Perron-Frobenius Theorem for Spectral Radius
Analysis*

Jose Divasón, Ondřej Kunčar, René Thiemann and Akihisa Yamada

August 16, 2018

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

1 Introduction 2
2 Elimination of CARD('n) 3
3 Connecting HMA-matrices with JNF-matrices 4
   3.1 Bijections between index types of HMA and natural numbers 4
   3.2 Transfer rules to convert theorems from JNF to HMA and vice-versa . 8

*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 $\chi_A$ 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, 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 $\lambda$ 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 $\chi_A$. 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 $\chi_A$ has no real roots in the interval $(1, \infty)$.

This criterion is computationally far less expensive – one invocation of Sturm’s method on $\chi_A$ 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 $P[\text{CARD(’n)]}$ into $n! = 0 \Rightarrow P[n]$, 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: $n \neq 0 \Rightarrow \{0 \ldots < n :: \text{nat}\} \neq \{\}$ by auto
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.Numerical-Type
begin

lemma finite-set-to-list: \( \exists \, xs :: \text{‘a :: finite list}. \text{distinct} \, xs \wedge \text{set} \, xs = Y \)
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 exI1, rule i1)
    fix j
    assume i1: ?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

5
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 ∧ i < length (univ-list::'a list) by auto
show ?thesis unfolding to-nat
proof (rule someI2, rule i1)
fix x
assume x: ?ul ! x = a ∧ 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 ∧ n < length (univ-list::'a list)
  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 ∈ set (univ-list) unfolding univ-list by auto
  from this [unfolded set-cone-nth]
  obtain i where i1: univ-list ! i = a ∧ 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
  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 \Rightarrow 'a \ 'n \Rightarrow 'a \Rightarrow bool where
eigen-vector A v ev = (v \not= 0 \land A \ast v = ev \ast s v)
definition eigen-value :: 'a :: comm-ring-1 \ 'n \ 'n \Rightarrow 'a \Rightarrow bool where
eigen-value A k = (\exists v. eigen-vector A v k)
definition similar-matrix-wit
  :: 'a :: semiring-1 \ 'n \ 'n \Rightarrow 'a \ 'n \ 'n \Rightarrow 'a \ 'n \ 'n \Rightarrow 'a \ 'n \ 'n \Rightarrow bool
where
  similar-matrix-wit A B P Q = (P \ast\ast Q \equiv mat 1 \land Q \ast\ast P = mat 1 \land A = P \ast\ast B \ast\ast Q)
definition similar-matrix
  :: 'a :: semiring-1 \ 'n \ 'n \Rightarrow 'a \ 'n \ 'n \Rightarrow bool where
  similar-matrix A B = (\exists P Q. similar-matrix-wit A B P Q)
definition spectral-radius :: complex \ 'n \ 'n \Rightarrow real where
\[
\text{spectral-radius } A = \text{Max} \{ \text{norm } ev \mid v \text{ ev. eigen-vector } A \text{ } v \text{ ev} \}
\]

**definition** Spectrum :: \('a :: \text{field } \to 'n \to 'a) \Rightarrow 'a set where
Spectrum A = \text{Collect (eigen-value } A)\]

**definition** vec-elements-h :: \('a \to 'n \Rightarrow 'a) \Rightarrow 'a set where
vec-elements-h v = \text{range (vec-nth } v)\]

**lemma** vec-elements-h-def : vec-elements-h v = \{v \text{ } i \mid i. \text{True} \}
unfolding vec-elements-h-def by auto

**definition** elements-mat-h :: \('a \Rightarrow 'n \Rightarrow 'a) \Rightarrow 'a set where
elements-mat-h A = \text{range (λ (i,j). } A \text{ } h \text{ } i \text{ } h \text{ } j)\]

**lemma** elements-mat-h-def : elements-mat-h A = \{A \text{ } h \text{ } i \text{ } h \text{ } j \mid i j. \text{True} \}
unfolding elements-mat-h-def by auto

**definition** map-vector :: (\('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'b) \Rightarrow 'a set where
map-vector f v ≡ χ i. f (v \text{ } h \text{ } i)\]

**definition** map-matrix :: (\('a \Rightarrow 'b) \Rightarrow 'a \Rightarrow 'm \Rightarrow 'b \Rightarrow 'm) \Rightarrow 'a set where
map-matrix f A ≡ \chi i. map-vector f (A \text{ } h \text{ } i)\]

**definition** normbound :: \('a :: \text{real-normed-field } \to 'n \to 'a) \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool where
normbound A b ≡ ∀ x ∈ \text{elements-mat-h } A. \text{norm } x \leq b\]

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

**lemma** elements-mat : elements-mat A = \{A \text{ } h \text{ } i \text{ } h \text{ } j \mid i j. i < \text{dim-row } A \land j < \text{dim-col } A\}
unfolding elements-mat-def by force

**definition** vec-elements :: \('a Matrix.vec \Rightarrow 'a) \Rightarrow 'a set
where vec-elements v = \text{set [v } \text{ } i \text{ } i < - [0 ..< \text{dim-vec } v] \}

**lemma** vec-elements : vec-elements v = \{v \text{ } i \mid i. i < \text{dim-vec } v \}
unfolding vec-elements-def by auto

**context includes** vec.\text{lifting}
beginn
end

**definition** from-hma_v :: \('a \to 'n \Rightarrow 'a Matrix.vec \Rightarrow 'a) \Rightarrow 'a Matrix.vec where
from-hma_v v = Matrix.vec CARD('n) (λ i. v \text{ } h \text{ } from-nat } i)
definition from-hma_m :: 'a :: '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 = (λ i. v $v to-nat i)

definition to-hma_m :: 'a Matrix.mat ⇒ 'a :: 'nr ⇒ 'nc where
to-hma_m a = (λ i j. a $(to-nat i, to-nat j))

declare vec-lambda-eta

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 a) = a
  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 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 :: '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
HMA-V = (λ v w. v = from-hma_v w)

definition HMA-M :: 'a Matrix.mat ⇒ 'a :: 'nr ⇒ 'nc ⇒ bool
HMA-M = (λ a b. a = from-hma_m b)

definition HMA-I :: nat ⇒ 'n := finite ⇒ bool
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 )))

10
by (intro ext iffI, insert from-hma-to-hma \[\text{[symmetric]},\] auto simp: from-hma_v-def HMA-V-def)

lemma Domainp-HMA-M [transfer-domain-rule]:
Domainp (HMA-M :: 'a Matrix.mat ⇒ 'a ∘ 'nc ∘ 'nr ⇒ bool) = (λ A. A ∈ carrier-mat CARD('nr) CARD('nc))
by (intro ext iffI, insert from-hma-to-hma \[\text{[symmetric]},\] auto simp: from-hma_m-def HMA-M-def)

lemma Domainp-HMA-I [transfer-domain-rule]:
Domainp (HMA-I :: nat ⇒ ′n :: finite ⇒ bool) = (λ i. i < CARD('n)) (is \(\_\) \(\_\))
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)
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 ≡ A \$h\ i \$h\ j
lemma HMA-M-index [transfer-rule]:
(HMA-M ===> HMA-I ===> HMA-I ===> (\lambda 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-hma-m-def)

lemma HMA-V-0 [transfer-rule]: HMA-V (0 \cdot CARD('n)) (\theta :: 'a :: zero ^ 'n)
  unfolding HMA-V-def from-hma by auto

lemma HMA-M-0 [transfer-rule]:
HMA-M (0 \cdot CARD('nr) CARD('nc)) (\theta :: 'a :: zero ^ 'nc ^ 'nr)
  unfolding HMA-M-def from-hma by auto

lemma HMA-M-1 [transfer-rule]:
HMA-M (1 \cdot (CARD('n))) (\mathbf{mat} 1 :: \{zero,one\} ^ 'n^ 'n)
  unfolding HMA-M-def by (auto simp add: mat-def from-hma-m-def from-nat-inj)

lemma from-hma-v-add: from-hma_v v + from-hma_v w = from-hma_v (v + w)
  unfolding from-hma_v-def by auto

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

lemma from-hma_v-diff: from-hma_v v - from-hma_v w = from-hma_v (v - w)
  unfolding from-hma_v-def by auto

lemma HMA-V-diff [transfer-rule]: (HMA-V ===> HMA-V ===> HMA-V) (-) (-)
  unfolding rel-fun-def HMA-V-def
  by (auto simp: from-hma_v-diff)

lemma from-hma_m-add: from-hma_m a + from-hma_m b = from-hma_m (a + b)
  unfolding from-hma_m-def by auto

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

lemma from-hma_m-diff: from-hma_m a - from-hma_m b = from-hma_m (a - b)
  unfolding from-hma_m-def by auto

lemma HMA-M-diff [transfer-rule]: (HMA-M ===> HMA-M ===> HMA-M) (-) (-)
  unfolding rel-fun-def HMA-M-def
by (auto simp: from-hma_m-diff)

lemma scalar-product: fixes v :: 'a :: semiring-1 "^ n"
  shows scalar-prod (from-hma_v v) (from-hma_w 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 "^ nc "^ nr) ∈ carrier-mat (CARD('nr)) (CARD('nc))
  unfolding from-hma_m-def by simp-all

lemma [simp]:
  from-hma_v (y :: 'a "^ n) ∈ carrier-vec (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 ⇒ 'j ⇒ 'c) ⇒ 'c "^ j "^ i
  mk-mat f = (χ i j. f i j)

definition mk-vec :: ('i ⇒ 'c) ⇒ 'c "^ i
  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 ⇒ 'nc ⇒ 'a) ⇒ 'a "^ 'nc "^ 'nr))
  proof
  { fix x y i j
  assume id: ∀ (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)
(\lambda f. \text{Matrix.vec} (\text{CARD}(\text{\textasciitilde} n)) (\lambda i. f i))
(mk-vec :: (\text{\textasciitilde} n \Rightarrow 'a\textasciitilde\text{\textasciitilde} n))
proof -
  { fix x y i
    assume id: \forall (ya :: 'n). (x (to-nat ya) :: 'a) = y ya
    and i : i < \text{CARD}(\text{\textasciitilde} 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-mat-def HMA-V-def HMA-I-def from-hma_v-def by auto
qed

lemma mat-mult-scalar: A ** B = mk-mat (\lambda i j. \text{scalar-product} (\text{row} i A) (\text{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 (\lambda i. \text{scalar-product} (\text{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 \textasciitilde \text{\textasciitilde} nc \textasciitilde \text{\textasciitilde} nr) \Rightarrow (\text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} nr)
unfolding HMA-M-def by auto

lemma dim-col-transfer-rule:
HMA-M A (A' :: 'a \textasciitilde \text{\textasciitilde} nc \textasciitilde \text{\textasciitilde} nr) \Rightarrow (\text{\textasciitilde} \text{\textasciitilde} \text{\textasciitilde} nc)
unfolding HMA-M-def by auto

lemma HMA-M-mult [transfer-rule]: (HMA-M ==>) (HMA-M ==>) (HMA-M)
((\star)) ((\star\star))
proof -
  { fix A B :: 'a :: \text{semiring-1 mat} and A' :: 'a \textasciitilde \text{\textasciitilde} 'n \textasciitilde \text{\textasciitilde} nr and B' :: 'a \textasciitilde \text{\textasciitilde} nc \textasciitilde \text{\textasciitilde} 'n
    assume I[transfer-rule]: HMA-M A A' HMA-M B B'

