Perron-Frobenius Theorem for Spectral Radius Analysis*

Jose Divasón, Ondej Kunar, René Thiemann and Akihisa Yamada

December 14, 2021

Abstract

The spectral radius of a matrix A is the maximum norm of all eigenvalues of A. In previous work we already formalized that for a complex matrix A, the values in A^n grow polynomially in n if and only if the spectral radius is at most one. One problem with the above characterization is the determination of all *complex* eigenvalues. In case A contains only non-negative real values, a simplification is possible with the help of the Perron-Frobenius theorem, which tells us that it suffices to consider only the *real* eigenvalues of A, i.e., applying Sturm's method can decide the polynomial growth of A^n.

We formalize the Perron-Frobenius theorem based on a proof via Brouwer's fixpoint theorem, which is available in the HOL multivariate analysis (HMA) library. Since the results on the spectral radius is based on matrices in the Jordan normal form (JNF) library, we further develop a connection which allows us to easily transfer theorems between HMA and JNF. With this connection we derive the combined result: if A is a non-negative real matrix, and no real eigenvalue of A is strictly larger than one, then A^n is polynomially bounded in n.

Contents

*Supported by FWF (Austrian Science Fund) project Y757.

1 Introduction

The spectral radius of a matrix A over $\mathbb{R}$ or $\mathbb{C}$ is defined as

$$\rho(A) = \max\{|x|.\ \chi_A(x) = 0, x \in \mathbb{C}\}$$

where χ_A is the characteristic polynomial of A. It is a central notion related to the growth rate of matrix powers. A matrix A has polynomial growth, i.e., all values of A^n can be bounded polynomially in n, if and only if $\rho(A) \leq 1$. It is quite easy to see that $\rho(A) \leq 1$ is a necessary criterion,[1] but it is more complicated to argue about sufficiency. In previous work we formalized this statement via Jordan normal forms [4].

Theorem 1 (in JNF)**.** *The values in A^n are polynomially bounded in n if $\rho(A) \leq 1$.*

In order to perform the proof via Jordan normal forms, we did not use the HMA library from the distribution to represent matrices. The reason is that already the definition of a Jordan normal form is naturally expressed via block-matrices, and arbitrary block-matrices are hard to express in HMA, if at all.

[1]Let λ and v be some eigenvalue and eigenvector pair such that $|\lambda| > 1$. Then $|A^n v| = |\lambda^n v| = |\lambda|^n |v|$ grows exponentially in n, where $|w|$ denotes the component-wise application of $|\cdot|$ to vector elements of w.

The problem in applying Theorem 1 in concrete examples is the determination of all complex roots of the polynomial χ_A. For instance, one can utilize complex algebraic numbers for this purpose, which however are computationally expensive. To avoid this problem, in this work we formalize the Perron Frobenius theorem. It states that for non-negative real-valued matrices, $\rho(A)$ is an eigenvalue of A.

Theorem 2 (in HMA)**.** *If $A \in \mathbb{R}_{\geq 0}^{k \times k}$, then $\chi_A(\rho(A)) = 0$.*

We decided to perform the formalization based on the HMA library, since there is a short proof of Theorem 2 via Brouwer's fixpoint theorem [2, Section 5.2]. The latter is a well-known but complex theorem that is available in HMA, but not in the JNF library.

Eventually we want to combine both theorems to obtain:

Corollary 1. *If $A \in \mathbb{R}_{\geq 0}^{k \times k}$, then the values in A^n are polynomially bounded in n if χ_A has no real roots in the interval $(1, \infty)$.*

This criterion is computationally far less expensive – one invocation of Sturm's method on χ_A suffices. Unfortunately, we cannot immediately combine both theorems. We first have to bridge the gap between the HMA-world and the JNF-world. To this end, we develop a setup for the transfer-tool which admits to translate theorems from JNF into HMA. Moreover, using a recent extension for local type definitions within proofs [1], we also provide a translation from HMA into JNF.

With the help of these translations, we prove Corollary 1 and make it available in both HMA and JNF. (In the formalization the corollary looks a bit more complicated as it also contains an estimation of the the degree of the polynomial growth.)

2 Elimination of CARD('n)

In the following theory we provide a method which modifies theorems of the form $P[CARD('n)]$ into $n! = 0 \Longrightarrow P[n]$, so that they can more easily be applied.

Known issues: there might be problems with nested meta-implications and meta-quantification.

theory *Cancel-Card-Constraint*
imports
  HOL−Types-To-Sets.Types-To-Sets
  HOL−Library.Cardinality
begin

lemma *n-zero-nonempty*: $n \neq 0 \Longrightarrow \{0 ..< n :: nat\} \neq \{\}$ **by** *auto*

lemma *type-impl-card-n*: **assumes** $\exists$ (*Rep* :: $'a \Rightarrow$ *nat*) *Abs*. *type-definition Rep Abs* $\{0 ..< n :: nat\}$
  shows *class.finite* (*TYPE*($'a$)) $\wedge$ *CARD*($'a$) = *n*
proof −
  from *assms* **obtain** *rep* :: $'a \Rightarrow$ *nat* **and** *abs* :: *nat* $\Rightarrow 'a$ **where** *t*: *type-definition rep abs* $\{0 ..< n\}$ **by** *auto*
  have *card* (*UNIV* :: $'a$ *set*) = *card* $\{0 ..< n\}$ **using** *t* **by** (*rule type-definition.card*)
  also have ... = *n* **by** *auto*
  finally have *bn*: *CARD* ($'a$) = *n* **.**
  have *finite* (*abs* ‘ $\{0 ..< n\}$) **by** *auto*
  also have *abs* ‘ $\{0 ..< n\}$ = *UNIV* **using** *t* **by** (*rule type-definition.Abs-image*)
  finally have *class.finite* (*TYPE*($'a$)) **unfolding** *class.finite-def* **.**
  with *bn* **show** *?thesis* **by** *blast*
qed

ML-file ‹*cancel-card-constraint.ML*›

end

3 Connecting HMA-matrices with JNF-matrices

The following theories provide a connection between the type-based representation of vectors and matrices in HOL multivariate-analysis (HMA) with the set-based representation of vectors and matrices with integer indices in the Jordan-normal-form (JNF) development.

3.1 Bijections between index types of HMA and natural numbers

At the core of HMA-connect, there has to be a translation between indices of vectors and matrices, which are via index-types on the one hand, and natural numbers on the other hand.

We some unspecified bijection in our application, and not the conversions to-nat and from-nat in theory Rank-Nullity-Theorem/Mod-Type, since our definitions below do not enforce any further type constraints.

theory *Bij-Nat*
imports
  HOL−Library.Cardinality
  HOL−Library.Numeral-Type
begin

lemma *finite-set-to-list*: $\exists$ *xs* :: $'a$ :: *finite list*. *distinct xs* $\wedge$ *set xs* = *Y*

```
proof −
  have finite Y by simp
  thus ?thesis
  proof (induct Y rule: finite-induct)
    case (insert y Y)
    then obtain xs where xs: distinct xs set xs = Y by auto
    show ?case
      by (rule exI[of - y # xs], insert xs insert(2), auto)
  qed simp
qed

definition univ-list :: 'a :: finite list where
  univ-list = (SOME xs. distinct xs ∧ set xs = UNIV)

lemma univ-list: distinct (univ-list :: 'a list) set univ-list = (UNIV :: 'a :: finite
set)
proof −
  let ?xs = univ-list :: 'a list
  have distinct ?xs ∧ set ?xs = UNIV
    unfolding univ-list-def
    by (rule someI-ex, rule finite-set-to-list)
  thus distinct ?xs set ?xs = UNIV by auto
qed

definition to-nat :: 'a :: finite ⇒ nat where
  to-nat a = (SOME i. univ-list ! i = a ∧ i < length (univ-list :: 'a list))

definition from-nat :: nat ⇒ 'a :: finite where
  from-nat i = univ-list ! i

lemma length-univ-list-card: length (univ-list :: 'a :: finite list) = CARD('a)
  using distinct-card[of univ-list :: 'a list, symmetric]
  by (auto simp: univ-list)

lemma to-nat-ex: ∃! i. univ-list ! i = (a :: 'a :: finite) ∧ i < length (univ-list :: 'a
list)
proof −
  let ?ul = univ-list :: 'a list
  have a-in-set: a ∈ set ?ul unfolding univ-list by auto
  from this [unfolded set-conv-nth]
  obtain i where i1: ?ul ! i = a ∧ i < length ?ul by auto
  show ?thesis
  proof (rule ex1I, rule i1)
    fix j
    assume ?ul ! j = a ∧ j < length ?ul
    moreover have distinct ?ul by (simp add: univ-list)
    ultimately show j = i using i1 nth-eq-iff-index-eq by blast
  qed
qed
```

lemma *to-nat-less-card*: *to-nat* (a :: $'a$ :: *finite*) $<$ *CARD*($'a$)
proof −
  let *?ul* = *univ-list* :: $'a$ *list*
  from *to-nat-ex*[*of a*] **obtain** *i* **where**
  i1: *univ-list* ! $i = a \wedge i <$*length* (*univ-list*::$'a$ *list*) **by** *auto*
  show *?thesis* **unfolding** *to-nat-def*
  proof (*rule someI2*, *rule i1*)
   fix *x*
   assume *x*: *?ul* ! $x = a \wedge x <$ *length ?ul*
   thus $x <$ *CARD* ($'a$) **using** *x* **by** (*simp add*: *univ-list length-univ-list-card*)
  qed
qed

lemma *to-nat-from-nat-id*:
  assumes *i*: $i <$ *CARD*($'a$ :: *finite*)
  shows *to-nat* (*from-nat i* :: $'a$) = *i*
  unfolding *to-nat-def from-nat-def*
proof (*rule some-equality*, *simp*)
  have *l*: *length* (*univ-list*::$'a$ *list*) = *card* (*set* (*univ-list*::$'a$ *list*))
    by (*rule distinct-card*[*symmetric*], *simp add*: *univ-list*)
  thus *i2*: $i <$ *length* (*univ-list*::$'a$ *list*)
    using *i* **unfolding** *univ-list* **by** *simp*
   fix *n*
  assume *n*: (*univ-list*::$'a$ *list*) ! *n* = (*univ-list*::$'a$ *list*) ! $i \wedge n <$ *length* (*univ-list*::$'a$ *list*)
   have *d*: *distinct* (*univ-list*::$'a$ *list*) **using** *univ-list* **by** *simp*
   show $n = i$ **using** *nth-eq-iff-index-eq*[*OF d* - *i2*] *n* **by** *auto*
qed

lemma *from-nat-inj*: **assumes** *i*: $i <$ *CARD*($'a$ :: *finite*)
  and *j*: $j <$ *CARD*($'a$ :: *finite*)
  and *id*: (*from-nat i* :: $'a$) = *from-nat j*
  shows $i = j$
proof −
  from *arg-cong*[*OF id*, *of to-nat*]
  show *?thesis* **using** *i j* **by** (*simp add*: *to-nat-from-nat-id*)
qed

lemma *from-nat-to-nat-id*[*simp*]:
  (*from-nat* (*to-nat a*)) = (a::$'a$ :: *finite*)
proof −
  have *a-in-set*: $a \in$ *set* (*univ-list*) **unfolding** *univ-list* **by** *auto*
  from *this* [*unfolded set-conv-nth*]
  obtain *i* **where** *i1*: *univ-list* ! $i = a \wedge i <$*length* (*univ-list*::$'a$ *list*) **by** *auto*
  show *?thesis*
  unfolding *to-nat-def from-nat-def*
  by (*rule someI2*, *rule i1*, *simp*)
qed

lemma *to-nat-inj*[*simp*]: **assumes** *to-nat a* = *to-nat b*
shows *a* = *b*
proof −
from *to-nat-ex*[*of a*] *to-nat-ex*[*of b*]
show *a* = *b* **unfolding** *to-nat-def* **by** (*metis assms from-nat-to-nat-id*)
qed

lemma *range-to-nat*: *range* (*to-nat* :: *'a* :: *finite* ⇒ *nat*) = {*0* ..< *CARD*(*'a*)} (**is** *?l* = *?r*)
proof −
{
fix *i*
assume *i* ∈ *?l*
hence *i* ∈ *?r* **using** *to-nat-less-card*[**where** *'a* = *'a*] **by** *auto*
}
moreover
{
fix *i*
assume *i* ∈ *?r*
hence *i* < *CARD*(*'a*) **by** *auto*
from *to-nat-from-nat-id*[*OF this*]
have *i* ∈ *?l* **by** (*metis range-eqI*)
}
ultimately show *?thesis* **by** *auto*
qed

lemma *inj-to-nat*: *inj to-nat* **by** (*simp add*: *inj-on-def*)

lemma *bij-to-nat*: *bij-betw to-nat* (*UNIV* :: *'a* :: *finite set*) {*0* ..< *CARD*(*'a*)}
unfolding *bij-betw-def* **by** (*auto simp*: *range-to-nat inj-to-nat*)

lemma *numeral-nat*: (*numeral m1* :: *nat*) ∗ *numeral n1* ≡ *numeral* (*m1* ∗ *n1*)
(*numeral m1* :: *nat*) + *numeral n1* ≡ *numeral* (*m1* + *n1*) **by** *simp-all*

lemmas *card-num-simps* =
card-num1 card-bit0 card-bit1
mult-num-simps
add-num-simps
eq-num-simps
mult-Suc-right mult-0-right One-nat-def add.right-neutral
numeral-nat Suc-numeral

end

3.2 Transfer rules to convert theorems from JNF to HMA and vice-versa.

theory *HMA-Connect*
imports
Jordan-Normal-Form.Spectral-Radius
HOL−Analysis.Determinants
HOL−Analysis.Cartesian-Euclidean-Space
Bij-Nat
Cancel-Card-Constraint
HOL−Eisbach.Eisbach
begin

Prefer certain constants and lemmas without prefix.

hide-const (**open**) *Matrix.mat*
hide-const (**open**) *Matrix.row*
hide-const (**open**) *Determinant.det*

lemmas *mat-def* = *Finite-Cartesian-Product.mat-def*
lemmas *det-def* = *Determinants.det-def*
lemmas *row-def* = *Finite-Cartesian-Product.row-def*

notation *vec-index* (**infixl** \$*v* *90*)
notation *vec-nth* (**infixl** \$*h* *90*)

Forget that *'a mat*, *'a Matrix.vec*, and *'a poly* have been defined via lifting

lifting-forget *vec.lifting*
lifting-forget *mat.lifting*

lifting-forget *poly.lifting*

Some notions which we did not find in the HMA-world.

definition *eigen-vector* :: *'a::comm-ring-1 ^ 'n ^ 'n ⇒ 'a ^ 'n ⇒ 'a ⇒ bool* **where**
eigen-vector A v ev = (*v* ≠ *0* ∧ *A* ∗*v* *v* = *ev* ∗*s* *v*)

definition *eigen-value* :: *'a* :: *comm-ring-1 ^ 'n ^ 'n ⇒ 'a ⇒ bool* **where**
eigen-value A k = (∃ *v*. *eigen-vector A v k*)

definition *similar-matrix-wit*
:: *'a* :: *semiring-1 ^ 'n ^ 'n ⇒ 'a ^ 'n ^ 'n ⇒ 'a ^ 'n ^ 'n ⇒ 'a ^ 'n ^ 'n ⇒ bool*
where
similar-matrix-wit A B P Q = (*P* ∗∗ *Q* = *mat 1* ∧ *Q* ∗∗ *P* = *mat 1* ∧ *A* = *P* ∗∗ *B* ∗∗ *Q*)

definition *similar-matrix*
:: *'a* :: *semiring-1 ^ 'n ^ 'n ⇒ 'a ^ 'n ^ 'n ⇒ bool* **where**
similar-matrix A B = (∃ *P Q*. *similar-matrix-wit A B P Q*)

definition *spectral-radius* :: *complex* ^ *'n* ^ *'n* ⇒ *real* **where**
spectral-radius A = *Max* { *norm ev* | *v ev*. *eigen-vector A v ev*}

definition *Spectrum* :: *'a* :: *field* ^ *'n* ^ *'n* ⇒ *'a set* **where**
Spectrum A = *Collect* (*eigen-value A*)

definition *vec-elements-h* :: *'a* ^ *'n* ⇒ *'a set* **where**
vec-elements-h v = *range* (*vec-nth v*)

lemma *vec-elements-h-def'*: *vec-elements-h v* = {*v* $*h i* | *i*. *True*}
unfolding *vec-elements-h-def* **by** *auto*

definition *elements-mat-h* :: *'a* ^ *'nc* ^ *'nr* ⇒ *'a set* **where**
elements-mat-h A = *range* (λ (*i*,*j*). *A* $*h i* $*h j*)

lemma *elements-mat-h-def'*: *elements-mat-h A* = {*A* $*h i* $*h j* | *i j*. *True*}
unfolding *elements-mat-h-def* **by** *auto*

definition *map-vector* :: (*'a* ⇒ *'b*) ⇒ *'a* ^*'n* ⇒ *'b* ^*'n* **where**
map-vector f v ≡ χ *i*. *f* (*v* $*h i*)

definition *map-matrix* :: (*'a* ⇒ *'b*) ⇒ *'a* ^ *'n* ^ *'m* ⇒ *'b* ^ *'n* ^ *'m* **where**
map-matrix f A ≡ χ *i*. *map-vector f* (*A* $*h i*)

definition *normbound* :: *'a* :: *real-normed-field* ^ *'nc* ^ *'nr* ⇒ *real* ⇒ *bool* **where**
normbound A b ≡ ∀ *x* ∈ *elements-mat-h A*. *norm x* ≤ *b*

lemma *spectral-radius-ev-def*: *spectral-radius A* = *Max* (*norm* ‘ (*Collect* (*eigen-value A*)))
unfolding *spectral-radius-def eigen-value-def*[*abs-def*]
by (*rule arg-cong*[**where** *f* = *Max*], *auto*)

lemma *elements-mat*: *elements-mat A* = {*A* $$ (*i*,*j*) | *i j*. *i* < *dim-row A* ∧ *j* < *dim-col A*}
unfolding *elements-mat-def* **by** *force*

definition *vec-elements* :: *'a Matrix.vec* ⇒ *'a set*
where *vec-elements v* = *set* [*v* $ *i*. *i* <− [*0* ..< *dim-vec v*]]

lemma *vec-elements*: *vec-elements v* = { *v* $ *i* | *i*. *i* < *dim-vec v*}
unfolding *vec-elements-def* **by** *auto*

context includes *vec.lifting*
begin
end

definition *from-hma*$_v$ :: *'a* ^ *'n* ⇒ *'a Matrix.vec* **where**

from-hma$_v$ *v* = *Matrix.vec CARD*('*n*) (λ *i*. *v* $*h* *from-nat i*)

definition *from-hma*$_m$ :: '*a* ^ '*nc* ^ '*nr* ⇒ '*a Matrix.mat* **where**
from-hma$_m$ *a* = *Matrix.mat CARD*('*nr*) *CARD*('*nc*) (λ (*i*,*j*). *a* $*h* *from-nat i* $*h* *from-nat j*)

definition *to-hma*$_v$ :: '*a Matrix.vec* ⇒ '*a* ^ '*n* **where**
to-hma$_v$ *v* = (χ *i*. *v* $*v* *to-nat i*)

definition *to-hma*$_m$ :: '*a Matrix.mat* ⇒ '*a* ^ '*nc* ^ '*nr* **where**
to-hma$_m$ *a* = (χ *i j*. *a* $$ (*to-nat i*, *to-nat j*))

declare *vec-lambda-eta*[*simp*]

lemma *to-hma-from-hma*$_v$[*simp*]: *to-hma*$_v$ (*from-hma*$_v$ *v*) = *v*
by (*auto simp*: *to-hma*$_v$*-def from-hma*$_v$*-def to-nat-less-card*)

lemma *to-hma-from-hma*$_m$[*simp*]: *to-hma*$_m$ (*from-hma*$_m$ *v*) = *v*
by (*auto simp*: *to-hma*$_m$*-def from-hma*$_m$*-def to-nat-less-card*)

lemma *from-hma-to-hma*$_v$[*simp*]:
v ∈ *carrier-vec* (*CARD*('*n*)) ⟹ *from-hma*$_v$ (*to-hma*$_v$ *v* :: '*a* ^ '*n*) = *v*
by (*auto simp*: *to-hma*$_v$*-def from-hma*$_v$*-def to-nat-from-nat-id*)

lemma *from-hma-to-hma*$_m$[*simp*]:
A ∈ *carrier-mat* (*CARD*('*nr*)) (*CARD*('*nc*)) ⟹ *from-hma*$_m$ (*to-hma*$_m$ *A* :: '*a* ^ '*nc* ^ '*nr*) = *A*
by (*auto simp*: *to-hma*$_m$*-def from-hma*$_m$*-def to-nat-from-nat-id*)

lemma *from-hma*$_v$*-inj*[*simp*]: *from-hma*$_v$ *x* = *from-hma*$_v$ *y* ⟷ *x* = *y*
by (*intro iffI*, *insert to-hma-from-hma*$_v$[*of x*], *auto*)

lemma *from-hma*$_m$*-inj*[*simp*]: *from-hma*$_m$ *x* = *from-hma*$_m$ *y* ⟷ *x* = *y*
by(*intro iffI*, *insert to-hma-from-hma*$_m$[*of x*], *auto*)

definition *HMA-V* :: '*a Matrix.vec* ⇒ '*a* ^ '*n* ⇒ *bool* **where**
HMA-V = (λ *v w*. *v* = *from-hma*$_v$ *w*)

definition *HMA-M* :: '*a Matrix.mat* ⇒ '*a* ^ '*nc* ^ '*nr* ⇒ *bool* **where**
HMA-M = (λ *a b*. *a* = *from-hma*$_m$ *b*)

definition *HMA-I* :: *nat* ⇒ '*n* :: *finite* ⇒ *bool* **where**
HMA-I = (λ *i a*. *i* = *to-nat a*)

context includes *lifting-syntax*
begin

lemma *Domainp-HMA-V* [*transfer-domain-rule*]:
Domainp (*HMA-V* :: '*a Matrix.vec* ⇒ '*a* ^ '*n* ⇒ *bool*) = (λ *v*. *v* ∈ *carrier-vec*

$(CARD('n\ )))$
by(*intro ext iffI*, *insert from-hma-to-hma*$_v$[*symmetric*], *auto simp*: *from-hma*$_v$*-def HMA-V-def*)

lemma *Domainp-HMA-M* [*transfer-domain-rule*]:
Domainp (*HMA-M* :: $'a$ *Matrix.mat* $\Rightarrow$ $'a$ ^ $'nc$ ^ $'nr$ $\Rightarrow$ *bool*)
= (λ *A*. *A* $\in$ *carrier-mat* *CARD*($'nr$) *CARD*($'nc$))
by (*intro ext iffI*, *insert from-hma-to-hma*$_m$[*symmetric*], *auto simp*: *from-hma*$_m$*-def HMA-M-def*)

lemma *Domainp-HMA-I* [*transfer-domain-rule*]:
Domainp (*HMA-I* :: *nat* $\Rightarrow$ $'n$ :: *finite* $\Rightarrow$ *bool*) = (λ *i*. *i* < *CARD*($'n$)) (**is** *?l* = *?r*)
proof (*intro ext*)
fix *i* :: *nat*
show *?l i* = *?r i*
unfolding *HMA-I-def Domainp-iff*
by (*auto intro*: *exI*[*of - from-nat i*] *simp*: *to-nat-from-nat-id to-nat-less-card*)
qed

lemma *bi-unique-HMA-V* [*transfer-rule*]: *bi-unique HMA-V left-unique HMA-V right-unique HMA-V*
unfolding *HMA-V-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

lemma *bi-unique-HMA-M* [*transfer-rule*]: *bi-unique HMA-M left-unique HMA-M right-unique HMA-M*
unfolding *HMA-M-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

lemma *bi-unique-HMA-I* [*transfer-rule*]: *bi-unique HMA-I left-unique HMA-I right-unique HMA-I*
unfolding *HMA-I-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

lemma *right-total-HMA-V* [*transfer-rule*]: *right-total HMA-V*
unfolding *HMA-V-def right-total-def* **by** *simp*

lemma *right-total-HMA-M* [*transfer-rule*]: *right-total HMA-M*
unfolding *HMA-M-def right-total-def* **by** *simp*

lemma *right-total-HMA-I* [*transfer-rule*]: *right-total HMA-I*
unfolding *HMA-I-def right-total-def* **by** *simp*

lemma *HMA-V-index* [*transfer-rule*]: (*HMA-V* ===> *HMA-I* ===> (=)) ($\$v$) ($\$h$)
unfolding *rel-fun-def HMA-V-def HMA-I-def from-hma*$_v$*-def*
by (*auto simp*: *to-nat-less-card*)

We introduce the index function to have pointwise access to HMA-matrices by a constant. Otherwise, the transfer rule with $\lambda A\ i.\ (\$h)\ (A\ \$h\ i)$ instead of index is not applicable.

definition *index-hma* $A\ i\ j \equiv A\ \$h\ i\ \$h\ j$

lemma *HMA-M-index* [*transfer-rule*]:
($HMA\text{-}M$ ===> $HMA\text{-}I$ ===> $HMA\text{-}I$ ===> (=)) (λ A i j. A \$\$ (i,j)) *index-hma*
by (*intro rel-funI*, *simp add*: *index-hma-def to-nat-less-card HMA-M-def HMA-I-def* $from\text{-}hma_m\text{-}def$)

lemma *HMA-V-0* [*transfer-rule*]: $HMA\text{-}V$ (0_v $CARD('n)$) (0 :: $'a$ :: $zero$ ^ $'n$)
unfolding *HMA-V-def* $from\text{-}hma_v\text{-}def$ **by** *auto*

lemma *HMA-M-0* [*transfer-rule*]:
$HMA\text{-}M$ (0_m $CARD('nr)$ $CARD('nc)$) (0 :: $'a$ :: $zero$ ^ $'nc$ ^ $'nr$)
unfolding *HMA-M-def* $from\text{-}hma_m\text{-}def$ **by** *auto*

lemma *HMA-M-1*[*transfer-rule*]:
$HMA\text{-}M$ (1_m ($CARD('n)$)) (mat 1 :: $'a$::{$zero$,one}^$'n$^$'n$)
unfolding *HMA-M-def*
by (*auto simp add*: *mat-def* $from\text{-}hma_m\text{-}def$ *from-nat-inj*)

lemma $from\text{-}hma_v\text{-}add$: $from\text{-}hma_v$ v + $from\text{-}hma_v$ w = $from\text{-}hma_v$ (v + w)
unfolding $from\text{-}hma_v\text{-}def$ **by** *auto*

lemma *HMA-V-add* [*transfer-rule*]: ($HMA\text{-}V$ ===> $HMA\text{-}V$ ===> $HMA\text{-}V$) (+) (+)
unfolding *rel-fun-def HMA-V-def*
by (*auto simp*: $from\text{-}hma_v\text{-}add$)

lemma $from\text{-}hma_v\text{-}diff$: $from\text{-}hma_v$ v − $from\text{-}hma_v$ w = $from\text{-}hma_v$ (v − w)
unfolding $from\text{-}hma_v\text{-}def$ **by** *auto*

lemma *HMA-V-diff* [*transfer-rule*]: ($HMA\text{-}V$ ===> $HMA\text{-}V$ ===> $HMA\text{-}V$) (−) (−)
unfolding *rel-fun-def HMA-V-def*
by (*auto simp*: $from\text{-}hma_v\text{-}diff$)

lemma $from\text{-}hma_m\text{-}add$: $from\text{-}hma_m$ a + $from\text{-}hma_m$ b = $from\text{-}hma_m$ (a + b)
unfolding $from\text{-}hma_m\text{-}def$ **by** *auto*

lemma *HMA-M-add* [*transfer-rule*]: ($HMA\text{-}M$ ===> $HMA\text{-}M$ ===> $HMA\text{-}M$) (+) (+)
unfolding *rel-fun-def HMA-M-def*
by (*auto simp*: $from\text{-}hma_m\text{-}add$)

lemma $from\text{-}hma_m\text{-}diff$: $from\text{-}hma_m$ a − $from\text{-}hma_m$ b = $from\text{-}hma_m$ (a − b)
unfolding $from\text{-}hma_m\text{-}def$ **by** *auto*

lemma *HMA-M-diff* [*transfer-rule*]: ($HMA\text{-}M$ ===> $HMA\text{-}M$ ===> $HMA\text{-}M$) (−) (−)

unfolding *rel-fun-def HMA-M-def*
by (*auto simp*: *from-hma*$_m$*-diff*)

lemma *scalar-product*: **fixes** v :: $'a$ :: *semiring-1* $\hat{}$ $'n$
shows *scalar-prod* (*from-hma*$_v$ v) (*from-hma*$_v$ w) = *scalar-product* v w
unfolding *scalar-product-def scalar-prod-def from-hma*$_v$*-def dim-vec*
by (*simp add*: *sum.reindex*[*OF inj-to-nat*, *unfolded range-to-nat*])

lemma [*simp*]:
from-hma$_m$ (y :: $'a$ $\hat{}$ $'nc$ $\hat{}$ $'nr$) $\in$ *carrier-mat* (*CARD*($'nr$)) (*CARD*($'nc$))
dim-row (*from-hma*$_m$ (y :: $'a$ $\hat{}$ $'nc$ $\hat{}$ $'nr$)) = *CARD*($'nr$)
dim-col (*from-hma*$_m$ (y :: $'a$ $\hat{}$ $'nc$ $\hat{}$ $'nr$)) = *CARD*($'nc$)
unfolding *from-hma*$_m$*-def* **by** *simp-all*

lemma [*simp*]:
from-hma$_v$ (y :: $'a$ $\hat{}$ $'n$) $\in$ *carrier-vec* (*CARD*($'n$))
dim-vec (*from-hma*$_v$ (y :: $'a$ $\hat{}$ $'n$)) = *CARD*($'n$)
unfolding *from-hma*$_v$*-def* **by** *simp-all*

declare *rel-funI* [*intro*!]

lemma *HMA-scalar-prod* [*transfer-rule*]:
(*HMA-V* ===> *HMA-V* ===> (=)) *scalar-prod scalar-product*
by (*auto simp*: *HMA-V-def scalar-product*)

lemma *HMA-row* [*transfer-rule*]: (*HMA-I* ===> *HMA-M* ===> *HMA-V*) (λ *i a*. *Matrix.row a i*) *row*
unfolding *HMA-M-def HMA-I-def HMA-V-def*
by (*auto simp*: *from-hma*$_m$*-def from-hma*$_v$*-def to-nat-less-card row-def*)

lemma *HMA-col* [*transfer-rule*]: (*HMA-I* ===> *HMA-M* ===> *HMA-V*) (λ *i a*. *col a i*) *column*
unfolding *HMA-M-def HMA-I-def HMA-V-def*
by (*auto simp*: *from-hma*$_m$*-def from-hma*$_v$*-def to-nat-less-card column-def*)

definition *mk-mat* :: ($'i \Rightarrow 'j \Rightarrow 'c$) $\Rightarrow$ $'c\hat{}'j\hat{}'i$ **where**
mk-mat f = (χ *i j*. *f i j*)

definition *mk-vec* :: ($'i \Rightarrow 'c$) $\Rightarrow$ $'c\hat{}'i$ **where**
mk-vec f = (χ *i*. *f i*)

lemma *HMA-M-mk-mat*[*transfer-rule*]: ((*HMA-I* ===> *HMA-I* ===> (=)) ===> *HMA-M*)
(λ *f*. *Matrix.mat* (*CARD*($'nr$)) (*CARD*($'nc$)) (λ (*i*,*j*). *f i j*))
(*mk-mat* :: (($'nr \Rightarrow 'nc \Rightarrow 'a$) $\Rightarrow$ $'a\hat{}'nc\hat{}'nr$))
proof −
{
fix *x y i j*
assume *id*: $\forall$ (*ya* :: $'nr$) (*yb* :: $'nc$). (*x* (*to-nat ya*) (*to-nat yb*) :: $'a$) = *y ya yb*

and i: $i < CARD('nr)$ **and** j: $j < CARD('nc)$
from *to-nat-from-nat-id*[*OF i*] *to-nat-from-nat-id*[*OF j*] *id*[*rule-format*, *of from-nat i from-nat j*]
have $x\ i\ j = y$ (*from-nat i*) (*from-nat j*) **by** *auto*
}
thus *?thesis*
unfolding *rel-fun-def mk-mat-def HMA-M-def HMA-I-def from-hma$_m$-def* **by** *auto*
qed

lemma *HMA-M-mk-vec*[*transfer-rule*]: ((*HMA-I* ===> (=)) ===> *HMA-V*)
(λ *f*. *Matrix.vec* (*CARD*('*n*)) (λ *i*. *f i*))
(*mk-vec* :: (('*n* ⇒ '*a*) ⇒ '*a*^'*n*))
proof−
{
fix *x y i*
assume *id*: ∀ (*ya* :: '*n*). (*x* (*to-nat ya*) :: '*a*) = *y ya*
and i: $i < CARD('n)$
from *to-nat-from-nat-id*[*OF i*] *id*[*rule-format*, *of from-nat i*]
have $x\ i = y$ (*from-nat i*) **by** *auto*
}
thus *?thesis*
unfolding *rel-fun-def mk-vec-def HMA-V-def HMA-I-def from-hma$_v$-def* **by** *auto*
qed

lemma *mat-mult-scalar*: *A* ∗∗ *B* = *mk-mat* (λ *i j*. *scalar-product* (*row i A*) (*column j B*))
unfolding *vec-eq-iff matrix-matrix-mult-def scalar-product-def mk-mat-def*
by (*auto simp*: *row-def column-def*)

lemma *mult-mat-vec-scalar*: *A* ∗*v v* = *mk-vec* (λ *i*. *scalar-product* (*row i A*) *v*)
unfolding *vec-eq-iff matrix-vector-mult-def scalar-product-def mk-mat-def mk-vec-def*
by (*auto simp*: *row-def column-def*)

lemma *dim-row-transfer-rule*:
HMA-M A (*A'* :: '*a* ^ '*nc* ^ '*nr*) ⟹ (=) (*dim-row A*) (*CARD*('*nr*))
unfolding *HMA-M-def* **by** *auto*

lemma *dim-col-transfer-rule*:
HMA-M A (*A'* :: '*a* ^ '*nc* ^ '*nr*) ⟹ (=) (*dim-col A*) (*CARD*('*nc*))
unfolding *HMA-M-def* **by** *auto*

lemma *HMA-M-mult* [*transfer-rule*]: (*HMA-M* ===> *HMA-M* ===> *HMA-M*) ((∗)) ((∗∗))
proof −
{
fix *A B* :: '*a* :: *semiring-1 mat* **and** *A'* :: '*a* ^ '*n* ^ '*nr* **and** *B'* :: '*a* ^ '*nc* ^ '*n*

assume *1*[*transfer-rule*]: *HMA-M A A′ HMA-M B B′*
note [*transfer-rule*] = *dim-row-transfer-rule*[*OF 1*(*1*)] *dim-col-transfer-rule*[*OF 1*(*2*)]
have *HMA-M* (*A* ∗ *B*) (*A′* ∗∗ *B′*)
unfolding *times-mat-def mat-mult-scalar*
by (*transfer-prover-start*, *transfer-step*+, *transfer*, *auto*)
}
thus *?thesis* **by** *blast*
qed

lemma *HMA-V-smult* [*transfer-rule*]: ((=) ===> *HMA-V* ===> *HMA-V*) ($\cdot_v$) ((∗*s*))
unfolding *smult-vec-def*
unfolding *rel-fun-def HMA-V-def from-hma*$_v$*-def*
by *auto*

lemma *HMA-M-mult-vec* [*transfer-rule*]: (*HMA-M* ===> *HMA-V* ===> *HMA-V*) (($*_v$)) ((∗*v*))
proof −
{
fix *A* :: *′a* :: *semiring-1 mat* **and** *v* :: *′a Matrix.vec*
and *A′* :: *′a* ^ *′nc* ^ *′nr* **and** *v′* :: *′a* ^ *′nc*
assume *1*[*transfer-rule*]: *HMA-M A A′ HMA-V v v′*
note [*transfer-rule*] = *dim-row-transfer-rule*
have *HMA-V* (*A* $*_v$ *v*) (*A′* ∗*v* *v′*)
unfolding *mult-mat-vec-def mult-mat-vec-scalar*
by (*transfer-prover-start*, *transfer-step*+, *transfer*, *auto*)
}
thus *?thesis* **by** *blast*
qed

lemma *HMA-det* [*transfer-rule*]: (*HMA-M* ===> (=)) *Determinant.det*
(*det* :: *′a* :: *comm-ring-1* ^ *′n* ^ *′n* ⇒ *′a*)
proof −
{
fix *a* :: *′a* ^ *′n* ^ *′n*
let *?tn* = *to-nat* :: *′n* :: *finite* ⇒ *nat*
let *?fn* = *from-nat* :: *nat* ⇒ *′n*
let *?zn* = {*0*..< *CARD*(*′n*)}
let *?U* = *UNIV* :: *′n set*
let *?p1* = {*p*. *p permutes ?zn*}
let *?p2* = {*p*. *p permutes ?U*}
let *?f*= λ *p i*. *if i* ∈ *?U then ?fn* (*p* (*?tn i*)) *else i*
let *?g* = λ *p i*. *?fn* (*p* (*?tn i*))
have *fg*: ⋀ *a b c*. (*if a* ∈ *?U then b else c*) = *b* **by** *auto*
have *?p2* = *?f* ‘ *?p1*
by (*rule permutes-bij′*, *auto simp*: *to-nat-less-card to-nat-from-nat-id*)
hence *id*: *?p2* = *?g* ‘ *?p1* **by** *simp*

```
  have inj-g: inj-on ?g ?p1
    unfolding inj-on-def
  proof (intro ballI impI ext, auto)
    fix p q i
    assume p: p permutes ?zn and q: q permutes ?zn
      and id: (λ i. ?fn (p (?tn i))) = (λ i. ?fn (q (?tn i)))
    {
      fix i
      from permutes-in-image[OF p] have pi: p (?tn i) < CARD('n) by (simp add: to-nat-less-card)
      from permutes-in-image[OF q] have qi: q (?tn i) < CARD('n) by (simp add: to-nat-less-card)
      from fun-cong[OF id] have ?fn (p (?tn i)) = from-nat (q (?tn i)) .
      from arg-cong[OF this, of ?tn] have p (?tn i) = q (?tn i)
        by (simp add: to-nat-from-nat-id pi qi)
    } note id = this
    show p i = q i
    proof (cases i < CARD('n))
      case True
      hence ?tn (?fn i) = i by (simp add: to-nat-from-nat-id)
      from id[of ?fn i, unfolded this] show ?thesis .
    next
      case False
      thus ?thesis using p q unfolding permutes-def by simp
    qed
  qed
  have mult-cong: ⋀ a b c d. a = b ⟹ c = d ⟹ a * c = b * d by simp
  have sum (λ p.
    signof p * (∏ i∈?zn. a $h ?fn i $h ?fn (p i))) ?p1
    = sum (λ p. of-int (sign p) * (∏ i∈UNIV. a $h i $h p i)) ?p2
    unfolding id sum.reindex[OF inj-g]
  proof (rule sum.cong[OF refl], unfold mem-Collect-eq o-def, rule mult-cong)
    fix p
    assume p: p permutes ?zn
    let ?q = λ i. ?fn (p (?tn i))
    from id p have q: ?q permutes ?U by auto
    from p have pp: permutation p unfolding permutation-permutes by auto
    let ?ft = λ p i. ?fn (p (?tn i))
    have fin: finite ?zn by simp
    have sign p = sign ?q ∧ p permutes ?zn
    using p fin proof (induction rule: permutes-induct)
      case id
      show ?case by (auto simp: sign-id[unfolded id-def] permutes-id[unfolded id-def])
    next
      case (swap a b p)
      then have ‹permutation p›
        by (auto intro: permutes-imp-permutation)
      let ?sab = Transposition.transpose a b
```

let *?sfab* = *Transposition.transpose* (*?fn a*) (*?fn b*)
have *p-sab*: *permutation ?sab* **by** (*rule permutation-swap-id*)
have *p-sfab*: *permutation ?sfab* **by** (*rule permutation-swap-id*)
from *swap*(*4*) **have** *IH1*: *p permutes ?zn* **and** *IH2*: *sign p* = *sign* (*?ft p*) **by** *auto*
have *sab-perm*: *?sab permutes ?zn* **using** *swap*(*1*−*2*) **by** (*rule permutes-swap-id*)
from *permutes-compose*[*OF IH1 this*] **have** *perm1*: *?sab o p permutes ?zn* **.**
from *IH1* **have** *p-p1*: *p* ∈ *?p1* **by** *simp*
hence *?ft p* ∈ *?ft* ‘ *?p1* **by** (*rule imageI*)
from *this*[*folded id*] **have** *?ft p permutes ?U* **by** *simp*
hence *p-ftp*: *permutation* (*?ft p*) **unfolding** *permutation-permutes* **by** *auto*
{
fix *a b*
assume *a*: *a* ∈ *?zn* **and** *b*: *b* ∈ *?zn*
hence (*?fn a* = *?fn b*) = (*a* = *b*) **using** *swap*(*1*−*2*)
by (*auto simp*: *from-nat-inj*)
} **note** *inj* = *this*
from *inj*[*OF swap*(*1*−*2*)] **have** *id2*: *sign ?sfab* = *sign ?sab* **unfolding** *sign-swap-id* **by** *simp*
have *id*: *?ft* (*Transposition.transpose a b* ∘ *p*) = *Transposition.transpose* (*?fn a*) (*?fn b*) ∘ *?ft p*
proof
fix *c*
show *?ft* (*Transposition.transpose a b* ∘ *p*) *c* = (*Transposition.transpose* (*?fn a*) (*?fn b*) ∘ *?ft p*) *c*
proof (*cases p* (*?tn c*) = *a* ∨ *p* (*?tn c*) = *b*)
case *True*
thus *?thesis* **by** (*cases*, *auto simp add*: *swap-id-eq*)
next
case *False*
hence *neq*: *p* (*?tn c*) ≠ *a p* (*?tn c*) ≠ *b* **by** *auto*
have *pc*: *p* (*?tn c*) ∈ *?zn* **unfolding** *permutes-in-image*[*OF IH1*]
by (*simp add*: *to-nat-less-card*)
from *neq*[*folded inj*[*OF pc swap*(*1*)] *inj*[*OF pc swap*(*2*)]]
have *?fn* (*p* (*?tn c*)) ≠ *?fn a ?fn* (*p* (*?tn c*)) ≠ *?fn b* **.**
with *neq* **show** *?thesis* **by** (*auto simp*: *swap-id-eq*)
qed
qed
show *?case* **unfolding** *IH2 id sign-compose*[*OF p-sab* ‹*permutation p*›] *sign-compose*[*OF p-sfab p-ftp*] *id2*
by (*rule conjI*[*OF refl perm1*])
qed
thus *signof p* = *of-int* (*sign ?q*) **unfolding** *sign-def* **by** *auto*
show ($\prod$ *i* = *0*..<*CARD*(′*n*). *a* \$*h* *?fn i* \$*h* *?fn* (*p i*)) =
($\prod$ *i*∈*UNIV*. *a* \$*h* *i* \$*h* *?q i*) **unfolding**
range-to-nat[*symmetric*] *prod.reindex*[*OF inj-to-nat*]
by (*rule prod.cong*[*OF refl*], *unfold o-def*, *simp*)
qed
}

thus *?thesis* **unfolding** *HMA-M-def*
  by (*auto simp*: *from-hma*$_m$*-def Determinant.det-def det-def*)
qed

lemma *HMA-mat*[*transfer-rule*]: ((=) ===> *HMA-M*) (λ *k*. *k* $\cdot_m$ 1_m *CARD*('*n*))

  (*Finite-Cartesian-Product.mat* :: '*a*::*semiring-1* $\Rightarrow$ '*a*^'*n*^'*n*)
  unfolding *Finite-Cartesian-Product.mat-def*[*abs-def*] *rel-fun-def HMA-M-def*
  by (*auto simp*: *from-hma*$_m$*-def from-nat-inj*)

lemma *HMA-mat-minus*[*transfer-rule*]: (*HMA-M* ===> *HMA-M* ===> *HMA-M*)

