c1$)
and *f2*: $(b + sqrt\ (b^2 + 4 * a)) < 0$
using *sqrt-ge-absD*[*of b* $b^2 + 4 * a$] *assms* **by** (*force*, *linarith*)
hence *f3*: $exp\ (t * ?c2 / 2) \leq exp\ (t * ?c1 / 2)$ (**is** $exp\ ?t1 \leq exp\ ?t2$)
unfolding *exp-le-cancel-iff*
using *assms*(*4*) **by** (*case-tac t=0*, *simp-all*)
hence $?c2 * exp\ ?t2 \leq ?c2 * exp\ ?t1$
using *f1 f2 mult-le-cancel-left-pos*[*of* $-?c2$ *exp ?t1 exp ?t2*] **by** *linarith*
also have ... $< ?c1 * exp\ ?t1$
using *f1* **by** *auto*
also have... $\leq ?c1 * exp\ ?t1$
using *f1 f2* **by** *auto*
ultimately show $0 \leq (?c1 * exp\ ?t1 - ?c2 * exp\ ?t2) * s1 / ?c3$
using *f0 f1 assms*(*5*) **by** *auto*
qed

abbreviation *open-door* $s \equiv \{s.\ s\$1 > 0 \wedge s\$2 = 0\}$

lemma *overdamped-door*:
assumes $b^2 + a * 4 > 0$ **and** $a < 0$ **and** $b \leq 0$

shows *PRE* (λ*s*. *s*$*1* = *0*)
HP (*LOOP open-door*; (*x´*=((∗$_V$) (*A a b*)) & *G*) *INV* (λ*s*. *0* ≤ *s*$*1*))
POST (λ*s*. *0* ≤ *s* $ *1*)
apply(*rule fbox-loopI*, *simp-all add*: *le-fun-def*)
apply(*subst local-flow.fbox-g-ode-subset*[*OF local-flow-mtx-hOsc*[*OF assms*(*1*)]])
using *assms* **apply**(*simp-all add*: *le-fun-def fbox-def*)
unfolding *sq-mtx-scaleR-eq UNIV-2 sq-mtx-vec-mult-eq*
by (*clarsimp simp*: *overdamped-door-arith*)

no-notation *mtx-hOsc* (*A*)
and *mtx-chB-hOsc* (*P*)

6.1.3 Flow of non-diagonalisable matrix.

abbreviation *mtx-cnst-acc* :: *3 sq-mtx* (*K*)
where *K* ≡ *mtx* (
[*0*,*1*,*0*] #
[*0*,*0*,*1*] #
[*0*,*0*,*0*] # [])

lemma *pow2-scaleR-mtx-cnst-acc*: (*t* ∗$_R$ *K*)^2 = *mtx* (
[*0*,*0*,*t*^{2}] #
[*0*,*0*,*0*] #
[*0*,*0*,*0*] # [])
unfolding *power2-eq-square* **apply**(*subst sq-mtx-eq-iff*)
unfolding *sq-mtx-times-eq UNIV-3* **by** *auto*

lemma *powN-scaleR-mtx-cnst-acc*: *n* > *2* ⟹ (*t* ∗$_R$ *K*)^*n* = *0*
apply(*induct n*, *simp*, *case-tac n* ≤ *2*)
apply(*subgoal-tac n* = *2*, *erule ssubst*)
unfolding *power-Suc2 pow2-scaleR-mtx-cnst-acc sq-mtx-times-eq UNIV-3*
by (*auto simp*: *sq-mtx-eq-iff*)

lemma *exp-mtx-cnst-acc*: *exp* (*t* ∗$_R$ *K*) = ((*t* ∗$_R$ *K*)^2/$_R$ *2*) + (*t* ∗$_R$ *K*) + *1*
unfolding *exp-def* **apply**(*subst suminf-eq-sum*[*of 2*])
using *powN-scaleR-mtx-cnst-acc* **by** (*simp-all add*: *numeral-2-eq-2*)

lemma *exp-mtx-cnst-acc-simps*:
exp (*t* ∗$_R$ *K*) $$ *1* $ *1* = *1* *exp* (*t* ∗$_R$ *K*) $$ *1* $ *2* = *t* *exp* (*t* ∗$_R$ *K*) $$ *1* $ *3* = *t*^*2*/*2*
exp (*t* ∗$_R$ *K*) $$ *2* $ *1* = *0* *exp* (*t* ∗$_R$ *K*) $$ *2* $ *2* = *1* *exp* (*t* ∗$_R$ *K*) $$ *2* $ *3* = *t*
exp (*t* ∗$_R$ *K*) $$ *3* $ *1* = *0* *exp* (*t* ∗$_R$ *K*) $$ *3* $ *2* = *0* *exp* (*t* ∗$_R$ *K*) $$ *3* $ *3* = *1*
unfolding *exp-mtx-cnst-acc one-mtx3 pow2-scaleR-mtx-cnst-acc* **by** *simp-all*

lemma *exp-mtx-cnst-acc-vec-mult-eq*: *exp* (*t* ∗$_R$ *K*) ∗$_V$ *s* =
vector [*s*$*3* ∗ *t*^*2*/*2* + *s*$*2* ∗ *t* + *s*$*1*, *s*$*3* ∗ *t* + *s*$*2*, *s*$*3*]