14
\[\text{note [transfer-rule] } = \ dim-row-transfer-rule[\text{OF } 1(1)] \ dim-col-transfer-rule[\text{OF } 1(2)]\]

\begin{verbatim}
  have \(HMA-M (A \ast B) (A' \ast B')\)
  unfolding \(\text{times-mat-def mat-mult-scalar}\)
  by \(\text{transfer-prover-start, transfer-step+, transfer, auto}\)
}\)

thus \(\text{thesis by blast}\)

qed
\end{verbatim}

\begin{verbatim}
lemma \(HMA-V\text{-smult [transfer-rule]}\): \((\ast) \implies> HMA-V \implies> HMA-V\) \((\ast\ast)\)
unfolding \(\text{smult-vec-def}\)
unfolding \(\text{rel-fun-def HMA-V-def from-hma}\)-\(\text{v-def}\)
by \(\text{auto}\)
\end{verbatim}

\begin{verbatim}
lemma \(HMA-M\text{-mult-vec [transfer-rule]}\): \((HMA-M \implies> HMA-V \implies> HMA-V)\)
\((\ast\ast)\) \(\ast\ast\)\(\text{v}\)\(\text{v}'\)
proof −
{ fix \(A : \text{'}a : \text{'}\text{semiring-1 mat and } v : \text{'}a \text{Matrix.vec}\)
 and \(A' : \text{'}a \text{'}\text{nc} \ast \text{'}nr and } v' : \text{'}a \text{'}\text{nc}\)
 assume \(I[\text{transfer-rule]}: HMA-M A A' HMA-V v v'\)
 note \(\text{[transfer-rule]} = \text{dim-row-transfer-rule}\)
 have \(HMA-V (A \ast v) (A' \ast v')\)
 unfolding \(\text{mult-mat-vec-def mult-mat-vec-scalar}\)
 by \(\text{transfer-prover-start, transfer-step+, transfer, auto}\)
}

thus \(\text{thesis by blast}\)

qed
\end{verbatim}

\begin{verbatim}
lemma \(HMA-det [transfer-rule]\): \((HMA-M \implies> (=)) \text{Determinant.det}\)
\((\text{det} : \text{'}a : \text{'}\text{comm-ring-1 \text{'nc} \ast \text{'nr} \Rightarrow \text{'a})\)
proof −
{ fix \(a : \text{'}a \ast \text{'}\text{nc} \ast \text{'nr}\)
 let \(?tn = \text{to-nat} : \text{'}n : \text{finite} \Rightarrow \text{nat}\)
 let \(?fn = \text{from-nat} : \text{nat} \Rightarrow \text{'n}\)
 let \(?zn = \{0..< \text{\text{CARD('n)\}}\)
 let \(?U = \text{UNIV} : \text{'}n \text{set}\)
 let \(?p1 = \{p. p \text{permutes } ?zn\}\)
 let \(?p2 = \{p. p \text{permutes } ?U\}\)
 let \(?f = \lambda p i. \text{if } i \in ?U \text{ then } ?fn (p (\text{?tn i) else i}\)
 let \(?g = \lambda p i. \text{?fn (p (\text{?tn i))}\)
 have \(\text{fg} : \bigwedge a b c. (\text{if } a \in ?U \text{ then } b else c) = b \text{ by } auto\)
 have \(?p2 = \text{?f ' ?p1}\)
 by \(\text{rule permutes-bij', auto simp: to-nat-less-card to-nat-from-nat-id}\)
 hence \(id : ?p2 = ?g ' ?p1 \text{ by simp}\)
 have \(\text{inj-g: inj-on ?g ?p1}\)
\end{verbatim}

15
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)
next
  case False
  thus ?thesis using p q unfolding permutes-def by simp
qed
qed
have mult-cong: ![\forall a b c d. a = b \rightarrow c = d \rightarrow a \ast c = b \ast d] by simp
have sum (λ p.
  sign p * (∏ i∈?zn. a $\$h i $\$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
proof (induct rule: permutes-induct[OF fin - - p])
  case 1
  show ?case by (auto simp: sign-id[unfolded id-def] permutes-id[unfolded id-def])
next
  case (2 a b p)
  let ?sab = Fun.swap a b id
  let ?sfab = Fun.swap (?fn a) (?fn b) id
  have p-sab: permutation ?sab by (rule permutation-swap-id)
  have p-sfab: permutation ?sfab by (rule permutation-swap-id)
from 2(3) have IH1: \( p \) permutes \(?zn \) and IH2: sign \( p = \text{sign}(\?ft p) \) by auto

have sab-perm: \( \?sab \) permutes \(?zn \) using 2(1-2) by (rule permutes-swap-id)
from permutes-compose[OF IH1 this] have perm1: \( \?sab \circ p \) permutes \(?zn \).
from IH1 have p-p1: \( p \in \?p1 \) by simp
hence \( \?ft p \in \?ft \circ \?p1 \) by (rule imageI)
from this[folded id] have \( \?ft p \) permutes \(?U \) by simp
hence p-fp: permutation \( (?ft p) \) unfolding permutation-permutes by auto
{
  fix a b
  assume a: \( a \in \?zn \) and b: \( b \in \?zn \)
  hence \( (?fn a = ?fn b) = (a = b) \) using 2(1-2)
  by (auto simp: from-nat-inj)
}
note inj = this
from inj[OF 2(1-2)] have id2: sign \( ?sab \) = sign \( \?sab \) unfolding sign-swap-id by simp
have id: \( \?ft (\text{Fun}. \circ \?tn c) = \text{Fun}. \circ \?ft p \) proof
  fix c
  show \( \?ft (\text{Fun}. \circ \?tn c) = (\text{Fun}. \circ ?fn a) \circ \text{Fun}. \circ \?ft p \)
  proof (cases p \( (?tn c) = a \lor p \) \( (?tn c) = b \))
    case True
    thus ?thesis by (cases, auto simp add: o-def swap-def)
  next
    case False
    hence neq: \( p \) \( (?tn c) \neq a \lor p \) \( (?tn c) \neq b \) by auto
    have pc: \( p \in \?zn \) unfolding permutes-in-image[OF IH1]
    by (simp add: to-nat-less-card)
    from neq[folded inj[OF pc 2(1)]] inj[OF pc 2(2)]
    have \( ?fn (p \circ ?tn c) \neq ?fn a \circ ?fn (p \circ ?tn c) \neq ?fn b \).
    with neq show ?thesis by (auto simp: o-def swap-def)
  qed
  qed
  by (rule conjI[OF refl perm1])
  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]: \((=) \Longrightarrow HMA-M) (\lambda k. k \cdot m. I_m. \text{CARD}(\langle n \rangle))

\((\text{Finite-Cartesian-Product.mat :: 'a::semiring-1 \Rightarrow ('a^\langle n \rangle))} \)
unfolding \text{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 \Longrightarrow HMA-M) \Longrightarrow HMA-M) \quad (\lambda A B. A + \text{map-mat uminus B}) \quad ((-) :: 'a::group-add \Rightarrow ('a^\langle n \rangle))
unfolding \text{rel-fun-def HMA-M-def from-hma-def by auto}
definition mat2matofpoly where mat2matofpoly A = \(\chi i j. [\vdash : A \vdash i \vdash j :])$
definition charpoly where charpoly-def: charpoly A = det (\text{mat}(\text{monom 1 (Suc 0)}) - \text{mat2matofpoly} A)
definition erase-mat :: \(\langle n :: \text{finite} \Rightarrow 'a::\text{comm-monoid-add} \Rightarrow \langle n \rangle\rangle \Rightarrow \langle n \rangle \Rightarrow \langle n \rangle \Rightarrow \langle a \rangle \Rightarrow \langle n \rangle \Rightarrow \langle n \rangle \Rightarrow \langle a \rangle \where \quad \text{erase-mat} A i j = \(\chi \ i'. \chi \ j'. \text{if} \ i' = i \text{ or} \ j' = j \text{ then} 0 \text{ else} A \vdash i' \vdash j'$\)
definition sum-UNIV-type :: \(\langle n :: \text{finite} \Rightarrow \langle a \rangle \Rightarrow \langle n \rangle \Rightarrow \langle a \rangle \Rightarrow \langle n \rangle \Rightarrow \langle a \rangle \where \quad \text{sum-UNIV-type} f - = \text{sum} f \text{ UNIV}$
definition sum-UNIV-set :: \(\text{nat} \Rightarrow \langle a \rangle \Rightarrow \langle n \rangle \Rightarrow \langle n \rangle \Rightarrow \langle a \rangle \where \quad \text{sum-UNIV-set} f n = \text{sum} f \{..<n\}$
definition HMA-T :: \(\text{nat} \Rightarrow \langle n :: \text{finite} \Rightarrow \langle bool \rangle \Rightarrow \langle n \rangle \Rightarrow \langle bool \rangle \Rightarrow \langle a \rangle \text{ where} \quad \text{HMA-T} n - = (n = \text{CARD}(\langle n \rangle))$
lemma HMA-mat2matofpoly[transfer-rule]: \((HMA-M \Longrightarrow HMA-M) \quad (\lambda x. \text{map-mat} (\lambda a. ['[a:]]) x) \text{mat2matofpoly}$
unfolding \text{rel-fun-def HMA-M-def from-hma-def mat2matofpoly-def by auto}
lemma HMA-char-poly [transfer-rule]:
\((HMA-M :: (\langle a :: \text{comm-ring-1} \Rightarrow \langle a \rangle \Rightarrow \langle n \rangle \Rightarrow \langle bool \rangle)) \Longrightarrow \langle = \rangle) \quad \text{char-poly}$
proof -
{ 
  \fix A :: 'a \text{ mat and} \quad A' :: 'a^\langle n \rangle
  \assume \text{[transfer-rule]}: HMA-M A A'
  \hence \text{[simp]}: \text{dim-row} A = \text{CARD}(\langle n \rangle) \text{ by} \ (\text{simp add: HMA-M-def})
  \have \text{[simp]}: \text{monom 1 (Suc 0)} = :\emptyset, 1 :: 'a :]
  \by \text{[simp add: monom-Suc]}
  \have \text{[simp]}: \text{map-mat uminus} (\text{map-mat} (\lambda a. ['[a:]]) A) = \text{map-mat} (\lambda a. ['[-a:]])
  \by \text{[rule eq-matI, auto]}
  \have \text{char-poly} A = \text{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 = eigenvector A' v'
    unfolding eigenvector-def[abs-def] eigenvector-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) = (eigenvalue 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

\[ \Rightarrow \text{bool} \] \overset{=} {\Rightarrow} (=)

\text{elements-mat elements-mat-h}
\text{proof} -
\{ 
\text{fix } y :: \text{'a } \text{'}n c \text{'}nr \text{ and } i j :: \text{nat} 
\text{assume } i :: i < \text{CARD('nr) and } j :: j < \text{CARD('nc)} 
\text{hence } \text{from-hma_m y } \text{($) (i, j) } \in \text{ range (} \lambda(i, y) . y \text{ $h i $h ya)} 
\text{using } \text{to-nat-from-nat-id[OF i]} \text{ to-nat-from-nat-id[OF j] by (auto simp: from-hma_m-def)} 
\}
\text{moreover}
\{ 
\text{fix } y :: \text{'a } \text{'}n c \text{'}nr \text{ and } a 
\text{have } \exists i j . y \text{ $h a $h b = from-hma_m y } \text{($) (i, j) and } i < \text{CARD('nr) and } j < \text{CARD('nc)} 
\text{unfolding } \text{from-hma_m-def} 
\text{by (rule exI[of - Bij-Nat.to-nat a], rule exI[of - Bij-Nat.to-nat b], auto simp: to-nat-less-card)} 
\}
\text{ultimately show } ?\text{thesis} 
\text{unfolding } \text{elements-mat[abs-def] elements-mat-h-def[abs-def] HMA-M-def} 
\text{by auto} 
\text{qed}