  (λ *A B*. *A* + *map-mat uminus B*) (($-$) :: '*a* :: *group-add* ^'*nc*^'*nr* $\Rightarrow$ '*a*^'*nc*^'*nr*
$\Rightarrow$ '*a*^'*nc*^'*nr*)
  unfolding *rel-fun-def HMA-M-def from-hma*$_m$*-def* **by** *auto*

definition *mat2matofpoly* **where** *mat2matofpoly A* = (χ *i j*. [: *A* \$ *i* \$ *j* :])

definition *charpoly* **where** *charpoly-def*: *charpoly A* = *det* (*mat* (*monom 1* (*Suc 0*)) $-$ *mat2matofpoly A*)

definition *erase-mat* :: '*a* :: *zero* ^ '*nc* ^ '*nr* $\Rightarrow$ '*nr* $\Rightarrow$ '*nc* $\Rightarrow$ '*a* ^ '*nc* ^ '*nr*
  where *erase-mat A i j* = (χ *i*'. χ *j*'. *if i*' = *i* $\vee$ *j*' = *j then 0 else A* \$ *i*' \$ *j*')

definition *sum-UNIV-type* :: ('*n* :: *finite* $\Rightarrow$ '*a* :: *comm-monoid-add*) $\Rightarrow$ '*n itself*
$\Rightarrow$ '*a* **where**
  sum-UNIV-type f - = *sum f UNIV*

definition *sum-UNIV-set* :: (*nat* $\Rightarrow$ '*a* :: *comm-monoid-add*) $\Rightarrow$ *nat* $\Rightarrow$ '*a* **where**
  sum-UNIV-set f n = *sum f* {..<*n*}

definition *HMA-T* :: *nat* $\Rightarrow$ '*n* :: *finite itself* $\Rightarrow$ *bool* **where**
  HMA-T n - = (*n* = *CARD*('*n*))

lemma *HMA-mat2matofpoly*[*transfer-rule*]: (*HMA-M* ===> *HMA-M*) (λ*x*. *map-mat* (λ*a*. [:*a*:]) *x*) *mat2matofpoly*
  unfolding *rel-fun-def HMA-M-def from-hma*$_m$*-def mat2matofpoly-def* **by** *auto*

lemma *HMA-char-poly* [*transfer-rule*]:
  ((*HMA-M* :: ('*a*:: *comm-ring-1 mat* $\Rightarrow$ '*a*^'*n*^'*n* $\Rightarrow$ *bool*)) ===> (=)) *char-poly charpoly*
proof $-$
  {
    fix *A* :: '*a mat* **and** *A*' :: '*a*^'*n*^'*n*
    assume [*transfer-rule*]: *HMA-M A A*'
    hence [*simp*]: *dim-row A* = *CARD*('*n*) **by** (*simp add*: *HMA-M-def*)
    have [*simp*]: *monom 1* (*Suc 0*) = [:*0*, *1* :: '*a* :]
      by (*simp add*: *monom-Suc*)

have [*simp*]: *map-mat uminus* (*map-mat* (λa. [:*a*:]) *A*) = *map-mat* (λa. [:$-a$:]) *A*
by (*rule eq-matI*, *auto*)
have *char-poly A* = *charpoly A'*
unfolding *char-poly-def*[*abs-def*] *char-poly-matrix-def charpoly-def*[*abs-def*]
by (*transfer*, *simp*)
}
thus *?thesis* **by** *blast*
qed

lemma *HMA-eigen-vector* [*transfer-rule*]: (*HMA-M* ===> *HMA-V* ===> (=)) *eigenvector eigen-vector*
proof –
{
fix *A* :: *'a mat* **and** *v* :: *'a Matrix.vec*
and *A'* :: *'a* ^ *'n* ^ *'n* **and** *v'* :: *'a* ^ *'n* **and** *k* :: *'a*
assume *1*[*transfer-rule*]: *HMA-M A A'* **and** *2*[*transfer-rule*]: *HMA-V v v'*
hence [*simp*]: *dim-row A* = *CARD*(*'n*) *dim-vec v* = *CARD*(*'n*) **by** (*auto simp add*: *HMA-V-def HMA-M-def*)
have [*simp*]: *v* ∈ *carrier-vec CARD*(*'n*) **using** *2* **unfolding** *HMA-V-def* **by** *simp*
have *eigenvector A v* = *eigen-vector A' v'*
unfolding *eigenvector-def*[*abs-def*] *eigen-vector-def*[*abs-def*]
by (*transfer*, *simp*)
}
thus *?thesis* **by** *blast*
qed

lemma *HMA-eigen-value* [*transfer-rule*]: (*HMA-M* ===> (=) ===> (=)) *eigenvalue eigen-value*
proof –
{
fix *A* :: *'a mat* **and** *A'* :: *'a* ^ *'n* ^ *'n* **and** *k*
assume *1*[*transfer-rule*]: *HMA-M A A'*
hence [*simp*]: *dim-row A* = *CARD*(*'n*) **by** (*simp add*: *HMA-M-def*)
note [*transfer-rule*] = *dim-row-transfer-rule*[*OF 1*(*1*)]
have (*eigenvalue A k*) = (*eigen-value A' k*)
unfolding *eigenvalue-def*[*abs-def*] *eigen-value-def*[*abs-def*]
by (*transfer*, *auto simp add*: *eigenvector-def*)
}
thus *?thesis* **by** *blast*
qed

lemma *HMA-spectral-radius* [*transfer-rule*]:
(*HMA-M* ===> (=)) *Spectral-Radius.spectral-radius spectral-radius*
unfolding *Spectral-Radius.spectral-radius-def*[*abs-def*] *spectrum-def*

spectral-radius-ev-def[*abs-def*]
by *transfer-prover*

lemma *HMA-elements-mat*[*transfer-rule*]: ((*HMA-M* :: ('*a mat* ⇒ '*a* ^ '*nc* ^ '*nr* ⇒ *bool*)) ===> (=))
elements-mat elements-mat-h
proof −
{
fix *y* :: '*a* ^ '*nc* ^ '*nr* **and** *i j* :: *nat*
assume *i*: *i* < *CARD*('*nr*) **and** *j*: *j* < *CARD*('*nc*)
hence $from\text{-}hma_m$ *y* $$ (*i*, *j*) ∈ *range* (λ(*i*, *ya*). *y* $*h* *i* $*h* *ya*)
using *to-nat-from-nat-id*[*OF i*] *to-nat-from-nat-id*[*OF j*] **by** (*auto simp*: $from\text{-}hma_m\text{-}def$)
}
moreover
{
fix *y* :: '*a* ^ '*nc* ^ '*nr* **and** *a b*
have ∃ *i j*. *y* $*h* *a* $*h* *b* = $from\text{-}hma_m$ *y* $$ (*i*, *j*) ∧ *i* < *CARD*('*nr*) ∧ *j* < *CARD*('*nc*)
unfolding $from\text{-}hma_m\text{-}def$
by (*rule exI*[*of - Bij-Nat.to-nat a*], *rule exI*[*of - Bij-Nat.to-nat b*], *auto simp*: *to-nat-less-card*)
}
ultimately show *?thesis*
unfolding *elements-mat*[*abs-def*] *elements-mat-h-def*[*abs-def*] *HMA-M-def*
by *auto*
qed

lemma *HMA-vec-elements*[*transfer-rule*]: ((*HMA-V* :: ('*a Matrix.vec* ⇒ '*a* ^ '*n* ⇒ *bool*)) ===> (=))
vec-elements vec-elements-h
proof −
{
fix *y* :: '*a* ^ '*n* **and** *i* :: *nat*
assume *i*: *i* < *CARD*('*n*)
hence $from\text{-}hma_v$ *y* $ *i* ∈ *range* (*vec-nth y*)
using *to-nat-from-nat-id*[*OF i*] **by** (*auto simp*: $from\text{-}hma_v\text{-}def$)
}
moreover
{
fix *y* :: '*a* ^ '*n* **and** *a*
have ∃ *i*. *y* $*h* *a* = $from\text{-}hma_v$ *y* $ *i* ∧ *i* < *CARD*('*n*)
unfolding $from\text{-}hma_v\text{-}def$
by (*rule exI*[*of - Bij-Nat.to-nat a*], *auto simp*: *to-nat-less-card*)
}
ultimately show *?thesis*
unfolding *vec-elements*[*abs-def*] *vec-elements-h-def*[*abs-def*] *rel-fun-def HMA-V-def*
by *auto*
qed

lemma *norm-bound-elements-mat*: *norm-bound A b* = (∀ *x* ∈ *elements-mat A*. *norm x* ≤ *b*)
  unfolding *norm-bound-def elements-mat* **by** *auto*

lemma *HMA-normbound* [*transfer-rule*]:
  ((*HMA-M* :: *'a* :: *real-normed-field mat* ⇒ *'a ^ 'nc ^ 'nr* ⇒ *bool*) ===> (=) ===> (=))
  norm-bound normbound
  unfolding *normbound-def*[*abs-def*] *norm-bound-elements-mat*[*abs-def*]
  by (*transfer-prover*)

lemma *HMA-map-matrix* [*transfer-rule*]:
  ((=) ===> *HMA-M* ===> *HMA-M*) *map-mat map-matrix*
  unfolding *map-vector-def map-matrix-def*[*abs-def*] *map-mat-def*[*abs-def*] *HMA-M-def from-hma*$_m$*-def*
  by *auto*

lemma *HMA-transpose-matrix* [*transfer-rule*]:
  (*HMA-M* ===> *HMA-M*) *transpose-mat transpose*
  unfolding *transpose-mat-def transpose-def HMA-M-def from-hma*$_m$*-def* **by** *auto*

lemma *HMA-map-vector* [*transfer-rule*]:
  ((=) ===> *HMA-V* ===> *HMA-V*) *map-vec map-vector*
  unfolding *map-vector-def*[*abs-def*] *map-vec-def*[*abs-def*] *HMA-V-def from-hma*$_v$*-def*
  by *auto*

lemma *HMA-similar-mat-wit* [*transfer-rule*]:
  ((*HMA-M* :: - ⇒ *'a* :: *comm-ring-1 ^ 'n ^ 'n* ⇒ -) ===> *HMA-M* ===> *HMA-M* ===> *HMA-M* ===> (=))
  similar-mat-wit similar-matrix-wit
proof (*intro rel-funI*, *goal-cases*)
  case (*1 a A b B c C d D*)
  note [*transfer-rule*] = *this*
  hence *id*: *dim-row a* = *CARD*(*'n*) **by** (*auto simp*: *HMA-M-def*)
  have ∗: (*c* ∗ *d* = *1*$_m$ (*dim-row a*) ∧ *d* ∗ *c* = *1*$_m$ (*dim-row a*) ∧ *a* = *c* ∗ *b* ∗ *d*) =
    (*C* ∗∗ *D* = *mat 1* ∧ *D* ∗∗ *C* = *mat 1* ∧ *A* = *C* ∗∗ *B* ∗∗ *D*) **unfolding** *id*
    by (*transfer*, *simp*)
  show *?case* **unfolding** *similar-mat-wit-def Let-def similar-matrix-wit-def* ∗
    using *1* **by** (*auto simp*: *HMA-M-def*)
qed

lemma *HMA-similar-mat* [*transfer-rule*]:
  ((*HMA-M* :: - ⇒ *'a* :: *comm-ring-1 ^ 'n ^ 'n* ⇒ -) ===> *HMA-M* ===> (=))

  similar-mat similar-matrix
proof (*intro rel-funI*, *goal-cases*)
  case (*1 a A b B*)
  note [*transfer-rule*] = *this*

hence *id*: *dim-row a* = *CARD*(*'n*) **by** (*auto simp*: *HMA-M-def*)
{
fix *c d*
assume *similar-mat-wit a b c d*
hence {*c*,*d*} ⊆ *carrier-mat CARD*(*'n*) *CARD*(*'n*) **unfolding** *similar-mat-wit-def id Let-def* **by** *auto*
} note ∗ = *this*
show *?case* **unfolding** *similar-mat-def similar-matrix-def*
by (*transfer*, *insert* ∗, *blast*)
qed

lemma *HMA-spectrum*[*transfer-rule*]: (*HMA-M* ===> (=)) *spectrum Spectrum*
unfolding *spectrum-def*[*abs-def*] *Spectrum-def*[*abs-def*]
by *transfer-prover*

lemma *HMA-M-erase-mat*[*transfer-rule*]: (*HMA-M* ===> *HMA-I* ===> *HMA-I* ===> *HMA-M*) *mat-erase erase-mat*
unfolding *mat-erase-def*[*abs-def*] *erase-mat-def*[*abs-def*]
by (*auto simp*: *HMA-M-def HMA-I-def from-hma*$_m$*-def to-nat-from-nat-id intro*!: *eq-matI*)

lemma *HMA-M-sum-UNIV*[*transfer-rule*]:
((*HMA-I* ===> (=)) ===> *HMA-T* ===> (=)) *sum-UNIV-set sum-UNIV-type*
unfolding *rel-fun-def*
proof (*clarify*, *rename-tac f fT n nT*)
fix *f* **and** *fT* :: *'b* ⇒ *'a* **and** *n* **and** *nT* :: *'b itself*
assume *f*: ∀ *x y*. *HMA-I x y* ⟶ *f x* = *fT y*
and *n*: *HMA-T n nT*
let *?f* = *from-nat* :: *nat* ⇒ *'b*
let *?t* = *to-nat* :: *'b* ⇒ *nat*
from *n*[*unfolded HMA-T-def*] **have** *n*: *n* = *CARD*(*'b*) **.**
from *to-nat-from-nat-id*[**where** *'a* = *'b*, *folded n*]
have *tf*: *i* < *n* ⟹ *?t* (*?f i*) = *i* **for** *i* **by** *auto*
have *sum-UNIV-set f n* = *sum f* (*?t* ‘ *?f* ‘ {..<*n*})
unfolding *sum-UNIV-set-def*
by (*rule arg-cong*[*of - - sum f*], *insert tf*, *force*)
also have ... = *sum* (*f* ∘ *?t*) (*?f* ‘ {..<*n*})
by (*rule sum.reindex*, *insert tf n*, *auto simp*: *inj-on-def*)
also have *?f* ‘ {..<*n*} = *UNIV*
using *range-to-nat*[**where** *'a* = *'b*, *folded n*] **by** *force*
also have *sum* (*f* ∘ *?t*) *UNIV* = *sum fT UNIV*
proof (*rule sum.cong*[*OF refl*])
fix *i* :: *'b*
show (*f* ∘ *?t*) *i* = *fT i* **unfolding** *o-def*
by (*rule f*[*rule-format*], *auto simp*: *HMA-I-def*)
qed
also have ... = *sum-UNIV-type fT nT*
unfolding *sum-UNIV-type-def* **..**
finally show *sum-UNIV-set f n* = *sum-UNIV-type fT nT* **.**

qed
end

Setup a method to easily convert theorems from JNF into HMA.

method *transfer-hma* **uses** *rule* = (
  (*fold index-hma-def*)*?*,
  transfer,
  rule rule,
  (*unfold carrier-vec-def carrier-mat-def*)*?*,
  auto)

Now it becomes easy to transfer results which are not yet proven in HMA, such as:

lemma *matrix-add-vect-distrib*: $(A + B) *v\ v = A *v\ v + B *v\ v$
  by (*transfer-hma rule*: *add-mult-distrib-mat-vec*)

lemma *matrix-vector-right-distrib*: $M *v\ (v + w) = M *v\ v + M *v\ w$
  by (*transfer-hma rule*: *mult-add-distrib-mat-vec*)

lemma *matrix-vector-right-distrib-diff*: $(M :: 'a :: \textit{ring-1}\ \widehat{}\ 'nr\ \widehat{}\ 'nc) *v\ (v - w) = M *v\ v - M *v\ w$
  by (*transfer-hma rule*: *mult-minus-distrib-mat-vec*)

lemma *eigen-value-root-charpoly*:
  eigen-value A k $\longleftrightarrow$ *poly* (*charpoly* $(A :: 'a :: \textit{field}\ \widehat{}\ 'n\ \widehat{}\ 'n)$) $k = 0$
  by (*transfer-hma rule*: *eigenvalue-root-char-poly*)

lemma *finite-spectrum*: **fixes** $A :: 'a :: \textit{field}\ \widehat{}\ 'n\ \widehat{}\ 'n$
  shows *finite* (*Collect* (*eigen-value A*))
  by (*transfer-hma rule*: *card-finite-spectrum*(*1*)[*unfolded spectrum-def*])

lemma *non-empty-spectrum*: **fixes** $A :: \textit{complex}\ \widehat{}\ 'n\ \widehat{}\ 'n$
  shows *Collect* (*eigen-value A*) $\neq \{\}$
  by (*transfer-hma rule*: *spectrum-non-empty*[*unfolded spectrum-def*])

lemma *charpoly-transpose*: *charpoly* (*transpose* $A :: 'a :: \textit{field}\ \widehat{}\ 'n\ \widehat{}\ 'n$) = *charpoly* A
  by (*transfer-hma rule*: *char-poly-transpose-mat*)

lemma *eigen-value-transpose*: *eigen-value* (*transpose* $A :: 'a :: \textit{field}\ \widehat{}\ 'n\ \widehat{}\ 'n$) v = *eigen-value* $A\ v$
  unfolding *eigen-value-root-charpoly charpoly-transpose* **by** *simp*

lemma *matrix-diff-vect-distrib*: $(A - B) *v\ v = A *v\ v - B *v\ (v :: 'a :: \textit{ring-1}\ \widehat{}\ 'n)$
  by (*transfer-hma rule*: *minus-mult-distrib-mat-vec*)

lemma *similar-matrix-charpoly*: *similar-matrix* $A\ B \Longrightarrow$ *charpoly* A = *charpoly* B

by (*transfer-hma rule*: *char-poly-similar*)

lemma *pderiv-char-poly-erase-mat*: **fixes** *A* :: *'a* :: *idom* ^ *'n* ^ *'n*
shows *monom 1 1* ∗ *pderiv* (*charpoly A*) = *sum* (λ *i*. *charpoly* (*erase-mat A i i*)) *UNIV*
proof −
let *?A* = $from\text{-}hma_m$ *A*
let *?n* = *CARD*(*'n*)
have *tA*[*transfer-rule*]: *HMA-M ?A A* **unfolding** *HMA-M-def* **by** *simp*
have *tN*[*transfer-rule*]: *HMA-T ?n TYPE*(*'n*) **unfolding** *HMA-T-def* **by** *simp*
have *A*: *?A* ∈ *carrier-mat ?n ?n* **unfolding** $from\text{-}hma_m$*-def* **by** *auto*
have *id*: *sum* (λ *i*. *charpoly* (*erase-mat A i i*)) *UNIV* =
sum-UNIV-type (λ *i*. *charpoly* (*erase-mat A i i*)) *TYPE*(*'n*)
unfolding *sum-UNIV-type-def* **..**
show *?thesis* **unfolding** *id*
by (*transfer*, *insert pderiv-char-poly-mat-erase*[*OF A*], *simp add*: *sum-UNIV-set-def*)
qed

lemma *degree-monic-charpoly*: **fixes** *A* :: *'a* :: *comm-ring-1* ^ *'n* ^ *'n*
shows *degree* (*charpoly A*) = *CARD*(*'n*) ∧ *monic* (*charpoly A*)
proof (*transfer*, *goal-cases*)
case *1*
from *degree-monic-char-poly*[*OF 1*] **show** *?case* **by** *auto*
qed

end

4 Perron-Frobenius Theorem

4.1 Auxiliary Notions

We define notions like non-negative real-valued matrix, both in JNF and in HMA. These notions will be linked via HMA-connect.

theory *Perron-Frobenius-Aux*
imports *HMA-Connect*
begin

definition *real-nonneg-mat* :: *complex mat* ⇒ *bool* **where**
real-nonneg-mat A ≡ ∀ *a* ∈ *elements-mat A*. *a* ∈ ℝ ∧ *Re a* ≥ *0*

definition *real-nonneg-vec* :: *complex Matrix.vec* ⇒ *bool* **where**
real-nonneg-vec v ≡ ∀ *a* ∈ *vec-elements v*. *a* ∈ ℝ ∧ *Re a* ≥ *0*

definition *real-non-neg-vec* :: *complex* ^ *'n* ⇒ *bool* **where**
real-non-neg-vec v ≡ (∀ *a* ∈ *vec-elements-h v*. *a* ∈ ℝ ∧ *Re a* ≥ *0*)

definition *real-non-neg-mat* :: *complex* ^ *'nr* ^ *'nc* ⇒ *bool* **where**
real-non-neg-mat A ≡ (∀ *a* ∈ *elements-mat-h A*. *a* ∈ ℝ ∧ *Re a* ≥ *0*)

lemma *real-non-neg-matD*: **assumes** *real-non-neg-mat A*
  shows *A $h i $h j* $\in \mathbb{R}$ *Re* (*A $h i $h j*) $\geq$ *0*
  using *assms* **unfolding** *real-non-neg-mat-def elements-mat-h-def* **by** *auto*

definition *nonneg-mat* :: *'a* :: *linordered-idom mat* ⇒ *bool* **where**
  nonneg-mat A ≡ ∀ *a* ∈ *elements-mat A*. *a* ≥ *0*

definition *non-neg-mat* :: *'a* :: *linordered-idom* ^ *'nr* ^ *'nc* ⇒ *bool* **where**
  non-neg-mat A ≡ (∀ *a* ∈ *elements-mat-h A*. *a* ≥ *0*)

context includes *lifting-syntax*
begin

lemma *HMA-real-non-neg-mat* [*transfer-rule*]:
  ((*HMA-M* :: *complex mat* ⇒ *complex* ^ *'nc* ^ *'nr* ⇒ *bool*) ===> (=))
  real-nonneg-mat real-non-neg-mat
  unfolding *real-nonneg-mat-def*[*abs-def*] *real-non-neg-mat-def*[*abs-def*]
  by *transfer-prover*

lemma *HMA-real-non-neg-vec* [*transfer-rule*]:
  ((*HMA-V* :: *complex Matrix.vec* ⇒ *complex* ^ *'n* ⇒ *bool*) ===> (=))
  real-nonneg-vec real-non-neg-vec
  unfolding *real-nonneg-vec-def*[*abs-def*] *real-non-neg-vec-def*[*abs-def*]
  by *transfer-prover*

lemma *HMA-non-neg-mat* [*transfer-rule*]:
  ((*HMA-M* :: *'a* :: *linordered-idom mat* ⇒ *'a* ^ *'nc* ^ *'nr* ⇒ *bool*) ===> (=))
  nonneg-mat non-neg-mat
  unfolding *nonneg-mat-def*[*abs-def*] *non-neg-mat-def*[*abs-def*]
  by *transfer-prover*

end

primrec *matpow* :: *'a*::*semiring-1*^*'n*^*'n* ⇒ *nat* ⇒ *'a*^*'n*^*'n* **where**
  matpow-0:   *matpow A 0* = *mat 1* |
  matpow-Suc: *matpow A* (*Suc n*) = (*matpow A n*) ** *A*

context includes *lifting-syntax*
begin
lemma *HMA-pow-mat*[*transfer-rule*]:
 ((*HMA-M* ::*'a*::{*semiring-1*} *mat* ⇒ *'a*^*'n*^*'n* ⇒ *bool*) ===> (=) ===> *HMA-M*)
pow-mat matpow
proof −
  {
    fix *A* :: *'a mat* **and** *A'* :: *'a*^*'n*^*'n* **and** *n* :: *nat*
    assume [*transfer-rule*]: *HMA-M A A'*
    hence [*simp*]: *dim-row A* = *CARD*(*'n*) **unfolding** *HMA-M-def* **by** *simp*

have *HMA-M* (*pow-mat A n*) (*matpow A′ n*)
proof (*induct n*)
case (*Suc n*)
note [*transfer-rule*] = *this*
show *?case* **by** (*simp*, *transfer-prover*)
qed (*simp*, *transfer-prover*)
}
thus *?thesis* **by** *blast*
qed
end

lemma *trancl-image*:
(i,j) ∈ R^+ ⟹ (f i, f j) ∈ (*map-prod f f* ‘ R)$^+$
proof (*induct rule*: *trancl-induct*)
case (*step j k*)
from *step*(*2*) **have** (f j, f k) ∈ *map-prod f f* ‘ R **by** *auto*
from *step*(*3*) *this* **show** *?case* **by** *auto*
qed *auto*

lemma *inj-trancl-image*: **assumes** *inj*: *inj f*
shows (f i, f j) ∈ (*map-prod f f* ‘ R)$^+$ = ((i,j) ∈ R^+) (**is** *?l* = *?r*)
proof
assume *?r* **from** *trancl-image*[*OF this*] **show** *?l* **.**
next
assume *?l* **from** *trancl-image*[*OF this*, *of the-inv f*]
show *?r* **unfolding** *image-image prod.map-comp o-def the-inv-f-f*[*OF inj*] **by** *auto*
qed

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

lemma *norm-smult*: *norm* ((*a* :: *real*) ∗*s* *x*) = *abs a* ∗ *norm x*
unfolding *norm-vec-def*
by (*metis norm-scaleR norm-vec-def scalar-mult-eq-scaleR*)

lemma *nonneg-mat-mult*:
nonneg-mat A ⟹ *nonneg-mat B* ⟹ A ∈ *carrier-mat nr n*
⟹ B ∈ *carrier-mat n nc* ⟹ *nonneg-mat* (A ∗ B)
unfolding *nonneg-mat-def*
by (*auto simp*: *elements-mat-def scalar-prod-def intro*!: *sum-nonneg*)

lemma *nonneg-mat-power*: **assumes** A ∈ *carrier-mat n n nonneg-mat A*
shows *nonneg-mat* (A $\widehat{}_m$ k)
proof (*induct k*)
case *0*
thus *?case* **by** (*auto simp*: *nonneg-mat-def*)
next
case (*Suc k*)

from *nonneg-mat-mult*[*OF this assms*(*2*) - *assms*(*1*), *of n*] *assms*(*1*)
show *?case* **by** *auto*
qed

lemma *nonneg-matD*: **assumes** *nonneg-mat A*
and $i <$ *dim-row A* **and** $j <$ *dim-col A*
shows *A* $$ (*i*,*j*) ≥ 0
using *assms* **unfolding** *nonneg-mat-def elements-mat* **by** *auto*

lemma (**in** *comm-ring-hom*) *similar-mat-wit-hom*: **assumes**
similar-mat-wit A B C D
shows *similar-mat-wit* (mat_h *A*) (mat_h *B*) (mat_h *C*) (mat_h *D*)
proof −
obtain *n* **where** *n*: *n* = *dim-row A* **by** *auto*
note ∗ = *similar-mat-witD*[*OF n assms*]
from ∗ **have** [*simp*]: *dim-row C* = *n* **by** *auto*
note *C* = ∗(*6*) **note** *D* = ∗(*7*)
note *id* = *mat-hom-mult*[*OF C D*] *mat-hom-mult*[*OF D C*]
note ∗∗ = ∗(*1*−*3*)[*THEN arg-cong*[*of* - - mat_h], *unfolded id*]
note *mult* = *mult-carrier-mat*[*of* - *n n*]
note *hom-mult* = *mat-hom-mult*[*of* - *n n* - *n*]
show *?thesis* **unfolding** *similar-mat-wit-def Let-def* **unfolding** ∗∗(*3*) **using**
∗∗(*1*,*2*)
by (*auto simp*: *n*[*symmetric*] *hom-mult simp*: ∗(*4*−) *mult*)
qed

lemma (**in** *comm-ring-hom*) *similar-mat-hom*:
similar-mat A B ⟹ *similar-mat* (mat_h *A*) (mat_h *B*)
using *similar-mat-wit-hom*[*of A B C D* **for** *C D*]
by (*smt similar-mat-def*)

lemma *det-dim-1*: **assumes** *A*: *A* ∈ *carrier-mat n n*
and *n*: *n* = *1*
shows *Determinant.det A* = *A* $$ (*0*,*0*)
by (*subst laplace-expansion-column*[*OF A*[*unfolded n*], *of 0*], *insert A n*,
auto simp: *cofactor-def mat-delete-def*)

lemma *det-dim-2*: **assumes** *A*: *A* ∈ *carrier-mat n n*
and *n*: *n* = *2*
shows *Determinant.det A* = *A* $$ (*0*,*0*) ∗ *A* $$ (*1*,*1*) − *A* $$ (*0*,*1*) ∗ *A* $$ (*1*,*0*)
proof −
have *set*: ($\sum$ *i*<(*2* :: *nat*). *f i*) = *f 0* + *f 1* **for** *f*
by (*subst sum.cong*[*of* - {*0*,*1*} *f f*], *auto*)
show *?thesis*
apply (*subst laplace-expansion-column*[*OF A*[*unfolded n*], *of 0*], *insert A n*,
auto simp: *cofactor-def mat-delete-def set*)
apply (*subst* (*1 2*) *det-dim-1*, *auto*)
done
qed

lemma *jordan-nf-root-char-poly*: **fixes** A :: $'a$ :: {*semiring-no-zero-divisors*, *idom*} *mat*
  assumes *jordan-nf A n-as*
  and (*m*, *lam*) ∈ *set n-as*
shows *poly* (*char-poly A*) *lam* = *0*
proof −
  from *assms* **have** *m0*: *m* ≠ *0* **unfolding** *jordan-nf-def* **by** *force*
  from *split-list*[*OF assms*(*2*)] **obtain** *as bs* **where** *nas*: *n-as* = *as* @ (*m*, *lam*) # *bs* **by** *auto*
  show *?thesis* **using** *m0*
    unfolding *jordan-nf-char-poly*[*OF assms*(*1*)] *nas poly-prod-list prod-list-zero-iff* **by** (*auto simp*: *o-def*)
qed

lemma *inverse-power-tendsto-zero*:
  (λx. *inverse* ((*of-nat x* :: $'a$ :: *real-normed-div-algebra*) ^ *Suc d*)) ⟶ *0*
proof (*rule filterlim-compose*[*OF tendsto-inverse-0*],
  intro filterlim-at-infinity[*THEN iffD2*, *of 0*] *allI impI*, *goal-cases*)
  case (*2 r*)
  let *?r* = *nat* (*ceiling r*) + *1*
  show *?case*
  proof (*intro eventually-sequentiallyI*[*of ?r*], *unfold norm-power norm-of-nat*)
    fix *x*
    assume *r*: *?r* ≤ *x*
    hence *x1*: *real x* ≥ *1* **by** *auto*
    have *r* ≤ *real ?r* **by** *linarith*
    also have ... ≤ *x* **using** *r* **by** *auto*
    also have ... ≤ *real x* ^ *Suc d* **using** *x1* **by** *simp*
    finally show *r* ≤ *real x* ^ *Suc d* **.**
  qed
qed *simp*

lemma *inverse-of-nat-tendsto-zero*:
  (λx. *inverse* (*of-nat x* :: $'a$ :: *real-normed-div-algebra*)) ⟶ *0*
  using *inverse-power-tendsto-zero*[*of 0*] **by** *auto*

lemma *poly-times-exp-tendsto-zero*: **assumes** *b*: *norm* (*b* :: $'a$ :: *real-normed-field*) < *1*
  shows (λ *x*. *of-nat x* ^ *k* ∗ *b* ^ *x*) ⟶ *0*
proof (*cases b* = *0*)
  case *False*
  define *nla* **where** *nla* = *norm b*
  define *s* **where** *s* = *sqrt nla*
  from *b False* **have** *nla*: *0* < *nla nla* < *1* **unfolding** *nla-def* **by** *auto*
  hence *s*: *0* < *s s* < *1* **unfolding** *s-def* **by** *auto*
  {
    fix *x*

have $s \hat{\ } x * s \hat{\ } x = sqrt\ (nla\ \hat{\ }\ (2 * x))$
unfolding *s-def power-add*[*symmetric*]
unfolding *real-sqrt-power*[*symmetric*]
by (*rule arg-cong*[*of* - - λ *x. sqrt* (*nla* ^ *x*)], *simp*)
also have ... $= nla \hat{\ } x$ **unfolding** *power-mult real-sqrt-power*
using *nla* **by** *simp*
finally have $nla \hat{\ } x = s \hat{\ } x * s \hat{\ } x$ **by** *simp*
} **note** *nla-s* = *this*
show ($\lambda x.$ *of-nat* x ^ $k * b$ ^ x) $\longrightarrow$ *0*
proof (*rule tendsto-norm-zero-cancel*, *unfold norm-mult norm-power norm-of-nat nla-def*[*symmetric*] *nla-s*
mult.assoc[*symmetric*])
from *poly-exp-constant-bound*[*OF s*, *of 1 k*] **obtain** *p* **where**
p: *real* x ^ $k * s \hat{\ } x \le p$ **for** *x* **by** (*auto simp*: *ac-simps*)
have *norm* (*real* x ^ $k * s$ ^ x) = *real* x ^ $k * s \hat{\ } x$ **for** *x* **using** *s* **by** *auto*
with *p* **have** *p*: *norm* (*real* x ^ $k * s$ ^ x) $\le p$ **for** *x* **by** *auto*
from *s* **have** *s*: *norm* $s < 1$ **by** *auto*
show ($\lambda x.$ *real* x ^ $k * s$ ^ $x * s$ ^ x) $\longrightarrow$ *0*
by (*rule lim-null-mult-left-bounded*[*OF* - *LIMSEQ-power-zero*[*OF s*], *of* - *p*], *insert p*, *auto*)
qed
next
case *True*
show *?thesis* **unfolding** *True*
by (*subst tendsto-cong*[*of* - λ *x. 0*], *rule eventually-sequentiallyI*[*of 1*], *auto*)
qed

lemma (**in** *linorder-topology*) *tendsto-Min*: **assumes** *I*: $I \neq \{\}$ **and** *fin*: *finite I*
shows ($\bigwedge i.\ i \in I \Longrightarrow$ ($f\ i \longrightarrow a\ i$) F) $\Longrightarrow$ (($\lambda x.$ *Min* ((λ *i. f i x*) ‘ *I*)) $\longrightarrow$
(*Min* (*a* ‘ *I*) :: '*a*)) *F*
using *fin I*
proof (*induct rule*: *finite-induct*)
case (*insert i I*)
hence *i*: ($f\ i \longrightarrow a\ i$) *F* **by** *auto*
show *?case*
proof (*cases* $I = \{\}$)
case *True*
show *?thesis* **unfolding** *True* **using** *i* **by** *auto*
next
case *False*
have $*$: *Min* (*a* ‘ *insert i I*) = *min* (*a i*) (*Min* (*a* ‘ *I*)) **using** *False insert*(*1*) **by** *auto*
have $**$: ($\lambda x.$ *Min* (($\lambda i.$ *f i x*) ‘ *insert i I*)) = ($\lambda x.$ *min* (*f i x*) (*Min* (($\lambda i.$ *f i x*) ‘ *I*)))
using *False insert*(*1*) **by** *auto*
have *IH*: (($\lambda x.$ *Min* (($\lambda i.$ *f i x*) ‘ *I*)) $\longrightarrow$ *Min* (*a* ‘ *I*)) *F*
using *insert*(*3*)[*OF insert*(*4*) *False*] **by** *auto*
show *?thesis* **unfolding** $*$ $**$

```
    by (auto intro!: tendsto-min i IH)
  qed
qed simp

lemma tendsto-mat-mult [tendsto-intros]:
  (f ⟶ a) F ⟹ (g ⟶ b) F ⟹ ((λx. f x ** g x) ⟶ a ** b) F
  for f :: 'a ⇒ 'b :: {semiring-1, real-normed-algebra} ^'n1 ^'n2
  unfolding matrix-matrix-mult-def[abs-def] by (auto intro!: tendsto-intros)

lemma tendsto-matpower [tendsto-intros]: (f ⟶ a) F ⟹ ((λx. matpow (f x)
n) ⟶ matpow a n) F
  for f :: 'a ⇒ 'b :: {semiring-1, real-normed-algebra} ^'n ^'n
  by (induct n, simp-all add: tendsto-mat-mult)

lemma continuous-matpow: continuous-on R (λ A :: 'a :: {semiring-1, real-normed-algebra-1}
^'n ^'n. matpow A n)
  unfolding continuous-on-def by (auto intro!: tendsto-intros)

lemma vector-smult-distrib: (A *v ((a :: 'a :: comm-ring-1) *s x)) = a *s ((A *v
x))
  unfolding matrix-vector-mult-def vector-scalar-mult-def
  by (simp add: ac-simps sum-distrib-left)

instance real :: ordered-semiring-strict
  by (intro-classes, auto)

lemma poly-tendsto-pinfty: fixes p :: real poly
  assumes lead-coeff p > 0 degree p ≠ 0
  shows poly p ⟶ ∞
  unfolding Lim-PInfty
proof
  fix b
  show ∃ N. ∀ n≥N. ereal b ≤ ereal (poly p (real n))
    unfolding ereal-less-eq using poly-pinfty-ge[OF assms, of b]
    by (meson of-nat-le-iff order-trans real-arch-simple)
qed

lemma div-lt-nat: (j :: nat) < x * y ⟹ j div x < y
  by (simp add: less-mult-imp-div-less mult.commute)


definition diagvector :: ('n ⇒ 'a :: semiring-0) ⇒ 'a ^'n ^'n where
  diagvector x = (χ i. χ j. if i = j then x i else 0)

lemma diagvector-mult-vector[simp]: diagvector x *v y = (χ i. x i * y $ i)
  unfolding diagvector-def matrix-vector-mult-def vec-eq-iff vec-lambda-beta
proof (rule, goal-cases)
  case (1 i)
  show ?case by (subst sum.remove[of - i], auto)
```

qed

lemma *diagvector-mult-left*: *diagvector x* $**$ *A* = (χ *i j*. *x i* $*$ *A* \$ *i* \$ *j*) (**is** *?A* = *?B*)
  unfolding *vec-eq-iff*
proof (*intro allI*)
  fix *i j*
  show *?A* \$*h* *i* \$*h* *j* = *?B* \$*h* *i* \$*h* *j*
   unfolding *map-vector-def diagvector-def matrix-matrix-mult-def vec-lambda-beta*
    by (*subst sum.remove*[*of - i*], *auto*)
qed

lemma *diagvector-mult-right*: *A* $**$ *diagvector x* = (χ *i j*. *A* \$ *i* \$ *j* $*$ *x j*) (**is** *?A* = *?B*)
  unfolding *vec-eq-iff*
proof (*intro allI*)
  fix *i j*
  show *?A* \$*h* *i* \$*h* *j* = *?B* \$*h* *i* \$*h* *j*
   unfolding *map-vector-def diagvector-def matrix-matrix-mult-def vec-lambda-beta*
    by (*subst sum.remove*[*of - j*], *auto*)
qed

lemma *diagvector-mult*[*simp*]: *diagvector x* $**$ *diagvector y* = *diagvector* (λ *i*. *x i* $*$ *y i*)
 unfolding *diagvector-mult-left* **unfolding** *diagvector-def* **by** (*auto simp*: *vec-eq-iff*)

lemma *diagvector-const*[*simp*]: *diagvector* (λ *x*. *k*) = *mat k*
  unfolding *diagvector-def mat-def* **by** *auto*

lemma *diagvector-eq-mat*: *diagvector x* = *mat a* $\longleftrightarrow$ *x* = (λ *x*. *a*)
  unfolding *diagvector-def mat-def* **by** (*auto simp*: *vec-eq-iff*)

lemma *cmod-eq-Re*: **assumes** *cmod x* = *Re x*
  shows *of-real* (*Re x*) = *x*
proof (*cases Im x* = *0*)
  case *False*
  hence (*cmod x*)^*2* $\neq$ (*Re x*)^*2* **unfolding** *norm-complex-def* **by** *simp*
  from *this*[*unfolded assms*] **show** *?thesis* **by** *auto*
qed (*cases x*, *auto simp*: *norm-complex-def complex-of-real-def*)

hide-fact (**open**) *Matrix.vec-eq-iff*

no-notation
  vec-index (**infixl** \$ *100*)

lemma *spectral-radius-ev*:
  $\exists$ *ev v*. *eigen-vector A v ev* $\wedge$ *norm ev* = *spectral-radius A*
proof −
  from *non-empty-spectrum*[*of A*] *finite-spectrum*[*of A*] **have**

spectral-radius A ∈ *norm* ‘ (*Collect* (*eigen-value A*))
unfolding *spectral-radius-ev-def* **by** *auto*
thus *?thesis* **unfolding** *eigen-value-def*[*abs-def*] **by** *auto*
qed

lemma *spectral-radius-max*: **assumes** *eigen-value A v*
shows *norm v* ≤ *spectral-radius A*
proof −
from *assms* **have** *norm v* ∈ *norm* ‘ (*Collect* (*eigen-value A*)) **by** *auto*
from *Max-ge*[*OF - this*, *folded spectral-radius-ev-def*]
finite-spectrum[*of A*] **show** *?thesis* **by** *auto*
qed

For Perron-Frobenius it is useful to use the linear norm, and not the Euclidean norm.

definition *norm1* :: *'a* :: *real-normed-field* ^ *'n* ⇒ *real* **where**
norm1 v = (∑ *i*∈*UNIV*. *norm* (*v* $ *i*))

lemma *norm1-ge-0*: *norm1 v* ≥ *0* **unfolding** *norm1-def*
by (*rule sum-nonneg*, *auto*)

lemma *norm1-0*[*simp*]: *norm1 0* = *0* **unfolding** *norm1-def* **by** *auto*

lemma *norm1-nonzero*: **assumes** *v* ≠ *0*
shows *norm1 v* > *0*
proof −
from ‹*v* ≠ *0*› **obtain** *i* **where** *vi*: *v* $ *i* ≠ *0* **unfolding** *vec-eq-iff*
using *Finite-Cartesian-Product.vec-eq-iff zero-index* **by** *force*
have *sum* (λ *i*. *norm* (*v* $ *i*)) (*UNIV* − {*i*}) ≥ *0*
by (*rule sum-nonneg*, *auto*)
moreover have *norm* (*v* $ *i*) > *0* **using** *vi* **by** *auto*
ultimately
have *0* < *norm* (*v* $ *i*) + *sum* (λ *i*. *norm* (*v* $ *i*)) (*UNIV* − {*i*}) **by** *arith*
also have . . . = *norm1 v* **unfolding** *norm1-def*
by (*simp add*: *sum.remove*)
finally show *norm1 v* > *0* **.**
qed

lemma *norm1-0-iff*[*simp*]: (*norm1 v* = *0*) = (*v* = *0*)
using *norm1-0 norm1-nonzero* **by** (*cases v* = *0*, *force+*)

lemma *norm1-scaleR*[*simp*]: *norm1* (*r* $*_R$ *v*) = *abs r* ∗ *norm1 v* **unfolding** *norm1-def sum-distrib-left*
by (*rule sum.cong*, *auto*)

lemma *abs-norm1*[*simp*]: *abs* (*norm1 v*) = *norm1 v* **using** *norm1-ge-0*[*of v*] **by** *arith*

lemma *normalize-eigen-vector*: **assumes** *eigen-vector* (*A* :: *'a* :: *real-normed-field*

$\hat{}$ $'n$ $\hat{}$ $'n$) *v ev*
shows *eigen-vector A* ((*1* / *norm1 v*) $*_R$ *v*) *ev norm1* ((*1* / *norm1 v*) $*_R$ *v*) = *1*
proof −
let *?v* = (*1* / *norm1 v*) $*_R$ *v*
from *assms*[*unfolded eigen-vector-def*]
have *nz*: *v* ≠ *0* **and** *id*: *A* ∗*v v* = *ev* ∗*s v* **by** *auto*
from *nz* **have** *norm1*: *norm1 v* ≠ *0* **by** *auto*
thus *norm1 ?v* = *1* **by** *simp*
from *norm1 nz* **have** *nz*: *?v* ≠ *0* **by** *auto*
have *A* ∗*v ?v* = (*1* / *norm1 v*) $*_R$ (*A* ∗*v v*)
by (*auto simp*: *vec-eq-iff matrix-vector-mult-def real-vector.scale-sum-right*)
also have *A* ∗*v v* = *ev* ∗*s v* **unfolding** *id* **..**
also have (*1* / *norm1 v*) $*_R$ (*ev* ∗*s v*) = *ev* ∗*s ?v*
by (*auto simp*: *vec-eq-iff*)
finally show *eigen-vector A ?v ev* **using** *nz* **unfolding** *eigen-vector-def* **by** *auto*
qed

lemma *norm1-cont*[*simp*]: *isCont norm1 v* **unfolding** *norm1-def*[*abs-def*] **by** *auto*

lemma *norm1-ge-norm*: *norm1 v* ≥ *norm v* **unfolding** *norm1-def norm-vec-def*
by (*rule L2-set-le-sum*, *auto*)

The following continuity lemmas have been proven with hints from Fabian Immler.

lemma *tendsto-matrix-vector-mult*[*tendsto-intros*]:
((∗*v*) (*A* :: *'a* :: *real-normed-algebra-1* $\hat{}$ *'n* $\hat{}$ *'k*) ⟶ *A* ∗*v v*) (*at v within S*)
unfolding *matrix-vector-mult-def*[*abs-def*]
by (*auto intro*!: *tendsto-intros*)

lemma *tendsto-matrix-matrix-mult*[*tendsto-intros*]:
((∗∗) (*A* :: *'a* :: *real-normed-algebra-1* $\hat{}$ *'n* $\hat{}$ *'k*) ⟶ *A* ∗∗ *B*) (*at B within S*)
unfolding *matrix-matrix-mult-def*[*abs-def*]
by (*auto intro*!: *tendsto-intros*)

lemma *matrix-vect-scaleR*: (*A* :: *'a* :: *real-normed-algebra-1* $\hat{}$ *'n* $\hat{}$ *'k*) ∗*v* (*a* $*_R$ *v*) = *a* $*_R$ (*A* ∗*v v*)
unfolding *vec-eq-iff*
by (*auto simp*: *matrix-vector-mult-def scaleR-vec-def scaleR-sum-right*
intro!: *sum.cong*)

lemma (**in** *inj-semiring-hom*) *map-vector-0*: (*map-vector hom v* = *0*) = (*v* = *0*)
unfolding *vec-eq-iff map-vector-def* **by** *auto*

lemma (**in** *inj-semiring-hom*) *map-vector-inj*: (*map-vector hom v* = *map-vector hom w*) = (*v* = *w*)
unfolding *vec-eq-iff map-vector-def* **by** *auto*

lemma (**in** *semiring-hom*) *matrix-vector-mult-hom*:

(*map-matrix hom A*) *∗v* (*map-vector hom v*) = *map-vector hom* (*A ∗v v*)
by (*transfer fixing*: *hom*, *auto simp*: *mult-mat-vec-hom*)

lemma (**in** *semiring-hom*) *vector-smult-hom*:
hom x ∗s (*map-vector hom v*) = *map-vector hom* (*x ∗s v*)
by (*transfer fixing*: *hom*, *auto simp*: *vec-hom-smult*)

lemma (**in** *inj-comm-ring-hom*) *eigen-vector-hom*:
eigen-vector (*map-matrix hom A*) (*map-vector hom v*) (*hom x*) = *eigen-vector A v x*
unfolding *eigen-vector-def matrix-vector-mult-hom vector-smult-hom map-vector-0 map-vector-inj*
by *auto*

end

4.2 Perron-Frobenius theorem via Brouwer's fixpoint theorem.

theory *Perron-Frobenius*
imports
HOL−Analysis.Brouwer-Fixpoint
Perron-Frobenius-Aux
begin

We follow the textbook proof of Serre [2, Theorem 5.2.1].

context
fixes *A* :: *complex* ^ *'n* ^ *'n* :: *finite*
assumes *rnnA*: *real-non-neg-mat A*
begin

private abbreviation(*input*) *sr* **where** *sr* ≡ *spectral-radius A*

private definition *max-v-ev* :: (*complex*^*'n*) × *complex* **where**
max-v-ev = (*SOME v-ev. eigen-vector A* (*fst v-ev*) (*snd v-ev*)
∧ *norm* (*snd v-ev*) = *sr*)

private definition *max-v* = (*1* / *norm1* (*fst max-v-ev*)) $*_R$ *fst max-v-ev*
private definition *max-ev* = *snd max-v-ev*

private lemma *max-v-ev*:
eigen-vector A max-v max-ev
norm max-ev = *sr*
norm1 max-v = *1*
proof −
obtain *v ev* **where** *id*: *max-v-ev* = (*v*,*ev*) **by** *force*
from *spectral-radius-ev*[*of A*] *someI-ex*[*of* λ *v-ev. eigen-vector A* (*fst v-ev*) (*snd v-ev*)
∧ *norm* (*snd v-ev*) = *sr*, *folded max-v-ev-def*, *unfolded id*]

have *v*: *eigen-vector A v ev* **and** *ev*: *norm ev* = *sr* **by** *auto*
from *normalize-eigen-vector*[*OF v*] *ev*
show *eigen-vector A max-v max-ev norm max-ev* = *sr norm1 max-v* = *1*
unfolding *max-v-def max-ev-def id* **by** *auto*
qed

In the definition of S, we use the linear norm instead of the default euclidean norm which is defined via the type-class. The reason is that S is not convex if one uses the euclidean norm.

private definition B :: *real* ^ *'n* ^ *'n* **where** $B \equiv \chi\ i\ j.\ Re\ (A\ \$\ i\ \$\ j)$
private definition S **where** $S = \{v :: real\ \hat{}\ 'n\ .\ norm1\ v = 1 \wedge (\forall\ i.\ v\ \$\ i \geq 0) \wedge$
$(\forall\ i.\ (B *v\ v)\ \$\ i \geq sr * (v\ \$\ i))\}$
private definition f :: *real* ^ *'n* ⇒ *real* ^ *'n* **where**
$f\ v = (1\ /\ norm1\ (B *v\ v)) *_R (B *v\ v)$

private lemma *closedS*: *closed S*
unfolding *S-def matrix-vector-mult-def*[*abs-def*]
proof (*intro closed-Collect-conj closed-Collect-all closed-Collect-le closed-Collect-eq*)
show *continuous-on UNIV norm1*
by (*simp add*: *continuous-at-imp-continuous-on*)
qed (*auto intro*!: *continuous-intros continuous-on-component*)

private lemma *boundedS*: *bounded S*
proof −
{
fix *v* :: *real* ^ *'n*
from *norm1-ge-norm*[*of v*] **have** $norm1\ v = 1 \Longrightarrow norm\ v \leq 1$ **by** *auto*
}
thus *?thesis*
unfolding *S-def bounded-iff*
by (*auto intro*!: *exI*[*of - 1*])
qed

private lemma *compactS*: *compact S*
using *boundedS closedS*
by (*simp add*: *compact-eq-bounded-closed*)

private lemmas *rnn* = *real-non-neg-matD*[*OF rnnA*]

lemma *B-norm*: $B\ \$\ i\ \$\ j = norm\ (A\ \$\ i\ \$\ j)$
using *rnn*[*of i j*]
by (*cases* $A\ \$\ i\ \$\ j$, *auto simp*: *B-def*)

lemma *mult-B-mono*: **assumes** $\bigwedge\ i.\ v\ \$\ i \geq w\ \$\ i$
shows $(B *v\ v)\ \$\ i \geq (B *v\ w)\ \$\ i$ **unfolding** *matrix-vector-mult-def vec-lambda-beta*
by (*rule sum-mono*, *rule mult-left-mono*[*OF assms*], *unfold B-norm*, *auto*)

```
private lemma non-emptyS: S ≠ {}
proof −
  let ?v = (χ i. norm (max-v $ i)) :: real ^ 'n
  have norm1 max-v = 1 by (rule max-v-ev(3))
  hence nv: norm1 ?v = 1 unfolding norm1-def by auto
  {
    fix i
    have sr ∗ (?v $ i) = sr ∗ norm (max-v $ i) by auto
    also have ... = (norm max-ev) ∗ norm (max-v $ i) using max-v-ev by auto
    also have ... = norm ((max-ev ∗s max-v) $ i) by (auto simp: norm-mult)
    also have max-ev ∗s max-v = A ∗v max-v using max-v-ev(1)[unfolded eigen-vector-def]
by auto
    also have norm ((A ∗v max-v) $ i) ≤ (B ∗v ?v) $ i
      unfolding matrix-vector-mult-def vec-lambda-beta
      by (rule sum-norm-le, auto simp: norm-mult B-norm)
    finally have sr ∗ (?v $ i) ≤ (B ∗v ?v) $ i .
  } note le = this
  have ?v ∈ S unfolding S-def using nv le by auto
  thus ?thesis by blast
qed

private lemma convexS: convex S
proof (rule convexI)
  fix v w a b
  assume ∗: v ∈ S w ∈ S 0 ≤ a 0 ≤ b a + b = (1 :: real)
  let ?lin = a ∗R v + b ∗R w
  from ∗ have 1: norm1 v = 1 norm1 w = 1 unfolding S-def by auto
  have norm1 ?lin = a ∗ norm1 v + b ∗ norm1 w
    unfolding norm1-def sum-distrib-left sum.distrib[symmetric]
  proof (rule sum.cong)
    fix i :: 'n
    from ∗ have v $ i ≥ 0 w $ i ≥ 0 unfolding S-def by auto
    thus norm (?lin $ i) = a ∗ norm (v $ i) + b ∗ norm (w $ i)
      using ∗(3−4) by auto
  qed simp
  also have ... = 1 using ∗(5) 1 by auto
  finally have norm1: norm1 ?lin = 1 .
  {
    fix i
    from ∗ have 0 ≤ v $ i sr ∗ v $ i ≤ (B ∗v v) $ i unfolding S-def by auto
    with ‹a ≥ 0› have a: a ∗ (sr ∗ v $ i) ≤ a ∗ (B ∗v v) $ i by (intro mult-left-mono)
    from ∗ have 0 ≤ w $ i sr ∗ w $ i ≤ (B ∗v w) $ i unfolding S-def by auto
      with ‹b ≥ 0› have b: b ∗ (sr ∗ w $ i) ≤ b ∗ (B ∗v w) $ i by (intro
mult-left-mono)
    from a b have a ∗ (sr ∗ v $ i) + b ∗ (sr ∗ w $ i) ≤ a ∗ (B ∗v v) $ i + b ∗
(B ∗v w) $ i by auto
  } note le = this
  have switch[simp]: ⋀ x y. x ∗ a ∗ y = a ∗ x ∗ y  ⋀ x y. x ∗ b ∗ y = b ∗ x ∗ y
by auto
```

have [*simp*]: $x \in \{v,w\} \Longrightarrow a * (r * x \,\$h\, i) = r * (a * x \,\$h\, i)$ **for** *a r i x* **by** *auto*
show $a *_R v + b *_R w \in S$ **using** $*$ *norm1 le* **unfolding** *S-def*
by (*auto simp*: *matrix-vect-scaleR matrix-vector-right-distrib ring-distribs*)
qed

private abbreviation (*input*) r :: *real* $\Rightarrow$ *complex* **where**
$r \equiv$ *of-real*

private abbreviation *rv* :: *real* ^'*n* $\Rightarrow$ *complex* ^'*n* **where**
$rv\ v \equiv \chi\ i.\ r\ (v\ \$\ i)$

private lemma *rv-0*: $(rv\ v = 0) = (v = 0)$
by (*simp add*: *of-real-hom.map-vector-0 map-vector-def vec-eq-iff*)

private lemma *rv-mult*: $A *v\ rv\ v = rv\ (B *v\ v)$
proof –
have *map-matrix r B* $= A$
using *rnnA* **unfolding** *map-matrix-def B-def real-non-neg-mat-def map-vector-def elements-mat-h-def*
by *vector*
thus *?thesis*
using *of-real-hom.matrix-vector-mult-hom*[*of B*, **where** *'a = complex*]
unfolding *map-vector-def* **by** *auto*
qed

context
assumes *zero-no-ev*: $\bigwedge v.\ v \in S \Longrightarrow A *v\ rv\ v \neq 0$
begin
private lemma *normB-S*: **assumes** v: $v \in S$
shows $norm1\ (B *v\ v) \neq 0$
proof –
from *zero-no-ev*[*OF v*, *unfolded rv-mult rv-0*]
show *?thesis* **by** *auto*
qed

private lemma *image-f*: f ‘ $S \subseteq S$
proof –
{
fix v
assume v: $v \in S$
hence *norm*: $norm1\ v = 1$ **and** *ge*: $\bigwedge i.\ v\ \$\ i \geq 0\ \bigwedge i.\ sr * v\ \$\ i \leq (B *v\ v)\ \$\ i$ **unfolding** *S-def* **by** *auto*
from *normB-S*[*OF v*] **have** *normB*: $norm1\ (B *v\ v) > 0$ **using** *norm1-nonzero* **by** *auto*
have *fv*: $f\ v = (1\ /\ norm1\ (B *v\ v)) *_R (B *v\ v)$ **unfolding** *f-def* **by** *auto*
from *normB* **have** *Bv0*: $B *v\ v \neq 0$ **unfolding** *norm1-0-iff*[*symmetric*] **by** *linarith*
have *norm*: $norm1\ (f\ v) = 1$ **unfolding** *fv* **using** *normB Bv0* **by** *simp*

```
    define c where c = (1 / norm1 (B *v v))
    have c: c > 0 unfolding c-def using normB by auto
    {
      fix i
      have 1: f v $ i ≥ 0 unfolding fv c-def[symmetric] using c ge
      by (auto simp: matrix-vector-mult-def sum-distrib-left B-norm intro!: sum-nonneg)
      have id1: ⋀ i. (B *v f v) $ i = c * ((B *v (B *v v)) $ i)
        unfolding f-def c-def matrix-vect-scaleR by simp
      have id3: ⋀ i. sr * f v $ i = c * ((B *v (sr *R v)) $ i)
        unfolding f-def c-def[symmetric] matrix-vect-scaleR by auto
      have 2: sr * f v $ i ≤ (B *v f v) $ i unfolding id1 id3
        unfolding mult-le-cancel-iff2[OF ‹c > 0›]
        by (rule mult-B-mono, insert ge(2), auto)
      note 1 2
    }
    with norm have f v ∈ S unfolding S-def by auto
  }
  thus ?thesis by blast
qed

private lemma cont-f: continuous-on S f
  unfolding f-def[abs-def] continuous-on using normB-S
  unfolding norm1-def
  by (auto intro!: tendsto-eq-intros)

qualified lemma perron-frobenius-positive-ev:
  ∃ v. eigen-vector A v (r sr) ∧ real-non-neg-vec v
proof −
  from brouwer[OF compactS convexS non-emptyS cont-f image-f]
    obtain v where v: v ∈ S and fv: f v = v by auto
  define ev where ev = norm1 (B *v v)
  from normB-S[OF v] have ev ≠ 0 unfolding ev-def by auto
  with norm1-ge-0[of B *v v, folded ev-def] have norm: ev > 0 by auto
  from arg-cong[OF fv[unfolded f-def], of λ (w :: real ^ 'n). ev *R w] norm
   have ev: B *v v = ev *s v unfolding ev-def[symmetric] scalar-mult-eq-scaleR
by simp
  with v[unfolded S-def] have ge: ⋀ i. sr * v $ i ≤ ev * v $ i by auto
  have A *v rv v = rv (B *v v) unfolding rv-mult ..
  also have ... = ev *s rv v unfolding ev vec-eq-iff
    by (simp add: scaleR-conv-of-real scaleR-vec-def)
  finally have ev: A *v rv v = ev *s rv v .
  from v have v0: v ≠ 0 unfolding S-def by auto
  hence rv v ≠ 0 unfolding rv-0 .
  with ev have ev: eigen-vector A (rv v) ev unfolding eigen-vector-def by auto
  hence eigen-value A ev unfolding eigen-value-def by auto
  from spectral-radius-max[OF this] have le: norm (r ev) ≤ sr .
  from v0 obtain i where v $ i ≠ 0 unfolding vec-eq-iff by auto
  from v have v $ i ≥ 0 unfolding S-def by auto
  with ‹v $ i ≠ 0› have v $ i > 0 by auto
```

with *ge*[*of i*] **have** *ge*: *sr* $\leq$ *ev* **by** *auto*
with *le* **have** *sr*: *r sr* = *ev* **by** *auto*
from *v* **have** $*$: *real-non-neg-vec* (*rv v*) **unfolding** *S-def real-non-neg-vec-def vec-elements-h-def* **by** *auto*
show *?thesis* **unfolding** *sr*
by (*rule exI*[*of - rv v*], *insert* $*$ *ev norm*, *auto*)
qed
end

qualified lemma *perron-frobenius-both*:
$\exists$ *v*. *eigen-vector A v* (*r sr*) $\wedge$ *real-non-neg-vec v*
proof (*cases* $\forall$ *v* $\in$ *S*. *A* $*v$ *rv v* $\neq$ *0*)
case *True*
show *?thesis*
by (*rule Perron-Frobenius.perron-frobenius-positive-ev*[*OF rnnA*], *insert True*, *auto*)
next
case *False*
then obtain *v* **where** *v*: *v* $\in$ *S* **and** *A0*: *A* $*v$ *rv v* = *0* **by** *auto*
hence *id*: *A* $*v$ *rv v* = *0* $*s$ *rv v* **and** *v0*: *v* $\neq$ *0* **unfolding** *S-def* **by** *auto*
from *v0* **have** *rv v* $\neq$ *0* **unfolding** *rv-0* **.**
with *id* **have** *ev*: *eigen-vector A* (*rv v*) *0* **unfolding** *eigen-vector-def* **by** *auto*
hence *eigen-value A 0* **unfolding** *eigen-value-def* **..**
from *spectral-radius-max*[*OF this*] **have** *0*: *0* $\leq$ *sr* **by** *auto*
from *v*[*unfolded S-def*] **have** *ge*: $\bigwedge$ *i*. *sr* $*$ *v* \$ *i* $\leq$ (*B* $*v$ *v*) \$ *i* **by** *auto*
from *v*[*unfolded S-def*] **have** *rnn*: *real-non-neg-vec* (*rv v*)
unfolding *real-non-neg-vec-def vec-elements-h-def* **by** *auto*
from *v0* **obtain** *i* **where** *v* \$ *i* $\neq$ *0* **unfolding** *vec-eq-iff* **by** *auto*
from *v* **have** *v* \$ *i* $\geq$ *0* **unfolding** *S-def* **by** *auto*
with ‹*v* \$ *i* $\neq$ *0*› **have** *vi*: *v* \$ *i* > *0* **by** *auto*
from *rv-mult*[*of v*, *unfolded A0*] **have** *rv* (*B* $*v$ *v*) = *0* **by** *simp*
hence *B* $*v$ *v* = *0* **unfolding** *rv-0* **.**
from *ge*[*of i*, *unfolded this*] *vi* **have** *ge*: *sr* $\leq$ *0* **by** (*simp add*: *mult-le-0-iff*)
with ‹*0* $\leq$ *sr*› **have** *sr* = *0* **by** *auto*
show *?thesis* **unfolding** ‹*sr* = *0*› **using** *rnn ev* **by** *auto*
qed
end