apply(*subst exp-mtx-cnst-acc*, *subst pow2-scaleR-mtx-cnst-acc*)
apply(*simp add*: *sq-mtx-vec-mult-eq vector-def*)
unfolding *UNIV-3* **by** (*simp add*: *fun-eq-iff*)

lemma *local-flow-mtx-cnst-acc*:
local-flow ((*$_V$) *K*) *UNIV UNIV* (λ*t s*. ((*t* *$_R$ *K*)^2/$_R$ *2* + (*t* *$_R$ *K*) + *1*) *$_V$ s*)
using *local-flow-sq-mtx-linear*[*of K*] **unfolding** *exp-mtx-cnst-acc* **.**

lemma *docking-station-arith*:
assumes (*d*::*real*) > *x* **and** *v* > *0*
shows (*v* = *v*^{2} * *t* / (*2* * *d* − *2* * *x*)) ⟷ (*v* * *t* − *v*^{2} * *t*^{2} / (*4* * *d* − *4* * *x*) + *x* = *d*)
proof
assume *v* = *v*^{2} * *t* / (*2* * *d* − *2* * *x*)
hence *v* * *t* = *2* * (*d* − *x*)
using *assms* **by** (*simp add*: *eq-divide-eq power2-eq-square*)
hence *v* * *t* − *v*^{2} * *t*^{2} / (*4* * *d* − *4* * *x*) + *x* = *2* * (*d* − *x*) − *4* * (*d* − *x*)^2 / (*4* * (*d* − *x*)) + *x*
apply(*subst power-mult-distrib*[*symmetric*])
by (*erule ssubst*, *subst power-mult-distrib*, *simp*)
also have ... = *d*
apply(*simp only*: *mult-divide-mult-cancel-left-if*)
using *assms* **by** (*auto simp*: *power2-eq-square*)
finally show *v* * *t* − *v*^{2} * *t*^{2} / (*4* * *d* − *4* * *x*) + *x* = *d* **.**
next
assume *v* * *t* − *v*^{2} * *t*^{2} / (*4* * *d* − *4* * *x*) + *x* = *d*
hence *0* = *v*^{2} * *t*^{2} / (*4* * (*d* − *x*)) + (*d* − *x*) − *v* * *t*
by *auto*
hence *0* = (*4* * (*d* − *x*)) * (*v*^{2} * *t*^{2} / (*4* * (*d* − *x*)) + (*d* − *x*) − *v* * *t*)
by *auto*
also have ... = *v*^{2} * *t*^{2} + *4* * (*d* − *x*)^2 − (*4* * (*d* − *x*)) * (*v* * *t*)
using *assms* **apply**(*simp add*: *distrib-left right-diff-distrib*)
apply(*subst right-diff-distrib*[*symmetric*])+
by (*simp add*: *power2-eq-square*)
also have ... = (*v* * *t* − *2* * (*d* − *x*))^2
by (*simp only*: *power2-diff*, *auto simp*: *field-simps power2-diff*)
finally have *0* = (*v* * *t* − *2* * (*d* − *x*))^2 **.**
hence *v* * *t* = *2* * (*d* − *x*)
by *auto*
thus *v* = *v*^{2} * *t* / (*2* * *d* − *2* * *x*)
apply(*subst power2-eq-square*, *subst mult.assoc*)
apply(*erule ssubst*, *subst right-diff-distrib*[*symmetric*])
using *assms* **by** *auto*
qed

lemma *docking-station*:
assumes *d* > x_0 **and** v_0 > *0*
shows *PRE* (λ*s*. *s*$*1* = x_0 ∧ *s*$*2* = v_0)
HP ((*3* ::= (λ*s*. −(v_0 ^ *2*/(*2**(*d*−x_0))))); *x´*=(*$_V$) *K* & *G*)

POST ($\lambda s.\ s\$2 = 0 \longleftrightarrow s\$1 = d$)
apply(*clarsimp simp*: *le-fun-def local-flow.fbox-g-ode-subset*[*OF local-flow-sq-mtx-linear*[*of K*]])
unfolding *exp-mtx-cnst-acc-vec-mult-eq* **using** *assms* **by** (*simp add*: *docking-station-arith*)

no-notation *mtx-cnst-acc* (K)

end

References

[1] A. Armstrong, V. B. F. Gomes, and G. Struth. Building program construction and verification tools from algebraic principles. *Formal Aspects of Computing*, 28(2):265–293, 2016.

[2] S. Foster, J. J. H. y Munive, and G. Struth. Differential Hoare logics and refinement calculi for hybrid systems with Isabelle/HOL. arXiv:1909.05618[cs.LO], 2019.

[3] J. J. Huerta y Munive. Verification components for hybrid systems. *Archive of Formal Proofs*, 2019.

[4] J. J. Huerta y Munive. Affine systems of ODEs in Isabelle/HOL for hybrid-program verification. In *SEFM 2020*, volume 12310 of *LNCS*, pages 77–92. Springer, 2020.

[5] J. J. Huerta y Munive and G. Struth. Predicate transformer semantics for hybrid systems: Verification components for Isabelle/HOL. arXiv:1909.05618 [cs.LO], 2019.

[6] F. Immler and J. Hölzl. Ordinary differential equations. *Archive of Formal Proofs*, 2012.

[7] A. Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.