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

\text{lemma } \text{norm-bound-elements-mat}:: \text{norm-bound A b } = (\forall x \in \text{elements-mat A. norm x } \leq b) 
\text{unfolding } \text{norm-bound-def elements-mat by auto}
lemma HMA-normbound [transfer-rule]:
\[(HMA-M :: 'a :: real-normed-field mat \Rightarrow 'a ^ 'nc ^ 'nr \Rightarrow bool) \Longrightarrow (=)\]
unfolding normbound_def[abs-def] norm-bound-elements-mat[abs-def]
by (transfer-prover)

lemma HMA-map-matrix [transfer-rule]:
\[(=) \Longrightarrow HMA-M \Longrightarrow HMA-M\]
from-hma_m-def
by auto

lemma HMA-transpose-matrix [transfer-rule]:
\[(HMA-M \Longrightarrow HMA-M)\]
unfolding transpose-mat-def transpose-def HMA-M-def from-hma_m-def
by auto

lemma HMA-map-vector [transfer-rule]:
\[(=) \Longrightarrow HMA-V \Longrightarrow HMA-V\]
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]:
\[(HMA-M :: - \Rightarrow 'a :: comm-ring-1 ^ 'n ^ 'n \Rightarrow -) \Longrightarrow HMA-M \Longrightarrow (=)\]
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]:
\[(HMA-M :: - \Rightarrow 'a :: comm-ring-1 ^ 'n ^ 'n \Rightarrow -) \Longrightarrow HMA-M \Longrightarrow (=)\]
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 i) = i for i by auto

have sum-UNIV-set f n = sum f (?t i) i < n by auto

unfolding sum-UNIV-set-def

by (rule arg-cong[of - - sum f], insert tf, force)

also have ... = sum (f o (?t)) (?f i) i < n by auto

also have ?f i i = UNIV using range-to-nat[where 'a = 'b, folded n] by force

also have sum (f o (?t)) UNIV = sum fT UNIV

proof (rule sum.cong[OF refl])

fix i :: 'b

show (f o (?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 :: ring-I 'nr '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 ←→ poly (charpoly (A :: 'a :: field 'n 'n)) k = 0  
by (transfer-hma rule: eigenvalue-root-char-poly)

lemma finite-spectrum: fixes A :: 'a :: field 'n 'n  
shows finite (Collect (eigen-value A))  
by (transfer-hma rule: card-finite-spectrum(1)[unfolded spectrum-def])

lemma non-empty-spectrum: fixes A :: complex 'n 'n  
shows Collect (eigen-value A) ≠ {}  
by (transfer-hma rule: spectrum-non-empty[unfolded spectrum-def])

lemma charpoly-transpose: charpoly (transpose A :: 'a :: field 'n 'n) = charpoly A  
by (transfer-hma rule: char-poly-transpose-mat)

lemma eigen-value-transpose: eigen-value (transpose A :: 'a :: field 'n '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 :: ring-I 'n)  
by (transfer-hma rule: minus-mult-distrib-mat-vec)

lemma similar-matrix-charpoly: similar-matrix A B ⇒ 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 $?A = \text{from-hma}_n A$
let $?n = \text{CARD}(\prime n)$
have $tA[\text{transfer-rule}]: HMA-M ?A A \text{ unfolding HMA-M-def by simp}$
have $t\i[\text{transfer-rule}]: HMA-T ?n \text{ TYPE}(\prime n) \text{ unfolding HMA-T-def by simp}$
have $A: \forall A \in \text{carrier-mat} ?n ?n \text{ unfolding from-hma_m-def by auto}$
have $id: \text{sum} (\lambda i. \text{charpoly} (\text{erase-mat} A i i)) UNIV = \text{sum-UNIV-type} (\lambda i. \text{charpoly} (\text{erase-mat} A i i)) \text{ TYPE}(\prime n)$

unfolding $\text{sum-UNIV-type-def}$ ..
show $?\text{thesis}$$\text{ unfolding id by (transfer, insert pderiv-char-poly-mat-erase[OF A], simp add: sum-UNIV-set-def)}$
qed

lemma $\text{degree-monic-charpoly}$: \textbf{fixes} $A :: \prime a :: \text{comm-ring-1} \times \prime n \times \prime n$
\textbf{shows} $\text{degree} (\text{charpoly} A) = \text{CARD}(\prime n) \land \text{monic} (\text{charpoly} A)$
proof (transfer, goal-cases)
case 1
from $\text{degree-monic-char-poly[OF 1]}$ show $?\text{case}$ 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 $\text{Perron-Frobenius-Aux}$
imports $\text{HMA-Connect}$
begin

definition $\text{real-non-neg-mat} :: \text{complex mat} \Rightarrow \text{bool}$ where
$\text{real-non-neg-mat} A \equiv \forall a \in \text{elements-mat} A. a \in \mathbb{R} \land \text{Re } a \geq 0$

definition $\text{real-non-neg-vec} :: \text{complex Matrix.vec} \Rightarrow \text{bool}$ where
$\text{real-non-neg-vec} v \equiv \forall a \in \text{vec-elements} v. a \in \mathbb{R} \land \text{Re } a \geq 0$

definition $\text{real-non-neg-vec} :: \text{complex } \prime n \Rightarrow \text{bool}$ where
$\text{real-non-neg-vec} v \equiv (\forall a \in \text{vec-elements-h} v. a \in \mathbb{R} \land \text{Re } a \geq 0)$

definition $\text{real-non-neg-mat} :: \text{complex } \prime nr \times \prime nc \Rightarrow \text{bool}$ where
$\text{real-non-neg-mat} A \equiv (\forall a \in \text{elements-mat-h} A. a \in \mathbb{R} \land \text{Re } a \geq 0)$

lemma $\text{real-non-neg-matD}$: \textbf{assumes} $\text{real-non-neg-mat} A$
shows $A \forall h i \forall h j \in \mathbb{R} \ Re (A \forall h i \forall h j) \geq 0$

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)

countext 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

countext 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 \(\text{Suc } n\)

note \([\text{transfer-rule}] = \text{this}\)

show \(?\text{case}\ \text{by } (\text{simp, transfer-prover})\)

qed \((\text{simp, transfer-prover})\)

\}

thus \(?\text{thesis}\ \text{by } \text{blast}\)

qed

\end

\textbf{lemma} \\texttt{trancl-image:}

\((i,j) \in R^+ \implies (f \ i, f \ j) \in (\text{map-prod } f \ f \cdot R)^+\)

\textbf{proof} \((\text{induct rule: trancl-induct})\)

case \((\text{step } j \ k)\)

from \(\text{step}(2)\) have \((f \ j, f \ k) \in \text{map-prod } f \ f \cdot R\) by \text{auto}

from \(\text{step}(3)\) this \show \(?\text{case}\ \text{by } \text{auto}\)

qed \text{auto}

\textbf{lemma} \\texttt{inj-trancl-image:} \assumes inj: inj f

\shows \((f \ i, f \ j) \in (\text{map-prod } f \ f \cdot R)^+ = ((i,j) \in R^+)\) \((\text{is } l = ?r)\)

\textbf{proof} \assumes ?r from \(\text{trancl-image}[\text{OF } \text{this}]\) \show \(?l\).

\next

\assumes ?l from \(\text{trancl-image}[\text{OF } \text{this}, \text{of the-inv } f]\)

\show \(?r\) unfolding \text{image-image prod.map-comp o-def the-inv-f-f}[\text{OF } \text{inj}] \text{by}\ \text{auto}

qed \text{auto}

\textbf{lemma} \\texttt{matrix-add-rdistrib:} \((B + C) ** A) = (B ** A) + (C ** A)\)

by \((\text{vector matrix-matrix-mult-def sum.distrib[symmetric] field-simps})\)

\textbf{lemma} \\texttt{norm-smult:} \text{norm } ((a :: real) *s x) = abs a * \text{norm } x

unfolding \text{norm-vec-def}

by \((\text{metis norm-scaleR norm-vec-def scalar-mult-eq-scaleR})\)

\textbf{lemma} \\texttt{nonnull-mat-mult:}

\assumes \((\text{nonnull-mat } A \implies \text{nonnull-mat } B \implies A \in \text{carrier-mat } nr \ n)\)

\implies B \in \text{carrier-mat } n \ nc \implies \text{nonnull-mat } (A \cdot B)\)

\textbf{proof} \((\text{auto simp: elements-mat-def scalar-prod-def intro!: sum-nonneg})\)

\textbf{lemma} \\texttt{nonnull-mat-power:} \assumes \((A \in \text{carrier-mat } n \ n \ \text{nonnull-mat } A)\)

\shows \((A ^\ m \ k)\)

\textbf{proof} \((\text{induct } k)\)

case \(\emptyset\)

thus \(?\text{case}\ \text{by } (\text{auto simp: nonnull-mat-def})\)

\next

case \((\text{Suc } k)\)

from \(\text{nonnull-mat-mult}[\text{OF } \text{this } \text{assms}(2) - \text{assms}(1), \text{of } \text{n}] \text{assms}(1)\)

\show \(?\text{case}\ \text{by } \text{auto}\)
qed

lemma nonneg-matD: assumes nonneg-mat A
  and $i < \text{dim-row } A$ and $j < \text{dim-col } A$
shows $A \begin{pmatrix} i, j \end{pmatrix} \geq 0$
  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 $(\text{mat}_h A) (\text{mat}_h B) (\text{mat}_h C) (\text{mat}_h D)$
proof —
  obtain $n$ where $n = \text{dim-row } A$ by auto
note * = similar-mat-witD[OF $n$ assms]
from * have [simp]: dim-row C = $n$ by auto
note $C = \ast(6)$ note $D = \ast(7)$
note id = mat-hom-mult[OF $C$ $D$]
note ** = $\ast(1-3)[\text{THEN arg-cong[of - - mat}_h], \text{unfolded id}]$
note mult = $\text{mult-carrier-mat[of - n n}$
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 $\implies$ similar-mat $(\text{mat}_h A) (\text{mat}_h B)$
using similar-mat-wit-hom[of $A$ $B$ $C$ $D$ for $C$ $D$]
by (smt similar-mat-def)

lemma det-dim-1: assumes $A: A \in \text{carrier-mat } n n$
  and $n: n = 1$
shows Determinant.det $A = A \begin{pmatrix} 0, 0 \end{pmatrix}$
  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 $A: A \in \text{carrier-mat } n n$
  and $n: n = 2$
shows Determinant.det $A = A \begin{pmatrix} 0, 0 \end{pmatrix} * A \begin{pmatrix} 1, 1 \end{pmatrix} - A \begin{pmatrix} 0, 1 \end{pmatrix} * A \begin{pmatrix} 1, 0 \end{pmatrix}$
proof —
  have set: $(\sum i<\{0 :: nat\}. f \begin{pmatrix} i \end{pmatrix} = f \begin{pmatrix} 0 \end{pmatrix} + f \begin{pmatrix} 1 \end{pmatrix} \text{ for } f$
    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