Perron Frobenius: The largest complex eigenvalue of a real-valued non-negative matrix is a real one, and it has a real-valued non-negative eigenvector.

lemma *perron-frobenius*:
assumes *real-non-neg-mat A*
shows $\exists$ *v*. *eigen-vector A v* (*of-real* (*spectral-radius A*)) $\wedge$ *real-non-neg-vec v*
by (*rule Perron-Frobenius.perron-frobenius-both*[*OF assms*])

And a version which ignores the eigenvector.

lemma *perron-frobenius-eigen-value*:
assumes *real-non-neg-mat A*

shows *eigen-value A* (*of-real* (*spectral-radius A*))
using *perron-frobenius*[*OF assms*] **unfolding** *eigen-value-def* **by** *blast*

end

5 Roots of Unity

theory *Roots-Unity*
imports
Polynomial-Factorization.Order-Polynomial
HOL−Computational-Algebra.Fundamental-Theorem-Algebra
Polynomial-Interpolation.Ring-Hom-Poly
begin

lemma *cis-mult-cmod-id*: *cis* (*Arg x*) ∗ *of-real* (*cmod x*) = *x*
using *rcis-cmod-Arg*[*unfolded rcis-def*] **by** (*simp add*: *ac-simps*)

lemma *rcis-mult-cis*[*simp*]: *rcis n a* ∗ *cis b* = *rcis n* (*a* + *b*) **unfolding** *cis-rcis-eq rcis-mult* **by** *simp*
lemma *rcis-div-cis*[*simp*]: *rcis n a* / *cis b* = *rcis n* (*a* − *b*) **unfolding** *cis-rcis-eq rcis-divide* **by** *simp*

lemma *cis-plus-2pi*[*simp*]: *cis* (*x* + *2* ∗ *pi*) = *cis x* **by** (*auto simp*: *complex-eq-iff*)
lemma *cis-plus-2pi-neq-1*: **assumes** *x*: *0* < *x x* < *2* ∗ *pi*
shows *cis x* ≠ *1*
proof −
from *x* **have** *cos x* ≠ *1* **by** (*smt cos-2pi-minus cos-monotone-0-pi cos-zero*)
thus *?thesis* **by** (*auto simp*: *complex-eq-iff*)
qed

lemma *cis-times-2pi*[*simp*]: *cis* (*of-nat n* ∗ *2* ∗ *pi*) = *1*
proof (*induct n*)
case (*Suc n*)
have *of-nat* (*Suc n*) ∗ *2* ∗ *pi* = *of-nat n* ∗ *2* ∗ *pi* + *2* ∗ *pi* **by** (*simp add*: *distrib-right*)
also have *cis* . . . = *1* **unfolding** *cis-plus-2pi Suc* **..**
finally show *?case* **.**
qed *simp*

lemma *cis-add-pi*[*simp*]: *cis* (*pi* + *x*) = − *cis x*
by (*auto simp*: *complex-eq-iff*)

lemma *cis-3-pi-2*[*simp*]: *cis* (*pi* ∗ *3* / *2*) = − i
proof −
have *cis* (*pi* ∗ *3* / *2*) = *cis* (*pi* + *pi* / *2*)
by (*rule arg-cong*[*of - - cis*], *simp*)
also have . . . = − i **unfolding** *cis-add-pi* **by** *simp*
finally show *?thesis* **.**
qed

lemma *rcis-plus-2pi*[*simp*]: *rcis y* (*x* + *2* ∗ *pi*) = *rcis y x* **unfolding** *rcis-def* **by** *simp*

lemma *rcis-times-2pi*[*simp*]: *rcis r* (*of-nat n* ∗ *2* ∗ *pi*) = *of-real r*
  unfolding *rcis-def cis-times-2pi* **by** *simp*

lemma *arg-rcis-cis*: **assumes** *n*: *n* > *0* **shows** *Arg* (*rcis n x*) = *Arg* (*cis x*)
  using *Arg-bounded cis-Arg-unique cis-Arg complex-mod-rcis n rcis-def sgn-eq* **by** *auto*

lemma *arg-eqD*: **assumes** *Arg* (*cis x*) = *Arg* (*cis y*) −*pi* < *x x* ≤ *pi* −*pi* < *y y* ≤ *pi*
  shows *x* = *y*
  using *assms*(*1*) **unfolding** *cis-Arg-unique*[*OF sgn-cis assms*(*2*−*3*)] *cis-Arg-unique*[*OF sgn-cis assms*(*4*−*5*)] **.**

lemma *rcis-inj-on*: **assumes** *r*: *r* ≠ *0* **shows** *inj-on* (*rcis r*) {*0* ..< *2* ∗ *pi*}
proof (*rule inj-onI*, *goal-cases*)
  case (*1 x y*)
  from *arg-cong*[*OF 1*(*3*), *of* λ *x*. *x* / *r*] **have** *cis x* = *cis y* **using** *r* **by** (*simp add*: *rcis-def*)
  from *arg-cong*[*OF this*, *of* λ *x*. *inverse x*] **have** *cis* (−*x*) = *cis* (−*y*) **by** *simp*
  from *arg-cong*[*OF this*, *of uminus*] **have** ∗: *cis* (−*x* + *pi*) = *cis* (−*y* + *pi*)
    by (*auto simp*: *complex-eq-iff*)
  have − *x* + *pi* = − *y* + *pi*
    by (*rule arg-eqD*[*OF arg-cong*[*OF* ∗, *of Arg*]], *insert 1*(*1*−*2*), *auto*)
  thus *?case* **by** *simp*
qed

lemma *cis-inj-on*: *inj-on cis* {*0* ..< *2* ∗ *pi*}
  using *rcis-inj-on*[*of 1*] **unfolding** *rcis-def* **by** *auto*

definition *root-unity* :: *nat* ⇒ ′*a* :: *comm-ring-1 poly* **where**
  root-unity n = *monom 1 n* − *1*

lemma *poly-root-unity*: *poly* (*root-unity n*) *x* = *0* ⟷ *x*^*n* = *1*
  unfolding *root-unity-def* **by** (*simp add*: *poly-monom*)

lemma *degree-root-unity*[*simp*]: *degree* (*root-unity n*) = *n* (**is** *degree ?p* = -)
proof −
  have *p*: *?p* = *monom 1 n* + (−*1*) **unfolding** *root-unity-def* **by** *auto*
  show *?thesis*
  proof (*cases n*)
    case *0*
    thus *?thesis* **unfolding** *p* **by** *simp*
  next
    case (*Suc m*)
    show *?thesis* **unfolding** *p* **unfolding** *Suc*
      by (*subst degree-add-eq-left*, *auto simp*: *degree-monom-eq*)

qed
qed

lemma *zero-root-unit*[*simp*]: *root-unity n* = *0* ⟷ *n* = *0* (**is** *?p* = *0* ⟷ -)
proof (*cases n* = *0*)
  case *True*
  thus *?thesis* **unfolding** *root-unity-def* **by** *simp*
next
  case *False*
  from *degree-root-unity*[*of n*] *False*
  have *degree ?p* ≠ *0* **by** *auto*
  hence *?p* ≠ *0* **by** *fastforce*
  thus *?thesis* **using** *False* **by** *auto*
qed

definition *prod-root-unity* :: *nat list* ⇒ *'a* :: *idom poly* **where**
  prod-root-unity ns = *prod-list* (*map root-unity ns*)

lemma *poly-prod-root-unity*: *poly* (*prod-root-unity ns*) *x* = *0* ⟷ (∃ *k*∈*set ns*. *x* ^ *k* = *1*)
  unfolding *prod-root-unity-def*
  by (*simp add*: *poly-prod-list prod-list-zero-iff o-def image-def poly-root-unity*)

lemma *degree-prod-root-unity*[*simp*]: *0* ∉ *set ns* ⟹ *degree* (*prod-root-unity ns*) = *sum-list ns*
  unfolding *prod-root-unity-def*
  by (*subst degree-prod-list-eq*, *auto simp*: *o-def*)

lemma *zero-prod-root-unit*[*simp*]: *prod-root-unity ns* = *0* ⟷ *0* ∈ *set ns*
  unfolding *prod-root-unity-def prod-list-zero-iff* **by** *auto*

lemma *roots-of-unity*: **assumes** *n*: *n* ≠ *0*
  shows (λ *i*. (*cis* (*of-nat i* ∗ *2* ∗ *pi* / *n*))) ‘ {*0* ..< *n*} = { *x* :: *complex*. *x* ^ *n* = *1*} (**is** *?prod* = *?Roots*)
    {*x*. *poly* (*root-unity n*) *x* = *0*} = { *x* :: *complex*. *x* ^ *n* = *1*}
    card { *x* :: *complex*. *x* ^ *n* = *1*} = *n*
proof (*atomize*(*full*), *goal-cases*)
  case *1*
  let *?one* = *1* :: *complex*
  let *?p* = *monom ?one n* − *1*
  have *degM*: *degree* (*monom ?one n*) = *n* **by** (*rule degree-monom-eq*, *simp*)
  have *degree ?p* = *degree* (*monom ?one n* + (−*1*)) **by** *simp*
  also have . . . = *degree* (*monom ?one n*)
    by (*rule degree-add-eq-left*, *insert n*, *simp add*: *degM*)
  finally have *degp*: *degree ?p* = *n* **unfolding** *degM* **.**
  with *n* **have** *p*: *?p* ≠ *0* **by** *auto*
  have *roots*: *?Roots* = {*x*. *poly ?p x* = *0*}
    unfolding *poly-diff poly-monom* **by** *simp*
  also have *finite* . . . **by** (*rule poly-roots-finite*[*OF p*])

finally have *fin*: *finite ?Roots* **.**
have *sub*: *?prod* ⊆ *?Roots*
proof
fix *x*
assume *x* ∈ *?prod*
then obtain *i* **where** *x*: *x* = *cis* (*real i* ∗ *2* ∗ *pi* / *n*) **by** *auto*
have *x* ^ *n* = *cis* (*real i* ∗ *2* ∗ *pi*) **unfolding** *x DeMoivre* **using** *n* **by** *simp*
also have ... = *1* **by** *simp*
finally show *x* ∈ *?Roots* **by** *auto*
qed
have *Rn*: *card ?Roots* ≤ *n* **unfolding** *roots*
by (*rule poly-roots-degree*[*of ?p*, *unfolded degp*, *OF p*])
have ... = *card* {*0* ..< *n*} **by** *simp*
also have ... = *card ?prod*
proof (*rule card-image*[*symmetric*], *rule inj-onI*, *goal-cases*)
case (*1 x y*)
{
fix *m*
assume *m* < *n*
hence *real m* < *real n* **by** *simp*
from *mult-strict-right-mono*[*OF this*, *of 2* ∗ *pi* / *real n*] *n*
have *real m* ∗ *2* ∗ *pi* / *real n* < *real n* ∗ *2* ∗ *pi* / *real n* **by** *simp*
hence *real m* ∗ *2* ∗ *pi* / *real n* < *2* ∗ *pi* **using** *n* **by** *simp*
} **note** [*simp*] = *this*
have *0*: (*1* :: *real*) ≠ *0* **using** *n* **by** *auto*
have *real x* ∗ *2* ∗ *pi* / *real n* = *real y* ∗ *2* ∗ *pi* / *real n*
by (*rule inj-onD*[*OF rcis-inj-on 1*(*3*)[*unfolded cis-rcis-eq*]], *insert 1*(*1*−*2*), *auto*)
with *n* **show** *x* = *y* **by** *auto*
qed
finally have *cn*: *card ?prod* = *n* **..**
with *Rn* **have** *card ?prod* ≥ *card ?Roots* **by** *auto*
with *card-mono*[*OF fin sub*] **have** *card*: *card ?prod* = *card ?Roots* **by** *auto*
have *?prod* = *?Roots*
by (*rule card-subset-eq*[*OF fin sub card*])
from *this roots*[*symmetric*] *cn*[*unfolded this*]
show *?case* **unfolding** *root-unity-def* **by** *blast*
qed

lemma *poly-roots-dvd*: **fixes** *p* :: *'a* :: *field poly*
assumes *p* ≠ *0* **and** *degree p* = *n*
and *card* {*x*. *poly p x* = *0*} ≥ *n* **and** {*x*. *poly p x* = *0*} ⊆ {*x*. *poly q x* = *0*}
shows *p dvd q*
proof −
from *poly-roots-degree*[*OF assms*(*1*)] *assms*(*2*−*3*) **have** *card* {*x*. *poly p x* = *0*} = *n* **by** *auto*
from *assms*(*1*−*2*) *this assms*(*4*)
show *?thesis*
proof (*induct n arbitrary*: *p q*)

case (*0 p q*)
from *is-unit-iff-degree*[*OF 0*(*1*)] *0*(*2*) **show** *?case* **by** *blast*
next
case (*Suc n p q*)
let *?P* = {*x*. *poly p x* = *0*}
let *?Q* = {*x*. *poly q x* = *0*}
from *Suc*(*4*−*5*) *card-gt-0-iff*[*of ?P*] **obtain** *x* **where**
x: *poly p x* = *0 poly q x* = *0* **and** *fin*: *finite ?P* **by** *auto*
define *r* **where** *r* = [:−*x*, *1*:]
from *x*[*unfolded poly-eq-0-iff-dvd r-def*[*symmetric*]] **obtain** *p′ q′* **where**
p: *p* = *r* ∗ *p′* **and** *q*: *q* = *r* ∗ *q′* **unfolding** *dvd-def* **by** *auto*
from *Suc*(*2*) **have** *degree p* = *degree r* + *degree p′* **unfolding** *p*
by (*subst degree-mult-eq*, *auto*)
with *Suc*(*3*) **have** *deg*: *degree p′* = *n* **unfolding** *r-def* **by** *auto*
from *Suc*(*2*) *p* **have** *p′0*: *p′* ≠ *0* **by** *auto*
let *?P′* = {*x*. *poly p′ x* = *0*}
let *?Q′* = {*x*. *poly q′ x* = *0*}
have *P*: *?P* = *insert x ?P′* **unfolding** *p poly-mult* **unfolding** *r-def* **by** *auto*
have *Q*: *?Q* = *insert x ?Q′* **unfolding** *q poly-mult* **unfolding** *r-def* **by** *auto*
{
assume *x* ∈ *?P′*
hence *?P* = *?P′* **unfolding** *P* **by** *auto*
from *arg-cong*[*OF this*, *of card*, *unfolded Suc*(*4*)] *deg* **have** *False*
using *poly-roots-degree*[*OF p′0*] **by** *auto*
} **note** *xp′* = *this*
hence *xP′*: *x* ∉ *?P′* **by** *auto*
have *card ?P* = *Suc* (*card ?P′*) **unfolding** *P*
by (*rule card-insert-disjoint*[*OF - xP′*], *insert fin*[*unfolded P*], *auto*)
with *Suc*(*4*) **have** *card*: *card ?P′* = *n* **by** *auto*
from *Suc*(*5*)[*unfolded P Q*] *xP′* **have** *?P′* ⊆ *?Q′* **by** *auto*
from *Suc*(*1*)[*OF p′0 deg card this*]
have *IH*: *p′ dvd q′* **.**
show *?case* **unfolding** *p q* **using** *IH* **by** *simp*
qed
qed

lemma *root-unity-decomp*: **assumes** *n*: *n* ≠ *0*
shows *root-unity n* =
prod-list (*map* (λ *i*. [:−*cis* (*of-nat i* ∗ *2* ∗ *pi* / *n*), *1*:]) [*0* ..< *n*]) (**is** *?u* = *?p*)
proof −
have *deg*: *degree ?u* = *n* **by** *simp*
note *main* = *roots-of-unity*[*OF n*]
have *dvd*: *?u dvd ?p*
proof (*rule poly-roots-dvd*[*OF - deg*])
show *n* ≤ *card* {*x*. *poly ?u x* = *0*} **using** *main* **by** *auto*
show *?u* ≠ *0* **using** *n* **by** *auto*
show {*x*. *poly ?u x* = *0*} ⊆ {*x*. *poly ?p x* = *0*}
unfolding *main*(*2*) *main*(*1*)[*symmetric*] *poly-prod-list prod-list-zero-iff* **by** *auto*

qed
have *deg'*: *degree ?p* = *n*
by (*subst degree-prod-list-eq*, *auto simp*: *o-def sum-list-triv*)
have *mon*: *monic ?u* using *deg* unfolding *root-unity-def* using *n* by *auto*
have *mon'*: *monic ?p* by (*rule monic-prod-list*, *auto*)
from *dvd*[*unfolded dvd-def*] obtain *f* where *puf*: *?p* = *?u* ∗ *f* by *auto*
have *degree ?p* = *degree ?u* + *degree f* using *mon' n* unfolding *puf*
by (*subst degree-mult-eq*, *auto*)
with *deg deg'* have *degree f* = *0* by *auto*
from *degree0-coeffs*[*OF this*] obtain *a* where *f*: *f* = [:*a*:] by *blast*
from *arg-cong*[*OF puf*, *of lead-coeff*] *mon mon'*
have *a* = *1* unfolding *puf f* by (*cases a* = *0*, *auto*)
with *f* have *f*: *f* = *1* by *auto*
with *puf* show *?thesis* by *auto*
qed

lemma *order-monic-linear*: *order x* [:*y*,*1*:] = (*if y* + *x* = *0 then 1 else 0*)
proof (*cases y* + *x* = *0*)
case *True*
hence *poly* [:*y*,*1*:] *x* = *0* by *simp*
from *this*[*unfolded order-root*] have *order x* [:*y*,*1*:] ≠ *0* by *auto*
moreover from *order-degree*[*of* [:*y*,*1*:] *x*] have *order x* [:*y*,*1*:] ≤ *1* by *auto*
ultimately show *?thesis* unfolding *True* by *auto*
next
case *False*
hence *poly* [:*y*,*1*:] *x* ≠ *0* by *auto*
from *order-0I*[*OF this*] *False* show *?thesis* by *auto*
qed

lemma *order-root-unity*: fixes *x* :: *complex* assumes *n*: *n* ≠ *0*
shows *order x* (*root-unity n*) = (*if x^n* = *1 then 1 else 0*)
(is *order* - *?u* = -)
proof (*cases x^n* = *1*)
case *False*
with *roots-of-unity*(*2*)[*OF n*] have *poly ?u x* ≠ *0* by *auto*
from *False order-0I*[*OF this*] show *?thesis* by *auto*
next
case *True*
let *?phi* = λ *i* :: *nat*. *i* ∗ *2* ∗ *pi* / *n*
from *True roots-of-unity*(*1*)[*OF n*] obtain *i* where *i*: *i* < *n*
and *x*: *x* = *cis* (*?phi i*) by *force*
from *i* have *n-split*: [*0* ..< *n*] = [*0* ..< *i*] @ *i* # [*Suc i* ..< *n*]
by (*metis le-Suc-ex less-imp-le-nat not-le-imp-less not-less0 upt-add-eq-append upt-conv-Cons*)
{
fix *j*
assume *j*: *j* < *n* ∨ *j* < *i* and *eq*: *cis* (*?phi i*) = *cis* (*?phi j*)
from *inj-onD*[*OF cis-inj-on eq*] *i j n* have *i* = *j* by (*auto simp*: *field-simps*)
} note *inj* = *this*

have *order x ?u* = *1* **unfolding** *root-unity-decomp*[*OF n*]
unfolding *x n-split* **using** *inj*
by (*subst order-prod-list*, *force*, *fastforce simp*: *order-monic-linear*)
with *True* **show** *?thesis* **by** *auto*
qed

lemma *order-prod-root-unity*: **assumes** *0*: $0 \notin$ *set ks*
shows *order* (*x* :: *complex*) (*prod-root-unity ks*) = *length* (*filter* (λ *k*. *x*^*k* = *1*) *ks*)
proof −
have *order x* (*prod-root-unity ks*) = ($\sum$ *k*←*ks*. *order x* (*root-unity k*))
unfolding *prod-root-unity-def*
by (*subst order-prod-list*, *insert 0*, *auto simp*: *o-def*)
also have ... = ($\sum$ *k*←*ks*. (*if x*^*k* = *1 then 1 else 0*))
by (*rule arg-cong*, *rule map-cong*, *insert 0*, *force*, *intro order-root-unity*, *metis*)
also have ... = *length* (*filter* (λ *k*. *x*^*k* = *1*) *ks*)
by (*subst sum-list-map-filter'*[*symmetric*], *simp add*: *sum-list-triv*)
finally show *?thesis* **.**
qed

lemma *root-unity-witness*: **fixes** *xs* :: *complex list*
assumes *prod-list* (*map* (λ *x*. [:−*x*,*1*:]) *xs*) = *monom 1 n* − *1*
shows *x*^*n* = *1* $\longleftrightarrow$ *x* $\in$ *set xs*
proof −
from *assms* **have** *n0*: *n* $\neq$ *0* **by** (*cases n* = *0*, *auto simp*: *prod-list-zero-iff*)
have *x* $\in$ *set xs* $\longleftrightarrow$ *poly* (*prod-list* (*map* (λ *x*. [:−*x*,*1*:]) *xs*)) *x* = *0*
unfolding *poly-prod-list prod-list-zero-iff* **by** *auto*
also have ... $\longleftrightarrow$ *x*^*n* = *1* **using** *roots-of-unity*(*2*)[*OF n0*] **unfolding** *assms root-unity-def* **by** *auto*
finally show *?thesis* **by** *auto*
qed

lemma *root-unity-explicit*: **fixes** *x* :: *complex*
shows
(*x* ^ *1* = *1*) $\longleftrightarrow$ *x* = *1*
(*x* ^ *2* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, −*1*})
(*x* ^ *3* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, *Complex* (−*1*/*2*) (*sqrt 3* / *2*), *Complex* (−*1*/*2*) (− *sqrt 3* / *2*)})
(*x* ^ *4* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, −*1*, i, − i})
proof −
show (*x* ^ *1* = *1*) $\longleftrightarrow$ *x* = *1*
by (*subst root-unity-witness*[*of* [*1*]], *code-simp*, *auto*)
show (*x* ^ *2* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, −*1*})
by (*subst root-unity-witness*[*of* [*1*,−*1*]], *code-simp*, *auto*)
show (*x* ^ *4* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, −*1*, i, − i})
by (*subst root-unity-witness*[*of* [*1*,−*1*, i, − i]], *code-simp*, *auto*)
have *3*: *3* = *Suc* (*Suc* (*Suc 0*)) *1* = [:*1*:] **by** *auto*
show (*x* ^ *3* = *1*) $\longleftrightarrow$ (*x* $\in$ {*1*, *Complex* (−*1*/*2*) (*sqrt 3* / *2*), *Complex* (−*1*/*2*) (− *sqrt 3* / *2*)})

by (*subst root-unity-witness*[*of*
[*1*, *Complex* (−*1*/*2*) (*sqrt 3* / *2*), *Complex* (−*1*/*2*) (− *sqrt 3* / *2*)]],
auto simp: *3 monom-altdef complex-mult complex-eq-iff*)
qed

definition *primitive-root-unity* :: *nat* ⇒ ′*a* :: *power* ⇒ *bool* **where**
primitive-root-unity k x = (*k* ≠ *0* ∧ *x*^*k* = *1* ∧ (∀ *k*′ < *k*. *k*′ ≠ *0* ⟶ *x*^*k*′ ≠ *1*))

lemma *primitive-root-unityD*: **assumes** *primitive-root-unity k x*
shows *k* ≠ *0 x*^*k* = *1 k*′ ≠ *0* ⟹ *x*^*k*′ = *1* ⟹ *k* ≤ *k*′
proof −
note ∗ = *assms*[*unfolded primitive-root-unity-def*]
from ∗ **have** ∗∗: *k*′ < *k* ⟹ *k*′ ≠ *0* ⟹ *x* ^ *k*′ ≠ *1* **by** *auto*
show *k* ≠ *0 x*^*k* = *1* **using** ∗ **by** *auto*
show *k*′ ≠ *0* ⟹ *x*^*k*′ = *1* ⟹ *k* ≤ *k*′ **using** ∗∗ **by** *force*
qed

lemma *primitive-root-unity-exists*: **assumes** *k* ≠ *0 x* ^ *k* = *1*
shows ∃ *k*′. *k*′ ≤ *k* ∧ *primitive-root-unity k*′ *x*
proof −
let *?P* = λ *k*. *x* ^ *k* = *1* ∧ *k* ≠ *0*
define *k*′ **where** *k*′ = (*LEAST k*. *?P k*)
from *assms* **have** *Pk*: ∃ *k*. *?P k* **by** *auto*
from *LeastI-ex*[*OF Pk*, *folded k*′*-def*]
have *k*′ ≠ *0 x* ^ *k*′ = *1* **by** *auto*
with *not-less-Least*[*of - ?P*, *folded k*′*-def*]
have *primitive-root-unity k*′ *x* **unfolding** *primitive-root-unity-def* **by** *auto*
with *primitive-root-unityD*(*3*)[*OF this assms*]
show *?thesis* **by** *auto*
qed

lemma *primitive-root-unity-dvd*: **fixes** *x* :: *complex*
assumes *k*: *primitive-root-unity k x*
shows *x* ^ *n* = *1* ⟷ *k dvd n*
proof
assume *k dvd n* **then obtain** *j* **where** *n*: *n* = *k* ∗ *j* **unfolding** *dvd-def* **by** *auto*
have *x* ^ *n* = (*x* ^ *k*) ^ *j* **unfolding** *n power-mult* **by** *simp*
also have ... = *1* **unfolding** *primitive-root-unityD*[*OF k*] **by** *simp*
finally show *x* ^ *n* = *1* **.**
next
assume *n*: *x* ^ *n* = *1*
note *k* = *primitive-root-unityD*[*OF k*]
show *k dvd n*
proof (*cases n* = *0*)
case *n0*: *False*
from *k*(*3*)[*OF n0*] *n* **have** *nk*: *n* ≥ *k* **by** *force*
from *roots-of-unity*[*OF k*(*1*)] *k*(*2*) **obtain** *i* :: *nat* **where** *xk*: *x* = *cis* (*i* ∗ *2* ∗ *pi* / *k*)

and *ik*: $i < k$ **by** *force*
from *roots-of-unity*[*OF n0*] *n* **obtain** *j* :: *nat* **where** *xn*: $x = cis\ (j * 2 * pi\ /\ n)$
and *jn*: $j < n$ **by** *force*
have *cop*: *coprime i k*
proof (*rule gcd-eq-1-imp-coprime*)
from *k*(*1*) **have** $gcd\ i\ k \neq 0$ **by** *auto*
from *gcd-coprime-exists*[*OF this*] *this* **obtain** i' k' *g* **where**
$*$: $i = i' * g$ $k = k' * g$ $g \neq 0$ **and** *g*: $g = gcd\ i\ k$ **by** *blast*
from $*(2)$ *k*(*1*) **have** k': $k' \neq 0$ **by** *auto*
have $x = cis\ (i * 2 * pi\ /\ k)$ **by** *fact*
also have $i * 2 * pi\ /\ k = i' * 2 * pi\ /\ k'$ **unfolding** $*$ **using** $*(3)$ **by** *auto*
finally have $x\ \hat{}\ k' = 1$ **by** (*simp add*: *DeMoivre* k')
with *k*(*3*)[*OF* k'] **have** $k' \geq k$ **by** *linarith*
moreover with $*$ *k*(*1*) **have** $g = 1$ **by** *auto*
then show $gcd\ i\ k = 1$ **by** (*simp add*: *g*)
qed
from *inj-onD*[*OF cis-inj-on xk*[*unfolded xn*]] *n0 k*(*1*) *ik jn*
have $j * real\ k = i * real\ n$ **by** (*auto simp*: *field-simps*)
hence $real\ (j * k) = real\ (i * n)$ **by** *simp*
hence *eq*: $j * k = i * n$ **by** *linarith*
with *cop* **show** *k dvd n*
by (*metis coprime-commute coprime-dvd-mult-right-iff dvd-triv-right*)
qed *auto*
qed

lemma *primitive-root-unity-simple-computation*:
primitive-root-unity k x = (*if* $k = 0$ *then False else*
$x\ \hat{}\ k = 1 \wedge (\forall\ i \in \{1\ ..< k\}.\ x\ \hat{}\ i \neq 1)$)
unfolding *primitive-root-unity-def* **by** *auto*

lemma *primitive-root-unity-explicit*: **fixes** *x* :: *complex*
shows *primitive-root-unity 1 x* $\longleftrightarrow x = 1$
primitive-root-unity 2 x $\longleftrightarrow x = -1$
primitive-root-unity 3 x $\longleftrightarrow (x \in \{Complex\ (-1/2)\ (sqrt\ 3\ /\ 2),\ Complex\ (-1/2)\ (-\ sqrt\ 3\ /\ 2)\})$
primitive-root-unity 4 x $\longleftrightarrow (x \in \{\mathrm{i}, -\ \mathrm{i}\})$
proof (*atomize*(*full*), *goal-cases*)
case *1*
{
fix *P* :: *nat* ⇒ *bool*
have $*$: $\{1\ ..< 2 :: nat\} = \{1\}$ $\{1\ ..< 3 :: nat\} = \{1,2\}$ $\{1\ ..< 4 :: nat\} = \{1,2,3\}$
by *code-simp+*
have $(\forall i\in \{1\ ..< 2\}.\ P\ i) = P\ 1$ $(\forall i\in \{1\ ..< 3\}.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2$
$(\forall i\in \{1\ ..< 4\}.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3$
unfolding $*$ **by** *auto*
} **note** $*$ = *this*
show *?case* **unfolding** *primitive-root-unity-simple-computation root-unity-explicit*

*
  by (*auto simp*: *complex-eq-iff*)
qed

function *decompose-prod-root-unity-main* ::
  'a :: *field poly* ⇒ *nat* ⇒ *nat list* × *'a poly* **where**
  decompose-prod-root-unity-main p k = (
    if k = *0 then* ([], *p*) *else*
   let q = *root-unity k in if q dvd p then if p* = *0 then* ([],*0*) *else*
     map-prod (*Cons k*) *id* (*decompose-prod-root-unity-main* (*p div q*) *k*) *else*
     decompose-prod-root-unity-main p (*k* − *1*))
  by *pat-completeness auto*

termination by (*relation measure* (λ (*p*,*k*). *degree p* + *k*), *auto simp*: *degree-div-less*)

declare *decompose-prod-root-unity-main.simps*[*simp del*]

lemma *decompose-prod-root-unity-main*: **fixes** *p* :: *complex poly*
  assumes *p*: *p* = *prod-root-unity ks* ∗ *f*
  and *d*: *decompose-prod-root-unity-main p k* = (*ks'*,*g*)
  and *f*: ⋀ *x*. *cmod x* = *1* ⟹ *poly f x* ≠ *0*
  and *k*: ⋀ *k'*. *k'* > *k* ⟹ ¬ *root-unity k' dvd p*
shows *p* = *prod-root-unity ks'* ∗ *f* ∧ *f* = *g* ∧ *set ks* = *set ks'*
  using *d p k*
proof (*induct p k arbitrary*: *ks ks' rule*: *decompose-prod-root-unity-main.induct*)
  case (*1 p k ks ks'*)
  note *p* = *1*(*4*)
  note *k* = *1*(*5*)
  from *k*[*of Suc k*] **have** *p0*: *p* ≠ *0* **by** *auto*
  hence *p* = *0* ⟷ *False* **by** *auto*
  note *d* = *1*(*3*)[*unfolded decompose-prod-root-unity-main.simps*[*of p k*] *this if-False Let-def*]
  from *p0*[*unfolded p*] **have** *ks0*: *0* ∉ *set ks* **by** *simp*
  from *f*[*of 1*] **have** *f0*: *f* ≠ *0* **by** *auto*
  note *IH* = *1*(*1*)[*OF* - *refl* - *p0*] *1*(*2*)[*OF* - *refl*]
  show *?case*
  proof (*cases k* = *0*)
    case *True*
    with *p k*[*unfolded this*, *of hd ks*] *p0* **have** *ks* = []
      by (*cases ks*, *auto simp*: *prod-root-unity-def*)
    with *d p True* **show** *?thesis* **by** (*auto simp*: *prod-root-unity-def*)
  next
    case *k0*: *False*
    note *IH* = *IH*[*OF k0*]
    from *k0* **have** *k* = *0* ⟷ *False* **by** *auto*
    note *d* = *d*[*unfolded this if-False*]
    let *?u* = *root-unity k* :: *complex poly*
    show *?thesis*
    proof (*cases ?u dvd p*)

case *True*
note *IH* = *IH*(*1*)[*OF True*]
let *?call* = *decompose-prod-root-unity-main* (*p div ?u*) *k*
from *True d* **obtain** *Ks* **where** *rec*: *?call* = (*Ks*,*g*) **and** *ks'*: *ks'* = (*k* # *Ks*)
  by (*cases ?call*, *auto*)
from *True* **have** *?u dvd p* ⟷ *True* **by** *simp*
note *d* = *d*[*unfolded this if-True rec*]
let *?x* = *cis* (*2* ∗ *pi* / *k*)
have *rt*: *poly ?u ?x* = *0* **unfolding** *poly-root-unity* **using** *cis-times-2pi*[*of 1*]
  by (*simp add*: *DeMoivre*)
with *True* **have** *poly p ?x* = *0* **unfolding** *dvd-def* **by** *auto*
from *this*[*unfolded p*] *f*[*of ?x*] *rt* **have** *poly* (*prod-root-unity ks*) *?x* = *0*
  unfolding *poly-root-unity* **by** *auto*
from *this*[*unfolded poly-prod-root-unity*] *ks0* **obtain** *k'* **where** *k'*: *k'* ∈ *set ks*
  and *rt*: *?x* ^ *k'* = *1* **and** *k'0*: *k'* ≠ *0* **by** *auto*
let *?u'* = *root-unity k'* :: *complex poly*
from *k' rt k'0* **have** *rtk'*: *poly ?u' ?x* = *0* **unfolding** *poly-root-unity* **by** *auto*
{
  let *?phi* = *k'* ∗ (*2* ∗ *pi* / *k*)
  assume *k'* < *k*
  hence *0* < *?phi ?phi* < *2* ∗ *pi* **using** *k0 k'0* **by** (*auto simp*: *field-simps*)
  from *cis-plus-2pi-neq-1*[*OF this*] *rtk'*
  have *False* **unfolding** *poly-root-unity DeMoivre* **..**
}
hence *kk'*: *k* ≤ *k'* **by** *presburger*
{
  assume *k'* > *k*
  from *k*[*OF this*, *unfolded p*]
  have ¬ *?u' dvd prod-root-unity ks* **using** *dvd-mult2* **by** *auto*
  with *k'* **have** *False* **unfolding** *prod-root-unity-def*
    using *prod-list-dvd*[*of ?u' map root-unity ks*] **by** *auto*
}
with *kk'* **have** *kk'*: *k'* = *k* **by** *presburger*
with *k'* **have** *k* ∈ *set ks* **by** *auto*
from *split-list*[*OF this*] **obtain** *ks1 ks2* **where** *ks*: *ks* = *ks1* @ *k* # *ks2* **by**
auto
hence *p div ?u* = (*?u* ∗ (*prod-root-unity* (*ks1* @ *ks2*) ∗ *f*)) *div ?u*
  by (*simp add*: *ac-simps p prod-root-unity-def*)
also have . . . = *prod-root-unity* (*ks1* @ *ks2*) ∗ *f*
  by (*rule nonzero-mult-div-cancel-left*, *insert k0*, *auto*)
finally have *id*: *p div ?u* = *prod-root-unity* (*ks1* @ *ks2*) ∗ *f* **.**
from *d* **have** *ks'*: *ks'* = *k* # *Ks* **by** *auto*
have *k* < *k'* ⟹ ¬ *root-unity k' dvd p div ?u* **for** *k'*
  using *k*[*of k'*] *True* **by** (*metis dvd-div-mult-self dvd-mult2*)
from *IH*[*OF rec id this*]
have *id*: *p div root-unity k* = *prod-root-unity Ks* ∗ *f* **and**
  ∗: *f* = *g* ∧ *set* (*ks1* @ *ks2*) = *set Ks* **by** *auto*
from *arg-cong*[*OF id*, *of* λ *x*. *x* ∗ *?u*] *True*
have *p* = *prod-root-unity Ks* ∗ *f* ∗ *root-unity k* **by** *auto*

thus *?thesis* **using** ∗ **unfolding** *ks ks′* **by** (*auto simp*: *prod-root-unity-def*)
next
case *False*
from *d False* **have** *decompose-prod-root-unity-main p* (*k* − *1*) = (*ks′*,*g*) **by** *auto*
note *IH* = *IH*(*2*)[*OF False this p*]
have *k*: *k* − *1* < *k′* ⟹ ¬ *root-unity k′ dvd p* **for** *k′* **using** *False k*[*of k′*] *k0*
by (*cases k′* = *k*, *auto*)
show *?thesis* **by** (*rule IH*, *insert False k*, *auto*)
qed
qed
qed

definition *decompose-prod-root-unity p* = *decompose-prod-root-unity-main p* (*degree p*)

lemma *decompose-prod-root-unity*: **fixes** *p* :: *complex poly*
assumes *p*: *p* = *prod-root-unity ks* ∗ *f*
and *d*: *decompose-prod-root-unity p* = (*ks′*,*g*)
and *f*: ⋀ *x*. *cmod x* = *1* ⟹ *poly f x* ≠ *0*
and *p0*: *p* ≠ *0*
shows *p* = *prod-root-unity ks′* ∗ *f* ∧ *f* = *g* ∧ *set ks* = *set ks′*
proof (*rule decompose-prod-root-unity-main*[*OF p d*[*unfolded decompose-prod-root-unity-def*] *f*])
fix *k*
assume *deg*: *degree p* < *k*
hence *degree p* < *degree* (*root-unity k*) **by** *simp*
with *p0* **show** ¬ *root-unity k dvd p*
by (*simp add*: *poly-divides-conv0*)
qed

lemma (**in** *comm-ring-hom*) *hom-root-unity*: *map-poly hom* (*root-unity n*) = *root-unity n*
proof −
interpret *p*: *map-poly-comm-ring-hom hom* **..**
show *?thesis* **unfolding** *root-unity-def*
by (*simp add*: *hom-distribs*)
qed

lemma (**in** *idom-hom*) *hom-prod-root-unity*: *map-poly hom* (*prod-root-unity n*) = *prod-root-unity n*
proof −
interpret *p*: *map-poly-comm-ring-hom hom* **..**
show *?thesis* **unfolding** *prod-root-unity-def p.hom-prod-list map-map o-def hom-root-unity* **..**
qed

lemma (**in** *field-hom*) *hom-decompose-prod-root-unity-main*:
decompose-prod-root-unity-main (*map-poly hom p*) *k* = *map-prod id* (*map-poly*

```
hom)
    (decompose-prod-root-unity-main p k)
proof (induct p k rule: decompose-prod-root-unity-main.induct)
  case (1 p k)
  let ?h = map-poly hom
  let ?p = ?h p
  let ?u = root-unity k :: 'a poly
  let ?u' = root-unity k :: 'b poly
  interpret p: map-poly-inj-idom-divide-hom hom ..
  have u': ?u' = ?h ?u unfolding hom-root-unity ..
  note simp = decompose-prod-root-unity-main.simps
  let ?rec1 = decompose-prod-root-unity-main (p div ?u) k
  have 0: ?p = 0 ⟷ p = 0 by simp
  show ?case
    unfolding simp[of ?p k] simp[of p k] if-distrib[of map-prod id ?h] Let-def u'
    unfolding 0 p.hom-div[symmetric] p.hom-dvd-iff
    by (rule if-cong[OF refl], force, rule if-cong[OF refl if-cong[OF refl]], force,
     (subst 1(1), auto, cases ?rec1, auto)[1],
     (subst 1(2), auto))
qed

lemma (in field-hom) hom-decompose-prod-root-unity:
  decompose-prod-root-unity (map-poly hom p) = map-prod id (map-poly hom)
    (decompose-prod-root-unity p)
  unfolding decompose-prod-root-unity-def
  by (subst hom-decompose-prod-root-unity-main, simp)


end
```