27
lemma jordan-nf-root-char-poly: fixes \( A :: 'a :: \{\text{semiring-no-zero-divisors, idom}\} \)
mat
assumes jordan-nf \( A \ n-as \)
and \((m, \text{lam}) \in \text{set} \ n-as\)
shows \( \text{poly} \ (\text{char-poly} A) \ \text{lam} = 0 \)
proof –
from assms have \( m0: m \neq 0 \) unfolding jordan-nf-def by force
from \text{split-list}(OF \ assms(2)) \ obtain \( \text{as bs} \) where \( \text{nas: n-as = as @ (m, \text{lam}) # bs} \) 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:
\( (\lambda x. \text{inverse } ((\text{of-nat } x :: 'a :: \text{real-normed-div-algebra}) \ ^ \ \text{Suc } d)) \longrightarrow 0 \)
proof (rule \text{filterlim-compose}[OF \ tendsto-inverse-0],
introd \text{filterlim-at-infinity}[THEN \ \text{iffD2}, \ \text{of } \text{0}] \ \text{allI} \ \text{implI}, \ \text{goal-cases})
\text{case (2 } r) \hspace{1cm}
let \( \text{?r = nat (ceiling } r) + 1 \)
show ?case
proof (intro \text{eventually-sequentiallyI}[of \ ?r], \text{unfold norm-power norm-of-nat})
\text{fix } x \\
\text{assume } r: \text{?r } \leq x \\
\text{hence } \text{xt: real } x \geq 1 \ \text{by auto} \\
\text{have } r \leq \text{real } ?r \ \text{by linarith} \\
\text{also have } \ldots \leq x \ \text{using } r \ \text{by auto} \\
\text{also have } \ldots \leq x \ ^ \ \text{Suc } d \ \text{using } \text{xt by simp} \\
\text{finally show } r \leq \text{real } x \ ^ \ \text{Suc } d .
\text{qed}
\text{qed simp}

lemma inverse-of-nat-tendsto-zero:
\( (\lambda x. \text{inverse } (\text{of-nat } x :: 'a :: \text{real-normed-div-algebra})) \longrightarrow 0 \)
using inverse-power-tendsto-zero[of \ 0] by auto

lemma poly-times-exp-tendsto-zero: \text{assumes } b: \text{norm } (b :: 'a :: \text{real-normed-field}) < 1
\text{shows } (\lambda x. \text{of-nat } x \ ^ \ k \ast (b \ ^ x)) \longrightarrow 0 \)
proof (cases \( b = 0 \))
\text{case False} \\
\text{define } nla \text{ where } nla = \text{norm } b \\
\text{define } s \text{ where } s = \text{sqrt } nla \\
\text{from } b \text{ False have } \text{nla: } 0 < nla \text{ nla } < 1 \text{ unfolding } nla-def \ \text{by auto} \\
\text{hence } s: 0 < s \ s < 1 \text{ unfolding } s-def \ \text{by auto} \\
\text{fix } x \\
\text{have } s \ast x \ast s \ast x = \text{sqrt } (nla \ ^ (2 \ast x)) \\
\text{unfolding } s-def \ \text{power-add[symmetric]}

28
unfolding real-sqrt-power[symmetric]
by (rule arg-cong[of - x, sqrt (nla x)], simp)
also have \ldots = nla x unfolding power-mult real-sqrt-power
using nla by simp
finally have nla x = s x * s x by simp
\}

note nla-s = this

show \(\lambda x. \text{of-nat} x \cdot k \ast b \cdot x\) \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 \cdot k \ast s \cdot x \leq p for x by (auto simp: ac-simps)
have p have p: norm (real x \cdot k \ast s \cdot x) \leq p for x by auto
from s have s: norm s < 1 by auto
show \((\lambda x. \text{real} x \cdot k \ast s \cdot x \ast s \cdot x)\) \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 - \lambda x. 0], rule eventually-sequentiallyI[of 1], auto)
qed

lemma (in linorder-topology) tendsto-Min: assumes I: I \neq {} and fin: finite I
shows \(\forall i. i \in I \implies (f i \longrightarrow a i) F \implies ((\lambda x. \text{Min} ((\lambda i. f i x) \cdot I)) \longrightarrow (\lambda x. (a \cdot I) :: 'a)) F\)
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 \cdot insert i I) = min (a i) (Min (a \cdot I)) using False insert(I)
  by auto
  have **: (\lambda x. \text{Min} ((\lambda i. f i x) \cdot insert i I)) = (\lambda x. min (f i x) (Min ((\lambda i. f i x) \cdot I)))
  using False insert(1) by auto
  have IH: ((\lambda x. Min ((\lambda i. f i x) \cdot I)) \longrightarrow Min (a \cdot 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]:
\[(f \to a) F \Longrightarrow (g \to b) F \Longrightarrow ((\lambda x. f x \ast g x) \to a \ast b) F\]
for \(f : 'a \ightarrow 'b :: \{ \text{semiring-1, real-normed-algebra} \} \cdot 'n1 \cdot 'n2\)
unfolding matrix-matrix-mult-def [abs-def] by (auto intro!: tendsto-intros)

lemma tendsto-matpower [tendsto-intros]:
\[(f \to a) F \Longrightarrow ((\lambda x. \text{matpow}(f x) n) \to \text{matpow}(a n) F)\]
for \(f : 'a \ightarrow 'b :: \{ \text{semiring-1, real-normed-algebra} \} \cdot 'n \cdot 'n\)
by (induct n, simp-all add: tendsto-mat-mult)

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

lemma vector-smult-distrib:
\[ (A \ast (a :: 'a :: \text{comm-ring-1} \ast s x)) = a \ast s ((A \ast v) x) \]
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:
\[\text{fixes } p :: \text{real poly}\]
assumes lead-coeff p \(> 0 \land \text{degree } p \neq 0\)
shows \(\text{poly } p \to \infty\)
unfolding Lim-PInfty
proof
  fix b
  show \(\exists N \cdot \forall n \geq N. \text{ereal } b \leq \text{ereal } (\text{poly } p (\text{real } n))\)
    unfolding real-arch-simple
  by (meson real-arch-simple)
qed

lemma div-lt-nat:
\[(j :: \text{nat}) < x \ast y \Longrightarrow j \div x < y\]
by (simp add: less-mult-imp-div-less mult.commute)

definition diagvector :: ('n \Rightarrow 'a :: \text{semiring-0}) \Rightarrow 'a \cdot 'n \cdot 'n where
\[\text{diagvector} x = (\chi i. \chi j. \text{if } i = j \text{ then } x i \text{ else } 0)\]

lemma diagvector-mult-vector[simp]:
\[\text{diagvector} x \ast v y = (\chi i. x i \ast y \cdot i)\]
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 $x$ ** $A = (\chi i j. x i * A \$ i \$ j)$ (is $?A = ?B)
unfolding vec-eq-iff
proof (intro allI)
 fix $i$ $j$
 show $?A \$ 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 $x = (\chi i j. A \$ i \$ j * x j)$ (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 $y = diagvector (\lambda i. x i * y i)$
unfolding diagvector-mult-left unfolding diagvector-def by (auto simp: vec-eq-iff)

lemma diagvector-const[simp]: diagvector $(\lambda x. k) = mat k$
unfolding diagvector-def mat-def by auto

lemma diagvector-eq-mat: diagvector $x = mat a \iff x = (\lambda x. a)$
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:
 $\exists ev v. eigen-vector A v ev \land norm ev = spectral-radius A$
proof --
 from non-empty-spectrum[of $A$] finite-spectrum[of $A$] have
 spectral-radius $A \in norm \cdot (Collect (eigen-value A))$
 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 'n 'n) v ev
shows eigen-vector A (((1 / norm1 v) *R v) ev norm1 (((1 / norm1 v) *R v) =

32
proof
  let \( ?v = (1 / \text{norm1 } v) \ast_R v \)
  from assms[unfolded eigen-vector-def]
  have \( nz: v \neq 0 \) and id: \( A \ast v v = ev \ast s v \) by auto
  from \( nz \) have norm1: \( \text{norm1 } v \neq 0 \) by auto
  thus \( \text{norm1 } ?v = 1 \) by simp
  from \( \text{norm1 } nz \)
  have \( \text{nz} : ?v \neq 0 \)
  by auto
  have \( \text{A} \ast v ?v = (1 / \text{norm1 } v) \ast_R (A \ast v v) \)
  by (auto simp: vec-eq-iff matrix-vector-mult-def real-vector.scale-sum-right)
  also have \( A \ast v v = ev \ast s v \) unfolding id ..
  also have \( (1 / \text{norm1 } v) \ast_R (ev \ast s v) = ev \ast s ?v \)
  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 \( \text{norm1 } v \)
  unfolding norm1-def[abs-def] by auto

lemma norm1-ge-norm: \( \text{norm1 } v \geq \text{norm } v \)
  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]:
  \((\ast v) (A :: 'a :: real-normed-algebra-1 \ast 'n \ast 'k) \longrightarrow A \ast v v) \) (at \( v \) within \( S \))
  unfolding matrix-vector-mult-def[abs-def]
  by (auto intro!: tendsto-intros)

lemma tendsto-matrix-matrix-mult[tendsto-intros]:
  \((\ast\ast) (A :: 'a :: real-normed-algebra-1 \ast 'n \ast 'k) \longrightarrow A \ast\ast B) \) (at \( B \) within \( S \))
  unfolding matrix-matrix-mult-def[abs-def]
  by (auto intro!: tendsto-intros)

lemma matrix-vect-scaleR: \((A :: 'a :: real-normed-algebra-1 \ast 'n \ast 'k) \ast (a \ast_R v) = a \ast_R (A \ast v v)\)
  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: \((\text{map-vector } hom \ v = 0) = (v = 0)\)
  unfolding vec-eq-iff map-vector-def by auto

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

lemma (in semiring-hom) matrix-vector-mult-hom:
  \((\text{map-matrix } hom \ A) \ast v (\text{map-vector } hom \ v) = \text{map-vector } hom (A \ast v v)\)
lemma (in semiring-hom) vector-smult-hom:
  \( \text{hom } x * s (\text{map-vector hom } v) = \text{map-vector hom } (x * s v) \)
by (transfer fixing: hom, auto simp: mult-mat-vec-hom)

lemma (in inj-comm-ring-hom) eigen-vector-hom:
  \( \text{eigen-vector } (\text{map-matrix hom } A) (\text{map-vector hom } v) (\text{hom } x) = \text{eigen-vector } A v x \)
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 \equiv spectral-radius A
private definition max-v-ev :: (complex \^ 'n) \times complex where
  max-v-ev = (SOME v-ev. eigen-vector A (fst v-ev) (snd v-ev)
  \xrightarrow{-} \text{norm (snd v-ev) = sr})
private definition max-v = (1 / norm1 (fst max-v-ev)) \text{ of } R \text{ of } fst max-v-ev
private definition max-ev = snd max-v-ev

private lemma max-v-ev:
  \text{eigen-vector } A \text{ of } max-v max-ev
  \text{norm max-ev = sr}
  \text{norm1 max-v = 1}
proof
  obtain v ev where id: max-v-ev = (v, ev) by force
  from spectral-radius-ev[of A] some1-ex[of \lambda v-ev. \text{eigen-vector } A (fst v-ev) (snd v-ev)
  \xrightarrow{-} \text{norm (snd v-ev) = sr, folded max-v-ev-def, unfolded id}]
  have v: \text{eigen-vector } A v ev and ev: \text{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 ≡ \chi i j. Re (A \$ i \$ j)
private definition S where S = {v :: real 'n . norm1 v = 1 \wedge (\forall i. v \$ i \geq 0)} \wedge
(\forall i. (B * v \$ i \geq sr * (v \$ i)))
private definition f :: real 'n \Rightarrow real 'n where
f v = (1 / norm1 (B * v \$ )) \ast R (B * v \$ )

private lemma closedS: closed S
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 \implies 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 \land i. v \$ i \geq w \$ i
shows (B * 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 \neq {}
proof
  let \(?v = (x. \text{norm}(\text{max-v} \, $i$)) :: \text{real} ^ \sum\)
have \text{norm1 max-v} = 1 by (rule max-v-ev(3))
  hence \(?v = 1\) unfolding \text{norm1-def by auto}\{
    fix \(i\)
    have \(sr * (?v \, $i$) = sr * \text{norm}(\text{max-v} \, $i$)\) by auto
    also have \(\ldots = (\text{norm max-ev} \, \text{norm}(\text{max-v} \, $i$))\) using max-v-ev by auto
    also have \(\ldots = \text{norm} \, ((\text{max-ev} * \text{max-v}) \, $i$)\) by (auto simp; norm-mult)
also have \(\text{max-ev} * \text{max-v} = A * v \, \text{max-v}\) using max-v-ev(1)[unfolded eigen-vector-def] by auto
    also have \(\text{norm} \, ((A * v \, \text{max-v}) \, $i$) \leq (B * v \, ?v) \, $i$\)
    unfolding matrix-vector-mult-def vec-lambda-beta
    by (rule sum-norm-le, auto simp; norm-mult B-norm)
finally have \(sr * (?v \, $i$) \leq (B * v \, ?v) \, $i$\).
  }
  note \(le = \text{this}\)
have \(?v \in S\) unfolding \(S\)-def using \(\text{nv le by auto}\)
thus \(?\text{thesis\ by blast}\)

qed

private lemma \text{convexS: convex S}\nproof (rule convexI)
  fix \(v\, w\, a\, b\)
assume \(*\, : v \in S\, w \in S\, 0 \leq a \, 0 \leq b \, a + b = (1 :: \text{real})\)
  let \(?\text{lin} = a *R v + b *R w\)
from \(*\) have \(1:\) \text{norm1 v = 1 norm1 w = 1 unfolding S-def by auto}\nhave \(\text{norm1 ?lin} = a * \text{norm1 v} + b * \text{norm1 w}\)
  unfolding \text{norm1-def sum-distrib-left sum-distrib[symmetric]}\nproof (rule sum.cong)
  fix \(i :: ^\text{'}n\)
from \(*\) have \(v \, $i \geq 0\) \(w \, $i \geq 0\) unfolding \(S\)-def by auto
thus \(\text{norm} \, (?\text{lin} \, $i$) = a * \text{norm} (v \, $i$) + b * \text{norm} (w \, $i$)\)
  using \text{(*)(5) \ by auto}\n  qed simp
also have \(\ldots = 1\) using \text{(*)(5) \ by auto}\nfinally have \text{norm1: norm1 ?lin = 1}\{
  fix \(i\)
  from \(*\) have \(0 \leq v \, $i \, sr * v \, $i \leq (B * v \, v) \, $i\) unfolding \(S\)-def by auto
    with \(?a \geq 0\) have \(a : a * (sr * v \, $i$) \leq a * (B * v \, v) \, $i\) by (intro \text{mult-left-mono})\)
from \(*\) have \(0 \leq w \, $i \, sr * w \, $i \leq (B * v \, w) \, $i\) unfolding \(S\)-def by auto
    with \(?b \geq 0\) have \(b : b * (sr * w \, $i$) \leq b * (B * v \, w) \, $i\) by (intro \text{mult-left-mono})\)
from \(a b\) have \(a * (sr * v \, $i$) + b * (sr * w \, $i$) \leq a * (B * v \, v) \, $i + b * (B * v \, w) \, $i\) by auto\}
  note \(le = \text{this}\)
have \text{switch[simp]:} \(\land x y. x * a * y = a * x * y\ \land x y. x * b * y = b * x * y\)
by auto

36
\[ x \in \{v, w\} \implies a \cdot (r \cdot x \oplus h) = r \cdot (a \cdot x \oplus h) \]

\begin{verbatim}
by (auto simp: matrix-vect-scaleR matrix-vector-right-distrib ring-distribs)
qed

private abbreviation (input) r :: real \implies complex where
r \equiv of-real

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

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

private lemma rv-mult: A \cdot v rv v = rv (B \cdot 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: \(\forall v. v \in S \implies A \cdot v rv v \neq 0\)

begin

private lemma normB-S: assumes v: v \in S
  shows norm1 (B \cdot 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: \(\forall i. v \$ i \geq 0 \land i. sr \cdot v \$ i \leq (B \cdot v v)\)
    unfolding S-def by auto
    from normB-S[OF v] have normB: norm1 (B \cdot v v) > 0 using norm1-nonzero
    by auto
    have fv: f v = (1 / norm1 (B \cdot v v)) \cdot R (B \cdot v v)
      unfolding f-def by auto
    from normB have Bv0: B \cdot v v \neq 0
      unfolding norm1-0-iff[symmetric]
      by linarith
    have norm: norm1 (f v) = 1
      unfolding fv using normB Bv0 by simp

\end{verbatim}

37
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-sum-distrib-left B-norm intro!: sum-nonneg)

have id1: \(i \mapsto (B *v f v) \leq c * (B *v (B *v v))\) $ i

unfolding f-def c-def matrix-vector-scaleR by simp

have id3: \(i \mapsto s r * f v \leq c * (B *v (s r *v v))\) $ i

unfolding f-def c-def[symmetric] matrix-vector-scaleR by auto

have 2: sr * f v $ i ≤ (B *v f v) $ i unfolding id1 id3

unfolding real-mult-cancel-iff2[OF (c > 0)]

by (rule mult-B-mono, insert ge(2), auto)

note 1 2
}

with norm have f v \in 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:
\(\exists v. \textnormal{eigen-vector} A v (v, s r) \land \textnormal{real-non-neg-vec} v\)

proof –

from brouwer[OF compactS convexS non-emptyS cont-f image-f]

obtain v where \(v\): \(v \in S\) and \(fv f v = v\) by auto

define ev where \(ev = norm1 (B *v v)\)

from normB-S[OF ev] have ev \(\neq 0\) unfolding ev-def by auto

with norm1-ge-0[def] have norm: \(ev > 0\) by auto

from arg-comp[OF \(fv\), unfolded f-def], of \(\lambda w: \textnormal{real } ^{\sim} n. ev *w w\) 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: \(\exists i. s r * v \leq ev * v \leq ev * v \leq i\) by auto

have A *v rv v = rv (B *v v) unfolding rv-mult ..

also have \(_\ldots = ev *s rv v\) 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 \neq 0 unfolding S-def by auto

hence rv v \neq 0 unfolding rv-0 .

with ev have ev: \(\text{eigen-vector} A (rv v)\) ev unfolding \(\text{eigen-vector-def}\) by auto

hence eigen-value A ev unfolding eigen-value-def by auto

from spectral-radius-max[OF this] have le: \(\text{norm} (rv v) \leq s r\).

from v0 obtain i where \(v \leq ev \leq v \leq v \leq i\) unfolding vec-eq-iff by auto

from v have v $ i \geq 0 unfolding S-def by auto

...
with \( \langle v \mid i \neq 0 \rangle \) have \( v \mid i > 0 \) by auto
with ge[\ of \ i] have ge: \( sr \leq ev \) by auto
with le have sr: \( v \mid sr = ev \) 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:
\( \exists \ v. \text{eigen-vector } A \ v (v \ sr) \land \text{real-non-neg-vec } v \)

proof (cases \( v \in S. \ A \ast v \ v \neq 0 \))
case True
show ?thesis
by (rule Perron-Frobenius.perron-frobenius-positive-ev[\ OF \ rnnA], insert True, auto)
next
case False
then obtain \( v \) where \( v \in S \) and \( A0: \ A \ast v \ v = 0 \) by auto
hence id: \( A \ast v \ v = 0 \ast s \ v \ v \) and \( v \neq 0 \) 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 \( \theta: 0 \leq sr \) by auto
from \( v[\ unfolding \ S-def] \) have ge: \( \bigwedge i. \ sr \ast v \ S \ i \leq (B \ast v \ v) \ S \ i \) by auto
from \( v[\ unfolding \ S-def] \) have \( \text{rnn: real-non-neg-vec (rv } v) \)
unfolding real-non-neg-vec-def vec-elements-h-def by auto
from \( v0 \) obtain \( i \) where \( v \ S \ i \neq 0 \) unfolding vec-eq-iff by auto
from \( v \) have \( v \ S \ i \geq 0 \) unfolding S-def by auto
with \( \langle v \ S \ i \neq 0 \rangle \) have \( vi: v \ S \ i > 0 \) by auto
from rv-mult[\ of \ v, \ unfolded \ A0] have rv \ v \ast \ v = 0 \ by simp
hence \( B \ast v \ v = 0 \) unfolding rv-0.
from ge[\ of \ i, \ unfolded \ this] \( vi \) have ge: \( sr \leq 0 \) by (simp add: mult-le-0-iff)
with \( \langle 0 \leq sr \rangle \) have sr = 0 by auto
show ?thesis unfolding \( \langle sr = 0 \rangle \) using \( \text{rnn ev by auto} \)

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 \( \exists v. \text{eigen-vector } A \ v (of-real (\text{spectral-radius A})) \land \text{real-non-neg-vec } v \)
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 \neq 1 proof –
    from x have cos x \neq 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

declare cis-pi[simp]

lemma cis-pi-2[simp]: cis (pi / 2) = i
  by (auto simp: complex-eq-iff)

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

40
proof

have \( \ cis (\pi \ast 3 / 2) = \ cis (\pi + \pi / 2) \)
  by (rule arg-cong[of - - cis], simp)
also have \( \ldots = -i \) 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 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 arg-unique[OF sgn-cis assms(2-3)] 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 \lambda x. x / r] have cis x = cis y using r by (simp add: rcis-def)
  from arg-cong[OF OF this, of \lambda x. inverse x] have cis (−x) = cis (−y) by simp
  from arg-cong[OF 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 *, 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 \( \text{degp}: \) degree ?p = n unfolding \( \text{degM} \).

with n have p: ?p \neq 0 by auto

have roots: ?Roots = \{ x. poly ?p x = 0 \}

unfolding poly-diff poly-monom by simp

also have finite \ldots by \( \text{rule poly-roots-finite[OF p]} \)

finally have fin: finite ?Roots.

have sub: ?prod \subseteq ?Roots.

proof

fix x

assume x \in ?prod

then obtain i where x = cis (real i \cdot 2 \cdot pi / n) by auto

have x ^ n = cis (real i \cdot 2 \cdot pi) unfolding x DeMoivre using n by simp

also have \ldots = 1 by simp

finally show x \in ?Roots by auto

qed

have Rn: \( \text{card } ?\text{Roots} \leq n \) unfolding roots

by \( \text{rule poly-roots-degree[of ?p, unfolded degp, OF p]} \)

have \ldots = \{ 0 ..< n \} by simp

also have \ldots = card ?prod.

proof \( \text{rule card-image[symmetric], rule inj-onI, goal-cases} \)

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 \cdot pi / real n] n

have real m \cdot 2 \cdot pi / real n < real n \cdot 2 \cdot pi / real n by simp

hence real m \cdot 2 \cdot pi / real n < 2 \cdot pi using n by simp

} note [simp] = this

have 0: \( \{ 1 :: real \} \neq 0 \) using n by auto

have real x \cdot 2 \cdot pi / real n = real y \cdot 2 \cdot pi / real n

by \( \text{rule inj-onD[OF rcis-inj-on 1(3)[unfolded cis-rcis-eq]], insert 1(1-2), auto} \)

with n show x = y by auto

qed

finally have cn: \( \text{card } ?\text{prod} = n \).