5.1 The Perron Frobenius Theorem for Irreducible Matrices

```
theory Perron-Frobenius-Irreducible
imports
  Perron-Frobenius
  Roots-Unity
  Rank-Nullity-Theorem.Miscellaneous
begin

lifting-forget vec.lifting
lifting-forget mat.lifting
lifting-forget poly.lifting

lemma charpoly-of-real: charpoly (map-matrix complex-of-real A) = map-poly of-real
(charpoly A)
  by (transfer-hma rule: of-real-hom.char-poly-hom)

context includes lifting-syntax
begin
```

lemma *HMA-M-smult*[*transfer-rule*]: ((=) ===> *HMA-M* ===> *HMA-M*) $(\cdot_m)$ ((*∗k*))
unfolding *smult-mat-def*
unfolding *rel-fun-def HMA-M-def from-hma*$_m$*-def*
by (*auto simp*: *matrix-scalar-mult-def*)
end

lemma *order-charpoly-smult*: **fixes** *A* :: *complex* ^ *'n* ^ *'n*
assumes *k*: $k \neq 0$
shows *order x* (*charpoly* (*k ∗k A*)) = *order* (*x* / *k*) (*charpoly A*)
by (*transfer fixing*: *k*, *rule order-char-poly-smult*[*OF* - *k*])

lemma *smult-eigen-vector*: **fixes** *a* :: *'a* :: *field*
assumes *eigen-vector A v x*
shows *eigen-vector* (*a ∗k A*) *v* (*a ∗ x*)
proof −
from *assms*[*unfolded eigen-vector-def*] **have** *v*: $v \neq 0$ **and** *id*: *A ∗v v* = *x ∗s v*
by *auto*
from *arg-cong*[*OF id*, *of* (*∗s*) *a*] **have** *id*: (*a ∗k A*) *∗v v* = (*a ∗ x*) *∗s v*
unfolding *scalar-matrix-vector-assoc* **by** *simp*
thus *eigen-vector* (*a ∗k A*) *v* (*a ∗ x*) **using** *v* **unfolding** *eigen-vector-def* **by** *auto*
qed

lemma *smult-eigen-value*: **fixes** *a* :: *'a* :: *field*
assumes *eigen-value A x*
shows *eigen-value* (*a ∗k A*) (*a ∗ x*)
using *assms smult-eigen-vector*[*of A* - *x a*] **unfolding** *eigen-value-def* **by** *blast*

locale *fixed-mat* = **fixes** *A* :: *'a* :: *zero* ^ *'n* ^ *'n*
begin
definition *G* :: *'n rel* **where**
G = { (*i*,*j*). *A* \$ *i* \$ *j* $\neq$ *0*}

definition *irreducible* :: *bool* **where**
irreducible = (*UNIV* ⊆ *G*^+)
end

lemma *G-transpose*:
fixed-mat.G (*transpose A*) = ((*fixed-mat.G A*))^−*1*
unfolding *fixed-mat.G-def* **by** (*force simp*: *transpose-def*)

lemma *G-transpose-trancl*:
(*fixed-mat.G* (*transpose A*))^+ = ((*fixed-mat.G A*)^+)^−*1*
unfolding *G-transpose trancl-converse* **by** *auto*

locale *pf-nonneg-mat* = *fixed-mat A* **for**
A :: *'a* :: *linordered-idom* ^ *'n* ^ *'n* +

assumes *non-neg-mat*: *non-neg-mat A*
begin
lemma *nonneg*: *A* $ *i* $ *j* ≥ *0*
using *non-neg-mat* **unfolding** *non-neg-mat-def elements-mat-h-def* **by** *auto*

lemma *nonneg-matpow*: *matpow A n* $ *i* $ *j* ≥ *0*
by (*induct n arbitrary*: *i j*, *insert nonneg*,
auto intro!: *sum-nonneg simp*: *matrix-matrix-mult-def mat-def*)

lemma *G-relpow-matpow-pos*: (*i*,*j*) ∈ *G* ^^ *n* ⟹ *matpow A n* $ *i* $ *j* > *0*
proof (*induct n arbitrary*: *i j*)
case (*0 i*)
thus *?case* **by** (*auto simp*: *mat-def*)
next
case (*Suc n i j*)
from *Suc*(*2*) **have** (*i*,*j*) ∈ *G* ^^ *n O G*
by (*simp add*: *relpow-commute*)
then obtain *k* **where**
ik: *A* $ *k* $ *j* ≠ *0* **and** *kj*: (*i*, *k*) ∈ *G* ^^ *n* **by** (*auto simp*: *G-def*)
from *ik nonneg*[*of k j*] **have** *ik*: *A* $ *k* $ *j* > *0* **by** *auto*
from *Suc*(*1*)[*OF kj*] **have** *IH*: *matpow A n* $*h i* $*h k* > *0* **.**
thus *?case* **using** *ik* **by** (*auto simp*: *nonneg-matpow nonneg matrix-matrix-mult-def*

intro!: *sum-pos2*[*of - k*] *mult-nonneg-nonneg*)
qed

lemma *matpow-mono*: **assumes** *B*: ⋀ *i j*. *B* $ *i* $ *j* ≥ *A* $ *i* $ *j*
shows *matpow B n* $ *i* $ *j* ≥ *matpow A n* $ *i* $ *j*
proof (*induct n arbitrary*: *i j*)
case (*Suc n i j*)
thus *?case* **using** *B nonneg-matpow*[*of n*] *nonneg*
by (*auto simp*: *matrix-matrix-mult-def intro*!: *sum-mono mult-mono′*)
qed *simp*

lemma *matpow-sum-one-mono*: *matpow* (*A* + *mat 1*) (*n* + *k*) $ *i* $ *j* ≥ *matpow* (*A* + *mat 1*) *n* $ *i* $ *j*
proof (*induct k*)
case (*Suc k*)
have (*matpow* (*A* + *mat 1*) (*n* + *k*) ** *A*) $*h i* $*h j* ≥ *0* **unfolding** *matrix-matrix-mult-def*
using *order.trans*[*OF nonneg-matpow matpow-mono*[*of A* + *mat 1 n* + *k*]]
by (*auto intro*!: *sum-nonneg mult-nonneg-nonneg nonneg simp*: *mat-def*)
thus *?case* **using** *Suc* **by** (*simp add*: *matrix-add-ldistrib matrix-mul-rid*)
qed *simp*

lemma *G-relpow-matpow-pos-ge*:
assumes (*i*,*j*) ∈ *G* ^^ *m n* ≥ *m*
shows *matpow* (*A* + *mat 1*) *n* $ *i* $ *j* > *0*
proof −

from *assms*(*2*) **obtain** *k* **where** *n*: *n* = *m* + *k* **using** *le-Suc-ex* **by** *blast*
have *0* < *matpow A m* $ *i* $ *j* **by** (*rule G-relpow-matpow-pos*[*OF assms*(*1*)])
also have ... ≤ *matpow* (*A* + *mat 1*) *m* $ *i* $ *j*
by (*rule matpow-mono*, *auto simp*: *mat-def*)
also have ... ≤ *matpow* (*A* + *mat 1*) *n* $ *i* $ *j* **unfolding** *n* **using** *matpow-sum-one-mono* **.**
finally show *?thesis* **.**
qed
end

locale *perron-frobenius* = *pf-nonneg-mat A*
for *A* :: *real* ^ *'n* ^ *'n* +
assumes *irr*: *irreducible*
begin

definition *N* **where** *N* = (*SOME N*. ∀ *ij*. ∃ *n* ≤ *N*. *ij* ∈ *G* ^^ *n*)

lemma *N*: ∃ *n* ≤ *N*. *ij* ∈ *G* ^^ *n*
proof −
{
fix *ij*
have *ij* ∈ *G*^+ **using** *irr*[*unfolded irreducible-def*] **by** *auto*
from *this*[*unfolded trancl-power*] **have** ∃ *n*. *ij* ∈ *G* ^^ *n* **by** *blast*
}
hence ∀ *ij*. ∃ *n*. *ij* ∈ *G* ^^ *n* **by** *auto*
from *choice*[*OF this*] **obtain** *f* **where** *f*: ⋀ *ij*. *ij* ∈ *G* ^^ (*f ij*) **by** *auto*
define *N* **where** *N*: *N* = *Max* (*range f*)
{
fix *ij*
from *f*[*of ij*] **have** *ij* ∈ *G* ^^ *f ij* **.**
moreover have *f ij* ≤ *N* **unfolding** *N*
by (*rule Max-ge*, *auto*)
ultimately have ∃ *n* ≤ *N*. *ij* ∈ *G* ^^ *n* **by** *blast*
} note *main* = *this*
let *?P* = λ *N*. ∀ *ij*. ∃ *n* ≤ *N*. *ij* ∈ *G* ^^ *n*
from *main* **have** *?P N* **by** *blast*
from *someI*[*of ?P*, *OF this*, *folded N-def*]
show *?thesis* **by** *blast*
qed

lemma *irreducible-matpow-pos*: **assumes** *irreducible*
shows *matpow* (*A* + *mat 1*) *N* $ *i* $ *j* > *0*
proof −
from *N* **obtain** *n* **where** *n*: *n* ≤ *N* **and** *reach*: (*i*,*j*) ∈ *G* ^^ *n* **by** *auto*
show *?thesis* **by** (*rule G-relpow-matpow-pos-ge*[*OF reach n*])
qed

lemma *pf-transpose*: *perron-frobenius* (*transpose A*)
proof

show *fixed-mat.irreducible* (*transpose A*)
unfolding *fixed-mat.irreducible-def G-transpose-trancl* **using** *irr*[*unfolded irreducible-def*]
by *auto*
qed (*insert nonneg*, *auto simp*: *transpose-def non-neg-mat-def elements-mat-h-def*)

abbreviation *le-vec* :: *real* ^ *'n* ⇒ *real* ^ *'n* ⇒ *bool* **where**
le-vec x y ≡ (∀ *i*. *x* $ *i* ≤ *y* $ *i*)

abbreviation *lt-vec* :: *real* ^ *'n* ⇒ *real* ^ *'n* ⇒ *bool* **where**
lt-vec x y ≡ (∀ *i*. *x* $ *i* < *y* $ *i*)

definition *A1n* = *matpow* (*A* + *mat 1*) *N*

lemmas *A1n-pos* = *irreducible-matpow-pos*[*OF irr*, *folded A1n-def*]

definition *r* :: *real* ^ *'n* ⇒ *real* **where**
r x = *Min* { (*A* ∗*v x*) $ *j* / *x* $ *j* | *j*. *x* $ *j* ≠ *0* }

definition *X* :: (*real* ^ *'n*)*set* **where**
X = { *x* . *le-vec 0 x* ∧ *x* ≠ *0* }

lemma *nonneg-Ax*: *x* ∈ *X* ⟹ *le-vec 0* (*A* ∗*v x*)
unfolding *X-def* **using** *nonneg*
by (*auto simp*: *matrix-vector-mult-def intro*!: *sum-nonneg*)

lemma *A-nonzero-fixed-i*: ∃ *j*. *A* $ *i* $ *j* ≠ *0*
proof −
from *irr*[*unfolded irreducible-def*] **have** (*i*,*i*) ∈ *G*^+ **by** *auto*
then obtain *j* **where** (*i*,*j*) ∈ *G* **by** (*metis converse-tranclE*)
hence *Aij*: *A* $ *i* $ *j* ≠ *0* **unfolding** *G-def* **by** *auto*
thus *?thesis* **..**
qed

lemma *A-nonzero-fixed-j*: ∃ *i*. *A* $ *i* $ *j* ≠ *0*
proof −
from *irr*[*unfolded irreducible-def*] **have** (*j*,*j*) ∈ *G*^+ **by** *auto*
then obtain *i* **where** (*i*,*j*) ∈ *G* **by** (*cases*, *auto*)
hence *Aij*: *A* $ *i* $ *j* ≠ *0* **unfolding** *G-def* **by** *auto*
thus *?thesis* **..**
qed

lemma *Ax-pos*: **assumes** *x*: *lt-vec 0 x*
shows *lt-vec 0* (*A* ∗*v x*)
proof
fix *i*
from *A-nonzero-fixed-i*[*of i*] **obtain** *j* **where** *A* $ *i* $ *j* ≠ *0* **by** *auto*
with *nonneg*[*of i j*] **have** *A*: *A* $ *i* $ *j* > *0* **by** *simp*
from *x* **have** *x* $ *j* ≥ *0* **for** *j* **by** (*auto simp*: *order.strict-iff-order*)

note *nonneg* = *mult-nonneg-nonneg*[*OF nonneg*[*of i*] *this*]
have (*A* ∗*v x*) $ *i* = (∑*j*∈*UNIV*. *A* $ *i* $ *j* ∗ *x* $ *j*)
unfolding *matrix-vector-mult-def* **by** *simp*
also have ... = *A* $ *i* $ *j* ∗ *x* $ *j* + (∑*j*∈*UNIV* − {*j*}. *A* $ *i* $ *j* ∗ *x* $ *j*)
by (*subst sum.remove*, *auto*)
also have ... > *0* + *0*
by (*rule add-less-le-mono*, *insert A x*[*rule-format*] *nonneg*,
auto intro!: *sum-nonneg mult-pos-pos*)
finally show *0* $ *i* < (*A* ∗*v x*) $ *i* **by** *simp*
qed

lemma *nonzero-Ax*: **assumes** *x*: *x* ∈ *X*
shows *A* ∗*v x* ≠ *0*
proof
assume *0*: *A* ∗*v x* = *0*
from *x*[*unfolded X-def*] **have** *x*: *le-vec 0 x x* ≠ *0* **by** *auto*
from *x*(*2*) **obtain** *j* **where** *xj*: *x* $ *j* ≠ *0*
by (*metis vec-eq-iff zero-index*)
from *A-nonzero-fixed-j*[*of j*] **obtain** *i* **where** *Aij*: *A* $ *i* $ *j* ≠ *0* **by** *auto*
from *arg-cong*[*OF 0*, *of* λ *v*. *v* $ *i*, *unfolded matrix-vector-mult-def*]
have *0* = (∑ *k* ∈ *UNIV*. *A* $*h i* $*h k* ∗ *x* $*h k*) **by** *auto*
also have ... = *A* $*h i* $*h j* ∗ *x* $*h j* + (∑ *k* ∈ *UNIV* − {*j*}. *A* $*h i* $*h k* ∗ *x*
$*h k*)
by (*subst sum.remove*[*of - j*], *auto*)
also have ... > *0* + *0*
by (*rule add-less-le-mono*, *insert nonneg*[*of i*] *Aij x*(*1*) *xj*,
auto intro!: *sum-nonneg mult-pos-pos simp*: *dual-order.not-eq-order-implies-strict*)

finally show *False* **by** *simp*
qed

lemma *r-witness*: **assumes** *x*: *x* ∈ *X*
shows ∃ *j*. *x* $ *j* > *0* ∧ *r x* = (*A* ∗*v x*) $ *j* / *x* $ *j*
proof −
from *x*[*unfolded X-def*] **have** *x*: *le-vec 0 x x* ≠ *0* **by** *auto*
let *?A* = { (*A* ∗*v x*) $ *j* / *x* $ *j* | *j*. *x* $ *j* ≠ *0* }
from *x*(*2*) **obtain** *j* **where** *x* $ *j* ≠ *0*
by (*metis vec-eq-iff zero-index*)
hence *empty*: *?A* ≠ {} **by** *auto*
from *Min-in*[*OF - this*, *folded r-def*]
obtain *j* **where** *x* $ *j* ≠ *0* **and** *rx*: *r x* = (*A* ∗*v x*) $ *j* / *x* $ *j* **by** *auto*
with *x* **have** *x* $ *j* > *0* **by** (*auto simp*: *dual-order.not-eq-order-implies-strict*)
with *rx* **show** *?thesis* **by** *auto*
qed

lemma *rx-nonneg*: **assumes** *x*: *x* ∈ *X*
shows *r x* ≥ *0*
proof −

from *x*[*unfolded X-def*] **have** *x*: *le-vec 0 x x* ≠ *0* **by** *auto*
let *?A* = { (*A* ∗*v x*) $ *j* / *x* $ *j* | *j*. *x* $ *j* ≠ *0* }
from *r-witness*[*OF* ‹*x* ∈ *X*›]
have *empty*: *?A* ≠ {} **by** *force*
show *?thesis* **unfolding** *r-def X-def*
proof (*subst Min-ge-iff*, *force*, *use empty* **in** *force*, *intro ballI*)
fix *y*
assume *y* ∈ *?A*
then obtain *j* **where** *y* = (*A* ∗*v x*) $ *j* / *x* $ *j* **and** *x* $ *j* ≠ *0* **by** *auto*
from *nonneg-Ax*[*OF* ‹*x* ∈ *X*›] *this x*
show *0* ≤ *y* **by** *simp*
qed
qed

lemma *rx-pos*: **assumes** *x*: *lt-vec 0 x*
shows *r x* > *0*
proof −
from *Ax-pos*[*OF x*] **have** *lt*: *lt-vec 0* (*A* ∗*v x*) **.**
from *x* **have** *x′*: *x* ∈ *X* **unfolding** *X-def order.strict-iff-order* **by** *auto*
let *?A* = { (*A* ∗*v x*) $ *j* / *x* $ *j* | *j*. *x* $ *j* ≠ *0* }
from *r-witness*[*OF* ‹*x* ∈ *X*›]
have *empty*: *?A* ≠ {} **by** *force*
show *?thesis* **unfolding** *r-def X-def*
proof (*subst Min-gr-iff*, *force*, *use empty* **in** *force*, *intro ballI*)
fix *y*
assume *y* ∈ *?A*
then obtain *j* **where** *y* = (*A* ∗*v x*) $ *j* / *x* $ *j* **and** *x* $ *j* ≠ *0* **by** *auto*
from *lt this x* **show** *0* < *y* **by** *simp*
qed
qed

lemma *rx-le-Ax*: **assumes** *x*: *x* ∈ *X*
shows *le-vec* (*r x* ∗*s x*) (*A* ∗*v x*)
proof (*intro allI*)
fix *i*
show (*r x* ∗*s x*) $*h i* ≤ (*A* ∗*v x*) $*h i*
proof (*cases x* $ *i* = *0*)
case *True*
with *nonneg-Ax*[*OF x*] **show** *?thesis* **by** *auto*
next
case *False*
with *x*[*unfolded X-def*] **have** *pos*: *x* $ *i* > *0*
by (*auto simp*: *dual-order.not-eq-order-implies-strict*)
from *False* **have** (*A* ∗*v x*) $*h i* / *x* $ *i* ∈ { (*A* ∗*v x*) $ *j* / *x* $ *j* | *j*. *x* $ *j* ≠ *0*
} **by** *auto*
hence (*A* ∗*v x*) $*h i* / *x* $ *i* ≥ *r x* **unfolding** *r-def* **by** *simp*
hence *x* $ *i* ∗ *r x* ≤ *x* $ *i* ∗ ((*A* ∗*v x*) $*h i* / *x* $ *i*) **unfolding** *mult-le-cancel-left-pos*[*OF pos*] **.**
also have ... = (*A* ∗*v x*) $*h i* **using** *pos* **by** *simp*

```
    finally show ?thesis by (simp add: ac-simps)
  qed
qed

lemma rho-le-x-Ax-imp-rho-le-rx: assumes x: x ∈ X
  and ϱ: le-vec (ϱ *s x) (A *v x)
shows ϱ ≤ r x
proof −
  from r-witness[OF x] obtain j where
    rx: r x = (A *v x) $ j / x $ j and xj: x $ j > 0 x $ j ≥ 0 by auto
  from divide-right-mono[OF ϱ[rule-format, of j] xj(2)]
  show ?thesis unfolding rx using xj by simp
qed

lemma rx-Max: assumes x: x ∈ X
  shows r x = Sup { ϱ . le-vec (ϱ *s x) (A *v x) } (is - = Sup ?S)
proof −
  have r x ∈ ?S using rx-le-Ax[OF x] by auto
  moreover {
    fix y
    assume y ∈ ?S
    hence y: le-vec (y *s x) (A *v x) by auto
    from rho-le-x-Ax-imp-rho-le-rx[OF x this]
    have y ≤ r x .
  }
  ultimately show ?thesis by (metis (mono-tags, lifting) cSup-eq-maximum)
qed

lemma r-smult: assumes x: x ∈ X
  and a: a > 0
shows r (a *s x) = r x
  unfolding r-def
  by (rule arg-cong[of - - Min], unfold vector-smult-distrib, insert a, simp)

definition X1 = (X ∩ {x. norm x = 1})

lemma bounded-X1: bounded X1 unfolding bounded-iff X1-def by auto

lemma closed-X1: closed X1
proof −
  have X1: X1 = {x. le-vec 0 x ∧ norm x = 1}
    unfolding X1-def X-def by auto
  show ?thesis unfolding X1
   by (intro closed-Collect-conj closed-Collect-all  closed-Collect-le closed-Collect-eq,
      auto intro: continuous-intros)
qed

lemma compact-X1: compact X1 using bounded-X1 closed-X1
  by (simp add: compact-eq-bounded-closed)
```

definition *pow-A-1 x* = *A1n* ∗*v x*

lemma *continuous-pow-A-1*: *continuous-on R pow-A-1*
unfolding *pow-A-1-def continuous-on*
by (*auto intro*: *tendsto-intros*)

definition *Y* = *pow-A-1* ‘ *X1*

lemma *compact-Y*: *compact Y*
unfolding *Y-def* **using** *compact-X1 continuous-pow-A-1*[*of X1*]
by (*metis compact-continuous-image*)

lemma *Y-pos-main*: **assumes** *y*: *y* ∈ *pow-A-1* ‘ *X*
shows *y* $ *i* > *0*
proof −
from *y* **obtain** *x* **where** *x*: *x* ∈ *X* **and** *y*: *y* = *pow-A-1 x* **unfolding** *Y-def X1-def* **by** *auto*
from *r-witness*[*OF x*] **obtain** *j* **where** *xj*: *x* $ *j* > *0* **by** *auto*
from *x*[*unfolded X-def*] **have** *xi*: *x* $ *i* ≥ *0* **for** *i* **by** *auto*
have *nonneg*: *0* ≤ *A1n* $ *i* $ *k* ∗ *x* $ *k* **for** *k* **using** *A1n-pos*[*of i k*] *xi*[*of k*] **by** *auto*
have *y* $ *i* = ($\sum$*j*∈*UNIV*. *A1n* $ *i* $ *j* ∗ *x* $ *j*)
unfolding *y pow-A-1-def matrix-vector-mult-def* **by** *simp*
also have ... = *A1n* $ *i* $ *j* ∗ *x* $ *j* + ($\sum$*j*∈*UNIV* − {*j*}. *A1n* $ *i* $ *j* ∗ *x* $ *j*)
by (*subst sum.remove*, *auto*)
also have ... > *0* + *0*
by (*rule add-less-le-mono*, *insert xj A1n-pos nonneg*,
auto intro!: *sum-nonneg mult-pos-pos simp*: *dual-order.not-eq-order-implies-strict*)
finally show *?thesis* **by** *simp*
qed

lemma *Y-pos*: **assumes** *y*: *y* ∈ *Y*
shows *y* $ *i* > *0*
using *Y-pos-main*[*of y i*] *y* **unfolding** *Y-def X1-def* **by** *auto*

lemma *Y-nonzero*: **assumes** *y*: *y* ∈ *Y*
shows *y* $ *i* ≠ *0*
using *Y-pos*[*OF y*, *of i*] **by** *auto*

definition *r′* :: *real* ^ ′*n* ⇒ *real* **where**
r′ x = *Min* (*range* (λ *j*. (*A* ∗*v x*) $ *j* / *x* $ *j*))

lemma *r′-r*: **assumes** *x*: *x* ∈ *Y* **shows** *r′ x* = *r x*
unfolding *r′-def r-def*
proof (*rule arg-cong*[*of - - Min*])
have *range* (λ*j*. (*A* ∗*v x*) $ *j* / *x* $ *j*) ⊆ {(*A* ∗*v x*) $ *j* / *x* $ *j* |*j*. *x* $ *j* ≠ *0*} (**is**

?L ⊆ *?R*)
  proof
    fix *y*
    assume $y \in$ *?L*
    then obtain *j* **where** $y = (A *v\ x)\ \$\ j\ /\ x\ \$\ j$ **by** *auto*
    with *Y-pos*[*OF x*, *of j*] **show** $y \in$ *?R* **by** *auto*
  qed
  moreover have *?L* ⊇ *?R* **by** *auto*
  ultimately show *?L* = *?R* **by** *blast*
qed

lemma *continuous-Y-r*: *continuous-on Y r*
proof −
  have ∗: (∀ $y \in Y$. *P y* (*r y*)) = (∀ $y \in Y$. *P y* (*r′ y*)) **for** *P* **using** *r′-r* **by** *auto*
  have *continuous-on Y r* = *continuous-on Y r′*
    by (*rule continuous-on-cong*[*OF refl r′-r*[*symmetric*]])
  also have . . .
    unfolding *continuous-on r′-def* **using** *Y-nonzero*
    by (*auto intro*!: *tendsto-Min tendsto-intros*)
  finally show *?thesis* **.**
qed

lemma *X1-nonempty*: *X1* ≠ {}
proof −
  define *x* **where** *x* = ((χ *i*. *if i* = *undefined then 1 else 0*) :: *real* ^ ′*n*)
  {
    assume *x* = *0*
    from *arg-cong*[*OF this*, *of* λ *x*. *x* $ *undefined*] **have** *False* **unfolding** *x-def* **by**
auto
  }
  hence *x*: *x* ≠ *0* **by** *auto*
  moreover have *le-vec 0 x* **unfolding** *x-def* **by** *auto*
  moreover have *norm x* = *1* **unfolding** *norm-vec-def L2-set-def*
    by (*auto*, *subst sum.remove*[*of - undefined*], *auto simp*: *x-def*)
  ultimately show *?thesis* **unfolding** *X1-def X-def* **by** *auto*
qed

lemma *Y-nonempty*: *Y* ≠ {}
  unfolding *Y-def* **using** *X1-nonempty* **by** *auto*

definition *z* **where** *z* = (*SOME z*. $z \in Y$ ∧ (∀ $y \in Y$. *r y* ≤ *r z*))

abbreviation *sr* ≡ *r z*

lemma *z*: $z \in Y$ **and** *sr-max-Y*: ⋀ *y*. $y \in Y$ ⟹ *r y* ≤ *sr*
proof −
  let *?P* = λ *z*. $z \in Y$ ∧ (∀ $y \in Y$. *r y* ≤ *r z*)
  from *continuous-attains-sup*[*OF compact-Y Y-nonempty continuous-Y-r*]
  obtain *y* **where** *?P y* **by** *blast*

from *someI*[*of ?P*, *OF this*, *folded z-def*]
show $z \in Y \bigwedge y.\ y \in Y \Longrightarrow r\ y \leq r\ z$ **by** *blast+*
qed

lemma *Y-subset-X*: $Y \subseteq X$
proof
fix *y*
assume $y \in Y$
from *Y-pos*[*OF this*] **show** $y \in X$ **unfolding** *X-def*
by (*auto simp*: *order.strict-iff-order*)
qed

lemma *zX*: $z \in X$
using *z*(*1*) *Y-subset-X* **by** *auto*

lemma *le-vec-mono-left*: **assumes** *B*: $\bigwedge$ *i j*. *B* \$ *i* \$ *j* $\geq$ *0*
and *le-vec x y*
shows *le-vec* (*B* ∗*v x*) (*B* ∗*v y*)
proof (*intro allI*)
fix *i*
show (*B* ∗*v x*) \$ *i* $\leq$ (*B* ∗*v y*) \$ *i* **unfolding** *matrix-vector-mult-def* **using** *B*[*of i*] *assms*(*2*)
by (*auto intro*!: *sum-mono mult-left-mono*)
qed

lemma *matpow-1-commute*: *matpow* (*A* + *mat 1*) *n* ∗∗ *A* = *A* ∗∗ *matpow* (*A* + *mat 1*) *n*
by (*induct n*, *auto simp*: *matrix-add-rdistrib matrix-add-ldistrib matrix-mul-rid matrix-mul-lid*
matrix-mul-assoc[*symmetric*])

lemma *A1n-commute*: *A1n* ∗∗ *A* = *A* ∗∗ *A1n*
unfolding *A1n-def* **by** (*rule matpow-1-commute*)

lemma *le-vec-pow-A-1*: **assumes** *le*: *le-vec* (*rho* ∗*s x*) (*A* ∗*v x*)
shows *le-vec* (*rho* ∗*s pow-A-1 x*) (*A* ∗*v pow-A-1 x*)
proof −
have *A1n* \$ *i* \$ *j* $\geq$ *0* **for** *i j* **using** *A1n-pos*[*of i j*] **by** *auto*
from *le-vec-mono-left*[*OF this le*]
have *le-vec* (*A1n* ∗*v* (*rho* ∗*s x*)) (*A1n* ∗*v* (*A* ∗*v x*)) **.**
also have *A1n* ∗*v* (*A* ∗*v x*) = (*A1n* ∗∗ *A*) ∗*v x* **by** (*simp add*: *matrix-vector-mul-assoc*)
also have ... = *A* ∗*v* (*A1n* ∗*v x*) **unfolding** *A1n-commute* **by** (*simp add*: *matrix-vector-mul-assoc*)
also have ... = *A* ∗*v* (*pow-A-1 x*) **unfolding** *pow-A-1-def* **..**
also have *A1n* ∗*v* (*rho* ∗*s x*) = *rho* ∗*s* (*A1n* ∗*v x*) **unfolding** *vector-smult-distrib* **..**
also have ... = *rho* ∗*s pow-A-1 x* **unfolding** *pow-A-1-def* **..**
finally show *le-vec* (*rho* ∗*s pow-A-1 x*) (*A* ∗*v pow-A-1 x*) **.**

qed

lemma *r-pow-A-1*: **assumes** *x*: $x \in X$
  shows $r\ x \leq r\ (pow\text{-}A\text{-}1\ x)$
proof −
  let $?y = pow\text{-}A\text{-}1\ x$
  have $?y \in pow\text{-}A\text{-}1$ ' X **using** *x* **by** *auto*
  from *Y-pos-main*[*OF this*]
  have *y*: $?y \in X$ **unfolding** *X-def* **by** (*auto simp*: *order.strict-iff-order*)
  let $?A = \{\varrho.\ le\text{-}vec\ (\varrho *s\ x)\ (A *v\ x)\}$
  let $?B = \{\varrho.\ le\text{-}vec\ (\varrho *s\ pow\text{-}A\text{-}1\ x)\ (A *v\ pow\text{-}A\text{-}1\ x)\}$
  show *?thesis* **unfolding** *rx-Max*[*OF x*] *rx-Max*[*OF y*]
  proof (*rule cSup-mono*)
    show *bdd-above ?B* **using** *rho-le-x-Ax-imp-rho-le-rx*[*OF y*] **by** *fast*
    show $?A \neq \{\}$ **using** *rx-le-Ax*[*OF x*] **by** *auto*
    fix *rho*
    assume $rho \in ?A$
    hence $le\text{-}vec\ (rho *s\ x)\ (A *v\ x)$ **by** *auto*
    from *le-vec-pow-A-1*[*OF this*] **have** $rho \in ?B$ **by** *auto*
    thus $\exists\ rho' \in ?B.\ rho \leq rho'$ **by** *auto*
  qed
qed

lemma *sr-max*: **assumes** *x*: $x \in X$
  shows $r\ x \leq sr$
proof −
  let $?n = norm\ x$
  define x' **where** $x' = inverse\ ?n *s\ x$
  from *x*[*unfolded X-def*] **have** *x0*: $x \neq 0$ **by** *auto*
  hence *n*: $?n > 0$ **by** *auto*
  have x': $x' \in X1$ $x' \in X$ **using** *x n* **unfolding** *X1-def X-def* x'*-def* **by** (*auto simp*: *norm-smult*)
  have *id*: $r\ x = r\ x'$ **unfolding** x'*-def*
    by (*rule sym*, *rule r-smult*[*OF x*], *insert n*, *auto*)
  define *y* **where** $y = pow\text{-}A\text{-}1\ x'$
  from x' **have** *y*: $y \in Y$ **unfolding** *Y-def y-def* **by** *auto*
  note *id*
  also have $r\ x' \leq r\ y$ **using** *r-pow-A-1*[*OF* $x'(2)$] **unfolding** *y-def* **.**
  also have $\ldots \leq r\ z$ **using** *sr-max-Y*[*OF y*] **.**
  finally show $r\ x \leq r\ z$ **.**
qed

lemma *z-pos*: $z\ \$\ i > 0$
  using *Y-pos*[*OF z*(*1*)] **by** *auto*

lemma *sr-pos*: $sr > 0$
  by (*rule rx-pos*, *insert z-pos*, *auto*)

context fixes *u*

assumes *u*: $u \in X$ **and** *ru*: $r\ u = sr$
begin

lemma *sr-imp-eigen-vector-main*: $sr *s\ u = A *v\ u$
proof (*rule ccontr*)
  assume $*$: $sr *s\ u \neq A *v\ u$
  let $?x = A *v\ u - sr *s\ u$
  from $*$ **have** *0*: $?x \neq 0$ **by** *auto*
  let $?y = pow\text{-}A\text{-}1\ u$
  have *le-vec* $(sr *s\ u)$ $(A *v\ u)$ **using** *rx-le-Ax*[*OF u*] **unfolding** *ru* **.**
  hence *le*: *le-vec 0 ?x* **by** *auto*
  from *0 le* **have** *x*: $?x \in X$ **unfolding** *X-def* **by** *auto*
  have *y-pos*: *lt-vec 0 ?y* **using** *Y-pos-main*[*of ?y*] *u* **by** *auto*
  hence *y*: $?y \in X$ **unfolding** *X-def* **by** (*auto simp*: *order.strict-iff-order*)
  from *Y-pos-main*[*of pow-A-1 ?x*] *x*
  have *lt-vec 0* (*pow-A-1 ?x*) **by** *auto*
  hence *lt*: *lt-vec* $(sr *s\ ?y)$ $(A *v\ ?y)$ **unfolding** *pow-A-1-def matrix-vector-right-distrib-diff*
    matrix-vector-mul-assoc A1n-commute vector-smult-distrib **by** *simp*
  let $?f = (\lambda\ i.\ (A *v\ ?y - sr *s\ ?y)\ \$\ i\ /\ ?y\ \$\ i)$
  let $?U = UNIV :: 'n\ set$
  define *eps* **where** $eps = Min\ (?f\ `\ ?U)$
  have *U*: *finite* $(?f\ `\ ?U)$ $?f\ `\ ?U \neq \{\}$ **by** *auto*
  have *eps*: $eps > 0$ **unfolding** *eps-def Min-gr-iff*[*OF U*]
    using *lt sr-pos y-pos* **by** *auto*
  have *le*: *le-vec* $((sr + eps) *s\ ?y)$ $(A *v\ ?y)$
  proof
    fix *i*
    have $((sr + eps) *s\ ?y)\ \$\ i = sr * ?y\ \$\ i + eps * ?y\ \$\ i$
      by (*simp add*: *comm-semiring-class.distrib*)
    also have $\ldots \leq sr * ?y\ \$\ i + ?f\ i * ?y\ \$\ i$
    proof (*rule add-left-mono*[*OF mult-right-mono*])
      show $0 \leq ?y\ \$\ i$ **using** *y-pos*[*rule-format*, *of i*] **by** *auto*
      show $eps \leq ?f\ i$ **unfolding** *eps-def* **by** (*rule Min-le*, *auto*)
    qed
    also have $\ldots = (A *v\ ?y)\ \$\ i$ **using** *sr-pos y-pos*[*rule-format*, *of i*]
      by *simp*
    finally
    show $((sr + eps) *s\ ?y)\ \$\ i \leq (A *v\ ?y)\ \$\ i$ **.**
  qed
  from *rho-le-x-Ax-imp-rho-le-rx*[*OF y le*]
  have $r\ ?y \geq sr + eps$ **.**
  with *sr-max*[*OF y*] *eps* **show** *False* **by** *auto*
qed

lemma *sr-imp-eigen-vector*: *eigen-vector A u sr*
  unfolding *eigen-vector-def sr-imp-eigen-vector-main* **using** *u* **unfolding** *X-def*
by *auto*

lemma *sr-u-pos*: *lt-vec 0 u*

```
proof −
  let ?y = pow-A-1 u
  define n where n = N
  define c where c = (sr + 1)^N
  have c: c > 0 using sr-pos unfolding c-def by auto
  have lt-vec 0 ?y using Y-pos-main[of ?y] u by auto
  also have ?y = A1n *v u unfolding pow-A-1-def ..
  also have ... = c *s u unfolding c-def A1n-def n-def[symmetric]
  proof (induct n)
    case (Suc n)
    then show ?case
      by (simp add: matrix-vector-mul-assoc[symmetric] algebra-simps vec.scale
          sr-imp-eigen-vector-main[symmetric])
  qed auto
  finally have lt: lt-vec 0 (c *s u) .
 have 0 < u $ i for i using lt[rule-format, of i] c by simp (metis zero-less-mult-pos)
  thus lt-vec 0 u by simp
qed
end

lemma eigen-vector-z-sr: eigen-vector A z sr
  using sr-imp-eigen-vector[OF zX refl] by auto

lemma eigen-value-sr: eigen-value A sr
  using eigen-vector-z-sr unfolding eigen-value-def by auto

abbreviation c ≡ complex-of-real
abbreviation cA ≡ map-matrix c A
abbreviation norm-v ≡ map-vector (norm :: complex ⇒ real)

lemma norm-v-ge-0: le-vec 0 (norm-v v) by (auto simp: map-vector-def)
lemma norm-v-eq-0: norm-v v = 0 ⟷ v = 0 by (auto simp: map-vector-def
vec-eq-iff)

lemma cA-index: cA $ i $ j = c (A $ i $ j)
  unfolding map-matrix-def map-vector-def by simp

lemma norm-cA[simp]: norm (cA $ i $ j) = A $ i $ j
  using nonneg[of i j] by (simp add: cA-index)

context fixes α v
  assumes ev: eigen-vector cA v α
begin

lemma evD: α *s v = cA *v v v ≠ 0
  using ev[unfolded eigen-vector-def] by auto

lemma ev-alpha-norm-v: norm-v (α *s v) = (norm α *s norm-v v)
  by (auto simp: map-vector-def norm-mult vec-eq-iff)
```

lemma *ev-A-norm-v*: *norm-v* $(cA *v\ v)$ \$ $j \le (A *v$ *norm-v* $v)$ \$ j
proof −
  have *norm-v* $(cA *v\ v)$ \$ j = *norm* $(\sum i{\in}UNIV.\ cA$ \$ j \$ $i * v$ \$ $i)$
    unfolding *map-vector-def* **by** (*simp add*: *matrix-vector-mult-def*)
  also have ... $\le (\sum i{\in}UNIV.$ *norm* $(cA$ \$ j \$ $i * v$ \$ $i))$ **by** (*rule norm-sum*)
  also have ... $= (\sum i{\in}UNIV.\ A$ \$ j \$ $i *$ *norm-v* v \$ $i)$
    by (*rule sum.cong*[*OF refl*], *auto simp*: *norm-mult map-vector-def*)
  also have ... $= (A *v$ *norm-v* $v)$ \$ j **by** (*simp add*: *matrix-vector-mult-def*)
  finally show *?thesis* **.**
qed

lemma *ev-le-vec*: *le-vec* (*norm* α *∗s norm-v v*) (*A ∗v norm-v v*)
  using *arg-cong*[*OF evD*(*1*), *of norm-v*, *unfolded ev-alpha-norm-v*] *ev-A-norm-v*
by *auto*

lemma *norm-v-X*: *norm-v v* $\in X$
  using *norm-v-ge-0*[*of v*] *evD*(*2*) *norm-v-eq-0*[*of v*] **unfolding** *X-def* **by** *auto*

lemma *ev-inequalities*: *norm* $\alpha \le r$ (*norm-v v*) *r* (*norm-v v*) $\le$ *sr*
proof −
  have *v*: *norm-v v* $\in X$ **by** (*rule norm-v-X*)
  from *rho-le-x-Ax-imp-rho-le-rx*[*OF v ev-le-vec*]
  show *norm* $\alpha \le r$ (*norm-v v*) **.**
  from *sr-max*[*OF v*]
  show *r* (*norm-v v*) $\le$ *sr* **.**
qed

lemma *eigen-vector-norm-sr*: *norm* $\alpha \le$ *sr* **using** *ev-inequalities* **by** *auto*
end

lemma *eigen-value-norm-sr*: **assumes** *eigen-value cA* α
  shows *norm* $\alpha \le$ *sr*
  using *eigen-vector-norm-sr*[*of* - α] *assms* **unfolding** *eigen-value-def* **by** *auto*

lemma *le-vec-trans*: *le-vec x y* $\Longrightarrow$ *le-vec y u* $\Longrightarrow$ *le-vec x u*
  using *order.trans*[*of x* \$ *i y* \$ *i u* \$ *i* **for** *i*] **by** *auto*

lemma *eigen-vector-z-sr-c*: *eigen-vector cA* (*map-vector c z*) (*c sr*)
  unfolding *of-real-hom.eigen-vector-hom* **by** (*rule eigen-vector-z-sr*)

lemma *eigen-value-sr-c*: *eigen-value cA* (*c sr*)
  using *eigen-vector-z-sr-c* **unfolding** *eigen-value-def* **by** *auto*

definition *w* = *perron-frobenius.z* (*transpose A*)

lemma *w*: *transpose A ∗v w* = *sr ∗s w lt-vec 0 w perron-frobenius.sr* (*transpose A*) = *sr*

proof −
  interpret *t*: *perron-frobenius transpose A*
    by (*rule pf-transpose*)
  from *eigen-vector-z-sr-c t.eigen-vector-z-sr-c*
  have *ev*: *eigen-value cA* (*c sr*) *eigen-value t.cA* (*c t.sr*)
    unfolding *eigen-value-def* **by** *auto*
  {
    fix *x*
    have *eigen-value* (*t.cA*) *x* = *eigen-value* (*transpose cA*) *x*
      unfolding *map-matrix-def map-vector-def transpose-def*
      by (*auto simp*: *vec-eq-iff*)
    also have ... = *eigen-value cA x* **by** (*rule eigen-value-transpose*)
    finally have *eigen-value* (*t.cA*) *x* = *eigen-value cA x* **.**
  } **note** *ev-id* = *this*
  with *ev* **have** *ev*: *eigen-value t.cA* (*c sr*) *eigen-value cA* (*c t.sr*) **by** *auto*
  from *eigen-value-norm-sr*[*OF ev*(*2*)] *t.eigen-value-norm-sr*[*OF ev*(*1*)]
  show *id*: *t.sr* = *sr* **by** *auto*
  from *t.eigen-vector-z-sr*[*unfolded id*, *folded w-def*] **show** *transpose A* ∗*v w* = *sr* ∗*s w*
    unfolding *eigen-vector-def* **by** *auto*
  from *t.z-pos*[*folded w-def*] **show** *lt-vec 0 w* **by** *auto*
qed

lemma *c-cmod-id*: $a \in \mathbb{R} \Longrightarrow Re\ a \geq 0 \Longrightarrow c\ (cmod\ a) = a$ **by** (*auto simp*: *Reals-def*)

lemma *pos-rowvector-mult-0*: **assumes** *lt*: *lt-vec 0 x*
  and *0*: (*rowvector x* :: *real* ^ ′*n* ^ ′*n*) ∗*v y* = *0* (**is** *?x* ∗*v* - = *0*) **and** *le*: *le-vec 0 y*
shows *y* = *0*
proof −
  {
    fix *i*
    assume *y* \$ *i* ≠ *0*
    with *le* **have** *yi*: *y* \$ *i* > *0* **by** (*auto simp*: *order.strict-iff-order*)
    have *0* = (*?x* ∗*v y*) \$ *i* **unfolding** *0* **by** *simp*
    also have ... = (∑*j*∈*UNIV*. *x* \$ *j* ∗ *y* \$ *j*)
      unfolding *rowvector-def matrix-vector-mult-def* **by** *simp*
    also have ... > *0*
      by (*rule sum-pos2*[*of* - *i*], *insert yi lt le*, *auto intro*!: *mult-nonneg-nonneg*
        simp: *order.strict-iff-order*)
    finally have *False* **by** *simp*
  }
  thus *?thesis* **by** (*auto simp*: *vec-eq-iff*)
qed

lemma *pos-matrix-mult-0*: **assumes** *le*: ⋀ *i j*. *B* \$ *i* \$ *j* ≥ *0*
  and *lt*: *lt-vec 0 x*
  and *0*: *B* ∗*v x* = *0*

```
shows B = 0
proof –
  {
    fix i j
    assume B $ i $ j ≠ 0
    with le have gt: B $ i $ j > 0 by (auto simp: order.strict-iff-order)
    have 0 = (B *v x) $ i unfolding 0 by simp
    also have ... = (∑ j∈UNIV. B $ i $ j * x $ j)
      unfolding matrix-vector-mult-def by simp
    also have ... > 0
      by (rule sum-pos2[of - j], insert gt lt le, auto intro!: mult-nonneg-nonneg
        simp: order.strict-iff-order)
    finally have False by simp
  }
  thus B = 0 unfolding vec-eq-iff by auto
qed

lemma eigen-value-smaller-matrix: assumes B: ⋀ i j. 0 ≤ B $ i $ j ∧ B $ i $ j
≤ A $ i $ j
  and AB: A ≠ B
  and ev: eigen-value (map-matrix c B) sigma
shows cmod sigma < sr
proof –
  let ?B = map-matrix c B
  let ?sr = spectral-radius ?B
  define σ where σ = ?sr
  have real-non-neg-mat ?B unfolding real-non-neg-mat-def elements-mat-h-def
    by (auto simp: map-matrix-def map-vector-def B)
  from perron-frobenius[OF this, folded σ-def] obtain x where ev-sr: eigen-vector
?B x (c σ)
    and rnn: real-non-neg-vec x by auto
  define y where y = norm-v x
  from rnn have xy: x = map-vector c y
    unfolding real-non-neg-vec-def vec-elements-h-def y-def
    by (auto simp: map-vector-def vec-eq-iff c-cmod-id)
  from spectral-radius-max[OF ev, folded σ-def] have sigma-sigma: cmod sigma ≤
σ .
  from ev-sr[unfolded xy of-real-hom.eigen-vector-hom]
  have ev-B: eigen-vector B y σ .
  from ev-B[unfolded eigen-vector-def] have ev-B': B *v y = σ *s y by auto
  have ypos: y $ i ≥ 0 for i unfolding y-def by (auto simp: map-vector-def)
  from ev-B this have y: y ∈ X unfolding eigen-vector-def X-def by auto

  have BA: (B *v y) $ i ≤ (A *v y) $ i for i
    unfolding matrix-vector-mult-def vec-lambda-beta
    by (rule sum-mono, rule mult-right-mono, insert B ypos, auto)
  hence le-vec: le-vec (σ *s y) (A *v y) unfolding ev-B' by auto
  from rho-le-x-Ax-imp-rho-le-rx[OF y le-vec]
  have σ ≤ r y by auto
```

also have ... $\le$ *sr* **using** *y* **by** (*rule sr-max*)
finally have *sig-le-sr*: $\sigma \le$ *sr* **.**
{
assume σ = *sr*
hence *r-sr*: *r y* = *sr* **and** *sr-sig*: *sr* = σ **using** ‹$\sigma \le$ *r y*› ‹*r y* $\le$ *sr*› **by** *auto*
from *sr-u-pos*[*OF y r-sr*] **have** *pos*: *lt-vec 0 y* **.**
from *sr-imp-eigen-vector*[*OF y r-sr*] **have** *ev'*: *eigen-vector A y sr* **.**
have (*A* − *B*) *∗v y* = *A ∗v y* − *B ∗v y* **unfolding** *matrix-vector-mult-def*
by (*auto simp*: *vec-eq-iff field-simps sum-subtractf*)
also have *A ∗v y* = *sr ∗s y* **using** *ev'*[*unfolded eigen-vector-def*] **by** *auto*
also have *B ∗v y* = *sr ∗s y* **unfolding** *ev-B' sr-sig* **..**
finally have *id*: (*A* − *B*) *∗v y* = *0* **by** *simp*
from *pos-matrix-mult-0*[*OF - pos id*] *assms*(*1*−*2*) **have** *False* **by** *auto*
}
with *sig-le-sr sigma-sigma* **show** *?thesis* **by** *argo*
qed

lemma *charpoly-erase-mat-sr*: *0* < *poly* (*charpoly* (*erase-mat A i i*)) *sr*
proof −
let *?A* = *erase-mat A i i*
let *?pos* = *poly* (*charpoly ?A*) *sr*
{
from *A-nonzero-fixed-j*[*of i*] **obtain** *k* **where** *A* \$ *k* \$ *i* ≠ *0* **by** *auto*
assume *A* = *?A*
hence *A* \$ *k* \$ *i* = *?A* \$ *k* \$ *i* **by** *simp*
also have *?A* \$ *k* \$ *i* = *0* **by** (*auto simp*: *erase-mat-def*)
also have *A* \$ *k* \$ *i* ≠ *0* **by** *fact*
finally have *False* **by** *simp*
}
hence *AA*: *A* ≠ *?A* **by** *auto*
have *le*: *0* ≤ *?A* \$ *i* \$ *j* ∧ *?A* \$ *i* \$ *j* ≤ *A* \$ *i* \$ *j* **for** *i j*
by (*auto simp*: *erase-mat-def nonneg*)
note *ev-small* = *eigen-value-smaller-matrix*[*OF le AA*]
{
fix *rho* :: *real*
assume *eigen-value ?A rho*
hence *ev*: *eigen-value* (*map-matrix c ?A*) (*c rho*)
unfolding *eigen-value-def* **using** *of-real-hom.eigen-vector-hom*[*of ?A - rho*]
by *auto*
from *ev-small*[*OF this*] **have** *abs rho* < *sr* **by** *auto*
} **note** *ev-small-real* = *this*
have *pos0*: *?pos* ≠ *0*
using *ev-small-real*[*of sr*] **by** (*auto simp*: *eigen-value-root-charpoly*)
{
define *p* **where** *p* = *charpoly ?A*
assume *pos*: *?pos* < *0*
hence *neg*: *poly p sr* < *0* **unfolding** *p-def* **by** *auto*
from *degree-monic-charpoly*[*of ?A*] **have** *mon*: *monic p* **and** *deg*: *degree p* ≠ *0*
unfolding *p-def* **by** *auto*

    let *?f* = *poly p*
    have *cont*: *continuous-on* {*a..b*} *?f* **for** *a b* **by** (*auto intro*: *continuous-intros*)
    from *pos* **have** *le*: *?f sr* ≤ *0* **by** (*auto simp*: *p-def*)
    from *mon* **have** *lc*: *lead-coeff p* > *0* **by** *auto*
    from *poly-pinfty-ge*[*OF this deg*, *of 0*] **obtain** *z* **where** *lez*: ⋀ *x*. *z* ≤ *x* ⟹ *0* ≤ *?f x* **by** *auto*
    define *y* **where** *y* = *max z sr*
    have *yr*: *y* ≥ *sr* **and** *y* ≥ *z* **unfolding** *y-def* **by** *auto*
    from *lez*[*OF this*(*2*)] **have** *y0*: *?f y* ≥ *0* **.**
    from *IVT'*[*of ?f*, *OF le y0 yr cont*] **obtain** *x* **where** *ge*: *x* ≥ *sr* **and** *rt*: *?f x* = *0*
      unfolding *p-def* **by** *auto*
    hence *eigen-value ?A x* **unfolding** *p-def* **by** (*simp add*: *eigen-value-root-charpoly*)
    from *ev-small-real*[*OF this*] *ge* **have** *False* **by** *auto*
  }
  with *pos0* **show** *?thesis* **by** *argo*
qed

lemma *multiplicity-sr-1*: *order sr* (*charpoly A*) = *1*
proof −
  {
    assume *poly* (*pderiv* (*charpoly A*)) *sr* = *0*
    hence *0* = *poly* (*monom 1 1* ∗ *pderiv* (*charpoly A*)) *sr* **by** *simp*
    also have ... = *sum* (λ *i*. *poly* (*charpoly* (*erase-mat A i i*)) *sr*) *UNIV*
      unfolding *pderiv-char-poly-erase-mat poly-sum* **..**
    also have ... > *0*
      by (*rule sum-pos*, (*force simp*: *charpoly-erase-mat-sr*)+)
    finally have *False* **by** *simp*
  }
  hence *nZ*: *poly* (*pderiv* (*charpoly A*)) *sr* ≠ *0* **and** *nZ'*: *pderiv* (*charpoly A*) ≠ *0* **by** *auto*
  from *eigen-vector-z-sr* **have** *eigen-value A sr* **unfolding** *eigen-value-def* **..**
  from *this*[*unfolded eigen-value-root-charpoly*]
  have *poly* (*charpoly A*) *sr* = *0* **.**
  hence *order sr* (*charpoly A*) ≠ *0* **unfolding** *order-root* **using** *nZ'* **by** *auto*
  from *order-pderiv*[*OF nZ' this*] *order-0I*[*OF nZ*]
  show *?thesis* **by** *simp*
qed