with Rn have card ?prod \geq card ?Roots by auto

with card-mono[OF fin sub] have card: card ?prod = card ?Roots by auto

have ?prod = ?Roots

by \( \text{rule card-subset-eq[OF fin sub card]} \)

from this roots[symmetric] cn[unfolded this]

show ?case unfolding root-unity-def by blast

qed

lemma poly-roots-dvd: \( \text{fixes } p :: 'a :: \text{field poly} \)

assumes p \neq 0 and degree p = n

and card \{ x. poly p x = 0 \} \geq n and \{ x. poly p x = 0 \} \subseteq \{ x. poly q x = 0 \}

shows p dvd q

proof –
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 \( n \leq \text{card} \{ x. \text{poly} ?u x = 0 \} \) using main by auto

show \(?u \neq 0\) using n by auto

show \( \{ x. \text{poly} ?u x = 0 \} \subseteq \{ x. \text{poly} ?p x = 0 \} \)

unfolding main(2) main(1)[symmetric] poly-prod-list prod-list-zero-iff by auto

qed

have \( \text{deg'}: \text{degree} ?p = n \)
  by (subst degree-prod-list-eq, auto simp :o-def sum-list-triv)

have \( \text{monic } ?p \text{ by (rule monic-prod-list, auto) \)
from dvd[unfolded dvd-def] obtain \( f \text{ where } ?p = ?u \ast f \text{ by auto} \)
have \( \text{degree } ?p = \text{degree } ?u + \text{degree } f \text{ using mon' n unfolding puf} \)
  by (subst degree-mult-eq, auto)

have \( \text{degree } ?u = 0 \text{ by auto} \)

from degree0-coeffs[OF this]
obtain \( a \) where \( f \): \( f = [a:] \) by blast

from \( \text{arg-cong}[OF puf] \)
mon mon'

have \( a = 1 \)
  unfolding puf f
  by (cases a = 0, auto)

with \( f \) have \( f = 1 \) by auto

with \( f \) have \( ?\text{thesis} \) by auto

qed

lemma \( \text{order-monic-linear}: \text{order } x [\{y,1\}] = (\text{if } y + x = 0 \text{ then } 1 \text{ else } 0) \)
proof (cases \( y + x = 0 \))
  case True
  hence \( \text{poly } [\{y,1\}] x = 0 \) by simp
  from this[unfolded order-root] have \( \text{order } x [\{y,1\}] \neq 0 \) by auto
  moreover from \( \text{order-deg}[OF [\{y,1\}] x] \) have \( \text{order } x [\{y,1\}] \leq 1 \) by auto
  ultimately show \( ?\text{thesis unfolding True by auto} \)
next
  case False
  hence \( \text{poly } [\{y,1\}] x \neq 0 \) by auto
  from order-0I[OF this] False show \( ?\text{thesis by auto} \)

qed

lemma \( \text{order-root-unity}: \text{fixes } x :: \text{complex assumes } n: n \neq 0 \text{ shows } \text{order } x (\text{root-unity } n) = (\text{if } x^?n = 1 \text{ then } 1 \text{ else } 0) \)
proof (cases \( x^?n = 1 \))
  case False
  with \( \text{roots-of-unity}(2)[OF n] \) have \( \text{poly } ?u x \neq 0 \) by auto
  from False order-0I[OF this] show \( ?\text{thesis by auto} \)
next
  case True
  let \( ?\phi = \lambda i :: \text{nat}. i * 2 * \pi / n \)
  from True roots-of-unity(1)[OF n] obtain \( i \text{ where } i: i < n \)
    and \( x: x = \text{cis } (?\phi i) \) by force
  from \( i \) have \( n\text{-split: } [0..<n] = [0..<\ i] @ i \# [\text{Suc } i..<n] \)
    by (metis le-Suc-ex less-imp-le-nat not-le-imp-less not-less0 upt-add-eq-append upt-cone-Cons)
\[
\{ \\
\text{fix } j \\
\text{assume } j: j < n \lor j < i \text{ and eq: cis (}\phi i) = cis (\phi j) \\
\text{from inj-onD[OF cis-inj-on eq]} i j n \text{ have } i = j \text{ by (auto simp: field-simps)} \\
\} \\
\text{note inj = this} \\
\text{have order } x \not\in u = 1 \text{ unfolding root-unity-decomp[OF n]} \\
\text{unfolding } x n\text{-split using inj} \\
\text{by (subst order-prod-list, force, fastforce simp: order-monic-linear)} \\
\text{with True show } ?\text{thesis by auto} \\
\} \\
\text{qed}
\]

\textbf{lemma order-prod-root-unity:} assumes \(0: 0 \not\in \text{set } ks\)
\text{shows } \((x :: \text{complex}) (\text{prod-root-unity } ks) = \text{length } (\text{filter } (\lambda k. x^k = 1) \text{ ks})\)
\text{proof} –
\text{have } order \(x\) (\text{prod-root-unity } \text{ks}) = (\sum k\in\text{ks. order } x \text{ (root-unity } k)) \\
\text{unfolding } \text{prod-root-unity-def} \\
\text{by (subst order-prod-list, insert 0, auto simp: o-def)} \\
\text{also have } \ldots = (\sum k\in\text{ks. (if } x^k = 1 \text{ then 1 else 0)}) \\
\text{by (rule arg-cong, rule map-cong, insert 0, force, intro order-root-unity, metis)} \\
\text{also have } \ldots = \text{length } (\text{filter } (\lambda k. x^k = 1) \text{ ks}) \\
\text{by (subst sum-list-map-filter[ symmetric], simp add: sum-list-triv)} \\
\text{finally show } ?\text{thesis } .
\text{qed}

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

\textbf{lemma root-unity-explicit:} fixes \(x :: \text{complex}\)
\text{shows } \\
(x^1 = 1) \iff x = 1 \\
(x^2 = 1) \iff (x \in \{1, -1\}) \\
(x^3 = 1) \iff (x \in \{1, \text{Complex } (-1/2) \text{ (sqrt 3 / 2)}, \text{Complex } (-1/2) (-\text{sqrt 3 / 2})\}) \\
(x^4 = 1) \iff (x \in \{1, -1, i, -i\}) \\
\text{proof} –
\text{show } (x^1 = 1) \iff x = 1 \\
\text{by (subst root-unity-witness[of } [1]\text{], code-simp, auto)} \\
\text{show } (x^2 = 1) \iff (x \in \{1, -1\}) \\
\text{by (subst root-unity-witness[of } [1, -1]\text{, code-simp, auto)}
\]
proof
primitive-root-unity-exists
lemma
qed

−
proof
primitive-root-unityD
lemma
qed

−
proof
primitive-root-unity-dvd
fixes
assumes
shows
proof
assume
then obtain
where
have
also have
and
finally show
next
assume
note
show
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 ≠ 0 by auto
from gcd-coprime-exists[of this] this obtain i' k' g where
*: i = i' * g k = k' * g g ≠ 0 and g = gcd i k by blast
from *'(2) k(1) have k': k' ≠ 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 ^ k' = 1 by (simp add: DeMoivre k')
with k(3)[OF k'] have k' ≥ 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 ^ k = 1 ∧ (∀ i ∈ {1..< k}. x ^ i ≠ 1))
unfolding primitive-root-unity-def by auto

lemma primitive-root-unity-explicit: fixes x :: complex
shows primitive-root-unity 1 x ─→ x = 1
primitive-root-unity 2 x ─→ x = -1
primitive-root-unity 3 x ─→ (x ∈ {Complex (-1/2) (sqrt 3 / 2), Complex (-1/2) (- sqrt 3 / 2))})
primitive-root-unity 4 x ─→ (x ∈ {i, -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 |
— 48
have \((\forall i \in \{1..<2\}, P i) = P 1 \ (\forall i \in \{1..<3\}, P i) \iff P 1 \land P 2\)
(\forall i \in \{1..<4\}, P i) \iff P 1 \land P 2 \land 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 ::
's'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: \(\forall x. \text{cmod} x = 1 \implies \text{poly} f x \neq 0\)
and k: \(\forall k'. k' > k \implies \neg \text{root-unity} k' \text{ dvd} p\)
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 \iff 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 dvd ?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 I]
    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 @ ks2 by auto
    hence p dvd ?u = (?u * (prod-root-unity (ks1 @ ks2) * f)) dvd ?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 dvd ?u = prod-root-unity (ks1 @ ks2) * f .
    from d have ks': ks' = k # Ks by auto
    have k < k' ⟷ ¬ root-unity k' dvd p dvd ?u for k'
      using k[of k'] True by (metis dvd-die-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-distrib)
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 ..

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 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
context includes lifting-syntax
begin
lemma HMA-M-smult[transfer-rule]: ((=) ===> HMA-M ===> HMA-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 ≠ 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 ≠ 0 and id: A * v v = x * s v
  by auto
  from arg-cong[OF id, of (S S) a] have: (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 ≠ 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: nonneg-matpow nonneg matrix-matrix-mult-def
intro!: sum-pos2[of - k mult-nonneg-nonneg]

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 simp

lemma matpow-mono: assumes B: Â 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) $ i $ j $ h ≥ 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)
lemma G-relpow-matpow-pos-ge:
  assumes \((i, j) \in G \rightarrow n \geq m\)
  shows \(\text{matpow} \ (A + \text{mat} 1) \ n \ \text{$_{i \ j}$} > 0\)
proof –
  from assms(2) obtain \(k\) where \(n = m + k\) using leSuc-ex by blast
  have \(0 < \text{matpow} \ A \ m \ \text{$_{i \ j}$}\) by (rule G-relpow-matpow-pos[OF assms(1)])
  also have \(\ldots \leq \text{matpow} \ (A + \text{mat} 1) \ m \ \text{$_{i \ j}$}\)
    by (rule matpow-mono, auto simp: mat-def)
also have \(\ldots \leq \text{matpow} \ (A + \text{mat} 1) \ n \ \text{$_{i \ j}$}\)
  unfolding \(n\) using matpow-sum-one-mono
  .
  finally show \(?thesis\) .
qed
locale perron-frobenius =
  pf-nonneg-mat A
for \(A::\text{real}{}^\prime\) \(\times\) \(\times\) \(\text{\"n}\) \(\times\) \(\text{\"n}\) +
  assumes irr: irreducible
begin
definition \(N\) where \(N = (SOME \ n. \ \forall \ ij. \ \exists \ n \leq N. \ ij \in G \rightarrow n)\)
lemma \(N\): \(\exists \ n \leq N. \ ij \in G \rightarrow n\)
proof –
  \{ \fix \(ij\)
      have \(ij \in G^+\) using irr[unfolded irreducible-def] by auto
      from this[unfolded trancl-power] have \(\exists \ n. \ ij \in G \rightarrow n\) by blast
  \}
  hence \(\forall \ ij. \ \exists \ n. \ ij \in G \rightarrow n\) by auto
  from choice[OF this] obtain \(f\) where \(f:\ \land \ ij. \ ij \in G \rightarrow f \ ij\) by auto
define \(N\) where \(N = \text{Max} \ (\text{range} \ f)\)
  \{ \fix \(ij\)
      from \(\text{of} \ ij\) have \(ij \in G \rightarrow f \ ij\).
      moreover have \(f \ ij \leq N\) unfolding \(N\)
        by (rule Max-ge, auto)
      ultimately have \(\exists \ n \leq N. \ ij \in G \rightarrow n\) by blast
  \}
  note main = this
  let \(?P = \lambda \ N. \ \forall \ ij. \ \exists \ n \leq N. \ ij \in G \rightarrow n\)
  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 \(\text{matpow} \ (A + \text{mat} 1) \ n \ \text{$_{i \ j}$} > 0\)
proof –
  from \(N\) obtain \(n\) where \(n \leq N\) and \(\text{reach}: \ (i, j) \in G \rightarrow n\) by auto
  show \(?thesis\) by (rule G-relpow-matpow-pos-ge[OF reach n])
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 → real ⇒ bool where
  le-vec x y ≡ (∀i. x $ i ≤ y $ i)