lemma *sr-spectral-radius*: *sr* = *spectral-radius cA*
proof −
  from *eigen-vector-z-sr-c* **have** *eigen-value cA* (*c sr*)
    unfolding *eigen-value-def* **by** *auto*
  from *spectral-radius-max*[*OF this*]
  have *sr*: *sr* ≤ *spectral-radius cA* **by** *auto*
  with *spectral-radius-ev*[*of cA*] *eigen-vector-norm-sr*
  show *?thesis* **by** *force*
qed

```
lemma le-vec-A-mu: assumes y: y ∈ X and le: le-vec (A *v y) (mu *s y)
  shows sr ≤ mu lt-vec 0 y
  mu = sr ∨ A *v y = mu *s y ⟹ mu = sr ∧ A *v y = mu *s y
proof −
  let ?w = rowvector w
  let ?w' = columnvector w
  have ?w ** A = transpose (transpose (?w ** A))
    unfolding transpose-transpose by simp
  also have transpose (?w ** A) = transpose A ** transpose ?w
    by (rule matrix-transpose-mul)
  also have transpose ?w = columnvector w by (rule transpose-rowvector)
  also have transpose A ** . . . = columnvector (transpose A *v w)
    unfolding dot-rowvector-columnvector[symmetric] ..
  also have transpose A *v w = sr *s w unfolding w by simp
  also have transpose (columnvector . . . ) = rowvector (sr *s w)
    unfolding transpose-def columnvector-def rowvector-def vector-scalar-mult-def
by auto
  finally have 1: ?w ** A = rowvector (sr *s w) .
  have sr *s (?w *v y) = ?w ** A *v y unfolding 1
   by (auto simp: rowvector-def vector-scalar-mult-def matrix-vector-mult-def vec-eq-iff
      sum-distrib-left mult.assoc)
  also have . . . = ?w *v (A *v y) by (simp add: matrix-vector-mul-assoc)
  finally have eq1: sr *s (rowvector w *v y) = rowvector w *v (A *v y) .
  have le-vec (rowvector w *v (A *v y)) (?w *v (mu *s y))
     by (rule le-vec-mono-left[OF - le], insert w(2), auto simp: rowvector-def or-
der.strict-iff-order)
   also have ?w *v (mu *s y) = mu *s (?w *v y) by (simp add: algebra-simps
vec.scale)
  finally have le1: le-vec (rowvector w *v (A *v y)) (mu *s (?w *v y)) .
  from le1[unfolded eq1[symmetric]]
  have 2: le-vec (sr *s (?w *v y)) (mu *s (?w *v y)) .
  {
    from y obtain i where yi: y $ i > 0 and y: ⋀ j. y $ j ≥ 0 unfolding X-def
      by (auto simp: order.strict-iff-order vec-eq-iff)
    from w(2) have wi: w $ i > 0 and w: ⋀ j. w $ j ≥ 0
      by (auto simp: order.strict-iff-order)
    have (?w *v y) $ i > 0 using yi y wi w
      by (auto simp: matrix-vector-mult-def rowvector-def
        intro!: sum-pos2[of - i] mult-nonneg-nonneg)
    moreover from 2[rule-format, of i] have sr * (?w *v y) $ i ≤ mu * (?w *v
y) $ i by simp
    ultimately have sr ≤ mu by simp
  }
  thus *: sr ≤ mu .
  define cc where cc = (mu + 1)^N
  define n where n = N
  from * sr-pos have mu: mu ≥ 0 mu > 0 by auto
  hence cc: cc > 0 unfolding cc-def by simp
  from y have pow-A-1 y ∈ pow-A-1 ‘ X by auto
```

from *Y-pos-main*[*OF this*] **have** *lt*: *0* < (*A1n* ∗*v* *y*) $ *i* **for** *i* **by** (*simp add*: *pow-A-1-def*)
have *le*: *le-vec* (*A1n* ∗*v* *y*) (*cc* ∗*s* *y*) **unfolding** *cc-def A1n-def n-def*[*symmetric*]
proof (*induct n*)
case (*Suc n*)
let *?An* = *matpow* (*A* + *mat 1*) *n*
let *?mu* = (*mu* + *1*)
have *id'*: *matpow* (*A* + *mat 1*) (*Suc n*) ∗*v* *y* = *A* ∗*v* (*?An* ∗*v* *y*) + *?An* ∗*v* *y* (**is** *?a* = *?b* + *?c*)
by (*simp add*: *matrix-add-ldistrib matrix-mul-rid matrix-add-vect-distrib matpow-1-commute*
matrix-vector-mul-assoc[*symmetric*])
have *le-vec ?b* (*?mu^n* ∗*s* (*A* ∗*v* *y*))
using *le-vec-mono-left*[*OF nonneg Suc*] **by** (*simp add*: *algebra-simps vec.scale*)
moreover have *le-vec* (*?mu^n* ∗*s* (*A* ∗*v* *y*)) (*?mu^n* ∗*s* (*mu* ∗*s* *y*))
using *le mu* **by** *auto*
moreover have *id*: *?mu^n* ∗*s* (*mu* ∗*s* *y*) = (*?mu^n* ∗ *mu*) ∗*s* *y* **by** *simp*
from *le-vec-trans*[*OF calculation*[*unfolded id*]]
have *le1*: *le-vec ?b* ((*?mu^n* ∗ *mu*) ∗*s* *y*) **.**
from *Suc* **have** *le2*: *le-vec ?c* ((*mu* + *1*) ^ *n* ∗*s* *y*) **.**
have *le*: *le-vec ?a* ((*?mu^n* ∗ *mu*) ∗*s* *y* + *?mu^n* ∗*s* *y*)
unfolding *id'* **using** *add-mono*[*OF le1*[*rule-format*] *le2*[*rule-format*]] **by** *auto*
have *id''*: (*?mu^n* ∗ *mu*) ∗*s* *y* + *?mu^n* ∗*s* *y* = *?mu^Suc n* ∗*s* *y* **by** (*simp add*: *algebra-simps*)
show *?case* **using** *le* **unfolding** *id''* **.**
qed (*simp add*: *matrix-vector-mul-lid*)
have *lt*: *0* < *cc* ∗ *y* $ *i* **for** *i* **using** *lt*[*of i*] *le*[*rule-format*, *of i*] **by** *auto*
have *y* $ *i* > *0* **for** *i* **using** *lt*[*of i*] *cc* **by** (*rule zero-less-mult-pos*)
thus *lt-vec 0 y* **by** *auto*
assume ∗∗: *mu* = *sr* ∨ *A* ∗*v* *y* = *mu* ∗*s* *y*
{
assume *A* ∗*v* *y* = *mu* ∗*s* *y*
with *y* **have** *eigen-vector A y mu* **unfolding** *X-def eigen-vector-def* **by** *auto*
hence *eigen-vector cA* (*map-vector c y*) (*c mu*) **unfolding** *of-real-hom.eigen-vector-hom* **.**
from *eigen-vector-norm-sr*[*OF this*] ∗ **have** *mu* = *sr* **by** *auto*
}
with ∗∗ **have** *mu-sr*: *mu* = *sr* **by** *auto*
from *eq1*[*folded vector-smult-distrib*]
have *0*: *?w* ∗*v* (*sr* ∗*s* *y* − *A* ∗*v* *y*) = *0*
unfolding *matrix-vector-right-distrib-diff* **by** *simp*
have *le0*: *le-vec 0* (*sr* ∗*s* *y* − *A* ∗*v* *y*) **using** *assms*(*2*)[*unfolded mu-sr*] **by** *auto*
have *sr* ∗*s* *y* − *A* ∗*v* *y* = *0* **using** *pos-rowvector-mult-0*[*OF w*(*2*) *0 le0*] **.**
hence *ev-y*: *A* ∗*v* *y* = *sr* ∗*s* *y* **by** *auto*
show *mu* = *sr* ∧ *A* ∗*v* *y* = *mu* ∗*s* *y* **using** *ev-y mu-sr* **by** *auto*
qed

lemma *nonnegative-eigenvector-has-ev-sr*: **assumes** *eigen-vector A v mu* **and** *le*: *le-vec 0 v*

shows *mu* = *sr*
proof −
from *assms*(*1*)[*unfolded eigen-vector-def*] **have** *v*: $v \neq 0$ **and** *ev*: *A* ∗*v v* = *mu* ∗*s v* **by** *auto*
from *le v* **have** *v*: $v \in X$ **unfolding** *X-def* **by** *auto*
from *ev* **have** *le-vec* (*A* ∗*v v*) (*mu* ∗*s v*) **by** *auto*
from *le-vec-A-mu*[*OF v this*] *ev* **show** *?thesis* **by** *auto*
qed

lemma *similar-matrix-rotation*: **assumes** *ev*: *eigen-value cA* α **and** α: *cmod* α = *sr*
shows *similar-matrix* (*cis* (*Arg* α) ∗*k cA*) *cA*
proof −
from *ev* **obtain** *y* **where** *ev*: *eigen-vector cA y* α **unfolding** *eigen-value-def* **by** *auto*
let *?y* = *norm-v y*
note *maps* = *map-vector-def map-matrix-def*
define *yp* **where** *yp* = *norm-v y*
let *?yp* = *map-vector c yp*
have *yp*: $yp \in X$ **unfolding** *yp-def* **by** (*rule norm-v-X*[*OF ev*])
from *ev*[*unfolded eigen-vector-def*] **have** *ev-y*: *cA* ∗*v y* = α ∗*s y* **by** *auto*
from *ev-le-vec*[*OF ev*, *unfolded* α, *folded yp-def*]
have *1*: *le-vec* (*sr* ∗*s yp*) (*A* ∗*v yp*) **by** *simp*
from *rho-le-x-Ax-imp-rho-le-rx*[*OF yp 1*] **have** $sr \leq r\ yp$ **by** *auto*
with *ev-inequalities*[*OF ev*, *folded yp-def*]
have *2*: *r yp* = *sr* **by** *auto*
have *ev-yp*: *A* ∗*v yp* = *sr* ∗*s yp*
and *pos-yp*: *lt-vec 0 yp*
using *sr-imp-eigen-vector-main*[*OF yp 2*] *sr-u-pos*[*OF yp 2*] **by** *auto*
define *D* **where** *D* = *diagvector* (λ *j*. *cis* (*Arg* (*y* $ *j*)))
define *inv-D* **where** *inv-D* = *diagvector* (λ *j*. *cis* (− *Arg* (*y* $ *j*)))
have *DD*: *inv-D* ∗∗ *D* = *mat 1 D* ∗∗ *inv-D* = *mat 1* **unfolding** *D-def inv-D-def*
by (*auto simp add*: *diagvector-eq-mat cis-mult*)
{
fix *i*
have (*D* ∗*v ?yp*) $ *i* = *cis* (*Arg* (*y* $ *i*)) ∗ *c* (*cmod* (*y* $ *i*))
unfolding *D-def yp-def* **by** (*simp add*: *maps*)
also have . . . = *y* $ *i* **by** (*simp add*: *cis-mult-cmod-id*)
also note *calculation*
}
hence *y-D-yp*: *y* = *D* ∗*v ?yp* **by** (*auto simp*: *vec-eq-iff*)
define φ **where** φ = *Arg* α
let *?*φ = *cis* (− φ)
have [*simp*]: *cis* (− φ) ∗ *rcis sr* φ = *sr* **unfolding** *cis-rcis-eq rcis-mult* **by** *simp*
have α: α = *rcis sr* φ **unfolding** φ*-def* α[*symmetric*] *rcis-cmod-Arg* **..**
define *F* **where** *F* = *?*φ ∗*k* (*inv-D* ∗∗ *cA* ∗∗ *D*)
have *cA* ∗*v* (*D* ∗*v ?yp*) = α ∗*s y* **unfolding** *y-D-yp*[*symmetric*] *ev-y* **by** *simp*
also have *inv-D* ∗*v* . . . = α ∗*s ?yp*
unfolding *vector-smult-distrib y-D-yp matrix-vector-mul-assoc DD matrix-vector-mul-lid*

..
also have *?φ ∗s ...* = *sr ∗s ?yp* **unfolding** *α* **by** *simp*
also have ... = *map-vector c* (*sr ∗s yp*) **unfolding** *vec-eq-iff* **by** (*auto simp*: *maps*)
also have ... = *cA ∗v ?yp* **unfolding** *ev-yp*[*symmetric*] **by** (*auto simp*: *maps matrix-vector-mult-def*)
finally have *F*: *F ∗v ?yp* = *cA ∗v ?yp* **unfolding** *F-def matrix-scalar-vector-ac*[*symmetric*]
unfolding *matrix-vector-mul-assoc*[*symmetric*] *vector-smult-distrib* **.**
have *prod*: *inv-D ∗∗ cA ∗∗ D* = (*χ i j. cis* (− *Arg* (*y* $ *i*)) ∗ *cA* $ *i* $ *j* ∗ *cis* (*Arg* (*y* $ *j*)))
unfolding *inv-D-def D-def diagvector-mult-right diagvector-mult-left* **by** *simp*
{
fix *i j*
have *cmod* (*F* $ *i* $ *j*) = *cmod* (*?φ* ∗ *cA* $h *i* $h *j* ∗ (*cis* (− *Arg* (*y* $h *i*)) ∗ *cis* (*Arg* (*y* $h *j*))))
unfolding *F-def prod vec-lambda-beta matrix-scalar-mult-def*
by (*simp only*: *ac-simps*)
also have ... = *A* $ *i* $ *j* **unfolding** *cis-mult* **unfolding** *norm-mult* **by** *simp*
also note *calculation*
}
hence *FA*: *map-matrix norm F* = *A* **unfolding** *maps* **by** *auto*
let *?F* = *map-matrix c* (*map-matrix norm F*)
let *?G* = *?F* − *F*
let *?Re* = *map-matrix Re*
from *F*[*folded FA*] **have** *0*: *?G ∗v ?yp* = *0* **unfolding** *matrix-diff-vect-distrib* **by** *simp*
have *?Re ?G ∗v yp* = *map-vector Re* (*?G ∗v ?yp*)
unfolding *maps matrix-vector-mult-def vec-lambda-beta Re-sum* **by** *auto*
also have ... = *0* **unfolding** *0* **by** (*simp add*: *vec-eq-iff maps*)
finally have *0*: *?Re ?G ∗v yp* = *0* **.**
have *?Re ?G* = *0*
by (*rule pos-matrix-mult-0*[*OF - pos-yp 0*], *auto simp*: *maps complex-Re-le-cmod*)
hence *?F* = *F* **by** (*auto simp*: *maps vec-eq-iff cmod-eq-Re*)
with *FA* **have** *AF*: *cA* = *F* **by** *simp*
from *arg-cong*[*OF this*, *of λ A. cis φ ∗k A*]
have *sim*: *cis φ ∗k cA* = *inv-D ∗∗ cA ∗∗ D* **unfolding** *F-def matrix.scale-scale cis-mult*
by *simp*
have *similar-matrix* (*cis φ ∗k cA*) *cA* **unfolding** *similar-matrix-def similar-matrix-wit-def sim*
by (*rule exI*[*of - inv-D*], *rule exI*[*of - D*], *auto simp*: *DD*)
thus *?thesis* **unfolding** *φ-def* **.**
qed

lemma assumes *ev*: *eigen-value cA α* **and** *α*: *cmod α* = *sr*
shows *maximal-eigen-value-order-1*: *order α* (*charpoly cA*) = *1*
and *maximal-eigen-value-rotation*: *eigen-value cA* (*x* ∗ *cis* (*Arg α*)) = *eigen-value cA x*
eigen-value cA (*x* / *cis* (*Arg α*)) = *eigen-value cA x*

proof −
  let *?a* = *cis* (*Arg* α)
  let *?p* = *charpoly cA*
  from *similar-matrix-rotation*[*OF ev* α]
  have *similar-matrix* (*?a* ∗*k cA*) *cA* **.**
  from *similar-matrix-charpoly*[*OF this*]
  have *id*: *charpoly* (*?a* ∗*k cA*) = *?p* **.**
  have *a*: *?a* ≠ *0* **by** *simp*
  from *order-charpoly-smult*[*OF this*, *of - cA*, *unfolded id*]
  have *order-neg*: *order x ?p* = *order* (*x* / *?a*) *?p* **for** *x* **.**
  have *order-pos*: *order x ?p* = *order* (*x* ∗ *?a*) *?p* **for** *x*
    using *order-neg*[*symmetric*, *of x* ∗ *?a*] **by** *simp*
  note *order-neg*[*of* α]
  also have *id*: α / *?a* = *sr* **unfolding** α[*symmetric*]
    by (*metis a cis-mult-cmod-id nonzero-mult-div-cancel-left*)
  also have *sr*: *order* ... *?p* = *1* **unfolding** *multiplicity-sr-1*[*symmetric*] *charpoly-of-real*
    by (*rule map-poly-inj-idom-divide-hom.order-hom*, *unfold-locales*)
  finally show ∗: *order* α *?p* = *1* **.**
  show *eigen-value cA* (*x* ∗ *?a*) = *eigen-value cA x* **using** *order-pos*
    unfolding *eigen-value-root-charpoly order-root* **by** *auto*
  show *eigen-value cA* (*x* / *?a*) = *eigen-value cA x* **using** *order-neg*
    unfolding *eigen-value-root-charpoly order-root* **by** *auto*
qed

lemma *maximal-eigen-values-group*: **assumes** *M*: *M* = {*ev* :: *complex*. *eigen-value cA ev* ∧ *cmod ev* = *sr*}
  and *a*: *rcis sr* α ∈ *M*
  and *b*: *rcis sr* β ∈ *M*
shows *rcis sr* (α + β) ∈ *M rcis sr* (α − β) ∈ *M rcis sr 0* ∈ *M*
proof −
  {
    fix *a*
    assume ∗: *rcis sr a* ∈ *M*
    have *id*: *cis* (*Arg* (*rcis sr a*)) = *cis a*
      by (*smt* ∗ *M mem-Collect-eq nonzero-mult-div-cancel-left of-real-eq-0-iff*
        rcis-cmod-Arg rcis-def sr-pos)
    from ∗[*unfolded assms*] **have** *eigen-value cA* (*rcis sr a*) *cmod* (*rcis sr a*) = *sr* **by** *auto*
    from *maximal-eigen-value-rotation*[*OF this*, *unfolded id*]
    have *eigen-value cA* (*x* ∗ *cis a*) = *eigen-value cA x*
      eigen-value cA (*x* / *cis a*) = *eigen-value cA x* **for** *x* **by** *auto*
  } **note** ∗ = *this*
  from ∗(*1*)[*OF b*, *of rcis sr* α] *a* **show** *rcis sr* (α + β) ∈ *M* **unfolding** *M* **by** *auto*
  from ∗(*2*)[*OF a*, *of rcis sr* α] *a* **show** *rcis sr 0* ∈ *M* **unfolding** *M* **by** *auto*
  from ∗(*2*)[*OF b*, *of rcis sr* α] *a* **show** *rcis sr* (α − β) ∈ *M* **unfolding** *M* **by** *auto*
qed

```
lemma maximal-eigen-value-roots-of-unity-rotation:
  assumes M: M = {ev :: complex. eigen-value cA ev ∧ cmod ev = sr}
   and kM: k = card M
 shows k ≠ 0
    k ≤ CARD('n)
    ∃ f. charpoly A = (monom 1 k − [:sr^k:]) ∗ f
       ∧ (∀ x. poly (map-poly c f) x = 0 ⟶ cmod x < sr)
    M = (∗) (c sr) ‘ (λ i. (cis (of-nat i ∗ 2 ∗ pi / k))) ‘ {0 ..< k}
    M = (∗) (c sr) ‘ { x :: complex. x ^ k = 1}
    (∗) (cis (2 ∗ pi / k)) ‘ Spectrum cA = Spectrum cA
  unfolding kM
proof −
  let ?M = card M
  note fin = finite-spectrum[of cA]
  note char = degree-monic-charpoly[of cA]
  have ?M ≤ card (Collect (eigen-value cA))
    by (rule card-mono[OF fin], unfold M, auto)
  also have Collect (eigen-value cA) = {x. poly (charpoly cA) x = 0}
    unfolding eigen-value-root-charpoly by auto
  also have card ... ≤ degree (charpoly cA)
    by (rule poly-roots-degree, insert char, auto)
  also have ... = CARD('n) using char by simp
  finally show ?M ≤ CARD ('n) .
  from finite-subset[OF - fin, of M]
  have finM: finite M unfolding M by blast
  from finite-distinct-list[OF this]
  obtain m where Mm: M = set m and dist: distinct m by auto
  from Mm dist have card: ?M = length m by (auto simp: distinct-card)
  have sr: sr ∈ set m using eigen-value-sr-c sr-pos unfolding Mm[symmetric] M
by auto
  define s where s = sort-key Arg m
  define a where a = map Arg s
  let ?k = length a
  from dist Mm card sr have s: M = set s distinct s  sr ∈ set s
    and card: ?M = ?k
    and sorted: sorted a
    unfolding s-def a-def by auto
  have map-s: map ((∗) (c sr)) (map cis a) = s unfolding map-map o-def a-def
  proof (rule map-idI)
    fix x
    assume x ∈ set s
    from this[folded s(1), unfolded M]
    have id: cmod x = sr by auto
    show sr ∗ cis (Arg x) = x
      by (subst (5) rcis-cmod-Arg[symmetric], unfold id[symmetric] rcis-def, simp)
  qed
  from s(2)[folded map-s, unfolded distinct-map] have a: distinct a inj-on cis (set
a) by auto
```

from $s(3)$ **obtain** $aa\ a'$ **where** $a\text{-}split$: $a = aa\ \#\ a'$ **unfolding** $a\text{-}def$ **by** $(cases\ s,\ auto)$
from $Arg\text{-}bounded$ **have** $bounded$: $x \in set\ a \Longrightarrow -\ pi < x \wedge x \leq pi$ **for** x **unfolding** $a\text{-}def$ **by** $auto$
from $bounded[of\ aa,\ unfolded\ a\text{-}split]$ **have** aa: $-\ pi < aa \wedge aa \leq pi$ **by** $auto$
let $?aa = aa + 2 * pi$
define $args$ **where** $args = a\ @\ [?aa]$
let $?diff = \lambda\ i.\ args\ !\ Suc\ i - args\ !\ i$
have bnd: $x \in set\ a \Longrightarrow x < ?aa$ **for** x **using** $aa\ bounded[of\ x]$ **by** $auto$
hence $aa\text{-}a$: $?aa \notin set\ a$ **by** $fast$
have $sorted$: $sorted\ args$ **unfolding** $args\text{-}def$ **using** $sorted$ **unfolding** $sorted\text{-}append$
by $(insert\ bnd,\ auto\ simp:\ order.strict\text{-}iff\text{-}order)$
have $dist$: $distinct\ args$ **using** $a\ aa\text{-}a$ **unfolding** $args\text{-}def\ distinct\text{-}append$ **by** $auto$
have sum: $(\sum\ i < ?k.\ ?diff\ i) = 2 * pi$
unfolding $sum\text{-}lessThan\text{-}telescope\ args\text{-}def\ a\text{-}split$ **by** $simp$
have k: $?k \neq 0$ **unfolding** $a\text{-}split$ **by** $auto$
let $?A = ?diff\ `\ \{..<\ ?k\}$
let $?Min = Min\ ?A$
define Min **where** $Min = ?Min$
have $?Min = (?k * ?Min)\ /\ ?k$ **using** k **by** $auto$
also have $?k * ?Min = (\sum\ i < ?k.\ ?Min)$ **by** $auto$
also have $\ldots\ /\ ?k \leq (\sum\ i < ?k.\ ?diff\ i)\ /\ ?k$
by $(rule\ divide\text{-}right\text{-}mono[OF\ sum\text{-}mono[OF\ Min\text{-}le]],\ auto)$
also have $\ldots = 2 * pi\ /\ ?k$ **unfolding** sum **..**
finally have Min: $Min \leq 2 * pi\ /\ ?k$ **unfolding** $Min\text{-}def$ **by** $auto$
have lt: $i < ?k \Longrightarrow args\ !\ i < args\ !\ (Suc\ i)$ **for** i
using $sorted[unfolded\ sorted\text{-}iff\text{-}nth\text{-}mono,\ rule\text{-}format,\ of\ i\ Suc\ i]$
$dist[unfolded\ distinct\text{-}conv\text{-}nth,\ rule\text{-}format,\ of\ Suc\ i\ i]$ **by** $(auto\ simp:\ args\text{-}def)$
let $?c = \lambda\ i.\ rcis\ sr\ (args\ !\ i)$
have $hda[simp]$: $hd\ a = aa$ **unfolding** $a\text{-}split$ **by** $simp$
have $Min0$: $Min > 0$ **using** lt **unfolding** $Min\text{-}def$ **by** $(subst\ Min\text{-}gr\text{-}iff,\ insert\ k,\ auto)$
have $Min\text{-}A$: $Min \in ?A$ **unfolding** $Min\text{-}def$ **by** $(rule\ Min\text{-}in,\ insert\ k,\ auto)$
{
fix $i :: nat$
assume i: $i < length\ args$
hence $?c\ i = rcis\ sr\ ((a\ @\ [hd\ a])\ !\ i)$
by $(cases\ i = ?k,\ auto\ simp:\ args\text{-}def\ nth\text{-}append\ rcis\text{-}def)$
also have $\ldots \in set\ (map\ (rcis\ sr)\ (a\ @\ [hd\ a]))$ **using** i
unfolding $args\text{-}def\ set\text{-}map$ **unfolding** $set\text{-}conv\text{-}nth$ **by** $auto$
also have $\ldots = rcis\ sr\ `\ set\ a$ **unfolding** $a\text{-}split$ **by** $auto$
also have $\ldots = M$ **unfolding** $s(1)\ map\text{-}s[symmetric]\ set\text{-}map\ image\text{-}image$
by $(rule\ image\text{-}cong[OF\ refl],\ auto\ simp:\ rcis\text{-}def)$
finally have $?c\ i \in M$ **by** $auto$
} **note** $ciM = this$
{
fix $i :: nat$
assume i: $i < ?k$
hence $i < length\ args\ Suc\ i < length\ args$ **unfolding** $args\text{-}def$ **by** $auto$

```
  from maximal-eigen-values-group[OF M ciM[OF this(2)] ciM[OF this(1)]]
  have rcis sr (?diff i) ∈ M by simp
}
hence Min-M: rcis sr Min ∈ M using Min-A by force
have rcisM: rcis sr (of-nat n * Min) ∈ M for n
proof (induct n)
  case 0
  show ?case using sr Mm by auto
next
  case (Suc n)
  have *: rcis sr (of-nat (Suc n) * Min) = rcis sr (of-nat n * Min) * cis Min
    by (simp add: rcis-mult ring-distribs add.commute)
  from maximal-eigen-values-group(1)[OF M Suc Min-M]
  show ?case unfolding * by simp
qed
let ?list = map (rcis sr) (map (λ i. of-nat i * Min) [0 ..< ?k])
define list where list = ?list
have len: length ?list = ?M unfolding card by simp
from sr-pos have sr0: sr ≠ 0 by auto
{
  fix i
  assume i: i < ?k
  hence *: 0 ≤ real i * Min using Min0 by auto
  from i have real i < real ?k by auto
  from mult-strict-right-mono[OF this Min0]
  have real i * Min < real ?k * Min by simp
  also have ... ≤ real ?k * (2 * pi / real ?k)
    by (rule mult-left-mono[OF Min], auto)
  also have ... = 2 * pi using k by simp
  finally have real i * Min < 2 * pi .
  note * this
} note prod-pi = this
have dist: distinct ?list
  unfolding distinct-map[of rcis sr]
proof (rule conjI[OF - inj-on-subset[OF rcis-inj-on[OF sr0]]])
  show distinct (map (λ i. of-nat i * Min) [0 ..< ?k]) using Min0
    by (auto simp: distinct-map inj-on-def)
  show set (map (λi. real i * Min) [0..<?k]) ⊆ {0..<2 * pi} using prod-pi
    by auto
qed
with len have card': card (set ?list) = ?M using distinct-card by fastforce
have listM: set ?list ⊆ M using rcisM by auto
from card-subset-eq[OF finM listM card']
have M-list: M = set ?list ..
let ?piM = 2 * pi / ?M
{
  assume Min ≠ ?piM
  with Min have lt: Min < 2 * pi / ?k unfolding card by simp
  from k have 0 < real ?k by auto
```

from *mult-strict-left-mono*[*OF lt this*] *k Min0*
have *k*: $0 \le ?k * Min$ $?k * Min < 2 * pi$ **by** *auto*
from *rcisM*[*of ?k*, *unfolded M-list*] **have** *rcis sr* $(?k * Min) \in$ *set ?list* **by** *auto*
then obtain *i* **where** *i*: $i < ?k$ **and** *id*: *rcis sr* $(?k * Min) =$ *rcis sr* $(i * Min)$
by *auto*
from *inj-onD*[*OF inj-on-subset*[*OF rcis-inj-on*[*OF sr0*], *of* $\{?k * Min, i * Min\}$]
id]
prod-pi[*OF i*] *k*
have $?k * Min = i * Min$ **by** *auto*
with *Min0 i* **have** *False* **by** *auto*
}
hence *Min*: *Min* = *?piM* **by** *auto*
show *cM*: $?M \neq 0$ **unfolding** *card* **using** *k* **by** *auto*
let *?f* = (λ *i*. *cis* (*of-nat i* $* 2 * pi$ / *?M*))
note *M-list*
also have *set ?list* = $(*)$ (*c sr*) ‘ (λ *i*. *cis* (*of-nat i* $*$ *Min*)) ‘ $\{0 ..< ?k\}$
unfolding *set-map image-image*
by (*rule image-cong*, *insert sr-pos*, *auto simp*: *rcis-mult rcis-def*)
finally show *M-cis*: *M* = $(*)$ (*c sr*) ‘ *?f* ‘ $\{0 ..< ?M\}$
unfolding *card Min* **by** (*simp add*: *mult.assoc*)
thus *M-pow*: *M* = $(*)$ (*c sr*) ‘ $\{$ *x* :: *complex*. $x \,\hat{}\, ?M = 1\}$ **using** *roots-of-unity*[*OF*
cM] **by** *simp*
let *?rphi* = *rcis sr* ($2 * pi$ / *?M*)
let *?phi* = *cis* ($2 * pi$ / *?M*)
from *Min-M*[*unfolded Min*]
have *ev*: *eigen-value cA ?rphi* **unfolding** *M* **by** *auto*
have *cm*: *cmod ?rphi* = *sr* **using** *sr-pos* **by** *simp*
have *id*: *cis* (*Arg ?rphi*) = *cis* (*Arg ?phi*) $*$ *cmod ?phi*
unfolding *arg-rcis-cis*[*OF sr-pos*] **by** *simp*
also have ... = *?phi* **unfolding** *cis-mult-cmod-id* **..**
finally have *id*: *cis* (*Arg ?rphi*) = *?phi* **.**
define *phi* **where** *phi* = *?phi*
have *phi*: $phi \neq 0$ **unfolding** *phi-def* **by** *auto*
note *max* = *maximal-eigen-value-rotation*[*OF ev cm*, *unfolded id phi-def*[*symmetric*]]
have (($*$) *phi*) ‘ *Spectrum cA* = *Spectrum cA* (**is** *?L* = *?R*)
proof −
{
fix *x*
have $*$: $x \in ?L \Longrightarrow x \in ?R$ **for** *x* **using** *max*(*2*)[*of x*] *phi* **unfolding**
Spectrum-def **by** *auto*
moreover
{
assume $x \in ?R$
hence *eigen-value cA x* **unfolding** *Spectrum-def* **by** *auto*
from *this*[*folded max*(*2*)[*of x*]] **have** *x* / *phi* $\in$ *?R* **unfolding** *Spectrum-def*
by *auto*
from *imageI*[*OF this*, *of* ($*$) *phi*]
have $x \in ?L$ **using** *phi* **by** *auto*
}

```
    note this *
  }
  thus ?thesis by blast
qed
from this[unfolded phi-def]
show (*) (cis (2 * pi / real (card M))) ‘ Spectrum cA = Spectrum cA .
let ?p = monom 1 k − [:sr^k:]
let ?cp = monom 1 k − [:(c sr)^k:]
let ?one = 1 :: complex
let ?list = map (rcis sr) (map (λ i. of-nat i * ?piM) [0 ..< card M])
interpret c: field-hom c ..
interpret p: map-poly-inj-idom-divide-hom c ..
have cp: ?cp = map-poly c ?p by (simp add: hom-distribs)
have M-list: M = set ?list using M-list[unfolded Min card[symmetric]] .
have dist: distinct ?list using dist[unfolded Min card[symmetric]] .
have k0: k ≠ 0 using k[folded card] assms by auto
have ?cp = (monom 1 k + (− [:(c sr)^k:])) by simp
also have degree . . . = k
  by (subst degree-add-eq-left, insert k0, auto simp: degree-monom-eq)
finally have deg: degree ?cp = k .
from deg k0 have cp0: ?cp ≠ 0 by auto
have {x. poly ?cp x = 0} = {x. x^k = (c sr)^k} unfolding poly-diff poly-monom

  by simp
also have . . . ⊆ M
proof −
  {
    fix x
    assume id: x^k = (c sr)^k
    from sr-pos k0 have (c sr)^k ≠ 0 by auto
    with arg-cong[OF id, of λ x. x / (c sr)^k]
    have (x / c sr)^k = 1
      unfolding power-divide by auto
    hence c sr * (x / c sr) ∈ M
      by (subst M-pow, unfold kM[symmetric], blast)
    also have c sr * (x / c sr) = x using sr-pos by auto
    finally have x ∈ M .
  }
  thus ?thesis by auto
qed
finally have cp-M: {x. poly ?cp x = 0} ⊆ M .
have k = card (set ?list) unfolding distinct-card[OF dist] by (simp add: kM)
also have . . . ≤ card {x. poly ?cp x = 0}
proof (rule card-mono[OF poly-roots-finite[OF cp0]])
  {
    fix x
    assume x ∈ set ?list
    then obtain i where x: x = rcis sr (real i * ?piM) by auto
    have x^k = (c sr)^k unfolding x DeMoivre2 kM
```

by *simp* (*metis mult.assoc of-real-power rcis-times-2pi*)
hence *poly ?cp x = 0* **unfolding** *poly-diff poly-monom* **by** *simp*
}
thus *set ?list* $\subseteq$ {*x. poly ?cp x = 0*} **by** *auto*
qed
finally have *k-card*: $k \leq$ *card* {*x. poly ?cp x = 0*} **.**
from *k-card cp-M finM* **have** *M-id*: *M* = {*x. poly ?cp x = 0*}
unfolding *kM* **by** (*metis card-seteq*)
have *dvdc*: *?cp dvd charpoly cA*
proof (*rule poly-roots-dvd*[*OF cp0 deg k-card*])
from *cp-M*
show {*x. poly ?cp x = 0*} $\subseteq$ {*x. poly* (*charpoly cA*) *x = 0*}
unfolding *M eigen-value-root-charpoly* **by** *auto*
qed
from *this*[*unfolded charpoly-of-real cp p.hom-dvd-iff*]
have *dvd*: *?p dvd charpoly A* **.**
from *this*[*unfolded dvd-def*] **obtain** *f* **where**
decomp: *charpoly A = ?p* ∗ *f* **by** *blast*
let *?f = map-poly c f*
have *decompc*: *charpoly cA = ?cp* ∗ *?f* **unfolding** *charpoly-of-real decomp p.hom-mult cp* **..**
show ∃ *f. charpoly A* = (*monom 1 ?M* − [:*sr*^*?M*:]) ∗ *f* ∧ (∀ *x. poly* (*map-poly c f*) *x = 0* ⟶ *cmod x* < *sr*)
unfolding *kM*[*symmetric*]
proof (*intro exI conjI allI impI*, *rule decomp*)
fix *x*
assume *f*: *poly ?f x = 0*
hence *ev*: *eigen-value cA x*
unfolding *decompc p.hom-mult eigen-value-root-charpoly* **by** *auto*
hence *le*: *cmod x* $\leq$ *sr* **using** *eigen-value-norm-sr* **by** *auto*
{
assume *max*: *cmod x = sr*
hence *x* ∈ *M* **unfolding** *M* **using** *ev* **by** *auto*
hence *poly ?cp x = 0* **unfolding** *M-id* **by** *auto*
hence *dvd1*: [: −*x*, *1* :] *dvd ?cp* **unfolding** *poly-eq-0-iff-dvd* **by** *auto*
from *f*[*unfolded poly-eq-0-iff-dvd*]
have *dvd2*: [: −*x*, *1* :] *dvd ?f* **by** *auto*
from *char* **have** *0*: *charpoly cA* ≠ *0* **by** *auto*
from *mult-dvd-mono*[*OF dvd1 dvd2*] **have** [: −*x*, *1* :]^*2 dvd* (*charpoly cA*)
unfolding *decompc power2-eq-square* **.**
from *order-max*[*OF this 0*] *maximal-eigen-value-order-1*[*OF ev max*]
have *False* **by** *auto*
}
with *le* **show** *cmod x* < *sr* **by** *argo*
qed
qed

lemmas *pf-main* =
eigen-value-sr eigen-vector-z-sr

eigen-value-norm-sr
z-pos
multiplicity-sr-1
nonnegative-eigenvector-has-ev-sr
maximal-eigen-value-order-1
maximal-eigen-value-roots-of-unity-rotation

lemmas *pf-main-connect* = *pf-main*(*1*,*3*,*5*,*7*,*8*−*10*)[*unfolded sr-spectral-radius*]
sr-pos[*unfolded sr-spectral-radius*]
end

end

5.2 Handling Non-Irreducible Matrices as Well

theory *Perron-Frobenius-General*
imports *Perron-Frobenius-Irreducible*
begin

We will need to take sub-matrices and permutations of matrices where the former can best be done via JNF-matrices. So, we first need the Perron-Frobenius theorem in the JNF-world. So, we first define irreducibility of a JNF-matrix.

definition *graph-of-mat* **where**
graph-of-mat A = (*let n* = *dim-row A*; *U* = {..<*n*} *in*
{ *ij*. *A* $$ *ij* ≠ *0*} ∩ *U* × *U*)

definition *irreducible-mat* **where**
irreducible-mat A = (*let n* = *dim-row A in*
(∀ *i j*. *i* < *n* ⟶ *j* < *n* ⟶ (*i*,*j*) ∈ (*graph-of-mat A*)^+))

definition *nonneg-irreducible-mat A* = (*nonneg-mat A* ∧ *irreducible-mat A*)

Next, we have to install transfer rules

context
includes *lifting-syntax*
begin
lemma *HMA-irreducible*[*transfer-rule*]: ((*HMA-M* :: - ⇒ - ^ ′*n* ^ ′*n* ⇒ -) ===> (=))
irreducible-mat fixed-mat.irreducible
proof (*intro rel-funI*, *goal-cases*)
case (*1 a A*)
interpret *fixed-mat A* **.**
let *?t* = *Bij-Nat.to-nat* :: ′*n* ⇒ *nat*
let *?f* = *Bij-Nat.from-nat* :: *nat* ⇒ ′*n*
from *1*[*unfolded HMA-M-def*]
have *a*: *a* = *from-hma*$_m$ *A* (**is** - = *?A*) **by** *auto*
let *?n* = *CARD*(′*n*)

have *dim*: *dim-row a* = *?n* **unfolding** *a* **by** *simp*
have *id*: {..<*?n*} = {*0*..<*?n*} **by** *auto*
have *Aij*: *A* $ *i* $ *j* = *?A* $$ (*?t i*, *?t j*) **for** *i j*
by (*metis* (*no-types*, *lifting*) *to-hma*$_m$*-def to-hma-from-hma*$_m$ *vec-lambda-beta*)
have *graph*: *graph-of-mat a* =
{(*?t i*,*?t j*) | *i j*. *A* $ *i* $ *j* ≠ *0*} (**is** *?G* = -) **unfolding** *graph-of-mat-def dim Let-def id range-to-nat*[*symmetric*]
unfolding *a Aij* **by** *auto*
have *irreducible-mat a* = (∀ *i j*. *i* ∈ *range ?t* ⟶ *j* ∈ *range ?t* ⟶ (*i*,*j*) ∈ *?G*^+)

unfolding *irreducible-mat-def dim Let-def range-to-nat* **by** *auto*
also have ... = (∀ *i j*. (*?t i*, *?t j*) ∈ *?G*^+) **by** *auto*
also note *part1* = *calculation*
have *G*: *?G* = *map-prod ?t ?t* ‘ *G* **unfolding** *graph G-def* **by** *auto*
have *part2*: (*?t i*, *?t j*) ∈ *?G*^+ ⟷ (*i*,*j*) ∈ *G*^+ **for** *i j*
unfolding *G* **by** (*rule inj-trancl-image*, *simp add*: *inj-on-def*)
show *?case* **unfolding** *part1 part2 irreducible-def* **by** *auto*
qed

lemma *HMA-nonneg-irreducible-mat*[*transfer-rule*]: (*HMA-M* ===> (=)) *nonneg-irreducible-mat perron-frobenius*
unfolding *perron-frobenius-def pf-nonneg-mat-def perron-frobenius-axioms-def nonneg-irreducible-mat-def*
by *transfer-prover*
end

The main statements of Perron-Frobenius can now be transferred to JNF-matrices

lemma *perron-frobenius-irreducible*: **fixes** *A* :: *real Matrix.mat* **and** *cA* :: *complex Matrix.mat*
assumes *A*: *A* ∈ *carrier-mat n n* **and** *n*: *n* ≠ *0* **and** *nonneg*: *nonneg-mat A*
and *irr*: *irreducible-mat A*
and *cA*: *cA* = *map-mat of-real A*
and *sr*: *sr* = *Spectral-Radius.spectral-radius cA*
shows
eigenvalue A sr
order sr (*char-poly A*) = *1*
0 < *sr*
eigenvalue cA α ⟹ *cmod* α ≤ *sr*
eigenvalue cA α ⟹ *cmod* α = *sr* ⟹ *order* α (*char-poly cA*) = *1*
∃ *k f*. *k* ≠ *0* ∧ *k* ≤ *n* ∧ *char-poly A* = (*monom 1 k* − [:*sr* ^ *k*:]) ∗ *f* ∧
(∀ *x*. *poly* (*map-poly complex-of-real f*) *x* = *0* ⟶ *cmod x* < *sr*)
proof (*atomize* (*full*), *goal-cases*)
case *1*
from *nonneg irr* **have** *irr*: *nonneg-irreducible-mat A* **unfolding** *nonneg-irreducible-mat-def* **by** *auto*
note *main* = *perron-frobenius.pf-main-connect*[*untransferred*, *cancel-card-constraint*, *OF A irr*,
folded sr cA]

from *main*(*5*,*6*,*7*)[*OF refl refl n*]
have $\exists$ *k f*. $k \neq 0 \wedge k \leq n \wedge$ *char-poly A* = (*monom 1 k* − [:*sr* ^ *k*:]) ∗ *f* ∧
(∀ *x*. *poly* (*map-poly complex-of-real f*) *x* = *0* ⟶ *cmod x* < *sr*) **by** *blast*
with *main*(*1*,*3*,*8*)[*OF n*] *main*(*2*)[*OF - n*] *main*(*4*)[*OF - - n*] **show** *?case* **by** *auto*
qed

We now need permutations on matrices to show that a matrix if a matrix is not irreducible, then it can be turned into a four-block-matrix by a permutation, where the lower left block is 0.

definition *permutation-mat* :: *nat* ⇒ (*nat* ⇒ *nat*) ⇒ *'a* :: *semiring-1 mat* **where**
permutation-mat n p = *Matrix.mat n n* (λ (*i*,*j*). (*if i* = *p j then 1 else 0*))

no-notation *m-inv* (*inv*ı - [*81*] *80*)

lemma *permutation-mat-dim*[*simp*]: *permutation-mat n p* ∈ *carrier-mat n n*
dim-row (*permutation-mat n p*) = *n*
dim-col (*permutation-mat n p*) = *n*
unfolding *permutation-mat-def* **by** *auto*

lemma *permutation-mat-row*[*simp*]: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹
Matrix.row (*permutation-mat n p*) *i* = *unit-vec n* (*inv p i*)
unfolding *permutation-mat-def unit-vec-def* **by** (*intro eq-vecI*, *auto simp*: *permutes-inverses*)

lemma *permutation-mat-col*[*simp*]: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹
Matrix.col (*permutation-mat n p*) *i* = *unit-vec n* (*p i*)
unfolding *permutation-mat-def unit-vec-def* **by** (*intro eq-vecI*, *auto simp*: *permutes-inverses*)

lemma *permutation-mat-left*: **assumes** *A*: *A* ∈ *carrier-mat n nc* **and** *p*: *p permutes* {..<*n*}
shows *permutation-mat n p* ∗ *A* = *Matrix.mat n nc* (λ (*i*,*j*). *A* $$ (*inv p i*, *j*))
proof −
{
fix *i j*
assume *ij*: *i* < *n j* < *nc*
from *p ij*(*1*) **have** *i*: *inv p i* < *n* **by** (*simp add*: *permutes-def*)
have (*permutation-mat n p* ∗ *A*) $$ (*i*,*j*) = *scalar-prod* (*unit-vec n* (*inv p i*)) (*col A j*)
by (*subst index-mult-mat*, *insert ij A p*, *auto*)
also have . . . = *A* $$ (*inv p i*, *j*)
by (*subst scalar-prod-left-unit*, *insert A ij i*, *auto*)
also note *calculation*
}
thus *?thesis* **using** *A*
by (*intro eq-matI*, *auto*)
qed

lemma *permutation-mat-right*: **assumes** *A*: *A* ∈ *carrier-mat nr n* **and** *p*: *p permutes* {..<*n*}
shows *A* ∗ *permutation-mat n p* = *Matrix.mat nr n* (λ (*i*,*j*). *A* $$ (*i*, *p j*))
proof −
{
fix *i j*
assume *ij*: *i* < *nr j* < *n*
from *p ij*(*2*) **have** *j*: *p j* < *n* **by** (*simp add*: *permutes-def*)
have (*A* ∗ *permutation-mat n p*) $$ (*i*,*j*) = *scalar-prod* (*Matrix.row A i*) (*unit-vec n* (*p j*))
by (*subst index-mult-mat*, *insert ij A p*, *auto*)
also have ... = *A* $$ (*i*, *p j*)
by (*subst scalar-prod-right-unit*, *insert A ij j*, *auto*)
also note *calculation*
}
thus *?thesis* **using** *A*
by (*intro eq-matI*, *auto*)
qed

lemma *permutes-lt*: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹ *p i* < *n*
by (*meson lessThan-iff permutes-in-image*)

lemma *permutes-iff*: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹ *j* < *n* ⟹ *p i* = *p j* ⟷ *i* = *j*
by (*metis permutes-inverses*(*2*))

lemma *permutation-mat-id-1*: **assumes** *p*: *p permutes* {..<*n*}
shows *permutation-mat n p* ∗ *permutation-mat n* (*inv p*) = *1*$_m$ *n*
by (*subst permutation-mat-left*[*OF - p*, *of - n*], *force*, *unfold permutation-mat-def*, *rule eq-matI*,
auto simp: *permutes-lt*[*OF permutes-inv*[*OF p*]] *permutes-iff*[*OF permutes-inv*[*OF p*]])

lemma *permutation-mat-id-2*: **assumes** *p*: *p permutes* {..<*n*}
shows *permutation-mat n* (*inv p*) ∗ *permutation-mat n p* = *1*$_m$ *n*
by (*subst permutation-mat-right*[*OF - p*, *of - n*], *force*, *unfold permutation-mat-def*, *rule eq-matI*,
insert p, *auto simp*: *permutes-lt*[*OF p*] *permutes-inverses*)

lemma *permutation-mat-both*: **assumes** *A*: *A* ∈ *carrier-mat n n* **and** *p*: *p permutes* {..<*n*}
shows *permutation-mat n p* ∗ *Matrix.mat n n* (λ (*i*,*j*). *A* $$ (*p i*, *p j*)) ∗ *permutation-mat n* (*inv p*) = *A*
unfolding *permutation-mat-left*[*OF mat-carrier p*]
by (*subst permutation-mat-right*[*OF - permutes-inv*[*OF p*], *of - n*], *force*, *insert A p*,
auto intro!: *eq-matI simp*: *permutes-inverses permutes-lt*[*OF permutes-inv*[*OF p*]])

lemma *permutation-similar-mat*: **assumes** *A*: *A* ∈ *carrier-mat n n* **and** *p*: *p permutes* {..<*n*}
  shows *similar-mat A* (*Matrix.mat n n* (λ (*i*,*j*). *A* $$ (*p i*, *p j*)))
 by (*rule similar-matI*[*OF - permutation-mat-id-1*[*OF p*] *permutation-mat-id-2*[*OF p*]
  permutation-mat-both[*symmetric*, *OF A p*]], *insert A*, *auto*)

lemma *det-four-block-mat-lower-left-zero*: **fixes** *A1* :: *'a* :: *idom mat*
  assumes *A1*: *A1* ∈ *carrier-mat n n*
  and *A2*: *A2* ∈ *carrier-mat n m* **and** *A30*: *A3* = 0_m *m n*
  and *A4*: *A4* ∈ *carrier-mat m m*
shows *Determinant.det* (*four-block-mat A1 A2 A3 A4*) = *Determinant.det A1* ∗ *Determinant.det A4*
proof −
  let *?det* = *Determinant.det*
  let *?t* = *transpose-mat*
  let *?A* = *four-block-mat A1 A2 A3 A4*
  let *?k* = *n* + *m*
  have *A3*: *A3* ∈ *carrier-mat m n* **unfolding** *A30* **by** *auto*
  have *A*: *?A* ∈ *carrier-mat ?k ?k*
    by (*rule four-block-carrier-mat*[*OF A1 A4*])
  have *?det ?A* = *?det* (*?t ?A*)
    by (*rule sym*, *rule Determinant.det-transpose*[*OF A*])
  also have *?t ?A* = *four-block-mat* (*?t A1*) (*?t A3*) (*?t A2*) (*?t A4*)
    by (*rule transpose-four-block-mat*[*OF A1 A2 A3 A4*])
  also have *?det* ... = *?det* (*?t A1*) ∗ *?det* (*?t A4*)
    by (*rule det-four-block-mat-upper-right-zero*[*of - n - m*], *insert A1 A2 A30 A4*, *auto*)
  also have *?det* (*?t A1*) = *?det A1*
    by (*rule Determinant.det-transpose*[*OF A1*])
  also have *?det* (*?t A4*) = *?det A4*
    by (*rule Determinant.det-transpose*[*OF A4*])
  finally show *?thesis* **.**
qed

lemma *char-poly-matrix-four-block-mat*: **assumes**
      A1: *A1* ∈ *carrier-mat n n*
  and *A2*: *A2* ∈ *carrier-mat n m*
  and *A3*: *A3* ∈ *carrier-mat m n*
  and *A4*: *A4* ∈ *carrier-mat m m*
shows *char-poly-matrix* (*four-block-mat A1 A2 A3 A4*) =
  four-block-mat (*char-poly-matrix A1*) (*map-mat* (λ *x*. [:−*x*:]) *A2*)
   (*map-mat* (λ *x*. [:−*x*:]) *A3*) (*char-poly-matrix A4*)
proof −
  from *A1 A4*
  have *dim*[*simp*]: *dim-row A1* = *n dim-col A1* = *n*
      dim-row A4 = *m dim-col A4* = *m* **by** *auto*
  show *?thesis*
    unfolding *char-poly-matrix-def four-block-mat-def Let-def dim*

by (*rule eq-matI*, *insert A2 A3*, *auto*)
qed

lemma *char-poly-four-block-mat-lower-left-zero*: **fixes** *A* :: *'a* :: *idom mat*
assumes *A*: *A* = *four-block-mat B C* (0_m *m n*) *D*
and *B*: *B* ∈ *carrier-mat n n*
and *C*: *C* ∈ *carrier-mat n m*
and *D*: *D* ∈ *carrier-mat m m*
shows *char-poly A* = *char-poly B* ∗ *char-poly D*
unfolding *A char-poly-def*
by (*subst char-poly-matrix-four-block-mat*[*OF B C - D*], *force*,
rule det-four-block-mat-lower-left-zero[*of - n - m*], *insert B C D*, *auto*)

lemma *elements-mat-permutes*: **assumes** *p*: *p permutes* {..< *n*}
and *A*: *A* ∈ *carrier-mat n n*
and *B*: *B* = *Matrix.mat n n* (λ (*i*,*j*). *A* $$ (*p i*, *p j*))
shows *elements-mat A* = *elements-mat B*
proof −
from *A B* **have** [*simp*]: *dim-row A* = *n dim-col A* = *n dim-row B* = *n dim-col B* = *n* **by** *auto*
{
fix *i j*
assume *ij*: *i* < *n j* < *n*
let *?p* = *inv p*
from *permutes-lt*[*OF p*] *ij* **have** ∗: *p i* < *n p j* < *n* **by** *auto*
from *permutes-lt*[*OF permutes-inv*[*OF p*]] *ij* **have** ∗∗: *?p i* < *n ?p j* < *n* **by** *auto*
have ∃ *i' j'*. *B* $$ (*i*,*j*) = *A* $$ (*i'*,*j'*) ∧ *i'* < *n* ∧ *j'* < *n*
∃ *i' j'*. *A* $$ (*i*,*j*) = *B* $$ (*i'*,*j'*) ∧ *i'* < *n* ∧ *j'* < *n*
by (*rule exI*[*of - p i*], *rule exI*[*of - p j*], *insert ij* ∗, *simp add*: *B*,
rule exI[*of - ?p i*], *rule exI*[*of - ?p j*], *insert* ∗∗ *p*, *simp add*: *B permutes-inverses*)
}
thus *?thesis* **unfolding** *elements-mat* **by** *auto*
qed

lemma *elements-mat-four-block-mat-supseteq*:
assumes *A1*: *A1* ∈ *carrier-mat n n*
and *A2*: *A2* ∈ *carrier-mat n m*
and *A3*: *A3* ∈ *carrier-mat m n*
and *A4*: *A4* ∈ *carrier-mat m m*
shows *elements-mat* (*four-block-mat A1 A2 A3 A4*) ⊇
(*elements-mat A1* ∪ *elements-mat A2* ∪ *elements-mat A3* ∪ *elements-mat A4*)
proof
let *?A* = *four-block-mat A1 A2 A3 A4*
have *A*: *?A* ∈ *carrier-mat* (*n* + *m*) (*n* + *m*) **using** *A1 A2 A3 A4* **by** *simp*
from *A1 A4*
have *dim*[*simp*]: *dim-row A1* = *n dim-col A1* = *n*
dim-row A4 = *m dim-col A4* = *m* **by** *auto*
fix *x*

assume *x*: *x* ∈ *elements-mat A1* ∪ *elements-mat A2* ∪ *elements-mat A3* ∪ *elements-mat A4*
{
assume *x* ∈ *elements-mat A1*
from *this*[*unfolded elements-mat*] *A1* **obtain** *i j* **where** *x*: *x* = *A1* $$ (*i*,*j*) **and**

ij: *i* < *n j* < *n* **by** *auto*
have *x* = *?A* $$ (*i*,*j*) **using** *ij* **unfolding** *x four-block-mat-def Let-def* **by** *simp*
from *elements-matI*[*OF A - - this*] *ij* **have** *x* ∈ *elements-mat ?A* **by** *auto*
}
moreover
{
assume *x* ∈ *elements-mat A2*
from *this*[*unfolded elements-mat*] *A2* **obtain** *i j* **where** *x*: *x* = *A2* $$ (*i*,*j*) **and**

ij: *i* < *n j* < *m* **by** *auto*
have *x* = *?A* $$ (*i*,*j* + *n*) **using** *ij* **unfolding** *x four-block-mat-def Let-def* **by** *simp*
from *elements-matI*[*OF A - - this*] *ij* **have** *x* ∈ *elements-mat ?A* **by** *auto*
}
moreover
{
assume *x* ∈ *elements-mat A3*
from *this*[*unfolded elements-mat*] *A3* **obtain** *i j* **where** *x*: *x* = *A3* $$ (*i*,*j*) **and**

ij: *i* < *m j* < *n* **by** *auto*
have *x* = *?A* $$ (*i*+*n*,*j*) **using** *ij* **unfolding** *x four-block-mat-def Let-def* **by** *simp*
from *elements-matI*[*OF A - - this*] *ij* **have** *x* ∈ *elements-mat ?A* **by** *auto*
}
moreover
{
assume *x* ∈ *elements-mat A4*
from *this*[*unfolded elements-mat*] *A4* **obtain** *i j* **where** *x*: *x* = *A4* $$ (*i*,*j*) **and**

ij: *i* < *m j* < *m* **by** *auto*
have *x* = *?A* $$ (*i*+*n*,*j* + *n*) **using** *ij* **unfolding** *x four-block-mat-def Let-def* **by** *simp*
from *elements-matI*[*OF A - - this*] *ij* **have** *x* ∈ *elements-mat ?A* **by** *auto*
}
ultimately show *x* ∈ *elements-mat ?A* **using** *x* **by** *blast*
qed

lemma *non-irreducible-mat-split*:
fixes *A* :: *'a* :: *idom mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *not*: ¬ *irreducible-mat A*
and *n*: *n* > *1*

shows ∃ *n1 n2 B B1 B2 B4*. *similar-mat A B* ∧ *elements-mat A* = *elements-mat B* ∧
B = *four-block-mat B1 B2* (0_m *n2 n1*) *B4* ∧
B1 ∈ *carrier-mat n1 n1* ∧ *B2* ∈ *carrier-mat n1 n2* ∧ *B4* ∈ *carrier-mat n2 n2* ∧
0 < *n1* ∧ *n1* < *n* ∧ *0* < *n2* ∧ *n2* < *n* ∧ *n1* + *n2* = *n*
proof −
from *A* **have** [*simp*]: *dim-row A* = *n* **by** *auto*
let *?G* = *graph-of-mat A*
let *?reachp* = λ *i j*. (*i*,*j*) ∈ *?G*^+
let *?reach* = λ *i j*. (*i*,*j*) ∈ *?G*^∗
have ∃ *i j*. *i* < *n* ∧ *j* < *n* ∧ ¬ *?reach i j*
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *reach*: ⋀ *i j*. *i* < *n* ⟹ *j* < *n* ⟹ *?reach i j* **by** *auto*
from *not*[*unfolded irreducible-mat-def Let-def*]
obtain *i j* **where** *i*: *i* < *n* **and** *j*: *j* < *n* **and** *nreach*: ¬ *?reachp i j* **by** *auto*
from *reach*[*OF i j*] *nreach* **have** *ij*: *i* = *j* **by** (*simp add*: *rtrancl-eq-or-trancl*)
from *n j* **obtain** *k* **where** *k*: *k* < *n* **and** *diff*: *j* ≠ *k* **by** *auto*
from *reach*[*OF j k*] *diff reach*[*OF k j*]
have *?reachp j j* **by** (*simp add*: *rtrancl-eq-or-trancl*)
with *nreach ij* **show** *False* **by** *auto*
qed
then obtain *i j* **where** *i*: *i* < *n* **and** *j*: *j* < *n* **and** *nreach*: ¬ *?reach i j* **by** *auto*
define *I* **where** *I* = {*k*. *k* < *n* ∧ *?reach i k*}
have *iI*: *i* ∈ *I* **unfolding** *I-def* **using** *nreach i* **by** *auto*
have *jI*: *j* ∉ *I* **unfolding** *I-def* **using** *nreach j* **by** *auto*
define *f* **where** *f* = (λ *i*. *if i* ∈ *I then 1 else 0* :: *nat*)
let *?xs* = [*0* ..< *n*]
from *mset-eq-permutation*[*OF mset-sort*, *of ?xs f*] **obtain** *p* **where** *p*: *p permutes* {..< *n*}
and *perm*: *permute-list p ?xs* = *sort-key f ?xs* **by** *auto*
from *p* **have** *lt*[*simp*]: *i* < *n* ⟹ *p i* < *n* **for** *i* **by** (*rule permutes-lt*)
let *?p* = *inv p*
have *ip*: *?p permutes* {..< *n*} **using** *permutes-inv*[*OF p*] **.**
from *ip* **have** *ilt*[*simp*]: *i* < *n* ⟹ *?p i* < *n* **for** *i* **by** (*rule permutes-lt*)
let *?B* = *Matrix.mat n n* (λ (*i*,*j*). *A* $$ (*p i*, *p j*))
define *B* **where** *B* = *?B*
from *permutation-similar-mat*[*OF A p*] **have** *sim*: *similar-mat A B* **unfolding** *B-def* **.**
let *?ys* = *permute-list p ?xs*
define *ys* **where** *ys* = *?ys*
have *len-ys*: *length ys* = *n* **unfolding** *ys-def* **by** *simp*
let *?k* = *length* (*filter* (λ *i*. *f i* = *0*) *ys*)
define *k* **where** *k* = *?k*
have *kn*: *k* ≤ *n* **unfolding** *k-def* **using** *len-ys*
using *length-filter-le*[*of - ys*] **by** *auto*
have *ys-p*: *i* < *n* ⟹ *ys* ! *i* = *p i* **for** *i* **unfolding** *ys-def permute-list-def* **by** *simp*

have *ys*: $ys = map\ (\lambda\ i.\ ys\ !\ i)\ [0\ ..<\ n]$ **unfolding** *len-ys*[*symmetric*]
by (*simp add*: *map-nth*)
also have $\ldots = map\ p\ [0\ ..<\ n]$
by (*rule map-cong*, *insert ys-p*, *auto*)
also have $[0\ ..<\ n] = [0\ ..<\ k]\ @\ [k\ ..<\ n]$ **using** *kn*
using *le-Suc-ex upt-add-eq-append* **by** *blast*
finally have *ys*: $ys = map\ p\ [0\ ..<\ k]\ @\ map\ p\ [k\ ..<\ n]$ **by** *simp*
{
fix *i*
assume *i*: $i < n$
let $?g = (\lambda\ i.\ f\ i = 0)$
let $?f = filter\ ?g$
from *i* **have** *pi*: $p\ i < n$ **using** *p* **by** *simp*
have $k = length\ (?f\ ys)$ **by** *fact*
also have $?f\ ys = ?f\ (map\ p\ [0\ ..<\ k])\ @\ ?f\ (map\ p\ [k\ ..<\ n])$ **unfolding** *ys*
by *simp*
also note $k = calculation$
finally have *True* **by** *blast*
from *perm*[*symmetric*, *folded ys-def*]
have *sorted* (*map f ys*) **using** *sorted-sort-key* **by** *metis*
from *this*[*unfolded ys map-append sorted-append set-map*]
have *sorted*: $\bigwedge\ x\ y.\ x < k \Longrightarrow y \in \{k..<n\} \Longrightarrow f\ (p\ x) \leq f\ (p\ y)$ **by** *auto*
have *0*: $\forall\ i < k.\ f\ (p\ i) = 0$
proof (*rule ccontr*)
assume $\neg\ ?thesis$
then obtain *i* **where** *i*: $i < k$ **and** *zero*: $f\ (p\ i) \neq 0$ **by** *auto*
hence $f\ (p\ i) = 1$ **unfolding** *f-def* **by** (*auto split*: *if-splits*)
from *sorted*[*OF i*, *unfolded this*] **have** *1*: $j \in \{k..<n\} \Longrightarrow f\ (p\ j) \geq 1$ **for** *j*
by *auto*
have *le*: $j \in \{k\ ..<\ n\} \Longrightarrow f\ (p\ j) = 1$ **for** *j* **using** *1*[*of j*] **unfolding** *f-def*
by (*auto split*: *if-splits*)
also have $?f\ (map\ p\ [k\ ..<\ n]) = []$ **using** *le* **by** *auto*
from *k*[*unfolded this*] **have** $length\ (?f\ (map\ p\ [0..<k])) = k$ **by** *simp*
from *length-filter-less*[*of p i map p* [*0* ..< *k*] *?g*, *unfolded this*] *i zero*
show *False* **by** *auto*
qed
hence $?f\ (map\ p\ [0..<k]) = map\ p\ [0..<k]$ **by** *auto*
from *arg-cong*[*OF k*[*unfolded this*, *simplified*], *of set*]
have *1*: $\bigwedge\ i.\ i \in \{k\ ..<\ n\} \Longrightarrow f\ (p\ i) \neq 0$ **by** *auto*
have *1*: $i < n \Longrightarrow \neg\ i < k \Longrightarrow f\ (p\ i) \neq 0$ **for** *i* **using** *1*[*of i*] **by** *auto*
have *0*: $i < n \Longrightarrow (f\ (p\ i) = 0) = (i < k)$ **for** *i* **using** *1*[*of i*] *0*[*rule-format*,
of i] **by** *blast*
have *main*: $(f\ i = 0) = (?p\ i < k)$ **using** *0*[*of ?p i*] *i p*
by (*auto simp*: *permutes-inverses*)
have $i \in I \longleftrightarrow f\ i \neq 0$ **unfolding** *f-def* **by** *simp*
also have $(f\ i = 0) \longleftrightarrow ?p\ i < k$ **using** *main* **by** *auto*
finally have $i \in I \longleftrightarrow ?p\ i \geq k$ **by** *auto*
} **note** $main = this$
from *main*[*OF j*] *jI*

have *k0*: $k \neq 0$ **by** *auto*
from *iI main*[*OF i*] **have** *?p* $i \geq k$ **by** *auto*
with *ilt*[*OF i*] **have** *kn*: $k < n$ **by** *auto*
{
fix *i j*
assume *i*: $i < n$ **and** *ik*: $k \leq i$ **and** *jk*: $j < k$
with *kn* **have** *j*: $j < n$ **by** *auto*
have *jI*: $p\ j \notin I$
by (*subst main*, *insert jk j p*, *auto simp*: *permutes-inverses*)
have *iI*: $p\ i \in I$
by (*subst main*, *insert i ik p*, *auto simp*: *permutes-inverses*)
from *i j* **have** *B* $$ (*i*,*j*) = *A* $$ (*p i*, *p j*) **unfolding** *B-def* **by** *auto*
also have ... = *0*
proof (*rule ccontr*)
assume *A* $$ (*p i*, *p j*) ≠ *0*
hence (*p i*, *p j*) ∈ *?G* **unfolding** *graph-of-mat-def Let-def* **using** *i j p* **by** *auto*
with *iI j* **have** *p j* ∈ *I* **unfolding** *I-def* **by** *auto*
with *jI* **show** *False* **by** *simp*
qed
finally have *B* $$ (*i*,*j*) = *0* **.**
} **note** *zero* = *this*
have *dimB*[*simp*]: *dim-row B* = *n dim-col B* = *n* **unfolding** *B-def* **by** *auto*
have *dim*: *dim-row B* = *k* + (*n* − *k*) *dim-col B* = *k* + (*n* − *k*) **using** *kn* **by** *auto*
obtain *B1 B2 B3 B4* **where** *spl*: *split-block B k k* = (*B1*,*B2*,*B3*,*B4*) (**is** *?tmp* = -) **by** (*cases ?tmp*, *auto*)
from *split-block*[*OF this dim*] **have**
Bs: *B1* ∈ *carrier-mat k k B2* ∈ *carrier-mat k* (*n* − *k*)
B3 ∈ *carrier-mat* (*n* − *k*) *k B4* ∈ *carrier-mat* (*n* − *k*) (*n* − *k*)
and *B*: *B* = *four-block-mat B1 B2 B3 B4* **by** *auto*
have *B3*: *B3* = 0_m (*n* − *k*) *k* **unfolding** *arg-cong*[*OF spl*[*symmetric*], *of* λ (-,-,*B*,-). *B*, *unfolded split*]
unfolding *split-block-def Let-def split*
by (*rule eq-matI*, *auto simp*: *kn zero*)
from *elements-mat-permutes*[*OF p A B-def*]
have *elem*: *elements-mat A* = *elements-mat B* **.**
show *?thesis*
by (*intro exI conjI*, *rule sim*, *rule elem*, *rule B*[*unfolded B3*], *insert Bs k0 kn*, *auto*)
qed

lemma *non-irreducible-nonneg-mat-split*:
fixes *A* :: *'a* :: *linordered-idom mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *not*: ¬ *irreducible-mat A*
and *n*: *n* > *1*
shows ∃ *n1 n2 A1 A2*. *char-poly A* = *char-poly A1* ∗ *char-poly A2*

$\wedge$ *nonneg-mat A1* $\wedge$ *nonneg-mat A2*
$\wedge$ *A1* $\in$ *carrier-mat n1 n1* $\wedge$ *A2* $\in$ *carrier-mat n2 n2*
$\wedge$ *0* $<$ *n1* $\wedge$ *n1* $<$ *n* $\wedge$ *0* $<$ *n2* $\wedge$ *n2* $<$ *n* $\wedge$ *n1* $+$ *n2* $=$ *n*
proof −
from *non-irreducible-mat-split*[*OF A not n*]
obtain *n1 n2 B B1 B2 B4*
where *sim*: *similar-mat A B* **and** *elem*: *elements-mat A* = *elements-mat B*
and *B*: *B* = *four-block-mat B1 B2* (0_m *n2 n1*) *B4*
and *Bs*: *B1* $\in$ *carrier-mat n1 n1 B2* $\in$ *carrier-mat n1 n2 B4* $\in$ *carrier-mat n2 n2*
and *n*: *0* $<$ *n1 n1* $<$ *n 0* $<$ *n2 n2* $<$ *n n1* $+$ *n2* $=$ *n* **by** *auto*
from *char-poly-similar*[*OF sim*]
have *AB*: *char-poly A* = *char-poly B* **.**
from *nonneg* **have** *nonneg*: *nonneg-mat B* **unfolding** *nonneg-mat-def elem* **by** *auto*
have *cB*: *char-poly B* = *char-poly B1* $*$ *char-poly B4*
by (*rule char-poly-four-block-mat-lower-left-zero*[*OF B Bs*])
from *nonneg* **have** *B1-B4*: *nonneg-mat B1 nonneg-mat B4* **unfolding** *B nonneg-mat-def*
using *elements-mat-four-block-mat-supseteq*[*OF Bs*(*1*−*2*) - *Bs*(*3*), *of* 0_m *n2 n1*] **by** *auto*
show *?thesis*
by (*intro exI conjI*, *rule AB*[*unfolded cB*], *rule B1-B4*, *rule B1-B4*,
rule Bs, *rule Bs*, *insert n*, *auto*)
qed

The main generalized theorem. The characteristic polynomial of a non-negative real matrix can be represented as a product of roots of unitys (scaled by the the spectral radius sr) and a polynomial where all roots are smaller than the spectral radius.

theorem *perron-frobenius-nonneg*: **fixes** *A* :: *real Matrix.mat*
assumes *A*: *A* $\in$ *carrier-mat n n* **and** *pos*: *nonneg-mat A* **and** *n*: *n* $\neq$ *0*
shows $\exists$ *sr ks f*.
sr $\geq$ *0* $\wedge$
0 $\notin$ *set ks* $\wedge$ *ks* $\neq$ [] $\wedge$
char-poly A = *prod-list* (*map* (λ *k*. *monom 1 k* − [:*sr* ^ *k*:]) *ks*) $*$ *f* $\wedge$
($\forall$ *x*. *poly* (*map-poly complex-of-real f*) *x* = *0* $\longrightarrow$ *cmod x* $<$ *sr*)
proof −
define *p* **where** *p* = (λ *sr k*. *monom 1 k* − [: (*sr* :: *real*) ^ *k*:])
let *?small* = λ *f sr*. ($\forall$ *x*. *poly* (*map-poly complex-of-real f*) *x* = *0* $\longrightarrow$ *cmod x* $<$ *sr*)
let *?wit* = λ *A sr ks f*. *sr* $\geq$ *0* $\wedge$ *0* $\notin$ *set ks* $\wedge$ *ks* $\neq$ [] $\wedge$
char-poly A = *prod-list* (*map* (*p sr*) *ks*) $*$ *f* $\wedge$ *?small f sr*
let *?c* = *complex-of-real*
interpret *c*: *field-hom ?c* **..**
interpret *p*: *map-poly-inj-idom-divide-hom ?c* **..**
have *map-p*: *map-poly ?c* (*p sr k*) = (*monom 1 k* − [:*?c sr*^*k*:]) **for** *sr k*
unfolding *p-def* **by** (*simp add*: *hom-distribs*)
{

```
    fix k x sr
    assume 0: poly (map-poly ?c (p sr k)) x = 0 and k: k ≠ 0 and sr: sr ≥ 0
    note 0 also note map-p
    finally have x^k = (?c sr)^k by (simp add: poly-monom)
    from arg-cong[OF this, of λ c. root k (cmod c), unfolded norm-power] k
    have cmod x = cmod (?c sr) using real-root-pos2 by auto
    also have ... = sr using sr by auto
    finally have cmod x = sr .
  } note p-conv = this
  have ∃ sr ks f. ?wit A sr ks f using A pos n
  proof (induct n arbitrary: A rule: less-induct)
    case (less n A)
    note pos = less(3)
    note A = less(2)
    note IH = less(1)
    note n = less(4)
    from n
    consider (1) n = 1
      | (irr) irreducible-mat A
      | (red) ¬ irreducible-mat A n > 1
      by force
    thus ∃ sr ks f. ?wit A sr ks f
    proof cases
      case irr
      from perron-frobenius-irreducible(3,6)[OF A n pos irr refl refl]
      obtain sr k f where
        *: sr > 0 k ≠ 0 char-poly A = p sr k * f ?small f sr unfolding p-def
        by auto
      hence ?wit A sr [k] f by auto
      thus ?thesis by blast
    next
      case red
      from non-irreducible-nonneg-mat-split[OF A pos red] obtain n1 n2 A1 A2
        where char:  char-poly A = char-poly A1 * char-poly A2
          and pos: nonneg-mat A1 nonneg-mat A2
          and A: A1 ∈ carrier-mat n1 n1 A2 ∈ carrier-mat n2 n2
          and n: n1 < n n2 < n
          and n0: n1 ≠ 0 n2 ≠ 0 by auto
      from IH[OF n(1) A(1) pos(1) n0(1)] obtain sr1 ks1 f1 where 1: ?wit A1
sr1 ks1 f1 by blast
      from IH[OF n(2) A(2) pos(2) n0(2)] obtain sr2 ks2 f2 where 2: ?wit A2
sr2 ks2 f2 by blast
      have ∃ A1 A2 sr1 ks1 f1 sr2 ks2 f2. ?wit A1 sr1 ks1 f1 ∧ ?wit A2 sr2 ks2 f2
∧
         sr1 ≥ sr2 ∧ char-poly A = char-poly A1 * char-poly A2
      proof (cases sr1 ≥ sr2)
        case True
        show ?thesis unfolding char
          by (intro exI, rule conjI[OF 1 conjI[OF 2]], insert True, auto)
```

next
case *False*
show *?thesis* **unfolding** *char*
by (*intro exI*, *rule conjI*[*OF 2 conjI*[*OF 1*]], *insert False*, *auto*)
qed
then obtain *A1 A2 sr1 ks1 f1 sr2 ks2 f2* **where**
1: *?wit A1 sr1 ks1 f1* **and** *2*: *?wit A2 sr2 ks2 f2* **and**
sr: *sr1* ≥ *sr2* **and** *char*: *char-poly A* = *char-poly A1* ∗ *char-poly A2* **by** *blast*
show *?thesis*
proof (*cases sr1* = *sr2*)
case *True*
have *?wit A sr2* (*ks1* @ *ks2*) (*f1* ∗ *f2*) **unfolding** *char*
by (*insert 1 2 True*, *auto simp*: *True p.hom-mult*)
thus *?thesis* **by** *blast*
next
case *False*
with *sr* **have** *sr1*: *sr1* > *sr2* **by** *auto*
have *lt*: *poly* (*map-poly ?c* (*p sr2 k*)) *x* = *0* ⟹ *k* ∈ *set ks2* ⟹ *cmod x* < *sr1* **for** *k x*
using *sr1 p-conv*[*of sr2 k x*] *2* **by** *auto*
have *?wit A sr1 ks1* (*f1* ∗ *f2* ∗ *prod-list* (*map* (*p sr2*) *ks2*)) **unfolding** *char*
by (*insert 1 2 sr1 lt*, *auto simp*: *p.hom-mult p.hom-prod-list*
poly-prod-list prod-list-zero-iff)
thus *?thesis* **by** *blast*
qed
next
case *1*
define *a* **where** *a* = *A* $$ (*0*,*0*)
have *A*: *A* = *Matrix.mat 1 1* (λ *x*. *a*)
by (*rule eq-matI*, *unfold a-def*, *insert A 1*(*1*), *auto*)
have *char*: *char-poly A* = [: − *a*, *1* :] **unfolding** *A*
by (*auto simp*: *Determinant.det-def char-poly-def char-poly-matrix-def*)
from *pos A* **have** *a*: *a* ≥ *0* **unfolding** *nonneg-mat-def elements-mat* **by** *auto*
have *?wit A a* [*1*] *1* **unfolding** *char* **using** *a* **by** (*auto simp*: *p-def monom-Suc*)
thus *?thesis* **by** *blast*
qed
qed
then obtain *sr ks f* **where** *wit*: *?wit A sr ks f* **by** *blast*
thus *?thesis* **using** *wit* **unfolding** *p-def* **by** *auto*
qed

And back to HMA world via transfer.

theorem *perron-frobenius-non-neg*: **fixes** *A* :: *real* ^ *'n* ^ *'n*
assumes *pos*: *non-neg-mat A*
shows ∃ *sr ks f*.
sr ≥ *0* ∧
0 ∉ *set ks* ∧ *ks* ≠ [] ∧
charpoly A = *prod-list* (*map* (λ *k*. *monom 1 k* − [:*sr* ^ *k*:]) *ks*) ∗ *f* ∧

(∀ x. poly (map-poly complex-of-real f) x = 0 ⟶ cmod x < sr)
using pos
proof (transfer, goal-cases)
case (1 A)
from perron-frobenius-nonneg[OF 1]
show ?case **by** auto
qed

We now specialize the theorem for complexity analysis where we are mainly interested in the case where the spectral radius is as most 1. Note that this can be checked by tested that there are no real roots of the characteristic polynomial which exceed 1.

Moreover, here the existential quantifier over the factorization is replaced by *decompose-prod-root-unity*, an algorithm which computes this factorization in an efficient way.

```
lemma perron-frobenius-for-complexity: fixes A :: real ^ 'n ^ 'n and f :: real poly

  defines cA ≡ map-matrix complex-of-real A
  defines cf ≡ map-poly complex-of-real f
  assumes pos: non-neg-mat A
   and sr: ⋀ x. poly (charpoly A) x = 0 ⟹ x ≤ 1
   and decomp: decompose-prod-root-unity (charpoly A) = (ks, f)
  shows 0 ∉ set ks
   charpoly A = prod-root-unity ks * f
   charpoly cA = prod-root-unity ks * cf
   ⋀ x. poly (charpoly cA) x = 0 ⟹ cmod x ≤ 1
   ⋀ x. poly cf x = 0 ⟹ cmod x < 1
   ⋀ x. cmod x = 1 ⟹ order x (charpoly cA) = length [k←ks . x ^ k = 1]
   ⋀ x. cmod x = 1 ⟹ poly (charpoly cA) x = 0 ⟹ ∃ k ∈ set ks. x^k = 1
  unfolding cf-def cA-def
proof (atomize(full), goal-cases)
  case 1
  let ?c = complex-of-real
  let ?cp = map-poly ?c
  let ?A = map-matrix ?c A
  let ?wit = λ ks f. 0 ∉ set ks ∧
    charpoly A = prod-root-unity ks * f ∧
    charpoly ?A = prod-root-unity ks * map-poly of-real f ∧
    (∀ x. poly (charpoly ?A) x = 0 ⟶ cmod x ≤ 1) ∧
    (∀ x. poly (?cp f) x = 0 ⟶ cmod x < 1)
  interpret field-hom ?c ..
  interpret p: map-poly-inj-idom-divide-hom ?c ..
  {
    from perron-frobenius-non-neg[OF pos] obtain sr ks f
      where *: sr ≥ 0 0 ∉ set ks ks ≠ []
       and cp: charpoly A = prod-list (map (λ k. monom 1 k − [:sr ^ k:]) ks) * f
       and small: ⋀ x. poly (?cp f) x = 0 ⟹ cmod x < sr by blast

    from arg-cong[OF cp, of map-poly ?c]
```

have *cpc*: *charpoly ?A* = *prod-list* (*map* (λ *k*. *monom 1 k* − [:*?c sr* ^ *k*:]) *ks*) ∗ *map-poly ?c f*
by (*simp add*: *charpoly-of-real hom-distribs p.prod-list-map-hom*[*symmetric*] *o-def*)
have *sr-le-1*: *sr* ≤ *1*
by (*rule sr*, *unfold cp*, *insert* ∗, *cases ks*, *auto simp*: *poly-monom*)
{
fix *x*
note [*simp*] = *prod-list-zero-iff o-def poly-monom*
assume *poly* (*charpoly ?A*) *x* = *0*
from *this*[*unfolded cpc poly-mult poly-prod-list*] *small*[*of x*]
consider (*lt*) *cmod x* < *sr* | (*mem*) *k* **where** *k* ∈ *set ks x* ^ *k* = (*?c sr*) ^ *k* **by** *force*
hence *cmod x* ≤ *sr*
proof (*cases*)
case (*mem k*)
with ∗ **have** *k*: *k* ≠ *0* **by** *metis*
with *arg-cong*[*OF mem*(*2*), *of* λ *x*. *root k* (*cmod x*), *unfolded norm-power*]
real-root-pos2[*of k*] ∗(*1*)
have *cmod x* = *sr* **by** *auto*
thus *?thesis* **by** *auto*
qed *simp*
} **note** *root* = *this*
have ∃ *ks f*. *?wit ks f*
proof (*cases sr* = *1*)
case *False*
with *sr-le-1* **have** ∗: *cmod x* ≤ *sr* ⟹ *cmod x* < *1* *cmod x* ≤ *sr* ⟹ *cmod x* ≤ *1* **for** *x* **by** *auto*
show *?thesis*
by (*rule exI*[*of - Nil*], *rule exI*[*of - charpoly A*], *insert* ∗ *root*,
auto simp: *prod-root-unity-def charpoly-of-real*)
next
case *sr*: *True*
from ∗ *cp cpc small root*
show *?thesis* **unfolding** *sr root-unity-def prod-root-unity-def* **by** (*auto simp*: *pCons-one*)
qed
}
then obtain *Ks F* **where** *wit*: *?wit Ks F* **by** *auto*
have *cA0*: *charpoly ?A* ≠ *0* **using** *degree-monic-charpoly*[*of ?A*] **by** *auto*
from *wit* **have** *id*: *charpoly ?A* = *prod-root-unity Ks* ∗ *?cp F* **by** *auto*
from *of-real-hom.hom-decompose-prod-root-unity*[*of charpoly A*, *unfolded decomp*]
have *decompc*: *decompose-prod-root-unity* (*charpoly ?A*) = (*ks*, *?cp f*)
by (*auto simp*: *charpoly-of-real*)
from *wit* **have** *small*: *cmod x* = *1* ⟹ *poly* (*?cp F*) *x* ≠ *0* **for** *x* **by** *auto*
from *decompose-prod-root-unity*[*OF id decompc this cA0*]
have *id*: *charpoly ?A* = *prod-root-unity ks* ∗ *?cp F F* = *f set Ks* = *set ks* **by** *auto*
have *?cp* (*charpoly A*) = *?cp* (*prod-root-unity ks* ∗ *f*) **unfolding** *id*
unfolding *charpoly-of-real*[*symmetric*] *id p.hom-mult of-real-hom.hom-prod-root-unity*

..
hence *idr*: *charpoly A* = *prod-root-unity ks* ∗ *f* **by** *auto*
have *wit*: *?wit ks f* **and** *idc*: *charpoly ?A* = *prod-root-unity ks* ∗ *?cp f*
using *wit* **unfolding** *id idr* **by** *auto*
{
fix *x*
assume *cmod x* = *1*
from *small*[*OF this*, *unfolded id*] **have** *poly* (*?cp f*) *x* ≠ *0* **by** *auto*
from *order-0I*[*OF this*] *this* **have** *ord*: *order x* (*?cp f*) = *0* **and** *cf0*: *?cp f* ≠ *0* **by** *auto*
have *order x* (*charpoly ?A*) = *order x* (*prod-root-unity ks*) **unfolding** *idc*
by (*subst order-mult*, *insert cf0 wit ord*, *auto*)
also have . . . = *length* [*k*←*ks* . *x* ^ *k* = *1*]
by (*subst order-prod-root-unity*, *insert wit*, *auto*)
finally have *ord*: *order x* (*charpoly ?A*) = *length* [*k*←*ks* . *x* ^ *k* = *1*] **.**
{
assume *poly* (*charpoly ?A*) *x* = *0*
with *cA0* **have** *order x* (*charpoly ?A*) ≠ *0* **unfolding** *order-root* **by** *auto*
from *this*[*unfolded ord*] **have** ∃ *k* ∈ *set ks*. *x* ^ *k* = *1*
by (*cases* [*k*←*ks* . *x* ^ *k* = *1*], *force*+)
}
note *this ord*
}
with *wit* **show** *?case* **by** *blast*
qed

and convert to JNF-world

lemmas *perron-frobenius-for-complexity-jnf* =
perron-frobenius-for-complexity[*unfolded atomize-imp atomize-all*,
untransferred, *cancel-card-constraint*, *rule-format*]

end

6 Combining Spectral Radius Theory with Perron Frobenius theorem

theory *Spectral-Radius-Theory*
imports
Polynomial-Factorization.Square-Free-Factorization
Jordan-Normal-Form.Spectral-Radius
Jordan-Normal-Form.Char-Poly
Perron-Frobenius
HOL−Computational-Algebra.Field-as-Ring
begin
abbreviation *spectral-radius* **where** *spectral-radius* ≡ *Spectral-Radius.spectral-radius*
hide-const (**open**) *Module.smult*

Via JNFs it has been proven that the growth of A^k is polynomially bounded, if all complex eigenvalues have a norm at most 1, i.e., the spectral radius must be at most 1. Moreover, the degree of the polynomial growth can be bounded by the order of those roots which have norm 1, cf. $[\![$*?A* $\in$ *carrier-mat ?n ?n*; *Spectral-Radius-Theory.spectral-radius ?A* $\leq$ *1*; $\bigwedge$*ev k.* $[\![$*poly* (*char-poly ?A*) *ev* = *0*; *cmod ev* = *1*$]\!]$ $\Longrightarrow$ *order ev* (*char-poly ?A*) $\leq$ *?d*$]\!]$ $\Longrightarrow$ $\exists$ *c1 c2*. $\forall$ *k*. *norm-bound* (*?A* $\widehat{\ }_m$ *k*) (*c1* + *c2* $*$ (*real k*)$^{?d\ -\ 1}$).

Perron Frobenius theorem tells us that for a real valued non negative matrix, the largest eigenvalue is a real non-negative one. Hence, we only have to check, that all real eigenvalues are at most one.

We combine both theorems in the following. To be more precise, the set-based complexity results from JNFs with the type-based Perron Frobenius theorem in HMA are connected to obtain a set based complexity criterion for real-valued non-negative matrices, where one only investigated the real valued eigenvalues for checking the eigenvalue-at-most-1 condition. Here, in the precondition of the roots of the polynomial, the type-system ensures that we only have to look at real-valued eigenvalues, and can ignore the complex-valued ones.

The linkage between set-and type-based is performed via HMA-connect.

```
lemma perron-frobenius-spectral-radius-complex: fixes A :: complex mat
  assumes A: A ∈ carrier-mat n n
  and real-nonneg: real-nonneg-mat A
  and ev-le-1: ⋀ x. poly (char-poly (map-mat Re A)) x = 0 ⟹ x ≤ 1
  and ev-order: ⋀ x. norm x = 1 ⟹ order x (char-poly A) ≤ d
  shows ∃ c1 c2. ∀ k. norm-bound (A ^m k) (c1 + c2 * real k ^ (d − 1))
proof (cases n = 0)
  case False
  hence n: n > 0 n ≠ 0 by auto
  define sr where sr = spectral-radius A
  note sr = spectral-radius-mem-max[OF A n(1), folded sr-def]
  show ?thesis
  proof (rule spectral-radius-poly-bound[OF A], unfold sr-def[symmetric])
    let ?cr = complex-of-real
```

here is the transition from type-based perron-frobenius to set-based

```
    from perron-frobenius[untransferred, cancel-card-constraint, OF A real-nonneg
n(2)]
      obtain v where v: v ∈ carrier-vec n and ev: eigenvector A v (?cr sr) and
      rnn: real-nonneg-vec v unfolding sr-def by auto
    define B where B = map-mat Re A
    let ?A = map-mat ?cr B
    have AB: A = ?A unfolding B-def
    by (rule eq-matI, insert real-nonneg[unfolded real-nonneg-mat-def elements-mat-def],
auto)
    define w where w = map-vec Re v
    let ?v = map-vec ?cr w
    have vw: v = ?v unfolding w-def
```

by (*rule eq-vecI*, *insert rnn*[*unfolded real-nonneg-vec-def vec-elements-def*], *auto*)
have *B*: $B \in$ *carrier-mat n n* **unfolding** *B-def* **using** *A* **by** *auto*
from *AB vw ev* **have** *ev*: *eigenvector ?A ?v* (*?cr sr*) **by** *simp*
have *eigenvector B w sr*
by (*rule of-real-hom.eigenvector-hom-rev*[*OF B ev*])
hence *eigenvalue B sr* **unfolding** *eigenvalue-def* **by** *blast*
from *ev-le-1*[*folded B-def*, *OF this*[*unfolded eigenvalue-root-char-poly*[*OF B*]]]
show $sr \leq 1$ **.**
next
fix *ev*
assume *cmod ev* = *1*
thus *order ev* (*char-poly A*) $\leq d$ **by** (*rule ev-order*)
qed
next
case *True*
with *A* **show** *?thesis*
by (*intro exI*[*of - 0*], *auto simp*: *norm-bound-def*)
qed

The following lemma is the same as ⟦*?A* ∈ *carrier-mat ?n ?n*; *real-nonneg-mat ?A*; $\bigwedge x$. *poly* (*char-poly* (*map-mat Re ?A*)) $x = 0 \Longrightarrow x \leq 1$; $\bigwedge x$. *cmod* $x = 1 \Longrightarrow$ *order x* (*char-poly ?A*) $\leq$ *?d*⟧ $\Longrightarrow \exists$ *c1 c2*. $\forall k$. *norm-bound* (*?A* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ (*real k*)$^{?d\ -\ 1}$), except that now the type *real* is used instead of *complex*.

lemma *perron-frobenius-spectral-radius*: **fixes** *A* :: *real mat*
assumes *A*: $A \in$ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *ev-le-1*: $\forall$ *x*. *poly* (*char-poly A*) $x = 0 \longrightarrow x \leq 1$
and *ev-order*: $\forall$ *x* :: *complex*. *norm x* = *1* $\longrightarrow$ *order x* (*map-poly of-real* (*char-poly A*)) $\leq d$
shows $\exists$ *c1 c2*. $\forall$ *k a*. $a \in$ *elements-mat* (*A* $\widehat{\ }_m$ *k*) $\longrightarrow$ *abs a* $\leq$ (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
proof −
let *?cr* = *complex-of-real*
let *?B* = *map-mat ?cr A*
have *B*: *?B* ∈ *carrier-mat n n* **using** *A* **by** *auto*
have *rnn*: *real-nonneg-mat ?B* **using** *nonneg* **unfolding** *real-nonneg-mat-def nonneg-mat-def*
by (*auto simp*: *elements-mat-def*)
have *id*: *map-mat Re ?B* = *A*
by (*rule eq-matI*, *auto*)
have $\exists$ *c1 c2*. $\forall$ *k*. *norm-bound* (*?B* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
by (*rule perron-frobenius-spectral-radius-complex*[*OF B rnn*], *unfold id*, *insert ev-le-1 ev-order*, *auto simp*: *of-real-hom.char-poly-hom*[*OF A*])
then obtain *c1 c2* **where** *nb*: $\bigwedge$ *k*. *norm-bound* (*?B* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *real k* ^ (*d* − *1*)) **by** *auto*
show *?thesis*
proof (*rule exI*[*of - c1*], *rule exI*[*of - c2*], *intro allI impI*)

fix k a
assume $a \in$ *elements-mat* $(A \ \widehat{}_m\ k)$
with *pow-carrier-mat*[*OF A*] **obtain** i j **where** a: $a = (A \ \widehat{}_m\ k)$ $$ (i,j) **and** ij: $i < n$ $j < n$
unfolding *elements-mat* **by** *force*
from ij nb[*of k*] A **have** *norm* $((?B \ \widehat{}_m\ k)$ $$ $(i,j)) \leq c1 + c2 * real\ k\ \widehat{}\ (d - 1)$
unfolding *norm-bound-def* **by** *auto*
also have $(?B \ \widehat{}_m\ k)$ $$ $(i,j) = ?cr\ a$
unfolding *of-real-hom.mat-hom-pow*[*OF A*, *symmetric*] a **using** ij A **by** *auto*
also have *norm* $(?cr\ a) = abs\ a$ **by** *auto*
finally show $abs\ a \leq (c1 + c2 * real\ k\ \widehat{}\ (d - 1))$ **.**
qed
qed

We can also convert the set-based lemma $[\![?A \in carrier\text{-}mat\ ?n\ ?n; nonneg\text{-}mat\ ?A; \forall x.\ poly\ (char\text{-}poly\ ?A)\ x = 0 \longrightarrow x \leq 1; \forall x.\ cmod\ x = 1 \longrightarrow order\ x\ (map\text{-}poly\ complex\text{-}of\text{-}real\ (char\text{-}poly\ ?A)) \leq ?d]\!] \Longrightarrow \exists c1\ c2.\ \forall k\ a.\ a \in elements\text{-}mat\ (?A \ \widehat{}_m\ k) \longrightarrow |a| \leq c1 + c2 * (real\ k)^{?d - 1}$ to a type-based version.

lemma *perron-frobenius-spectral-type-based*:
assumes *non-neg-mat* $(A :: real\ \widehat{}\ 'n\ \widehat{}\ 'n)$
and $\forall\ x.\ poly\ (charpoly\ A)\ x = 0 \longrightarrow x \leq 1$
and $\forall\ x :: complex.\ norm\ x = 1 \longrightarrow order\ x\ (map\text{-}poly\ of\text{-}real\ (charpoly\ A)) \leq d$
shows $\exists c1\ c2.\ \forall k\ a.\ a \in elements\text{-}mat\text{-}h\ (matpow\ A\ k) \longrightarrow abs\ a \leq (c1 + c2 * real\ k\ \widehat{}\ (d - 1))$
using *assms perron-frobenius-spectral-radius*
by (*transfer*, *blast*)

And of course, we can also transfer the type-based lemma back to a set-based setting, only that – without further case-analysis – we get the additional assumption $n \neq 0$.

lemma assumes $A \in$ *carrier-mat* n n
and *nonneg-mat* A
and $\forall\ x.\ poly\ (char\text{-}poly\ A)\ x = 0 \longrightarrow x \leq 1$
and $\forall\ x :: complex.\ norm\ x = 1 \longrightarrow order\ x\ (map\text{-}poly\ of\text{-}real\ (char\text{-}poly\ A)) \leq d$
and $n \neq 0$
shows $\exists c1\ c2.\ \forall k\ a.\ a \in elements\text{-}mat\ (A \ \widehat{}_m\ k) \longrightarrow abs\ a \leq (c1 + c2 * real\ k\ \widehat{}\ (d - 1))$
using *perron-frobenius-spectral-type-based*[*untransferred*, *cancel-card-constraint*, *OF assms*] **.**

Note that the precondition eigenvalue-at-most-1 can easily be formulated as a cardinality constraints which can be decided by Sturm's theorem. And in order to obtain a bound on the order, one can perform a square-free-factorization (via Yun's factorization algorithm) of the characteristic polynomial into $f_1^1 \cdot \ldots f_d^d$ where each f_i has precisely the roots of order i.

context
  fixes *A* :: *real mat* **and** *c* :: *real* **and** *fis* **and** *n* :: *nat*
  assumes *A*: $A \in$ *carrier-mat n n*
  and *nonneg*: *nonneg-mat A*
  and *yun*: *yun-factorization gcd* (*char-poly A*) = (*c*,*fis*)
  and *ev-le-1*: *card* {*x*. *poly* (*char-poly A*) *x* = *0* ∧ *x* > *1*} = *0*
begin

Note that *yun-factorization* has an offset by 1, so the pair $(f_i,\ i) \in$ *set fis* encodes $f_i^{Suc\ i}$.

lemma *perron-frobenius-spectral-radius-yun*:
  assumes *bnd*: $\bigwedge f_i\ i.\ (f_i,i) \in$ *set fis*
    $\Longrightarrow$ (∃ *x* :: *complex*. *poly* (*map-poly of-real* f_i) *x* = *0* ∧ *norm x* = *1*)
    $\Longrightarrow$ *Suc i* ≤ *d*
  shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* ($A\ \widehat{}_m\ k$) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *real* *k* ^ (*d* − *1*))
proof (*rule perron-frobenius-spectral-radius*[*OF A nonneg*]; *intro allI impI*)
  let *?cr* = *complex-of-real*
  let *?cp* = *map-poly ?cr* (*char-poly A*)
  fix *x* :: *complex*
  assume *x*: *norm x* = *1*
  have *A0*: *char-poly A* ≠ *0* **using** *degree-monic-char-poly*[*OF A*] **by** *auto*
  interpret *field-hom-0′ ?cr* **by** (*standard*, *auto*)
  from *A0* **have** *cp0*: *?cp* ≠ *0* **by** *auto*
  obtain *ox* **where** *ox*: *order x ?cp* = *ox* **by** *blast*
  note *sff* = *square-free-factorization-order-root*[*OF yun-factorization*(*1*)[*OF* *yun-factorization-hom*[*of char-poly A*, *unfolded yun map-prod-def split*]] *cp0*, *of* *x ox*, *unfolded ox*]
  show *order x ?cp* ≤ *d* **unfolding** *ox*
  proof (*cases ox*)
    case (*Suc oo*)
    with *sff* **obtain** *fi* **where** *mem*: (*fi*,*oo*) ∈ *set fis* **and** *rt*: *poly* (*map-poly ?cr fi*) *x* = *0* **by** *auto*
    from *bnd*[*OF mem exI*[*of* - *x*], *OF conjI*[*OF rt x*]]
    show *ox* ≤ *d* **unfolding** *Suc* **.**
  qed *auto*
next
  let *?L* = {*x*. *poly* (*char-poly A*) *x* = *0* ∧ *x* > *1*}
  fix *x* :: *real*
  assume *rt*: *poly* (*char-poly A*) *x* = *0*
  have *finite ?L*
    by (*rule finite-subset*[*OF* - *poly-roots-finite*[*of char-poly A*]],
      insert degree-monic-char-poly[*OF A*], *auto*)
  with *ev-le-1* **have** *?L* = {} **by** *simp*
  with *rt* **show** *x* ≤ *1* **by** *auto*
qed

Note that the only remaining problem in applying ($\bigwedge f_i\ i.$ ⟦$(f_i,\ i) \in$ *set fis*; ∃ *x*. *poly* (*map-poly complex-of-real* f_i) *x* = *0* ∧ *cmod x* = *1*⟧ ⟹ *Suc i* ≤ *?d*) ⟹ ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* ($A\ \widehat{}_m\ k$) ⟶ |*a*| ≤ *c1* + *c2* ∗

$(\mathit{real}\ k)^{?d\ -\ 1}$ is to check the condition $\exists\, x.\ \mathit{poly}\ (\mathit{map\text{-}poly}\ \mathit{complex\text{-}of\text{-}real}\ f_i)\ x = 0 \wedge \mathit{cmod}\ x = 1$. Here, there are at least three possibilities. First, one can just ignore this precondition and weaken the statement. Second, one can apply Sturm's theorem to determine whether all roots are real. This can be done by comparing the number of distinct real roots with the degree of f_i, since f_i is square-free. If all roots are real, then one can decide the criterion by checking the only two possible real roots with norm equal to 1, namely 1 and -1. If on the other hand there are complex roots, then we loose precision at this point. Third, one uses a factorization algorithm (e.g., via complex algebraic numbers) to precisely determine the complex roots and decide the condition.