abbreviation lt-vec :: real → real ⇒ 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 ⇒ real where
  r x = Min { (A ∗ v x) $ j | j. x $ j ≠ 0 }

definition X :: (real ⇒ real) 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 θ x
  shows lt-vec θ (A ∗ v x)
  proof
fix \( i \)

from \( A\)-nonzero-fixed-\( i \)\ of \( i \) obtain \( j \) where \( A \$ i \$ j \neq 0 \) by auto

with \( \text{nonneg} \text{ of} \ i \ j \) \( \text{have} \ A \$ i \$ j > 0 \) by simp

from \( x \) have \( x \$ j \geq 0 \) for \( j \) by (auto simp: order.strict-iff-order)

note \( \text{nonneg} = \text{mult-nonneg-nonneg} \text{of} \ i \ \text{this} \)

have \( (A * v \ x) \$ i = (\sum_{j \in \text{UNIV}}. \ A \$ i \$ j * x \$ j) \)

unfolding matrix-vector-mult-def by simp

also have \( \ldots = A \$ i \$ j * x \$ j + (\sum_{j \in \text{UNIV} - \{j\}}. \ A \$ i \$ j * x \$ j) \)

by (subst sum.remove, auto)

also have \( \ldots > 0 + 0 \)

by (rule add-less-le-mono, insert \( A \)\text{of} \rule-format \text{nonneg},

auto intro!: sum-nonneg mult-pos-pos)

finally show \( 0 \$ i < (A \ast v \ x) \$ i \) by simp

qed


\begin{itemize}
\item[] \textbf{lemma} nonzero-Ax: \textbf{assumes} \( x: x \in X \)
\textbf{shows} \( A \ast v \ x \neq 0 \)
\textbf{proof}
\item[] \textbf{assume} \( 0: A \ast v \ x = 0 \)
\item[] from \( x\text{[unfolded X-def]} \) \textbf{have} \( x: \text{le-vec} 0 \ x \ x \neq 0 \) by auto
\item[] from \( x(2) \) \textbf{obtain} \( j \) where \( xj: x \$ j \neq 0 \)
\item[] by (metis vec-eq-iff zero-index)
\item[] from \( A\)-nonzero-fixed-\( j \)\ of \( j \) obtain \( i \) where \( Aij: A \$ i \$ j \neq 0 \) by auto
\item[] from \( \text{arg-cong} \text{of} \ 0, \ \text{of} \ \lambda \ v. \ v \$ i, \ \text{unfolded matrix-vector-mult-def} \)
\item[] \textbf{have} \( 0 = (\sum_{k \in \text{UNIV}}. \ A \$ h \ i \$ k * x \$ h \ k) \) by auto
\item[] also have \( \ldots = A \$ h \ i \$ h \ j * x \$ h \ j + (\sum_{k \in \text{UNIV} - \{j\}}. \ A \$ h \ i \$ h \ k * x \$ h \ k) \)
\item[] by (subst sum.remove[\text{of} - \( j \)], auto)
\item[] also have \( \ldots > 0 + 0 \)
\item[] by (rule add-less-le-mono, insert \( \text{nonneg} \text{of} \ i \ \text{Aij x(1) xj}, \)

auto intro!: sum-nonneg mult-pos-pos simp: dual-order.not-eq-order-implies-strict)
\item[] finally show \( \text{False} \) by simp
\end{itemize}

\begin{itemize}
\item[] \textbf{lemma} r-witness: \textbf{assumes} \( x: x \in X \)
\textbf{shows} \( \exists \ j. \ x \$ j > 0 \land \text{rx} = (A * v \ x) \$ j / x \$ j \)
\textbf{proof}
\item[] from \( x\text{[unfolded X-def]} \) \textbf{have} \( x: \text{le-vec} 0 \ x \ x \neq 0 \) by auto
\item[] let \( \forall A = (A * v \ x) \$ j / x \$ j \ | \ j. \ x \$ j \neq 0 \)
\item[] from \( x(2) \) \textbf{obtain} \( j \) where \( x \$ j \neq 0 \)
\item[] by (metis vec-eq-iff zero-index)
\item[] hence empty: \( \forall A \neq \{\} \) by auto
\item[] from Min-in\( \text{OF} \ - \ this, \ \text{folded r-def} \)
\item[] obtain \( j \) where \( x \$ j \neq 0 \) and \( \text{rx}: \text{rx} = (A * v \ x) \$ j / x \$ j \) by auto
\item[] with \( x \) have \( x \$ j > 0 \) by (auto simp: dual-order.not-eq-order-implies-strict)
\item[] with \( \text{rx show} \ ?\text{thesis} \) by auto
\end{itemize}

\begin{itemize}
\item[] \textbf{qed}
\end{itemize}
lemma rx-nonneg: assumes $x: x \in X$
  shows $r x \geq 0$
proof -
  from $x$[unfolded X-def] have $le-vec 0 x x \neq 0$ by auto
  let $?A = \{ (A *v x) \ s j / x s j \ | j. x s j \neq 0 \}$
  from $r$-witness[OF $x \in X$]
  have empty: $?A \neq \{}$ by force
  show ?thesis unfolding $r$-def X-def
  proof (subst Min-ge-iff, force, use empty in force, intro ballI)
    fix $y$
    assume $y \in ?A$
    then obtain $j$ where $y = (A *v x) s j / x s j$ and $x s j \neq 0$ by auto
    from nonneg-Ax[OF $x \in X$] this $x$
    show $0 \leq y$ 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 $x$: $x \in X$ unfolding X-def.strict-iff-order by auto
  let $?A = \{ (A *v x) \ s j / x s j \ | j. x s j \neq 0 \}$
  from $r$-witness[OF $x \in X$]
  have empty: $?A \neq \{}$ by force
  show ?thesis unfolding $r$-def X-def
  proof (subst Min-gr-iff, force, use empty in force, intro ballI)
    fix $y$
    assume $y \in ?A$
    then obtain $j$ where $y = (A *v x) s j / x s j$ and $x s j \neq 0$ by auto
    from $lt$ this $x$ show $0 < y$ by simp
  qed
qed

lemma rx-le-Ax: assumes $x: x \in X$
  shows $le-vec (r x *s x) (A *v x)$
proof (intro allI)
  fix $i$
  show $(r x *s x) s h i \leq (A *v x) s h i$
  proof (cases $x s i = 0$)
    case True
    with nonneg-Ax[OF $x$] show ?thesis by auto
  next
    case False
    with $x$[unfolded X-def] have pos: $x s i > 0$
    by (auto simp: dual-order.not-eq-order-implies-strict)
    from False have $(A *v x) s h i / x s i \in \{ (A *v x) s j / x s j / j. x s j \neq 0 \}$
  qed
qed

58
hence \((A \ast v x) \# h i / x \# i \geq r x\) unfolding \(r\)-def by simp

hence \(x \# i \ast r x \leq x \# i \ast ((A \ast v x) \# h i / x \# i)\) unfolding mult-cancel-left-pos[OF pos].

also have \(\ldots = (A \ast v x) \# h i\) using pos by simp

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

qed

lemma \(\rho\)-le-x-Ax-imp-\(\rho\)-le-\(r\)x:
assumes \(x: x \in X\)
and \(\rho: \le-vec (\rho \ast s x) (A \ast v x)\)
shows \(\rho \leq r x\)

proof –

have \(r x \in \#S\) using \(\rho\)-le-x-Ax[OF \(x\)] by auto

moreover {

fix \(y\)

assume \(y \in \#S\)

hence \(y: \le-vec (y \ast s x) (A \ast v x)\) by auto

from \(\rho\)-le-x-Ax-imp-\(\rho\)-le-\(r\)x[OF \(x\) this]

have \(y \leq r x\).

}

ultimately show \(?thesis\) by (metis (mono-tags, lifting) cSup-eq-maximum)

qed

lemma \(\rho\)-smult:
assumes \(x: x \in X\)
and \(a: a > 0\)
shows \(r (a \ast s x) = r x\)

unfolding \(r\)-def

by (rule arg-cong[of \(-\ - Min\)], unfold vector-smult-distrib, insert \(a\), simp)

definition \(X1 = (X \cap \{x. \text{norm } x = 1\}\))

lemma bounded-\(X1\): bounded \(X1\) unfolding bounded-iff \(X1\)-def by auto

lemma closed-\(X1\): closed \(X1\)

proof –

have \(X1: X1 = \{x. \text{le-vec } 0 x \land \text{norm } x = 1\}\)

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)
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 = (∑ 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 + (∑ 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 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 \( r' \text{-} r \): assumes \( x \in Y \) shows \( r' \ x = r \ x \)
unfolding \( r' \text{-} def \ r \text{-} def \)
proof (rule arg-cong[of \( \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \cdot \·
Lemma $z : z \in Y$ and $sr-max-Y : \forall y. y \in Y \implies r y \leq sr$

Proof -
  let $?P = \lambda z. z \in Y \land (\forall y \in Y. r y \leq r z)$
  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 $z \in Y \land y \in Y \implies r y \leq r z$ 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 $B : \forall i j. B \leq i \leq j \geq 0$
and le-vec $x y$
shows le-vec $(B \ast v x) (B \ast v y)$
proof (intro allI)
  fix $i$
  show $(B \ast v x) \leq (B \ast v y) \leq i$
  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 $(A + mat \ 1)$ $n \ast\ast A = A \ast\ast matpow (A + mat \ 1)\ n$
by (induct $n$, auto simp: matrix-add-rdistrib matrix-add-ldistrib matrix-mul-rdistrib
matrix-mul-assoc[symmetric])

Lemma A1n-commute: $A1n \ast\ast A = A \ast\ast A1n$
unfolding A1n-def by (rule matpow-1-commute)

Lemma le-vec-pow-A-1: assumes le: le-vec $(\rho \ast s \ x) (A \ast v x)$
shows le-vec $(\rho \ast s \ pow-A-1 \ x) (A \ast v pow-A-1 \ x)$
proof -
  have $A1n \leq i \leq j$ for $i\ j$ using A1n-pos[of i j] by auto
  from le-vec-mono-left[OF this le]
  have le-vec $(A1n \ast v (\rho \ast s \ x)) (A1n \ast v (A \ast v x))$.
  also have $(A1n \ast v (A \ast v x) = (A1n \ast\ast A) \ast v x$ by (simp add: matrix-vector-mult-assoc)
  also have $\ldots = A \ast v (A1n \ast v x)$ unfolding A1n-commute by (simp add:
matrix-vector-mul-assoc
also have \( A \ast v \) unfolding pow-A-1-def ..
also have \( A_{in} \ast v (\rho \ast s x) = \rho \ast s (A_{in} \ast v x) \) unfolding vector-smult-distrib ..
also have \( \rho \ast s pow-A-1 x \) unfolding pow-A-1-def ..
finally show le-vec (\( \rho \ast s pow-A-1 x \) \( A \ast v pow-A-1 x \)) .
qed

lemma r-pow-A-1: assumes \( x : x \in X \) shows \( r x \leq r (pow-A-1 x) \)
proof –
let \( ?y = pow-A-1 x \)
have \( ?y \in pow-A-1 \ ' X \) 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 \( ?a = \{ \rho. \ le-vec (\rho \ast s x) (A \ast v x) \} \)
let \( ?b = \{ \rho. \ le-vec (\rho \ast s pow-A-1 x) (A \ast v pow-A-1 x) \} \)
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 \( le-vec (\rho \ast s x) (A \ast v x) \) by auto
from le-vec-power-A-1[OF this] have \( \rho \in ?b \) by auto
thus \( \exists \rho' \in ?b. \rho \leq \rho' \) by auto
qed

lemma sr-max: assumes \( x : x \in X \) shows \( r x \leq s r \)
proof –
let \( ?n = norm x \)
define \( x' \ where \ x' = inverse \ ?n \ast s x \)
from x[unfolded X-def] have x0: \( x \neq 0 \) by auto
hence n: \( ?n > 0 \) by auto
have x': \( x' \in X_1 \) \( x' \in X \) 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 \( y \ where \ y = pow-A-1 x' \)
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-power-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: \( z \notin i > 0 \)