The second approach is illustrated in the following theorem. Note that all preconditions – including the ones from the context – can easily be checked with the help of Sturm's method. This method is used as a fast approximative technique in CeTA [3]. Only if the desired degree cannot be ensured by this method, the more costly complex algebraic number based factorization is applied.

lemma *perron-frobenius-spectral-radius-yun-real-roots*:
  assumes *bnd*: $\bigwedge f_i\ i.\ (f_i,i) \in \mathit{set\ fis}$
    $\Longrightarrow \mathit{card}\ \{\ x.\ \mathit{poly}\ f_i\ x = 0\} \neq \mathit{degree}\ f_i \vee \mathit{poly}\ f_i\ 1 = 0 \vee \mathit{poly}\ f_i\ (-1) = 0$
    $\Longrightarrow \mathit{Suc}\ i \leq d$
  shows $\exists\, c1\ c2.\ \forall\, k\ a.\ a \in \mathit{elements\text{-}mat}\ (A\ \widehat{}_m\ k) \longrightarrow \mathit{abs}\ a \leq (c1 + c2 * \mathit{real}\ k\ \widehat{}\ (d - 1))$
proof (*rule perron-frobenius-spectral-radius-yun*)
  fix *fi i*
  let *?cr* = *complex-of-real*
  let *?cp* = *map-poly ?cr*
  assume *fi*: (*fi*, *i*) ∈ *set fis*
    and $\exists\ x.\ \mathit{poly}\ (\mathit{map\text{-}poly}\ ?cr\ fi)\ x = 0 \wedge \mathit{norm}\ x = 1$
  then obtain *x* **where** *rt*: $\mathit{poly}\ (?cp\ fi)\ x = 0$ **and** *x*: $\mathit{norm}\ x = 1$ **by** *auto*
  show $\mathit{Suc}\ i \leq d$
  proof (*rule bnd*[*OF fi*])
    consider $(c)\ x \notin \mathbb{R} \mid (1)\ x = 1 \mid (m1)\ x = -1 \mid (r)\ x \in \mathbb{R}\ x \notin \{1, -1\}$
      by (*cases* $x \in \mathbb{R}$; *auto*)
    thus $\mathit{card}\ \{x.\ \mathit{poly}\ fi\ x = 0\} \neq \mathit{degree}\ fi \vee \mathit{poly}\ fi\ 1 = 0 \vee \mathit{poly}\ fi\ (-\ 1) = 0$
    proof (*cases*)
      case *1*
      from *rt* **have** $\mathit{poly}\ fi\ 1 = 0$
        unfolding *1* **by** *simp*
      thus *?thesis* **by** *simp*
    next
      case *m1*
      have *id*: $-1 = ?cr\ (-1)$ **by** *simp*
      from *rt* **have** $\mathit{poly}\ fi\ (-1) = 0$
        unfolding *m1 id of-real-hom.hom-zero*[**where** *'a=complex,symmetric*] *of-real-hom.poly-map-poly* **by** *simp*
      thus *?thesis* **by** *simp*

```
    next
      case r
      then obtain y where xy: x = of-real y unfolding Reals-def by auto
      from r(2)[unfolded xy] have y: y ∉ {1,−1} by auto
      from x[unfolded xy] have abs y = 1 by auto
      with y have False by auto
      thus ?thesis ..
    next
      case c
      from yun-factorization(2)[OF yun] fi have monic fi by auto
      hence fi: ?cp fi ≠ 0 by auto
      hence fin: finite {x. poly (?cp fi) x = 0} by (rule poly-roots-finite)
      have ?cr ‘ {x. poly (?cp fi) (?cr x) = 0} ⊂ {x. poly (?cp fi) x = 0} (is ?l ⊂
?r)
      proof (rule, force)
        have x ∈ ?r using rt by auto
        moreover have x ∉ ?l using c unfolding Reals-def by auto
        ultimately show ?l ≠ ?r by blast
      qed
      from psubset-card-mono[OF fin this] have card ?l < card ?r .
      also have ... ≤ degree (?cp fi) by (rule poly-roots-degree[OF fi])
      also have ... = degree fi by simp
      also have ?l = ?cr ‘ {x. poly fi x = 0} by auto
      also have card ... = card {x. poly fi x = 0}
        by (rule card-image, auto simp: inj-on-def)
      finally have card {x. poly fi x = 0} ≠ degree fi by simp
      thus ?thesis by auto
    qed
  qed
qed

end

end
```

7 The Jordan Blocks of the Spectral Radius are Largest

Consider a non-negative real matrix, and consider any Jordan-block of any eigenvalues whose norm is the spectral radius. We prove that there is a Jordan block of the spectral radius which has the same size or is larger.

```
theory Spectral-Radius-Largest-Jordan-Block
imports
  Jordan-Normal-Form.Jordan-Normal-Form-Uniqueness
  Perron-Frobenius-General
  HOL−Real-Asymp.Real-Asymp
begin
```

lemma *poly-asymp-equiv*: $(\lambda x.\ poly\ p\ (real\ x)) \sim[at\text{-}top]\ (\lambda x.\ lead\text{-}coeff\ p * real\ x$ $\hat{}\,(degree\ p))$
proof $(cases\ degree\ p = 0)$
  case *False*
  hence *lc*: $lead\text{-}coeff\ p \neq 0$ **by** *auto*
  have *1*: $1 = (\sum n{\le}degree\ p.\ if\ n = degree\ p\ then\ (1 :: real)\ else\ 0)$ **by** *simp*
  from *False* **show** *?thesis*
  proof (*intro asymp-equivI′*, *unfold poly-altdef sum-divide-distrib*,
     subst 1 , *intro tendsto-sum*, *goal-cases*)
    case $(1\ n)$
    hence $n = degree\ p \lor n < degree\ p$ **by** *auto*
    thus *?case*
    proof
      assume $n = degree\ p$
      thus *?thesis* **using** *False lc*
        by (*simp*, *intro LIMSEQ-I exI*[*of - Suc 0*], *auto*)
    qed (*insert False lc*, *real-asymp*)
  qed
next
  case *True*
  then obtain *c* **where** *p*: $p = [:c:]$ **by** (*metis degree-eq-zeroE*)
  show *?thesis* **unfolding** *p* **by** *simp*
qed

lemma *sum-root-unity*: **fixes** $x :: 'a :: \{comm\text{-}ring, division\text{-}ring\}$
  assumes $x\hat{}n = 1$
  shows $sum\ (\lambda\ i.\ x\hat{}i)\ \{..< n\} = (if\ x = 1\ then\ of\text{-}nat\ n\ else\ 0)$
proof $(cases\ x = 1 \lor n = 0)$
  case *x*: *False*
  from *x* **obtain** *m* **where** *n*: $n = Suc\ m$ **by** (*cases n*, *auto*)
  have *id*: $\{..< n\} = \{0..m\}$ **unfolding** *n* **by** *auto*
  show *?thesis* **using** *assms x n* **unfolding** *id sum-gp* **by** (*auto simp*: *divide-inverse*)
qed *auto*

lemma *sum-root-unity-power-pos-implies-1* :
  assumes *sumpos*: $\bigwedge\ k.\ Re\ (sum\ (\lambda\ i.\ b\ i * x\ i\ \hat{}\ k)\ I) > 0$
  and *root-unity*: $\bigwedge\ i.\ i \in I \Longrightarrow \exists\ d.\ d \neq 0 \land x\ i\ \hat{}\ d = 1$
shows $1 \in x\ `\ I$
proof (*rule ccontr*)
  assume $\neg$ *?thesis*
  hence *x*: $i \in I \Longrightarrow x\ i \neq 1$ **for** *i* **by** *auto*
  from *sumpos*[*of 0*] **have** *I*: *finite I* $I \neq \{\}$
    using *sum.infinite* **by** *fastforce*+
  have $\forall\ i.\ \exists\ d.\ i \in I \longrightarrow d \neq 0 \land x\ i\ \hat{}\ d = 1$ **using** *root-unity* **by** *auto*
  from *choice*[*OF this*] **obtain** *d* **where** *d*: $\bigwedge\ i.\ i \in I \Longrightarrow d\ i \neq 0 \land x\ i\ \hat{}\,(d\ i)$ $= 1$ **by** *auto*
  define *D* **where** $D = prod\ d\ I$
  have *D0*: $0 < D$ **unfolding** *D-def*

by (*rule prod-pos*, *insert d*, *auto*)
have $0 < sum\ (\lambda\ k.\ Re\ (sum\ (\lambda\ i.\ b\ i * x\ i\ \hat{}\ k)\ I))\ \{..< D\}$
by (*rule sum-pos*[*OF - - sumpos*], *insert D0*, *auto*)
also have $\ldots = Re\ (sum\ (\lambda\ k.\ sum\ (\lambda\ i.\ b\ i * x\ i\ \hat{}\ k)\ I)\ \{..< D\})$ **by** *auto*
also have $sum\ (\lambda\ k.\ sum\ (\lambda\ i.\ b\ i * x\ i\ \hat{}\ k)\ I)\ \{..< D\}$
$= sum\ (\lambda\ i.\ sum\ (\lambda\ k.\ b\ i * x\ i\ \hat{}\ k)\ \{..< D\})\ I$ **by** (*rule sum.swap*)
also have $\ldots = sum\ (\lambda\ i.\ b\ i * sum\ (\lambda\ k.\ x\ i\ \hat{}\ k)\ \{..< D\})\ I$
by (*rule sum.cong*, *auto simp*: *sum-distrib-left*)
also have $\ldots = 0$
proof (*rule sum.neutral*, *intro ballI*)
fix *i*
assume *i*: $i \in I$
from *d*[*OF this*] *x*[*OF this*] **have** *d*: $d\ i \neq 0$ **and** *rt-unity*: $x\ i\ \hat{}\ d\ i = 1$
and *x*: $x\ i \neq 1$ **by** *auto*
have $\exists\ C.\ D = d\ i * C$ **unfolding** *D-def*
by (*subst prod.remove*[*of - i*], *insert i I*, *auto*)
then obtain *C* **where** *D*: $D = d\ i * C$ **by** *auto*
have *image*: $(\bigwedge x.\ f\ x = x) \Longrightarrow \{..< D\} = f\ `\ \{..< D\}$ **for** *f* **by** *auto*
let $?g = (\lambda\ (a,c).\ a + d\ i * c)$
have $\{..< D\} = ?g\ `\ (\lambda\ j.\ (j\ mod\ d\ i,\ j\ div\ d\ i))\ `\ \{..< d\ i * C\}$
unfolding *image-image split D*[*symmetric*] **by** (*rule image*, *insert d mod-mult-div-eq*, *blast*)
also have $(\lambda\ j.\ (j\ mod\ d\ i,\ j\ div\ d\ i))\ `\ \{..< d\ i * C\} = \{..< d\ i\} \times \{..< C\}$ (**is** $?f\ `\ ?A = ?B$)
proof −
{
fix *x*
assume $x \in ?B$ **then obtain** *a c* **where** *x*: $x = (a,c)$ **and** *a*: $a < d\ i$ **and** *c*: $c < C$ **by** *auto*
hence $a + c * d\ i < d\ i * (1 + c)$ **by** *simp*
also have $\ldots \leq d\ i * C$ **by** (*rule mult-le-mono2*, *insert c*, *auto*)
finally have $a + c * d\ i \in ?A$ **by** *auto*
hence $?f\ (a + c * d\ i) \in ?f\ `\ ?A$ **by** *blast*
also have $?f\ (a + c * d\ i) = x$ **unfolding** *x* **using** *a* **by** *auto*
finally have $x \in ?f\ `\ ?A$ **.**
}
thus *?thesis* **using** *d* **by** (*auto simp*: *div-lt-nat*)
qed
finally have *D*: $\{..< D\} = (\lambda\ (a,c).\ a + d\ i * c)\ `\ ?B$ **by** *auto*
have *inj*: *inj-on ?g ?B*
proof −
{
fix *a1 a2 c1 c2*
assume *id*: $?g\ (a1,c1) = ?g\ (a2,c2)$ **and** $*$: $(a1,c1) \in ?B\ (a2,c2) \in ?B$
from *arg-cong*[*OF id*, *of* $\lambda\ x.\ x\ div\ d\ i$] $*$ **have** *c*: $c1 = c2$ **by** *auto*
from *arg-cong*[*OF id*, *of* $\lambda\ x.\ x\ mod\ d\ i$] $*$ **have** *a*: $a1 = a2$ **by** *auto*
note *a c*
}
thus *?thesis* **by** (*smt SigmaE inj-onI*)

qed
have *sum* (λ *k*. *x i* ^ *k*) {..< *D*} = *sum* (λ (*a*,*c*). *x i* ^ (*a* + *d i* * *c*)) *?B*
unfolding *D* by (*subst sum.reindex*, *rule inj*, *auto intro*!: *sum.cong*)
also have ... = *sum* (λ (*a*,*c*). *x i* ^ *a*) *?B*
by (*rule sum.cong*, *auto simp*: *power-add power-mult rt-unity*)
also have ... = *0* unfolding *sum.cartesian-product*[*symmetric*] *sum.swap*[*of* - {..<*C*}]
by (*rule sum.neutral*, *intro ballI*, *subst sum-root-unity*[*OF rt-unity*], *insert x*, *auto*)
finally
show *b i* * *sum* (λ *k*. *x i* ^ *k*) {..< *D*} = *0* by *simp*
qed
finally show *False* by *simp*
qed

fun *j-to-jb-index* :: (*nat* × '*a*)*list* ⇒ *nat* ⇒ *nat* × *nat* where
j-to-jb-index ((*n*,*a*) # *n-as*) *i* = (*if i* < *n then* (*0*,*i*) *else*
let rec = *j-to-jb-index n-as* (*i* − *n*) *in* (*Suc* (*fst rec*), *snd rec*))

fun *jb-to-j-index* :: (*nat* × '*a*)*list* ⇒ *nat* × *nat* ⇒ *nat* where
jb-to-j-index n-as (*0*,*j*) = *j*
| *jb-to-j-index* ((*n*,-) # *n-as*) (*Suc i*, *j*) = *n* + *jb-to-j-index n-as* (*i*,*j*)

lemma *j-to-jb-index*: assumes *i* < *sum-list* (*map fst n-as*)
and *j* < *sum-list* (*map fst n-as*)
and *j-to-jb-index n-as i* = (*bi*, *li*)
and *j-to-jb-index n-as j* = (*bj*, *lj*)
and *n-as* ! *bj* = (*n*, *a*)
shows ((*jordan-matrix n-as*) $\widehat{\ }_m$ *r*) $$ (*i*,*j*) = (*if bi* = *bj then* ((*jordan-block n a*) $\widehat{\ }_m$ *r*) $$ (*li*, *lj*) *else 0*)
∧ (*bi* = *bj* ⟶ *li* < *n* ∧ *lj* < *n* ∧ *bj* < *length n-as* ∧ (*n*,*a*) ∈ *set n-as*)
unfolding *jordan-matrix-pow* using *assms*
proof (*induct n-as arbitrary*: *i j bi bj*)
case (*Cons mb n-as i j bi bj*)
obtain *m b* where *mb*: *mb* = (*m*,*b*) by *force*
note *Cons* = *Cons*[*unfolded mb*]
have [*simp*]: *dim-col* (*case x of* (*n*, *a*) ⇒ 1_m *n*) = *fst x* for *x* by (*cases x*, *auto*)
have [*simp*]: *dim-row* (*case x of* (*n*, *a*) ⇒ 1_m *n*) = *fst x* for *x* by (*cases x*, *auto*)
have [*simp*]: *dim-col* (*case x of* (*n*, *a*) ⇒ *jordan-block n a* $\widehat{\ }_m$ *r*) = *fst x* for *x* by (*cases x*, *auto*)
have [*simp*]: *dim-row* (*case x of* (*n*, *a*) ⇒ *jordan-block n a* $\widehat{\ }_m$ *r*) = *fst x* for *x* by (*cases x*, *auto*)
consider (*UL*) *i* < *m j* < *m* | (*UR*) *i* < *m j* ≥ *m* | (*LL*) *i* ≥ *m j* < *m*
| (*LR*) *i* ≥ *m j* ≥ *m* by *linarith*
thus *?case*
proof *cases*
case *UL*
with *Cons*(*2*−) show *?thesis* unfolding *mb* by (*auto simp*: *Let-def*)
next

case *UR*
with *Cons*(*2*−) **show** *?thesis* **unfolding** *mb* **by** (*auto simp*: *Let-def dim-diag-block-mat o-def*)
next
case *LL*
with *Cons*(*2*−) **show** *?thesis* **unfolding** *mb* **by** (*auto simp*: *Let-def dim-diag-block-mat o-def*)
next
case *LR*
let *?i* = *i* − *m*
let *?j* = *j* − *m*
from *LR Cons*(*2*−) **have** *bi*: *j-to-jb-index n-as ?i* = (*bi* − *1*, *li*) *bi* ≠ *0* **by** (*auto simp*: *Let-def*)
from *LR Cons*(*2*−) **have** *bj*: *j-to-jb-index n-as ?j* = (*bj* − *1*, *lj*) *bj* ≠ *0* **by** (*auto simp*: *Let-def*)
from *LR Cons*(*2*−) **have** *i*: *?i* < *sum-list* (*map fst n-as*) **by** *auto*
from *LR Cons*(*2*−) **have** *j*: *?j* < *sum-list* (*map fst n-as*) **by** *auto*
from *LR Cons*(*2*−) *bj*(*2*) **have** *nas*: *n-as* ! (*bj* − *1*) = (*n*, *a*) **by** (*cases bj*, *auto*)
from *bi*(*2*) *bj*(*2*) **have** *id*: (*bi* − *1* = *bj* − *1*) = (*bi* = *bj*) **by** *auto*
note *IH* = *Cons*(*1*)[*OF i j bi*(*1*) *bj*(*1*) *nas*, *unfolded id*]
have *id*: *diag-block-mat* (*map* (*λa*. *case a of* (*n*, *a*) ⇒ *jordan-block n a* $\widehat{\ }_m$ *r*) (*mb* # *n-as*)) $$ (*i*, *j*)
= *diag-block-mat* (*map* (*λa*. *case a of* (*n*, *a*) ⇒ *jordan-block n a* $\widehat{\ }_m$ *r*) *n-as*) $$ (*?i*, *?j*)
using *i j LR* **unfolding** *mb* **by** (*auto simp*: *Let-def dim-diag-block-mat o-def*)
show *?thesis* **using** *IH* **unfolding** *id* **by** *auto*
qed
qed *auto*

lemma *j-to-jb-index-rev*: **assumes** *j*: *j-to-jb-index n-as i* = (*bi*, *li*)
and *i*: *i* < *sum-list* (*map fst n-as*)
and *k*: *k* ≤ *li*
shows *li* ≤ *i* ∧ *j-to-jb-index n-as* (*i* − *k*) = (*bi*, *li* − *k*) ∧ (
j-to-jb-index n-as j = (*bi*,*li* − *k*) ⟶ *j* < *sum-list* (*map fst n-as*) ⟶ *j* = *i* − *k*)
using *j i*
proof (*induct n-as arbitrary*: *i bi j*)
case (*Cons mb n-as i bi j*)
obtain *m b* **where** *mb*: *mb* = (*m*,*b*) **by** *force*
note *Cons* = *Cons*[*unfolded mb*]
show *?case*
proof (*cases i* < *m*)
case *True*
thus *?thesis* **unfolding** *mb* **using** *Cons*(*2*−) **by** (*auto simp*: *Let-def*)
next
case *i-large*: *False*
let *?i* = *i* − *m*
have *i*: *?i* < *sum-list* (*map fst n-as*) **using** *Cons*(*2*−) *i-large* **by** *auto*
from *Cons*(*2*−) *i-large* **have** *j*: *j-to-jb-index n-as ?i* = (*bi* − *1*, *li*)

and *bi*: *bi* ≠ *0* **by** (*auto simp*: *Let-def*)
note *IH* = *Cons*(*1*)[*OF j i*]
from *IH* **have** *IH1*: *j-to-jb-index n-as* (*i* − *m* − *k*) = (*bi* − *1*, *li* − *k*) **and**
li: *li* ≤ *i* − *m* **by** *auto*
from *li* **have** *aim1*: *li* ≤ *i* **by** *auto*
from *li k i-large* **have** *i* − *k* ≥ *m* **by** *auto*
hence *aim2*: *j-to-jb-index* (*mb* # *n-as*) (*i* − *k*) = (*bi*, *li* − *k*)
using *IH1 bi* **by** (*auto simp*: *mb Let-def add.commute*)
{
assume ∗: *j-to-jb-index* (*mb* # *n-as*) *j* = (*bi*, *li* − *k*)
j < *sum-list* (*map fst* (*mb* # *n-as*))
from ∗ *bi* **have** *j*: *j* ≥ *m* **unfolding** *mb* **by** (*auto simp*: *Let-def split*: *if-splits*)
let *?j* = *j* − *m*
from *j* ∗ **have** *jj*: *?j* < *sum-list* (*map fst n-as*) **unfolding** *mb* **by** *auto*
from *j* ∗ **have** ∗∗: *j-to-jb-index n-as* (*j* − *m*) = (*bi* − *1*, *li* − *k*) **using** *bi mb*
by (*cases j-to-jb-index n-as* (*j* − *m*), *auto simp*: *Let-def*)
from *IH*[*of ?j*] *jj* ∗∗ **have** *j* − *m* = *i* − *m* − *k* **by** *auto*
with *j i-large k* **have** *j* = *i* − *k* **using** ‹*m* ≤ *i* − *k*› **by** *linarith*
} **note** *aim3* = *this*
show *?thesis* **using** *aim1 aim2 aim3* **by** *blast*
qed
qed *auto*

locale *spectral-radius-1-jnf-max* =
fixes *A* :: *real mat* **and** *n m* :: *nat* **and** *lam* :: *complex* **and** *n-as*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *jnf*: *jordan-nf* (*map-mat complex-of-real A*) *n-as*
and *mem*: (*m*, *lam*) ∈ *set n-as*
and *lam1*: *cmod lam* = *1*
and *sr1*: ⋀*x*. *poly* (*char-poly A*) *x* = *0* ⟹ *x* ≤ *1*
and *max-block*: ⋀ *k la*. (*k*,*la*) ∈ *set n-as* ⟹ *cmod la* ≤ *1* ∧ (*cmod la* = *1* ⟶
k ≤ *m*)
begin

lemma *n-as0*: *0* ∉ *fst* ‘ *set n-as*
using *jnf*[*unfolded jordan-nf-def*] **..**

lemma *m0*: *m* ≠ *0* **using** *mem n-as0* **by** *force*

abbreviation *cA* **where** *cA* ≡ *map-mat complex-of-real A*
abbreviation *J* **where** *J* ≡ *jordan-matrix n-as*

lemma *sim-A-J*: *similar-mat cA J*
using *jnf*[*unfolded jordan-nf-def*] **..**

lemma *sumlist-nf*: *sum-list* (*map fst n-as*) = *n*
proof −

have *sum-list* (*map fst n-as*) = *dim-row* (*jordan-matrix n-as*) **by** *simp*
also have ... = *dim-row cA* **using** *similar-matD*[*OF sim-A-J*] **by** *auto*
finally show *?thesis* **using** *A* **by** *auto*
qed

definition *p* :: *nat* ⇒ *real poly* **where**
p s = ($\prod$ *i* = *0..<s*. [: − *of-nat i* / *of-nat* (*s* − *i*), *1* / *of-nat* (*s* − *i*) :])

lemma *p-binom*: **assumes** *sk*: *s* ≤ *k*
shows *of-nat* (*k choose s*) = *poly* (*p s*) (*of-nat k*)
unfolding *binomial-altdef-of-nat*[*OF assms*] *p-def poly-prod*
by (*rule prod.cong*[*OF refl*], *insert sk*, *auto simp*: *field-simps*)

lemma *p-binom-complex*: **assumes** *sk*: *s* ≤ *k*
shows *of-nat* (*k choose s*) = *complex-of-real* (*poly* (*p s*) (*of-nat k*))
unfolding *p-binom*[*OF sk*, *symmetric*] **by** *simp*

lemma *deg-p*: *degree* (*p s*) = *s* **unfolding** *p-def*
by (*subst degree-prod-eq-sum-degree*, *auto*)

lemma *lead-coeff-p*: *lead-coeff* (*p s*) = ($\prod$ *i* = *0..<s*. *1* / (*of-nat s* − *of-nat i*))
unfolding *p-def lead-coeff-prod*
by (*rule prod.cong*[*OF refl*], *auto*)

lemma *lead-coeff-p-gt-0*: *lead-coeff* (*p s*) > *0* **unfolding** *lead-coeff-p*
by (*rule prod-pos*, *auto*)

definition *c* = *lead-coeff* (*p* (*m* − *1*))

lemma *c-gt-0*: *c* > *0* **unfolding** *c-def* **by** (*rule lead-coeff-p-gt-0*)
lemma *c0*: *c* ≠ *0* **using** *c-gt-0* **by** *auto*

definition *PP* **where** *PP* = (*SOME PP*. *similar-mat-wit cA J* (*fst PP*) (*snd PP*))

definition *P* **where** *P* = *fst PP*
definition *iP* **where** *iP* = *snd PP*

lemma *JNF*: *P* ∈ *carrier-mat n n iP* ∈ *carrier-mat n n J* ∈ *carrier-mat n n*
P ∗ *iP* = 1_m *n iP* ∗ *P* = 1_m *n cA* = *P* ∗ *J* ∗ *iP*
proof (*atomize* (*full*), *goal-cases*)
case *1*
have *n*: *n* = *dim-row cA* **using** *A* **by** *auto*
from *sim-A-J*[*unfolded similar-mat-def*] **obtain** *Q iQ*
where *similar-mat-wit cA J Q iQ* **by** *auto*
hence *similar-mat-wit cA J* (*fst* (*Q*,*iQ*)) (*snd* (*Q*,*iQ*)) **by** *auto*
hence *similar-mat-wit cA J P iP* **unfolding** *PP-def iP-def P-def*
by (*rule someI*)
from *similar-mat-witD*[*OF n this*]
show *?case* **by** *auto*

qed

definition C :: *nat set* **where**
$C = \{j \mid j\ bj\ lj\ nn\ la.\ j < n \wedge j\text{-}to\text{-}jb\text{-}index\ n\text{-}as\ j = (bj,\ lj)$
$\wedge\ n\text{-}as\ !\ bj = (nn,la) \wedge cmod\ la = 1 \wedge nn = m \wedge lj = nn - 1\}$

lemma *C-nonempty*: $C \neq \{\}$
proof –
from *split-list*[*OF mem*] **obtain** *as bs* **where** *n-as*: *n-as* = *as* @ (*m*,*lam*) # *bs*
by *auto*
let *?i* = *sum-list* (*map fst as*) + (*m* − *1*)
have *j-to-jb-index n-as ?i* = (*length as*, *m* − *1*)
unfolding *n-as* **by** (*induct as*, *insert m0*, *auto simp*: *Let-def*)
with *lam1* **have** *?i* ∈ *C* **unfolding** *C-def* **unfolding** *sumlist-nf*[*symmetric*] *n-as*
using *m0* **by** *auto*
thus *?thesis* **by** *blast*
qed

lemma *C-n*: $C \subseteq \{..<n\}$ **unfolding** *C-def* **by** *auto*

lemma *root-unity-cmod-1*: **assumes** *la*: *la* ∈ *snd* ‘ *set n-as* **and** *1*: *cmod la* = *1*
shows ∃ *d*. *d* ≠ *0* ∧ *la* ^ *d* = *1*
proof –
from *la* **obtain** *k* **where** *kla*: (*k*,*la*) ∈ *set n-as* **by** *force*
from *n-as0 kla* **have** *k0*: *k* ≠ *0* **by** *force*
from *split-list*[*OF kla*] **obtain** *as bs* **where** *nas*: *n-as* = *as* @ (*k*,*la*) # *bs* **by**
auto
have *rt*: *poly* (*char-poly cA*) *la* = *0* **using** *k0*
unfolding *jordan-nf-char-poly*[*OF jnf*] *nas poly-prod-list prod-list-zero-iff* **by**
auto
obtain *ks f* **where** *decomp*: *decompose-prod-root-unity* (*char-poly A*) = (*ks*, *f*)
by *force*
from *sumlist-nf*[*unfolded nas*] *k0* **have** *n0*: *n* ≠ *0* **by** *auto*
note *pf* = *perron-frobenius-for-complexity-jnf*(*1*,*7*)[*OF A n0 nonneg sr1 decomp*,
simplified]
from *pf*(*1*) *pf*(*2*)[*OF 1 rt*] **show** ∃ *d*. *d* ≠ *0* ∧ *la* ^ *d* = *1* **by** *metis*
qed

definition *d* **where** *d* = (*SOME d*. ∀ *la*. *la* ∈ *snd* ‘ *set n-as* ⟶ *cmod la* = *1* ⟶

d la ≠ *0* ∧ *la* ^ (*d la*) = *1*)

lemma *d*: **assumes** (*k*,*la*) ∈ *set n-as cmod la* = *1*
shows *la* ^ (*d la*) = *1* ∧ *d la* ≠ *0*
proof –
let *?P* = λ *d*. ∀ *la*. *la* ∈ *snd* ‘ *set n-as* ⟶ *cmod la* = *1* ⟶
d la ≠ *0* ∧ *la* ^ (*d la*) = *1*
from *root-unity-cmod-1* **have** ∀ *la*. ∃ *d*. *la* ∈ *snd* ‘ *set n-as* ⟶ *cmod la* = *1*
⟶

d ≠ *0* ∧ *la* ^ *d* = *1* **by** *blast*
from *choice*[*OF this*] **have** ∃ *d*. *?P d* **.**
from *someI-ex*[*OF this*] **have** *?P d* **unfolding** *d-def* **.**
from *this*[*rule-format*, *of la*, *OF* - *assms*(*2*)] *assms*(*1*) **show** *?thesis* **by** *force*
qed

definition *D* **where** *D* = *prod-list* (*map* (λ *na*. *if cmod* (*snd na*) = *1 then d* (*snd na*) *else 1*) *n-as*)

lemma *D0*: *D* ≠ *0* **unfolding** *D-def*
by (*unfold prod-list-zero-iff*, *insert d*, *force*)

definition *f* **where** *f off k* = *D* ∗ *k* + (*m*−*1*) + *off*

lemma *mono-f*: *strict-mono* (*f off*) **unfolding** *strict-mono-def f-def*
using *D0* **by** *auto*

definition *inv-op* **where** *inv-op off k* = *inverse* (*c* ∗ *real* (*f off k*) ^ (*m* − *1*))

lemma *limit-jordan-block*: **assumes** *kla*: (*k*, *la*) ∈ *set n-as*
and *ij*: *i* < *k j* < *k*
shows (λ*N*. (*jordan-block k la* $\widehat{\ }_m$ (*f off N*)) $$ (*i*, *j*) ∗ *inv-op off N*)
⟶ (*if i* = *0* ∧ *j* = *k* − *1* ∧ *cmod la* = *1* ∧ *k* = *m then la*^*off else 0*)
proof −
let *?c* = *of-nat* :: *nat* ⇒ *complex*
let *?r* = *of-nat* :: *nat* ⇒ *real*
let *?cr* = *complex-of-real*
from *ij* **have** *k0*: *k* ≠ *0* **by** *auto*
from *jordan-nf-char-poly*[*OF jnf*] **have** *cA*: *char-poly cA* = (∏ (*n*, *a*)←*n-as*. [:− *a*, *1*:] ^ *n*) **.**
from *degree-monic-char-poly*[*OF A*] **have** *degree* (*char-poly A*) = *n* **by** *auto*
have *deg*: *degree* (*char-poly cA*) = *n* **using** *A* **by** (*simp add*: *degree-monic-char-poly*)
from *this*[*unfolded cA*] **have** *n* = *degree* (∏ (*n*, *a*)←*n-as*. [:− *a*, *1*:] ^ *n*) **by** *auto*
also have . . . = *sum-list* (*map degree* (*map* (λ(*n*, *a*). [:− *a*, *1*:] ^ *n*) *n-as*))
by (*subst degree-prod-list-eq*, *auto*)
also have . . . = *sum-list* (*map fst n-as*)
by (*rule arg-cong*[*of* - - *sum-list*], *auto simp*: *degree-linear-power*)
finally have *sum*: *sum-list* (*map fst n-as*) = *n* **by** *auto*
with *split-list*[*OF kla*] *k0* **have** *n0*: *n* ≠ *0* **by** *auto*
obtain *ks small* **where** *decomp*: *decompose-prod-root-unity* (*char-poly A*) = (*ks*, *small*) **by** *force*
note *pf* = *perron-frobenius-for-complexity-jnf*[*OF A n0 nonneg sr1 decomp*]
define *ji* **where** *ji* = *j* − *i*
have *ji*: *j* − *i* = *ji* **unfolding** *ji-def* **by** *auto*
let *?f* = λ *N*. *c* ∗ (*?r N*)^(*m*−*1*)
let *?jb* = λ *N*. (*jordan-block k la* $\widehat{\ }_m$ *N*) $$ (*i*,*j*)
let *?jbc* = λ *N*. (*jordan-block k la* $\widehat{\ }_m$ *N*) $$ (*i*,*j*) / *?f N*
define *e* **where** *e* = (*if i* = *0* ∧ *j* = *k* − *1* ∧ *cmod la* = *1* ∧ *k* = *m then la*^*off else 0*)

let *?e1* = λ *N* :: *nat*. *?cr* (*poly* (*p* (*j* − *i*)) (*?r N*)) ∗ *la* ^ (*N* + *i* − *j*)
let *?e2* = λ *N*. *?cr* (*poly* (*p ji*) (*?r N*) / *?f N*) ∗ *la* ^ (*N* + *i* − *j*)
define *e2* **where** *e2* = *?e2*
let *?e3* = λ *N*. *poly* (*p ji*) (*?r N*) / (*c* ∗ *?r N* ^ (*m* − *1*)) ∗ *cmod la* ^ (*N* + *i* − *j*)
define *e3* **where** *e3* = *?e3*
define *e3'* **where** *e3'* = (λ *N*. (*lead-coeff* (*p ji*) ∗ (*?r N*) ^ *ji*) / (*c* ∗ *?r N* ^ (*m* − *1*)) ∗ *cmod la* ^ (*N* + *i* − *j*))
{
assume *ij'*: *i* ≤ *j* **and** *la0*: *la* ≠ *0*
{
fix *N*
assume *N* ≥ *k*
with *ij ij'* **have** *ji*: *j* − *i* ≤ *N* **and** *id*: *N* + *i* − *j* = *N* − *ji* **unfolding** *ji-def* **by** *auto*
have *?jb N* = (*?c* (*N choose* (*j* − *i*)) ∗ *la* ^ (*N* + *i* − *j*))
unfolding *jordan-block-pow* **using** *ij ij'* **by** *auto*
also have . . . = *?e1 N* **by** (*subst p-binom-complex*[*OF ji*], *auto*)
finally have *id*: *?jb N* = *?e1 N* **.**
have *?jbc N* = *e2 N*
unfolding *id e2-def ji-def* **using** *c-gt-0* **by** (*simp add*: *norm-mult norm-divide norm-power*)
} **note** *jbc* = *this*
have *cmod-e2-e3*: (λ *n*. *cmod* (*e2 n*)) ∼[*at-top*] *e3*
proof (*intro asymp-equivI LIMSEQ-I exI*[*of - ji*] *allI impI*)
fix *n r*
assume *n*: *n* ≥ *ji*
have *cmod* (*e2 n*) = |*poly* (*p ji*) (*?r n*) / (*c* ∗ *?r n* ^ (*m* − *1*))| ∗ *cmod la* ^ (*n* + *i* − *j*)
unfolding *e2-def norm-mult norm-power norm-of-real* **by** *simp*
also have |*poly* (*p ji*) (*?r n*) / (*c* ∗ *?r n* ^ (*m* − *1*))| = *poly* (*p ji*) (*?r n*) / (*c* ∗ *real n* ^ (*m* − *1*))
by (*intro abs-of-nonneg divide-nonneg-nonneg mult-nonneg-nonneg*, *insert c-gt-0*, *auto simp*: *p-binom*[*OF n*, *symmetric*])
finally have *cmod* (*e2 n*) = *e3 n* **unfolding** *e3-def* **by** *auto*
thus *r* > *0* ⟹ *norm* ((*if cmod* (*e2 n*) = *0* ∧ *e3 n* = *0 then 1 else cmod* (*e2 n*) / *e3 n*) − *1*) < *r* **by** *simp*
qed
have *e3'*: *e3* ∼[*at-top*] *e3'* **unfolding** *e3-def e3'-def*
by (*intro asymp-equiv-intros*, *insert poly-asymp-equiv*[*of p ji*], *unfold deg-p*)
{
assume *e3'* ⟶ *0*
hence *e3*: *e3* ⟶ *0* **using** *e3'* **by** (*meson tendsto-asymp-equiv-cong*)
have *e2* ⟶ *0*
by (*subst tendsto-norm-zero-iff*[*symmetric*], *subst tendsto-asymp-equiv-cong*[*OF cmod-e2-e3*], *rule e3*)
} **note** *e2-via-e3* = *this*

have (*e2 o f off*) ⟶ *e*

proof (*cases cmod la* = *1* ∧ *k* = *m* ∧ *i* = *0* ∧ *j* = *k* − *1*)
case *False*
then consider (*0*) *la* = *0* | (*small*) *la* ≠ *0 cmod la* < *1* |
(*medium*) *cmod la* = *1 k* < *m* ∨ *i* ≠ *0* ∨ *j* ≠ *k* − *1*
using *max-block*[*OF kla*] **by** *linarith*
hence *main*: *e2* ⟶ *e*
proof *cases*
case *0*
hence *e0*: *e* = *0* **unfolding** *e-def* **by** *auto*
show *?thesis* **unfolding** *e0 0 LIMSEQ-iff e2-def ji*
proof (*intro exI*[*of - Suc j*] *impI allI*, *goal-cases*)
case (*1 r n*) **thus** *?case* **by** (*cases n* + *i* − *j*, *auto*)
qed
next
case *small*
define *d* **where** *d* = *cmod la*
from *small* **have** *d*: *0* < *d d* < *1* **unfolding** *d-def* **by** *auto*
have *e0*: *e* = *0* **using** *small* **unfolding** *e-def* **by** *auto*
show *?thesis* **unfolding** *e0*
by (*intro e2-via-e3*, *unfold e3'-def d-def*[*symmetric*], *insert d c0*, *real-asymp*)
next
case *medium*
with *max-block*[*OF kla*] **have** *k* ≤ *m* **by** *auto*
with *ij medium* **have** *ji*: *ji* < *m* − *1* **unfolding** *ji-def* **by** *linarith*
have *e0*: *e* = *0* **using** *medium* **unfolding** *e-def* **by** *auto*
show *?thesis* **unfolding** *e0*
by (*intro e2-via-e3*, *unfold e3'-def medium power-one mult-1-right*, *insert ji c0*, *real-asymp*)
qed
show (*e2 o f off*) ⟶ *e*
by (*rule LIMSEQ-subseq-LIMSEQ*[*OF main mono-f*])
next
case *True*
hence *large*: *cmod la* = *1 k* = *m i* = *0 j* = *k* − *1* **by** *auto*
hence *e*: *e* = *la*^*off* **and** *ji*: *ji* = *m* − *1* **unfolding** *e-def ji-def* **by** *auto*
from *large k0* **have** *m0*: *m* ≥ *1* **by** *auto*
define *m1* **where** *m1* = *m* − *1*
have *id*: (*real* (*m* − *1*) − *real ia*) = *?r m* − *1* − *?r ia* **for** *ia* **using** *m0* **unfolding** *m1-def* **by** *auto*
define *q* **where** *q* = *p m1* − *monom c m1*
hence *pji*: *p ji* = *q* + *monom c m1* **unfolding** *q-def ji m1-def* **by** *simp*
let *?e4a* = λ *x*. (*complex-of-real* (*poly q* (*real x*) / (*c* ∗ *real x* ^ *m1*))) ∗ *la* ^ (*x* + *i* − *j*)
let *?e4b* = λ *x*. *la* ^ (*x* + *i* − *j*)
{
fix *x* :: *nat*
assume *x*: *x* ≠ *0*
have *e2 x* = *?e4a x* + *?e4b x*
unfolding *e2-def pji poly-add poly-monom m1-def*[*symmetric*] **using** *c0 x*

by (*simp add*: *field-simps*)
} **note** *e2-e4* = *this*
have *e2-e4*: $\forall_F$ *x in sequentially*. (*e2 o f off*) *x* = (*?e4a o f off*) *x* + (*?e4b o f off*) *x* **unfolding** *o-def*
by (*intro eventually-sequentiallyI*[*of Suc 0*], *rule e2-e4*, *insert D0*, *auto simp*: *f-def*)
have (*e2 o f off*) $\longrightarrow$ *0* + *e*
unfolding *tendsto-cong*[*OF e2-e4*]
proof (*rule tendsto-add*, *rule LIMSEQ-subseq-LIMSEQ*[*OF - mono-f*])
show *?e4a* $\longrightarrow$ *0*
proof (*subst tendsto-norm-zero-iff*[*symmetric*],
unfold norm-mult norm-power large power-one mult-1-right norm-divide norm-of-real
tendsto-rabs-zero-iff)
have *deg-q*: *degree q* $\leq$ *m1* **unfolding** *q-def* **using** *deg-p*[*of m1*]
by (*intro degree-diff-le degree-monom-le*, *auto*)
have *coeff-q-m1*: *coeff q m1* = *0* **unfolding** *q-def c-def m1-def*[*symmetric*] **using** *deg-p*[*of m1*] **by** *simp*
from *deg-q coeff-q-m1* **have** *deg*: *degree q* < *m1* $\vee$ *q* = *0* **by** *fastforce*
have *eq*: (λ*n*. *poly q* (*real n*) / (*c* ∗ *real n* ^ *m1*)) ∼[*at-top*]
(λ*n*. *lead-coeff q* ∗ *real n* ^ *degree q* / (*c* ∗ *real n* ^ *m1*))
by (*intro asymp-equiv-intros poly-asymp-equiv*)
show (λ*n*. *poly q* (*?r n*) / (*c* ∗ *?r n* ^ *m1*)) $\longrightarrow$ *0*
unfolding *tendsto-asymp-equiv-cong*[*OF eq*] **using** *deg*
by (*standard*, *insert c0*, *real-asymp*, *simp*)
qed
next
have *id*: *D* ∗ *x* + (*m* − *1*) + *off* + *i* − *j* = *D* ∗ *x* + *off* **for** *x*
unfolding *ji*[*symmetric*] *ji-def* **using** *ij'* **by** *auto*
from *d*[*OF kla large*(*1*)] **have** *1*: *la* ^ *d la* = *1* **by** *auto*
from *split-list*[*OF kla*] **obtain** *as bs* **where** *n-as*: *n-as* = *as* @ (*k*,*la*) # *bs* **by** *auto*
obtain *C* **where** *D*: *D* = *d la* ∗ *C* **unfolding** *D-def* **unfolding** *n-as* **using** *large* **by** *auto*
show (*?e4b o f off*) $\longrightarrow$ *e*
unfolding *e f-def o-def id*
unfolding *power-add power-mult D 1* **by** *auto*
qed
thus *?thesis* **by** *simp*
qed
also have ((*e2 o f off*) $\longrightarrow$ *e*) = ((*?jbc o f off*) $\longrightarrow$ *e*)
proof (*rule tendsto-cong*, *unfold eventually-at-top-linorder*, *rule exI*[*of - k*],
intro allI impI, *goal-cases*)
case (*1 n*)
from *mono-f*[*of off*] *1* **have** *f off n* $\geq$ *k* **using** *le-trans seq-suble* **by** *blast*
from *jbc*[*OF this*] **show** *?case* **by** (*simp add*: *o-def*)
qed
finally have (*?jbc o f off*) $\longrightarrow$ *e* **.**
} **note** *part1* = *this*

{
assume $i > j \vee la = 0$
hence e: $e = 0$ **and** jbn: $N \geq k \Longrightarrow$ *?jbc* $N = 0$ **for** N
unfolding *jordan-block-pow e-def* **using** *ij* **by** *auto*
have *?jbc* $\longrightarrow$ e **unfolding** *e LIMSEQ-iff* **by** (*intro exI*[*of - k*] *allI impI*, *subst jbn*, *auto*)
from *LIMSEQ-subseq-LIMSEQ*[*OF this mono-f*]
have (*?jbc o f off*) $\longrightarrow$ *e* **.**
} **note** *part2* = *this*
from *part1 part2* **have** (*?jbc o f off*) $\longrightarrow$ *e* **by** *linarith*
thus *?thesis* **unfolding** *e-def o-def inv-op-def* **by** (*simp add*: *field-simps*)
qed

definition *lambda* **where** *lambda i* = *snd* (*n-as* ! *fst* (*j-to-jb-index n-as i*))

lemma *cmod-lambda*: $i \in C \Longrightarrow$ *cmod* (*lambda i*) = *1*
unfolding *C-def lambda-def* **by** *auto*

lemma *R-lambda*: **assumes** *i*: $i \in C$
shows (*m*, *lambda i*) $\in$ *set n-as*
proof −
from *i*[*unfolded C-def*]
obtain *bi li la* **where** *i*: $i < n$ **and** *jb*: *j-to-jb-index n-as i* = (*bi*, *li*)
and *nth*: *n-as* ! *bi* = (*m*, *la*) **and** *cmod la* = *1* $\wedge$ *li* = *m* − *1* **by** *auto*
hence *lam*: *lambda i* = *la* **unfolding** *lambda-def* **by** *auto*
from *j-to-jb-index*[*of - n-as*, *unfolded sumlist-nf*, *OF i i jb jb nth*] *lam*
show *?thesis* **by** *auto*
qed

lemma *limit-jordan-matrix*: **assumes** *ij*: $i < n$ $j < n$
shows (λN. (J $\widehat{\ }_m$ (*f off N*)) \$\$ (*i*, *j*) $*$ *inv-op off N*)
$\longrightarrow$ (*if* $j \in C \wedge i = j - (m - 1)$ *then* (*lambda j*)^*off else 0*)
proof −
obtain *bi li* **where** *bi*: *j-to-jb-index n-as i* = (*bi*, *li*) **by** *force*
obtain *bj lj* **where** *bj*: *j-to-jb-index n-as j* = (*bj*, *lj*) **by** *force*
define *la* **where** *la* = *snd* (*n-as* ! *fst* (*j-to-jb-index n-as j*))
obtain *nn* **where** *nbj*: *n-as* ! *bj* = (*nn*,*la*) **unfolding** *la-def bj fst-conv* **by** (*metis prod.collapse*)
from *j-to-jb-index*[*OF ij*[*folded sumlist-nf*] *bi bj nbj*]
have *eq*: *bi* = *bj* $\Longrightarrow$ *li* < *nn* $\wedge$ *lj* < *nn* $\wedge$ *bj* < *length n-as* $\wedge$ (*nn*, *la*) $\in$ *set n-as* **and**
index: (J $\widehat{\ }_m$ *r*) \$\$ (*i*, *j*) =
(*if bi* = *bj then* (*jordan-block nn la* $\widehat{\ }_m$ *r*) \$\$ (*li*, *lj*) *else 0*) **for** *r*
by *auto*
note *index-rev* = *j-to-jb-index-rev*[*OF bj*, *unfolded sumlist-nf*, *OF ij*(*2*) *le-refl*]
show *?thesis*
proof (*cases bi* = *bj*)
case *False*
hence *id*: (*bi* = *bj*) = *False* **by** *auto*

```
    {
      assume j ∈ C i = j − (m − 1)
      from this[unfolded C-def] bj nbj have i = j − lj by auto
      from index-rev[folded this] bi False have False by auto
    }
    thus ?thesis unfolding index id if-False by auto
  next
    case True
    hence id: (bi = bj) = True by auto
   from eq[OF True] have eq: li < nn lj < nn (nn,la) ∈ set n-as bj < length n-as
by auto
    have (λN. (J ^m (f off N)) $$ (i, j) ∗ inv-op off N)
      ⟶ (if li = 0 ∧ lj = nn − 1 ∧ cmod la = 1 ∧ nn = m then la^off else 0)
      unfolding index id if-True using limit-jordan-block[OF eq(3,1,2)] .
    also have (li = 0 ∧ lj = nn − 1 ∧ cmod la = 1 ∧ nn = m) = (j ∈ C ∧ i =
j − (m − 1)) (is ?l = ?r)
    proof
      assume ?r
      hence j ∈ C ..
      from this[unfolded C-def] bj nbj
      have ∗: nn = m cmod la = 1 lj = nn − 1 by auto
      from ‹?r› ∗ have i = j − lj by auto
      with ∗ have li = 0 using index-rev bi by auto
      with ∗ show ?l by auto
    next
      assume ?l
      hence jI: j ∈ C using bj nbj ij by (auto simp: C-def)
      from ‹?l› have li = 0 by auto
      with index-rev[of i] bi ij(1) ‹?l› True
      have i = j − (m − 1) by auto
      with jI show ?r by auto
    qed
    finally show ?thesis unfolding la-def lambda-def .
  qed
qed

declare sumlist-nf[simp]

lemma A-power-P: cA ^m k ∗ P = P ∗ J ^m k
proof (induct k)
  case 0
  show ?case using A JNF by simp
next
  case (Suc k)
  have cA ^m Suc k ∗ P = cA ^m k ∗ cA ∗ P by simp
  also have . . . = cA ^m k ∗ (P ∗ J ∗ iP) ∗ P using JNF by simp
  also have . . . = (cA ^m k ∗ P) ∗ (J ∗ (iP ∗ P))
    using A JNF(1−3) by (simp add: assoc-mult-mat[of - n n - n - n])
  also have J ∗ (iP ∗ P) = J unfolding JNF using JNF by auto
```

finally show *?case* **unfolding** *Suc*
using *A JNF*$(1-3)$ **by** (*simp add*: *assoc-mult-mat*[*of - n n - n - n*])
qed

lemma *inv-op-nonneg*: *inv-op off k* $\geq$ *0* **unfolding** *inv-op-def* **using** *c-gt-0* **by** *auto*

lemma *P-nonzero-entry*: **assumes** *j*: $j < n$
shows $\exists\ i < n.\ P$ $$ $(i,j) \neq 0$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
hence *0*: $\bigwedge i.\ i < n \Longrightarrow P$ $$ $(i,j) = 0$ **by** *auto*
have $1 = (iP * P)$ $$ (j,j) **using** *j* **unfolding** *JNF* **by** *auto*
also have ... $= (\sum i = 0..<n.\ iP$ $$ $(j, i) * P$ $$ $(i, j))$
using *j JNF*$(1-2)$ **by** (*auto simp*: *scalar-prod-def*)
also have ... $= 0$ **by** (*rule sum.neutral*, *insert 0*, *auto*)
finally show *False* **by** *auto*
qed

definition *j* **where** $j = (SOME\ j.\ j \in C)$

lemma *j*: $j \in C$ **unfolding** *j-def* **using** *C-nonempty some-in-eq* **by** *blast*

lemma *j-n*: $j < n$ **using** *j* **unfolding** *C-def* **by** *auto*

definition $i = (SOME\ i.\ i < n \wedge P$ $$ $(i, j - (m - 1)) \neq 0)$

lemma *i*: $i < n$ **and** *P-ij0*: *P* $$ $(i, j - (m - 1)) \neq 0$
proof $-$
from *j-n* **have** *lt*: $j - (m - 1) < n$ **by** *auto*
show $i < n$ *P* $$ $(i, j - (m - 1)) \neq 0$
unfolding *i-def* **using** *someI-ex*[*OF P-nonzero-entry*[*OF lt*]] **by** *auto*
qed

definition $w = P *_v$ *unit-vec n j*

lemma *w*: $w \in$ *carrier-vec n* **using** *JNF* **unfolding** *w-def* **by** *auto*

definition $v =$ *map-vec cmod w*

lemma *v*: $v \in$ *carrier-vec n* **unfolding** *v-def* **using** *w* **by** *auto*

definition *u* **where** $u = iP *_v$ *map-vec of-real v*

lemma *u*: $u \in$ *carrier-vec n* **unfolding** *u-def* **using** *JNF*(*2*) *v* **by** *auto*

definition *a* **where** $a\ j = P$ $$ $(i, j - (m - 1)) * u$ $v *j* **for** *j*

lemma *main-step*: $0 < Re\ (\sum j{\in}C.\ a\ j * lambda\ j\ \widehat{}\ l)$

proof −
let *?c* = *complex-of-real*
let *?cv* = *map-vec ?c*
let *?cm* = *map-mat ?c*
let *?v* = *?cv v*
define *cc* **where**
cc = (λ *jj*. (($\sum$ *k* = *0*..<*n*. (*if k* = *jj* − (*m* − *1*) *then P* $$ (*i*, *k*) *else 0*)) ∗ *u* $*v jj*))
{
fix *off*
define *G* **where** *G* = (λ *k*. (*A* $\widehat{\ }_m$ *f off k* $*_v$ *v*) $*v i* ∗ *inv-op off k*)
define *F* **where** *F* = ($\sum$*j*∈*C*. *a j* ∗ *lambda j* ^ *off*)
{
fix *kk*
define *k* **where** *k* = *f off kk*
have ((*A* $\widehat{\ }_m$ *k*) $*_v$ *v*) $ *i* ∗ *inv-op off kk* = *Re* (*?c* (((*A* $\widehat{\ }_m$ *k*) $*_v$ *v*) $ *i* ∗ *inv-op off kk*)) **by** *simp*
also have *?c* (((*A* $\widehat{\ }_m$ *k*) $*_v$ *v*) $ *i* ∗ *inv-op off kk*) = *?cv* ((*A* $\widehat{\ }_m$ *k*) $*_v$ *v*) $ *i* ∗ *?c* (*inv-op off kk*)
using *i A* **by** *simp*
also have *?cv* ((*A* $\widehat{\ }_m$ *k*) $*_v$ *v*) = (*?cm* (*A* $\widehat{\ }_m$ *k*) $*_v$ *?v*) **using** *A*
by (*subst of-real-hom.mult-mat-vec-hom*[*OF* - *v*], *auto*)
also have ... = (*cA* $\widehat{\ }_m$ *k* $*_v$ *?v*)
by (*simp add*: *of-real-hom.mat-hom-pow*[*OF A*])
also have ... = (*cA* $\widehat{\ }_m$ *k* $*_v$ ((*P* ∗ *iP*) $*_v$ *?v*)) **unfolding** *JNF* **using** *v* **by** *auto*
also have ... = (*cA* $\widehat{\ }_m$ *k* $*_v$ (*P* $*_v$ *u*)) **unfolding** *u-def*
by (*subst assoc-mult-mat-vec*, *insert JNF v*, *auto*)
also have ... = (*P* ∗ *J* $\widehat{\ }_m$ *k* $*_v$ *u*) **unfolding** *A-power-P*[*symmetric*]
by (*subst assoc-mult-mat-vec*, *insert u JNF*(*1*) *A*, *auto*)
also have ... = (*P* $*_v$ (*J* $\widehat{\ }_m$ *k* $*_v$ *u*))
by (*rule assoc-mult-mat-vec*, *insert u JNF*(*1*) *A*, *auto*)
finally have (*A* $\widehat{\ }_m$ *k* $*_v$ *v*) $*v i* ∗ *inv-op off kk* = *Re* ((*P* $*_v$ (*J* $\widehat{\ }_m$ *k* $*_v$ *u*)) $ *i* ∗ *inv-op off kk*) **by** *simp*
also have ... = *Re* ($\sum$ *jj* = *0*..<*n*.
P $$ (*i*, *jj*) ∗ ($\sum$ *ia* = *0*..< *n*. (*J* $\widehat{\ }_m$ *k*) $$ (*jj*, *ia*) ∗ *u* $*v ia* ∗ *inv-op off kk*))
by (*subst index-mult-mat-vec*, *insert JNF*(*1*) *i u*, *auto simp*: *scalar-prod-def sum-distrib-right*[*symmetric*]
mult.assoc[*symmetric*])
finally have (*A* $\widehat{\ }_m$ *k* $*_v$ *v*) $*v i* ∗ *inv-op off kk* =
Re ($\sum$ *jj* = *0*..<*n*. *P* $$ (*i*, *jj*) ∗ ($\sum$ *ia* = *0*..<*n*. (*J* $\widehat{\ }_m$ *k*) $$ (*jj*, *ia*) ∗ *inv-op off kk* ∗ *u* $*v ia*))
unfolding *k-def*
by (*simp only*: *ac-simps*)
} **note** *A-to-u* = *this*
have *G* ⟶
Re ($\sum$ *jj* = *0*..<*n*. *P* $$ (*i*, *jj*) ∗
($\sum$ *ia* = *0*..<*n*. (*if ia* ∈ *C* ∧ *jj* = *ia* − (*m* − *1*) *then* (*lambda ia*)^*off else*

```
0) * u $v ia))
   unfolding A-to-u G-def
   by (intro tendsto-intros limit-jordan-matrix, auto)
  also have (∑ jj = 0..<n. P $$ (i, jj) *
      (∑ ia = 0..<n. (if ia ∈ C ∧ jj = ia − (m − 1) then (lambda ia)^off else
0) * u $v ia))
   = (∑ jj = 0..<n. (∑ ia ∈ C. (if ia ∈ C ∧ jj = ia − (m − 1) then P $$ (i,
jj) else 0) * ((lambda ia)^off * u $v ia)))
  by (rule sum.cong[OF refl], unfold sum-distrib-left, subst (2) sum.mono-neutral-left[of
{0..<n}],
     insert C-n, auto intro!: sum.cong)
  also have ... = (∑ ia ∈ C. (∑ jj = 0..<n. (if jj = ia − (m − 1) then P $$
(i, jj) else 0)) * ((lambda ia)^off * u $v ia))
   unfolding sum.swap[of - C] sum-distrib-right
   by (rule sum.cong[OF refl], auto)
  also have ... = (∑ ia ∈ C. cc ia * (lambda ia)^off) unfolding cc-def
   by (rule sum.cong[OF refl], simp)
  also have ... = F unfolding cc-def a-def F-def
   by (rule sum.cong[OF refl], insert C-n, auto)
  finally have tend3: G ———→ Re F .


  from j j-n have jR: j ∈ C and j: j < n by auto
  let ?exp = λ k. sum (λ ii. P $$ (i, ii) * (J ^m k) $$ (ii,j)) {..<n}
  define M where M = (λ k. cmod (?exp (f off k) * inv-op off k))
  {
   fix kk
   define k where k = f off kk
   define cAk where cAk = cA ^m k
   have cAk: cAk ∈ carrier-mat n n unfolding cAk-def using A by auto
   have ((A ^m k) *v v) $ i = ((map-mat cmod cAk) *v map-vec cmod w) $ i
    unfolding v-def[symmetric] cAk-def
    by (rule arg-cong[of - - λ x. (x *v v) $ i],
      unfold of-real-hom.mat-hom-pow[OF A, symmetric],
    insert nonneg-mat-power[OF A nonneg, of k], insert i j,
    auto simp: nonneg-mat-def elements-mat-def)
   also have ... ≥ cmod ((cAk *v w) $ i)
    by (subst (1 2) index-mult-mat-vec, insert i cAk w, auto simp: scalar-prod-def
    intro!: sum-norm-le norm-mult-ineq)
   also have cAk *v w = (cAk * P) *v unit-vec n j
    unfolding w-def using JNF cAk by simp
   also have ... = P *v (J ^m k *v unit-vec n j) unfolding cAk-def A-power-P
    using JNF by (subst assoc-mult-mat-vec[of - n n - n], auto)
   also have J ^m k *v unit-vec n j = col (J ^m k) j
    by (rule eq-vecI, insert j, auto)
   also have (P *v (col (J ^m k) j)) $ i = Matrix.row P i · col (J ^m k) j
    by (subst index-mult-mat-vec, insert i JNF, auto)
   also have ... = sum (λ ii. P $$ (i, ii) * (J ^m k) $$ (ii,j)) {..<n}
    unfolding scalar-prod-def by (rule sum.cong, insert i j JNF(1), auto)
```

finally have $(A\ \widehat{}_m\ k *_v\ v)$ $v $i \geq cmod\ (?exp\ k)$ **.**
from *mult-right-mono*[*OF this inv-op-nonneg*]
have $(A\ \widehat{}_m\ k *_v\ v)$ $v $i * inv\text{-}op\ off\ kk \geq cmod\ (?exp\ k * inv\text{-}op\ off\ kk)$
unfolding *norm-mult*
using *inv-op-nonneg* **by** *auto*
}
hence *ge*: $(A\ \widehat{}_m\ f\ off\ k *_v\ v)$ $v $i * inv\text{-}op\ off\ k \geq M\ k$ **for** *k* **unfolding** *M-def*
by *auto*
from *j* **have** *mem*: $j - (m - 1) \in \{..<n\}$ **by** *auto*
have $(\lambda\ k.\ ?exp\ (f\ off\ k) * inv\text{-}op\ off\ k) \longrightarrow$
$(\sum ii<n.\ P$ $$ $(i,\ ii) * (if\ j \in C \wedge ii = j - (m - 1)\ then\ lambda\ j\ \widehat{}\ off\ else$
$0))$
(**is** - $\longrightarrow$ *?sum*)
unfolding *sum-distrib-right mult.assoc*
by (*rule tendsto-sum*, *rule tendsto-mult*, *force*, *rule limit-jordan-matrix*[*OF* -
j], *auto*)
also have *?sum* = *P* $$ $(i,\ j - (m - 1)) * lambda\ j\ \widehat{}\ off$
by (*subst sum.remove*[*OF - mem*], *force*, *subst sum.neutral*, *insert jR*, *auto*)
finally have *tend1*: $(\lambda\ k.\ ?exp\ (f\ off\ k) * inv\text{-}op\ off\ k) \longrightarrow P$ $$ $(i,\ j - (m$
$- 1)) * lambda\ j\ \widehat{}\ off$ **.**
have *tend2*: $M \longrightarrow cmod\ (P$ $$ $(i,\ j - (m - 1)) * lambda\ j\ \widehat{}\ off)$ **unfolding**
M-def
by (*rule tendsto-norm*, *rule tend1*)
define *B* **where** $B = cmod\ (P$ $$ $(i,\ j - (m - 1)))\ /\ 2$
have *B*: $0 < B$ **unfolding** *B-def* **using** *P-ij0* **by** *auto*
{
from *P-ij0* **have** *0*: *P* $$ $(i,\ j - (m - 1)) \neq 0$ **by** *auto*
define *E* **where** $E = cmod\ (P$ $$ $(i,\ j - (m - 1)) * lambda\ j\ \widehat{}\ off)$
from *cmod-lambda*[*OF jR*] *0* **have** *E*: $E\ /\ 2 > 0$ **unfolding** *E-def* **by** *auto*
from *tend2*[*folded E-def*] **have** *tend2*: $M \longrightarrow E$ **.**
from *ge* **have** *ge*: $G\ k \geq M\ k$ **for** *k* **unfolding** *G-def* **.**
from *tend2*[*unfolded LIMSEQ-iff*, *rule-format*, *OF E*]
obtain k' **where** *diff*: $\bigwedge\ k.\ k \geq k' \Longrightarrow norm\ (M\ k - E) < E\ /\ 2$ **by** *auto*
{
fix *k*
assume $k \geq k'$
from *diff*[*OF this*] **have** *norm*: $norm\ (M\ k - E) < E\ /\ 2$ **.**
have $M\ k \geq 0$ **unfolding** *M-def* **by** *auto*
with *E norm* **have** $M\ k \geq E\ /\ 2$
by (*smt real-norm-def field-sum-of-halves*)
with *ge*[*of k*] *E* **have** $G\ k \geq E\ /\ 2$ **by** *auto*
also have $E\ /\ 2 = B$ **unfolding** *E-def B-def j norm-mult norm-power*
cmod-lambda[*OF jR*] **by** *auto*
finally have $G\ k \geq B$ **.**
}
hence $\exists\ k'.\ \forall\ k.\ k \geq k' \longrightarrow G\ k \geq B$ **by** *auto*
}
hence *Bound*: $\exists k'.\ \forall k{\geq}k'.\ B \leq G\ k$ **by** *auto*
from *tend3*[*unfolded LIMSEQ-iff*, *rule-format*, *of B / 2*] *B*

obtain *kk* **where** *kk*: $\bigwedge$ *k*. $k \geq kk \Longrightarrow$ *norm* $(G\ k - Re\ F) < B\ /\ 2$ **by** *auto*
from *Bound* **obtain** *kk'* **where** *kk'*: $\bigwedge$ *k*. $k \geq kk' \Longrightarrow B \leq G\ k$ **by** *auto*
define *k* **where** $k = max\ kk\ kk'$
with *kk kk'* **have** *1*: *norm* $(G\ k - Re\ F) < B\ /\ 2\ B \leq G\ k$ **by** *auto*
with *B* **have** $Re\ F > 0$ **by** (*smt real-norm-def field-sum-of-halves*)
}
thus *?thesis* **by** *blast*
qed

lemma *main-theorem*: $(m,\ 1) \in set\ n\text{-}as$
proof −
from *main-step* **have** *pos*: $0 < Re\ (\sum i{\in}C.\ a\ i * lambda\ i\ \widehat{}\ l)$ **for** *l* **by** *auto*
have $1 \in lambda$ ‘ *C*
proof (*rule sum-root-unity-power-pos-implies-1*[*of a lambda C*, *OF pos*])
fix *i*
assume $i \in C$
from *d*[*OF R-lambda*[*OF this*] *cmod-lambda*[*OF this*]]
show $\exists\, d.\ d \neq 0 \wedge lambda\ i\ \widehat{}\ d = 1$ **by** *auto*
qed
then obtain *i* **where** *i*: $i \in C$ **and** $lambda\ i = 1$ **by** *auto*
with *R-lambda*[*OF i*] **show** *?thesis* **by** *auto*
qed
end

lemma *nonneg-sr-1-largest-jb*:
assumes *nonneg*: *nonneg-mat A*
and *jnf*: *jordan-nf* (*map-mat complex-of-real A*) *n-as*
and *mem*: $(m,\ lam) \in set\ n\text{-}as$
and *lam1*: $cmod\ lam = 1$
and *sr1*: $\bigwedge x.\ poly\ (char\text{-}poly\ A)\ x = 0 \Longrightarrow x \leq 1$
shows $\exists\ M.\ M \geq m \wedge (M,1) \in set\ n\text{-}as$
proof −
note *jnf'* = *jnf*[*unfolded jordan-nf-def*]
from *jnf' similar-matD*[*OF jnf'*[*THEN conjunct2*]] **obtain** *n*
where *A*: $A \in carrier\text{-}mat\ n\ n$ **and** *n-as0*: $0 \notin fst$ ‘ *set n-as* **by** *auto*
let $?M = \{\ m.\ \exists\ lam.\ (m,lam) \in set\ n\text{-}as \wedge cmod\ lam = 1\}$
have *m*: $m \in\ ?M$ **using** *mem lam1* **by** *auto*
have *fin*: *finite ?M*
by (*rule finite-subset*[*OF - finite-set*[*of map fst n-as*]], *force*)
define *M* **where** $M = Max\ ?M$
have $M \in\ ?M$ **using** *fin m* **unfolding** *M-def* **using** *Max-in* **by** *blast*
then obtain *lambda* **where** *M*: $(M,lambda) \in set\ n\text{-}as\ cmod\ lambda = 1$ **by** *auto*
from *m fin* **have** *mM*: $m \leq M$ **unfolding** *M-def* **by** *simp*
interpret *spectral-radius-1-jnf-max A n M lambda*
proof (*unfold-locales*, *rule A*, *rule nonneg*, *rule jnf*, *rule M*, *rule M*, *rule sr1*)
fix *k la*
assume *kla*: $(k,\ la) \in set\ n\text{-}as$

with *fin* **have** *1*: *cmod la* = *1* ⟶ *k* ≤ *M* **unfolding** *M-def* **using** *Max-ge* **by** *blast*
obtain *ks f* **where** *decomp*: *decompose-prod-root-unity* (*char-poly A*) = (*ks*, *f*) **by** *force*
from *n-as0 kla* **have** *k0*: *k* ≠ *0* **by** *force*
let *?cA* = *map-mat complex-of-real A*
from *split-list*[*OF kla*] **obtain** *as bs* **where** *nas*: *n-as* = *as* @ (*k*,*la*) # *bs* **by** *auto*
have *rt*: *poly* (*char-poly ?cA*) *la* = *0* **using** *k0*
unfolding *jordan-nf-char-poly*[*OF jnf*] *nas poly-prod-list prod-list-zero-iff* **by** *auto*
have *sumlist-nf*: *sum-list* (*map fst n-as*) = *n*
proof −
have *sum-list* (*map fst n-as*) = *dim-row* (*jordan-matrix n-as*) **by** *simp*
also have ... = *dim-row ?cA* **using** *similar-matD*[*OF jnf'*[*THEN conjunct2*]] **by** *auto*
finally show *?thesis* **using** *A* **by** *auto*
qed
from *this*[*unfolded nas*] *k0* **have** *n0*: *n* ≠ *0* **by** *auto*
from *perron-frobenius-for-complexity-jnf*(*4*)[*OF A n0 nonneg sr1 decomp rt*]
have *cmod la* ≤ *1* **.**
with *1* **show** *cmod la* ≤ *1* ∧ (*cmod la* = *1* ⟶ *k* ≤ *M*) **by** *auto*
qed
from *main-theorem*
show *?thesis* **using** *mM* **by** *auto*
qed
hide-const(**open**) *spectral-radius*

lemma (**in** *ring-hom*) *hom-smult-mat*: mat_h (*a* $\cdot_m$ *A*) = *hom a* $\cdot_m$ mat_h *A*
by (*rule eq-matI*, *auto simp*: *hom-mult*)

lemma *root-char-poly-smult*: **fixes** *A* :: *complex mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *k*: *k* ≠ *0*
shows (*poly* (*char-poly* (*k* $\cdot_m$ *A*)) *x* = *0*) = (*poly* (*char-poly A*) (*x* / *k*) = *0*)
using *order-char-poly-smult*[*OF A k*, *of x*]
by (*metis A degree-0 degree-monic-char-poly monic-degree-0 order-root smult-carrier-mat*)

theorem *real-nonneg-mat-spectral-radius-largest-jordan-block*:
assumes *real-nonneg-mat A*
and *jordan-nf A n-as*
and (*m*, *lam*) ∈ *set n-as*
and *cmod lam* = *spectral-radius A*
shows ∃ *M* ≥ *m*. (*M*, *of-real* (*spectral-radius A*)) ∈ *set n-as*
proof −
from *similar-matD*[*OF assms*(*2*)[*unfolded jordan-nf-def*, *THEN conjunct2*]] **obtain** *n* **where**
A: *A* ∈ *carrier-mat n n* **by** *auto*
let *?c* = *complex-of-real*

define *B* **where** *B* = *map-mat Re A*
have *B*: *B* ∈ *carrier-mat n n* **unfolding** *B-def* **using** *A* **by** *auto*
have *AB*: *A* = *map-mat ?c B* **unfolding** *B-def* **using** *assms*(*1*)
by (*auto simp*: *real-nonneg-mat-def elements-mat-def*)
have *nonneg*: *nonneg-mat B* **using** *assms*(*1*) **unfolding** *AB*
by (*auto simp*: *real-nonneg-mat-def elements-mat-def nonneg-mat-def*)
let *?sr* = *spectral-radius A*
show *?thesis*
proof (*cases ?sr* = *0*)
case *False*
define *isr* **where** *isr* = *inverse ?sr*
let *?nas* = *map* (λ(*n*, *a*). (*n*, *?c isr* ∗ *a*)) *n-as*
from *False* **have** *isr0*: *isr* ≠ *0* **unfolding** *isr-def* **by** *auto*
hence *cisr0*: *?c isr* ≠ *0* **by** *auto*
from *False assms*(*4*) **have** *isr-pos*: *isr* > *0* **unfolding** *isr-def*
by (*smt norm-ge-zero positive-imp-inverse-positive*)
define *C* **where** *C* = *isr* $\cdot_m$ *B*
have *C*: *C* ∈ *carrier-mat n n* **using** *B* **unfolding** *C-def* **by** *auto*
have *BC*: *B* = *?sr* $\cdot_m$ *C* **using** *isr0* **unfolding** *C-def isr-def* **by** *auto*
have *nonneg*: *nonneg-mat C* **unfolding** *C-def* **using** *isr-pos nonneg*
unfolding *nonneg-mat-def elements-mat-def* **by** *auto*
from *jordan-nf-smult*[*OF assms*(*2*)[*unfolded AB*] *cisr0*]
have *jnf*: *jordan-nf* (*map-mat ?c C*) *?nas* **unfolding** *C-def* **by** (*auto simp*: *of-real-hom.hom-smult-mat*)
from *assms*(*3*) **have** *mem*: (*m*, *?c isr* ∗ *lam*) ∈ *set ?nas* **by** *auto*
have *1*: *cmod* (*?c isr* ∗ *lam*) = *1* **using** *False isr-pos* **unfolding** *isr-def norm-mult assms*(*4*)
by (*smt mult.commute norm-of-real right-inverse*)
{
fix *x*
have *B′*: *map-mat ?c B* ∈ *carrier-mat n n* **using** *B* **by** *auto*
assume *poly* (*char-poly C*) *x* = *0*
hence *poly* (*char-poly* (*map-mat ?c C*)) (*?c x*) = *0* **unfolding** *of-real-hom.char-poly-hom*[*OF C*] **by** *auto*
hence *poly* (*char-poly A*) (*?c x* / *?c isr*) = *0* **unfolding** *C-def of-real-hom.hom-smult-mat AB*
unfolding *root-char-poly-smult*[*OF B′ cisr0*] **.**
hence *eigenvalue A* (*?c x* / *?c isr*) **unfolding** *eigenvalue-root-char-poly*[*OF A*] **.**
hence *mem*: *cmod* (*?c x* / *?c isr*) ∈ *cmod* ‘ *spectrum A* **unfolding** *spectrum-def* **by** *auto*
from *Max-ge*[*OF finite-imageI this*]
have *cmod* (*?c x* / *?c isr*) ≤ *?sr* **unfolding** *Spectral-Radius.spectral-radius-def*
using *A card-finite-spectrum*(*1*) **by** *blast*
hence *cmod* (*?c x*) ≤ *1* **using** *isr0 isr-pos* **unfolding** *isr-def*
by (*auto simp*: *field-simps norm-divide norm-mult*)
hence *x* ≤ *1* **by** *auto*
} **note** *sr* = *this*
from *nonneg-sr-1-largest-jb*[*OF nonneg jnf mem 1 sr*] **obtain** *M* **where**

M: *M* ≥ *m* (*M*,*1*) ∈ *set ?nas* **by** *blast*
from *M*(*2*) **obtain** *a* **where** *mem*: (*M*,*a*) ∈ *set n-as* **and** *1* = *?c isr* ∗ *a* **by** *auto*
from *this*(*2*) **have** *a*: *a* = *?c ?sr* **using** *isr0* **unfolding** *isr-def* **by** (*auto simp*: *field-simps*)
show *?thesis*
by (*intro exI*[*of - M*], *insert mem a M*(*1*), *auto*)
next
case *True*
from *jordan-nf-root-char-poly*[*OF assms*(*2*,*3*)]
have *eigenvalue A lam* **unfolding** *eigenvalue-root-char-poly*[*OF A*] **.**
hence *cmod lam* ∈ *cmod ' spectrum A* **unfolding** *spectrum-def* **by** *auto*
from *Max-ge*[*OF finite-imageI this*]
have *cmod lam* ≤ *?sr* **unfolding** *Spectral-Radius.spectral-radius-def*
using *A card-finite-spectrum*(*1*) **by** *blast*
from *this*[*unfolded True*] **have** *lam0*: *lam* = *0* **by** *auto*
show *?thesis* **unfolding** *True* **using** *assms*(*3*)[*unfolded lam0*] **by** *auto*
qed
qed

end

8 Homomorphisms of Gauss-Jordan Elimination, Kernel and More

theory *Hom-Gauss-Jordan*
imports *Jordan-Normal-Form.Matrix-Kernel*
Jordan-Normal-Form.Jordan-Normal-Form-Uniqueness
begin

lemma (**in** *comm-ring-hom*) *similar-mat-wit-hom*: **assumes**
similar-mat-wit A B C D
shows *similar-mat-wit* (mat_h *A*) (mat_h *B*) (mat_h *C*) (mat_h *D*)
proof −
obtain *n* **where** *n*: *n* = *dim-row A* **by** *auto*
note ∗ = *similar-mat-witD*[*OF n assms*]
from ∗ **have** [*simp*]: *dim-row C* = *n* **by** *auto*
note *C* = ∗(*6*) **note** *D* = ∗(*7*)
note *id* = *mat-hom-mult*[*OF C D*] *mat-hom-mult*[*OF D C*]
note ∗∗ = ∗(*1*−*3*)[*THEN arg-cong*[*of - -* mat_h], *unfolded id*]
note *mult* = *mult-carrier-mat*[*of - n n*]
note *hom-mult* = *mat-hom-mult*[*of - n n - n*]
show *?thesis* **unfolding** *similar-mat-wit-def Let-def* **unfolding** ∗∗(*3*) **using** ∗∗(*1*,*2*)
by (*auto simp*: *n*[*symmetric*] *hom-mult simp*: ∗(*4*−) *mult*)
qed

lemma (**in** *comm-ring-hom*) *similar-mat-hom*:

similar-mat A B $\Longrightarrow$ *similar-mat* (*mat*$_h$ *A*) (*mat*$_h$ *B*)
using *similar-mat-wit-hom*[*of A B C D* **for** *C D*]
by (*smt similar-mat-def*)

context *field-hom*
begin
lemma *hom-swaprows*: *i* < *dim-row A* $\Longrightarrow$ *j* < *dim-row A* $\Longrightarrow$
swaprows i j (*mat*$_h$ *A*) = *mat*$_h$ (*swaprows i j A*)
unfolding *mat-swaprows-def* **by** (*rule eq-matI*, *auto*)

lemma *hom-gauss-jordan-main*: *A* ∈ *carrier-mat nr nc* $\Longrightarrow$ *B* ∈ *carrier-mat nr nc2* $\Longrightarrow$
gauss-jordan-main (*mat*$_h$ *A*) (*mat*$_h$ *B*) *i j* =
map-prod mat$_h$ *mat*$_h$ (*gauss-jordan-main A B i j*)
proof (*induct A B i j rule*: *gauss-jordan-main.induct*)
case (*1 A B i j*)
note *IH* = *1*(*1*−*4*)
note *AB* = *1*(*5*−*6*)
from *AB* **have** *dim*: *dim-row A* = *nr dim-col A* = *nc* **by** *auto*
let *?h* = *mat*$_h$
let *?hp* = *map-prod mat*$_h$ *mat*$_h$
show *?case* **unfolding** *gauss-jordan-main.simps*[*of A B i j*] *gauss-jordan-main.simps*[*of ?h A - i j*]
index-map-mat Let-def if-distrib[*of ?hp*] *dim*
proof (*rule if-cong*[*OF refl*], *goal-cases*)
case *1*
note *IH* = *IH*[*OF dim*[*symmetric*] *1 refl*]
from *1* **have** *ij*: *i* < *nr j* < *nc* **by** *auto*
hence *hij*: (*?h A*) $$ (*i*,*j*) = *hom* (*A* $$ (*i*,*j*)) **using** *AB* **by** *auto*
define *ixs* **where** *ixs* = *concat* (*map* (λ*i*′. *if A* $$ (*i*′, *j*) ≠ *0 then* [*i*′] *else* []) [*Suc i*..<*nr*])
have *id*: *map* (λ*i*′. *if mat*$_h$ *A* $$ (*i*′, *j*) ≠ *0 then* [*i*′] *else* []) [*Suc i*..<*nr*] =
map (λ*i*′. *if A* $$ (*i*′, *j*) ≠ *0 then* [*i*′] *else* []) [*Suc i*..<*nr*]
by (*rule map-cong*[*OF refl*], *insert ij AB*, *auto*)
show *?case* **unfolding** *hij hom-0-iff hom-1-iff id ixs-def*[*symmetric*]
proof (*rule if-cong*[*OF refl - if-cong*[*OF refl*]], *goal-cases*)
case *1*
note *IH* = *IH*(*1*,*2*)[*OF 1*, *folded ixs-def*]
show *?case*
proof (*cases ixs*)
case *Nil*
show *?thesis* **unfolding** *Nil* **using** *IH*(*1*)[*OF Nil AB*] **by** *auto*
next
case (*Cons I ix*)
hence *I* ∈ *set ixs* **by** *auto*
hence *I*: *I* < *nr* **unfolding** *ixs-def* **by** *auto*
from *AB* **have** *swap*: *swaprows i I A* ∈ *carrier-mat nr nc swaprows i I B* ∈ *carrier-mat nr nc2*
by *auto*

show *?thesis* **unfolding** *Cons list.simps IH*(*2*)[*OF Cons swap*,*symmetric*]
using *AB ij I*
by (*auto simp*: *hom-swaprows*)
qed
next
case *2*
from *AB* **have** *elim*: *eliminate-entries* (*λi. A* $$ (*i*, *j*)) *A i j* ∈ *carrier-mat nr nc*
eliminate-entries (*λi. A* $$ (*i*, *j*)) *B i j* ∈ *carrier-mat nr nc2*
unfolding *eliminate-entries-gen-def* **by** *auto*
show *?case* **unfolding** *IH*(*3*)[*OF 2 refl elim*, *symmetric*]
by (*rule arg-cong2*[*of - - - - λ x y. gauss-jordan-main x y* (*Suc i*) (*Suc j*)];
intro eq-matI, *insert AB ij*, *auto simp*: *eliminate-entries-gen-def hom-minus hom-mult*)
next
case *3*
from *AB* **have** *mult*: *multrow i* (*inverse* (*A* $$ (*i*, *j*))) *A* ∈ *carrier-mat nr nc*
multrow i (*inverse* (*A* $$ (*i*, *j*))) *B* ∈ *carrier-mat nr nc2* **by** *auto*
show *?case* **unfolding** *IH*(*4*)[*OF 3 refl mult*, *symmetric*]
by (*rule arg-cong2*[*of - - - - λ x y. gauss-jordan-main x y i j*];
intro eq-matI, *insert AB ij*, *auto simp*: *hom-inverse hom-mult*)
qed
qed *auto*
qed

lemma *hom-gauss-jordan*: *A* ∈ *carrier-mat nr nc* ⟹ *B* ∈ *carrier-mat nr nc2* ⟹
gauss-jordan (mat_h *A*) (mat_h *B*) = *map-prod* mat_h mat_h (*gauss-jordan A B*)
unfolding *gauss-jordan-def* **using** *hom-gauss-jordan-main* **by** *blast*

lemma *hom-gauss-jordan-single*[*simp*]: *gauss-jordan-single* (mat_h *A*) = mat_h (*gauss-jordan-single A*)
proof −
let *?nr* = *dim-row A* **let** *?nc* = *dim-col A*
have *0*: 0_m *?nr 0* ∈ *carrier-mat ?nr 0* **by** *auto*
have *dim*: *dim-row* (mat_h *A*) = *?nr* **by** *auto*
have *hom0*: mat_h (0_m *?nr 0*) = 0_m *?nr 0* **by** *auto*
have *A*: *A* ∈ *carrier-mat ?nr ?nc* **by** *auto*
from *hom-gauss-jordan*[*OF A 0*] *A*
show *?thesis* **unfolding** *gauss-jordan-single-def dim hom0* **by** (*metis fst-map-prod*)
qed

lemma *hom-pivot-positions-main-gen*: **assumes** *A*: *A* ∈ *carrier-mat nr nc*
shows *pivot-positions-main-gen 0* (mat_h *A*) *nr nc i j* = *pivot-positions-main-gen 0 A nr nc i j*
proof (*induct rule*: *pivot-positions-main-gen.induct*[*of nr nc A 0*])
case (*1 i j*)
note *IH* = *this*
show *?case* **unfolding** *pivot-positions-main-gen.simps*[*of - - nr nc i j*]
proof (*rule if-cong*[*OF refl if-cong*[*OF refl - refl*] *refl*], *goal-cases*)

case *1*
with *A* **have** *id*: $(mat_h\ A)\ \$\$\ (i,j) = hom\ (A\ \$\$\ (i,j))$ **by** *simp*
note *IH* = *IH*[*OF 1*]
show *?case* **unfolding** *id hom-0-iff*
by (*rule if-cong*[*OF refl IH*(*1*)], *force*, *subst IH*(*2*), *auto*)
qed
qed

lemma *hom-pivot-positions*[*simp*]: *pivot-positions* $(mat_h\ A)$ = *pivot-positions A*
unfolding *pivot-positions-def* **by** (*subst hom-pivot-positions-main-gen*, *auto*)

lemma *hom-kernel-dim*[*simp*]: *kernel-dim* $(mat_h\ A)$ = *kernel-dim A*
unfolding *kernel-dim-code* **by** *simp*

lemma *hom-char-matrix*: **assumes** *A*: $A \in$ *carrier-mat n n*
shows *char-matrix* $(mat_h\ A)$ (*hom x*) = mat_h (*char-matrix A x*)
unfolding *char-matrix-def*
by (*rule eq-matI*, *insert A*, *auto simp*: *hom-minus*)

lemma *hom-dim-gen-eigenspace*: **assumes** *A*: $A \in$ *carrier-mat n n*
shows *dim-gen-eigenspace* $(mat_h\ A)$ (*hom x*) = *dim-gen-eigenspace A x*
proof (*intro ext*)
fix *k*
show *dim-gen-eigenspace* $(mat_h\ A)$ (*hom x*) *k* = *dim-gen-eigenspace A x k*
unfolding *dim-gen-eigenspace-def hom-char-matrix*[*OF A*]
mat-hom-pow[*OF char-matrix-closed*[*OF A*], *symmetric*] **by** *simp*
qed
end
end

9 Combining Spectral Radius Theory with Perron Frobenius theorem

theory *Spectral-Radius-Theory-2*
imports
Spectral-Radius-Largest-Jordan-Block
Hom-Gauss-Jordan
begin

hide-const(**open**) *Coset.order*

lemma *jnf-complexity-generic*: **fixes** *A* :: *complex mat*
assumes *A*: $A \in$ *carrier-mat n n*
and *sr*: $\bigwedge x.$ *poly* (*char-poly A*) $x = 0 \Longrightarrow$ *cmod* $x \le 1$
and *1*: $\bigwedge x.$ *poly* (*char-poly A*) $x = 0 \Longrightarrow$ *cmod* $x = 1 \Longrightarrow$
order x (*char-poly A*) $> d + 1 \Longrightarrow$
($\forall$ *bsize* $\in$ *fst* ‘ *set* (*compute-set-of-jordan-blocks A x*). *bsize* $\le d + 1$)
shows $\exists$ *c1 c2*. $\forall$ *k*. *norm-bound* $(A\ \widehat{}_m\ k)$ (*c1* + *c2* $*$ *of-nat k* ^ *d*)

```
proof -
  from char-poly-factorized[OF A] obtain as where cA: char-poly A = (∏ a←as.
[:− a, 1:])
    and lenn: length as = n by auto
  from jordan-nf-exists[OF A cA] obtain n-xs where jnf: jordan-nf A n-xs ..
  have dd: x ^ d = x ^((d + 1) − 1) for x by simp
  show ?thesis unfolding dd
  proof (rule jordan-nf-matrix-poly-bound[OF A - - jnf])
    fix n x
    assume nx: (n,x) ∈ set n-xs
    from jordan-nf-block-size-order-bound[OF jnf nx]
    have no: n ≤ order x (char-poly A) by auto
    {
      assume 0 < n
      with no have order x (char-poly A) ≠ 0 by auto
      hence rt: poly (char-poly A) x = 0 unfolding order-root by auto
      from sr[OF this] show cmod x ≤ 1 .
      note rt
    } note sr = this
    assume c1: cmod x = 1
    show n ≤ d + 1
    proof (rule ccontr)
      assume ¬ n ≤ d + 1
      hence lt: n > d + 1 by auto
      with sr have rt: poly (char-poly A) x = 0 by auto
      from lt no have ord: d + 1 < order x (char-poly A) by auto
      from 1[OF rt c1 ord, unfolded compute-set-of-jordan-blocks[OF jnf]] nx lt
      show False by force
    qed
  qed
qed

lemma norm-bound-complex-to-real: fixes A :: real mat
  assumes A: A ∈ carrier-mat n n
    and bnd: ∃ c1 c2. ∀ k. norm-bound ((map-mat complex-of-real A) ^m k) (c1 +
c2 ∗ of-nat k ^ d)
  shows ∃ c1 c2. ∀ k a. a ∈ elements-mat (A ^m k) ⟶ abs a ≤ (c1 + c2 ∗ of-nat
k ^ d)
proof -
  let ?B = map-mat complex-of-real A
  from bnd obtain c1 c2 where nb: ⋀ k. norm-bound (?B ^m k) (c1 + c2 ∗ real
k ^ d) by auto
  show ?thesis
  proof (rule exI[of - c1], rule exI[of - c2], intro allI impI)
    fix k a
    assume a ∈ elements-mat (A ^m k)
    with pow-carrier-mat[OF A] obtain i j where a: a = (A ^m k) $$ (i,j) and
ij: i < n j < n
      unfolding elements-mat by force
```

from *ij nb*[*of k*] *A* **have** *norm* ((*?B* $\widehat{\ }_m$ *k*) $$ (*i*,*j*)) ≤ *c1* + *c2* ∗ *real k* ^ *d*
  unfolding *norm-bound-def* **by** *auto*
also have (*?B* $\widehat{\ }_m$ *k*) $$ (*i*,*j*) = *of-real a*
  unfolding *of-real-hom.mat-hom-pow*[*OF A*, *symmetric*] *a* **using** *ij A* **by** *auto*
also have *norm* (*complex-of-real a*) = *abs a* **by** *auto*
finally show *abs a* ≤ (*c1* + *c2* ∗ *real k* ^ *d*) **.**
qed
qed

lemma *dim-gen-eigenspace-max-jordan-block*: **assumes** *jnf*: *jordan-nf A n-as*
shows *dim-gen-eigenspace A l d* = *order l* (*char-poly A*) ⟷
(∀ *n*. (*n*,*l*) ∈ *set n-as* ⟶ *n* ≤ *d*)
proof −
let *?list* = [(*n*, *e*)←*n-as* . *e* = *l*]
define *list* **where** *list* = [*na*←*n-as* . *snd na* = *l*]
have *list*: *?list* = *list* **unfolding** *list-def* **by** (*induct n-as*, *force*+)
have *id*: (∀ *n*. (*n*, *l*) ∈ *set n-as* ⟶ *n* ≤ *d*) = (∀ *n* ∈ *set* (*map fst list*). *n* ≤ *d*)
unfolding *list-def* **by** *auto*
define *ns* **where** *ns* = *map fst list*
show *?thesis*
unfolding *dim-gen-eigenspace*[*OF jnf*] *jordan-nf-order*[*OF jnf*] *list list-def*[*symmetric*]
id
unfolding *ns-def*[*symmetric*]
proof (*induct ns*)
case (*Cons n ns*)
show *?case*
proof (*cases n* ≤ *d*)
case *True*
thus *?thesis* **using** *Cons* **by** *auto*
next
case *False*
hence *n* > *d* **by** *auto*
moreover have *sum-list* (*map* (*min d*) *ns*) ≤ *sum-list ns* **by** (*induct ns*, *auto*)
ultimately show *?thesis* **by** *auto*
qed
qed *auto*
qed

lemma *jnf-complexity-1-complex*: **fixes** *A* :: *complex mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *real-nonneg-mat A*
and *sr*: ⋀ *x*. *poly* (*char-poly A*) *x* = *0* ⟹ *cmod x* ≤ *1*
and *1*: *poly* (*char-poly A*) *1* = *0* ⟹
order 1 (*char-poly A*) > *d* + *1* ⟹
dim-gen-eigenspace A 1 (*d*+*1*) = *order 1* (*char-poly A*)
shows ∃ *c1 c2*. ∀ *k*. *norm-bound* (*A* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *of-nat k* ^ *d*)
proof −
from *char-poly-factorized*[*OF A*] **obtain** *as* **where** *cA*: *char-poly A* = (∏ *a*←*as*.
[:− *a*, *1*:])

and *lenn*: *length as* = *n* **by** *auto*
from *jordan-nf-exists*[*OF A cA*] **obtain** *n-as* **where** *jnf*: *jordan-nf A n-as* **..**
have *dd*: *x* ^ *d* = *x* ^((*d* + *1*) − *1*) **for** *x* **by** *simp*
let *?n* = *n*
show *?thesis* **unfolding** *dd*
proof (*rule jordan-nf-matrix-poly-bound*[*OF A - - jnf*])
fix *n a*
assume *na*: (*n*,*a*) ∈ *set n-as*
from *jordan-nf-root-char-poly*[*OF jnf na*]
have *rt*: *poly* (*char-poly A*) *a* = *0* **by** *auto*
with *degree-monic-char-poly*[*OF A*] **have** *n0*: *?n* > *0*
by (*cases ?n*, *auto dest*: *degree0-coeffs*)
from *sr*[*OF rt*] **show** *cmod a* ≤ *1* **.**
assume *a*: *cmod a* = *1*
from *rt* **have** *a* ∈ *spectrum A* **using** *A spectrum-root-char-poly* **by** *auto*
hence *11*: *1* ∈ *cmod* ' *spectrum A* **using** *a* **by** *auto*
note *spec* = *spectral-radius-mem-max*[*OF A n0*]
from *spec*(*2*)[*OF 11*] **have** *le*: *1* ≤ *spectral-radius A* **.**
from *spec*(*1*)[*unfolded spectrum-root-char-poly*[*OF A*]] *sr* **have** *spectral-radius A* ≤ *1* **by** *auto*
with *le* **have** *sr*: *spectral-radius A* = *1* **by** *auto*
show *n* ≤ *d* + *1*
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *nd*: *n* > *d* + *1* **by** *auto*
from *real-nonneg-mat-spectral-radius-largest-jordan-block*[*OF nonneg jnf na*, *unfolded sr a*]
obtain *N* **where** *N*: *N* ≥ *n* **and** *mem*: (*N*, *1*) ∈ *set n-as* **by** *auto*
from *jordan-nf-root-char-poly*[*OF jnf mem*] **have** *rt*: *poly* (*char-poly A*) *1* = *0* **.**
from *jordan-nf-block-size-order-bound*[*OF jnf mem*] **have** *N* ≤ *order 1* (*char-poly A*) **.**
with *N nd* **have** *d* + *1* < *order 1* (*char-poly A*) **by** *simp*
from *1*[*OF rt this*, *unfolded dim-gen-eigenspace-max-jordan-block*[*OF jnf*]] *mem N nd*
show *False* **by** *force*
qed
qed
qed

lemma *jnf-complexity-1-real*: **fixes** *A* :: *real mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *sr*: ⋀ *x*. *poly* (*char-poly A*) *x* = *0* ⟹ *x* ≤ *1*
and *jb*: *poly* (*char-poly A*) *1* = *0* ⟹
order 1 (*char-poly A*) > *d* + *1* ⟹
dim-gen-eigenspace A 1 (*d*+*1*) = *order 1* (*char-poly A*)
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (A ^$_m$ *k*) ⟶ |*a*| ≤ *c1* + *c2* ∗ *real k* ^ *d*
proof −

```
  let ?c = complex-of-real
  let ?A = map-mat ?c A
  have A': ?A ∈ carrier-mat n n using A by auto
  have nn: real-nonneg-mat ?A using nonneg A unfolding nonneg-mat-def real-nonneg-mat-def
    by (force simp: elements-mat)
  have 1: 1 = ?c 1 by auto
  note cp = of-real-hom.char-poly-hom[OF A]
  have hom: map-poly-inj-idom-divide-hom complex-of-real ..
  show ?thesis
  proof (rule norm-bound-complex-to-real[OF A jnf-complexity-1-complex[OF A'
nn]],
      unfold cp of-real-hom.poly-map-poly-1, unfold 1
      of-real-hom.hom-dim-gen-eigenspace[OF A]
      map-poly-inj-idom-divide-hom.order-hom[OF hom], goal-cases)
    case 2
    thus ?case using jb by auto
  next
    case (1 x)
    let ?cp = char-poly A
    assume rt: poly (map-poly ?c ?cp) x = 0
    with degree-monic-char-poly[OF A', unfolded cp] have n0: n ≠ 0
      using degree0-coeffs[of ?cp] by (cases n, auto)
    from perron-frobenius-nonneg[OF A nonneg n0]
    obtain sr ks f where sr0: 0 ≤ sr and ks: 0 ∉ set ks ks ≠ []
      and cp: ?cp = (∏ k←ks. monom 1 k − [:sr ^ k:]) ∗ f
      and rtf: poly (map-poly ?c f) x = 0 ⟹ cmod x < sr by auto
    have sr-rt: poly ?cp sr = 0 unfolding cp poly-prod-list-zero-iff poly-mult-zero-iff
using ks
      by (cases ks, auto simp: poly-monom)
    from sr[OF sr-rt] have sr1: sr ≤ 1 .
    interpret c: map-poly-comm-ring-hom ?c ..
    from rt[unfolded cp c.hom-mult c.hom-prod-list poly-mult-zero-iff poly-prod-list-zero-iff]
    show cmod x ≤ 1
    proof (standard, goal-cases)
      case 2
      with rtf sr1 show ?thesis by auto
    next
      case 1
      from this ks obtain p where p: p ∈ set ks
        and rt: poly (map-poly ?c (monom 1 p − [:sr ^ p:])) x = 0 by auto
      from p ks(1) have p: p ≠ 0 by metis
      from rt have x^p = (?c sr)^p unfolding c.hom-minus
        by (simp add: poly-monom of-real-hom.map-poly-pCons-hom)
      hence cmod x = cmod (?c sr) using p power-eq-imp-eq-norm by blast
      with sr0 sr1 show cmod x ≤ 1 by auto
    qed
  qed
```

qed
end

10 An efficient algorithm to compute the growth rate of A^n.

theory *Check-Matrix-Growth*
imports
  Spectral-Radius-Theory-2
  Sturm-Sequences.Sturm-Method
begin

hide-const (**open**) *Coset.order*

definition *check-matrix-complexity* :: *real mat* ⇒ *real poly* ⇒ *nat* ⇒ *bool* **where**
  check-matrix-complexity A cp d = (*count-roots-above cp 1* = *0*
    ∧ (*poly cp 1* = *0* ⟶ (*let ord* = *order 1 cp in*
      d + *1* < *ord* ⟶ *kernel-dim* ((*A* − 1_m (*dim-row A*)) $\widehat{}_m$ (*d* + *1*)) = *ord*)))

lemma *check-matrix-complexity*: **assumes** *A*: *A* ∈ *carrier-mat n n* **and** *nn*: *non-neg-mat A*
  and *check*: *check-matrix-complexity A* (*char-poly A*) *d*
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (*A* $\widehat{}_m$ *k*) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *of-nat k* ^ *d*)
proof (*rule jnf-complexity-1-real*[*OF A nn*])
  have *id*: *dim-gen-eigenspace A 1* (*d* + *1*) = *kernel-dim* ((*A* − 1_m (*dim-row A*)) $\widehat{}_m$ (*d* + *1*))
    unfolding *dim-gen-eigenspace-def*
   by (*rule arg-cong*[*of* - - λ *x*. *kernel-dim* (*x* $\widehat{}_m$ (*d* + *1*))], *unfold char-matrix-def*, *insert A*, *auto*)
  note *check* = *check*[*unfolded check-matrix-complexity-def*
     Let-def count-roots-above-correct, *folded id*]
  have *fin*: *finite* {*x*. *poly* (*char-poly A*) *x* = *0*}
    by (*rule poly-roots-finite*, *insert degree-monic-char-poly*[*OF A*], *auto*)
  from *check* **have** *card* {*x*. *1* < *x* ∧ *poly* (*char-poly A*) *x* = *0*} = *0* **by** *auto*
  from *this*[*unfolded card-eq-0-iff*] *fin*
  have {*x*. *1* < *x* ∧ *poly* (*char-poly A*) *x* = *0*} = {} **by** *auto*
  thus *poly* (*char-poly A*) *x* = *0* ⟹ *x* ≤ *1* **for** *x* **by** *force*
  assume *poly* (*char-poly A*) *1* = *0 d* + *1* < *order 1* (*char-poly A*)
  with *check* **show** *dim-gen-eigenspace A 1* (*d* + *1*) = *order 1* (*char-poly A*) **by** *auto*
qed
end

Acknowledgements We thank Fabian Immler for an introduction to continuity proving using HMA.

References

[1] O. Kunar and A. Popescu. From types to sets by local type definitions in higher-order logic. In *Proc. ITP 2016*. Springer, 2016. To appear.

[2] D. Serre. *Matrices: Theory and Applications*. Graduate texts in mathematics. Springer, 2002.

[3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs'09*, LNCS 5674, pages 452–468. Springer, 2009.

[4] R. Thiemann and A. Yamada. Formalizing Jordan normal forms in Isabelle/HOL. In *Proc. CPP 2016*, pages 88–99. ACM, 2016.