63
lemma sr-pos: $sr > 0$
by (rule rz-pos, insert z-pos, auto)

context fixes $u$
assumes $u$: $u \in X$ and $ru$: $r \cdot u = sr$
begin

lemma sr-imp-eigen-vector-main: $sr \cdot s \cdot u = A \cdot v \cdot u$
proof (rule ccontr)
assume $\neg$: $sr \cdot s \cdot u \neq A \cdot v \cdot u$
let $?x = A \cdot v \cdot u - sr \cdot s \cdot u$
from $\neg$ have $0$: $?x \neq 0$ by auto
let $?y = pow-A-1 \cdot u$
have le-vec ($sr \cdot s \cdot u$) ($A \cdot v \cdot u$) using rz-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)
have lt-vec $0$ ($pow-A-1 \cdot ?x$) by auto
hence lt: lt-vec ($sr \cdot s \cdot ?y$) ($A \cdot v \cdot ?y$) unfolding pow-A-1-def matrix-vector-right-distrib-diff
      matrix-vector-mul-assoc AIn-commute vector-smult-distrib by simp
let $?f = (\lambda i \cdot (A \cdot v \cdot ?y - sr \cdot s \cdot ?y) \cdot i) / ?y \cdot i)$
let $?U = UNIV :: 'n set$
define eps where eps = Min ($if \cdot {?U}$)
have U: finite ($\forall ?f \cdot {?U}$) $?f \cdot {?U} \neq \{\}$ by auto
have eps: eps > $0$ unfolding eps-def Min-le[OF U]
      using lt sr-pos y-pos by auto
have le: le-vec (($sr + eps$) $\cdot s \cdot ?y$) ($A \cdot v \cdot ?y$)
proof
fix $i$
have (($sr + eps$) $\cdot s \cdot ?y$) $\cdot i = sr \cdot s \cdot ?y$ $\cdot i + eps \cdot s \cdot ?y$ $\cdot i$
  by (simp add: comm-semiring-class.distrib)
also have $\ldots \leq sr \cdot ?y$ $\cdot i + if i \cdot ?y$ $\cdot i$
proof (rule add-left-mono[OF mult-right-mono])
  show $0 \leq ?y$ $\cdot i$ using y-pos[rule-format, of $i$] by auto
  show eps $\leq if i$ unfolding eps-def by (rule Min-le, auto)
qed
also have $\ldots = (A \cdot v \cdot ?y) \cdot i$ using sr-pos y-pos[rule-format, of $i$]
by simp
finally
show (($sr + eps$) $\cdot s \cdot ?y$) $\cdot i \leq (A \cdot v \cdot ?y) \cdot i$ .
qed
from rho-le-x-Ax-imp-rho-le-rx[of y le]
have $?y \geq sr + eps$ .
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 z X 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: $\alpha \ast s \, v = cA \ast v \, v \neq 0$
using ev[unfolded eigen-vector-def] by auto

lemma ev-alpha-norm-v: norm-v $(\alpha \ast s \, v) = (norm \, \alpha \ast s \, norm-v \, v)$
by (auto simp: map-vector-def norm-mult vec-eq-iff)

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

lemma ev-le-vec: le-vec (norm $\alpha \ast s \, norm-v \, v)$ $(A \ast v \, norm-v \, v)$
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 $\alpha \leq r \, (norm-v \, v) \, r \, (norm-v \, v) \leq sr$
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 $\alpha \leq r \, (norm-v \, v)$ .
  from sr-max[OF v]
  show r $\, (norm-v \, v) \leq sr$ .
qed

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

lemma eigen-value-norm-sr: assumes eigen-value $cA \, \alpha$
shows norm $\alpha \leq sr$
using eigen-vector-norm-sr[of - $\alpha$] 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 ∈ ℝ ⇒ Re a ≥ 0 ⇒ 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
}

67
thus \textit{thesis} by (auto simp: vec-eq-iff)

qed

lemma pos-matrix-mult-0: assumes \( \land \ i \ j \cdot B \$ i \$ j \geq 0 \)
and lt: \( \llvec 0 \ x \)
and 0: \( B \ast v \ x = 0 \)
shows \( B = 0 \)
proof –
{ 
  \fix \ i \ j 
  assume \( B \$ i \$ j \neq 0 \)
  with le have gt: \( B \$ i \$ j > 0 \) by (auto simp: order.strict_iff_order)
  have 0 = \((B \ast v \ x) \$ i\) unfolding 0 by simp
  also have \( \ldots = (\sum_{j \in \text{UNIV}}. B \$ i \$ j \ast x \$ j) \)
  unfolding matrix-vector-mult-def by simp
  also have \( \ldots > 0 \)
  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 \( \land \ i \ j \cdot 0 \leq B \$ i \$ j \land B \$ i \$ j \leq A \$ i \$ j \)
and AB: \( A \neq B \)
and ev: \( \text{eigen-value (map-matrix } c \ B) \ \sigma \)
shows \( c \text{mod } \sigma < sr \)
proof –
  let \( ?B = \text{map-matrix } c \ B \)
  let \( ?sr = \text{spectral-radius } ?B \)
  define \( \sigma \) where \( \sigma = ?sr \)
  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 \( \sigma \)-def]
  obtain \( x \) where \( \text{ev-sr: eigen-vector } ?B \ x \ (c \ \sigma) \)
  and rnn: \( \text{real-non-neg-vec } x \) by auto
  define \( y \) where \( y = \text{norm-v } x \)
  from rnn have xy: \( x = \text{map-vector } c \ y \)
  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 \( \sigma \)-def]
  have sigma-sigma: \( \text{cmod } \sigma \leq \sigma \).
  from ev-sr[unfolded xy of-real-hom.eigen-vector-hom]
  have ev-B: \( \text{eigen-vector } B \ y \ \sigma \).
  from ev-B[unfolded eigen-vector-def] have ev-B': \( B \ast v \ y = \sigma \ast s \ y \) by auto
  have ypos: \( y \$ i \geq 0 \) for \( i \) unfolding y-def by (auto simp: map-vector-def)
  from ev-B this have \( y \in X \) unfolding eigen-vector-def X-def by auto

68
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 . . . ≤ sr using y by (rule sr-max)
finally have sig-le-sr: σ ≤ sr .
{
  assume σ = sr
  hence r-sr: r y = sr and sr-sig: sr = σ using (σ ≤ r y, r y ≤ 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
}

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)
  by auto
  from ev-small[OF this] have abs rho < sr 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)
  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)
\{ 

\textbf{define} \( p \) \textbf{where} \( p = \text{charpoly} \ ?A \)
\textbf{assume} \( \text{pos} : ?\text{pos} < 0 \)
\textbf{hence} \( \text{neg} : \text{poly} \ p \ ?s r < 0 \) \textbf{unfolding} \( p\text{-def} \) \textbf{by} \textit{auto}
\textbf{from} \( \text{degree-monic-charpoly} \ [\text{of} \ ?A] \) \textbf{have} \( \text{monic} \ p \ \text{and} \) \text{deg} : \text{degree} \( p \neq 0 \)
\textbf{unfolding} \( p\text{-def} \) \textbf{by} \textit{auto}
\textbf{let} \( ?f = \text{poly} \ p \)
\textbf{have} \( \text{cont} : \text{continuous-on} \ \{a..b\} \ ?f \ \text{for} \ a \ b \) \textbf{by} \textit{(auto intro: continuous-intros)}
\textbf{from} \( \text{mon} \) \textbf{have} \( \text{lc} : \text{lead-coeff} \ p > 0 \) \textbf{by} \textit{auto}
\textbf{from} \( \text{poly-pinfty-ge} \ [\text{OF} \ this \ \text{deg}, \ \text{of} \ 0] \) \textbf{obtain} \( z \) \textbf{where} \( \text{lez} : \text{\( \forall x \geq \)} x \ \text{z \ \leq \} \ x = \Rightarrow \ 0 \)
\textbf{define} \( y \) \textbf{where} \( y = \max \ z \ ?s r \)
\textbf{have} \( \text{yr} : y \geq \ ?s r \ \text{and} \ y \geq \ z \) \textbf{unfolding} \( y\text{-def} \) \textbf{by} \textit{auto}
\textbf{from} \( \text{lez} \ [\text{OF} \ this \ (2)] \) \textbf{have} \( \text{y0} : ?f \ y \geq 0 \).
\textbf{from} \( \text{IVT}' \ [\text{of} \ ?f, \ \text{OF} \ le \ y0 \ yr \ \text{cont}] \) \textbf{obtain} \( x \) \textbf{where} \( \text{ge} : x \geq \ ?s r \ \text{and} \ \text{rt} : ?f \ x = 0 \)
\textbf{unfolding} \( p\text{-def} \) \textbf{by} \textit{auto}
\textbf{hence} \( \text{eigen-value} \ A x \) \textbf{unfolding} \( p\text{-def} \) \textbf{(simp add: eigen-value-root-charpoly)}
\textbf{from} \( \text{ev-small-real} \ [\text{OF} \ this] \) \textbf{ge have} \( \text{False by auto} \}
\}
\textbf{with} \( \text{pos0} \) \textbf{show} \( ?\text{thesis by argo} \)
\textbf{qed}

\textbf{lemma} \ \text{multiplicity-sr-1}: \text{order} \ ?s r \ (\text{charpoly} \ A) = 1
\textbf{proof} --
\{ 
\textbf{assume} \( \text{poly} \ (\text{pderiv} \ (\text{charpoly} \ A)) \ ?s r = 0 \)
\textbf{hence} \( 0 = \text{poly} \ (\text{monom} \ 1 \ \text{i} \ \text{pderiv} \ (\text{charpoly} \ A)) \ ?s r \) \textbf{by} \textit{simp}
\textbf{also have} \ldots \ = \text{sum} \ (\lambda i. \ \text{poly} \ (\text{charpoly} \ (\text{erase-mat} \ A \ i \ i)) \ ?s r) \ \text{UNIV}
\textbf{unfolding} \( \text{pderiv-char-poly-erase-mat} \ \text{poly-sum} \ldots \)
\textbf{also have} \ldots \ > 0
\textbf{by} \ (\text{rule sum-pos, (force simp: charpoly-erase-mat-sr)})
\textbf{finally have} \( \text{False by simp} \}
\}\n\textbf{hence} \( \text{nZ}: \text{poly} \ (\text{pderiv} \ (\text{charpoly} \ A)) \ ?s r \neq 0 \) \textbf{and} \( \text{nZ'}: \text{pderiv} \ (\text{charpoly} \ A) \neq 0 \)
\textbf{by} \textit{auto}
\textbf{from} \( \text{eigen-vector-z-sr} \) \textbf{have} \( \text{eigen-value} A \ ?s r \) \textbf{unfolding} \( \text{eigen-value-def} \ldots \)
\textbf{from} \( \text{this}\ [\text{unfolded eigen-value-root-charpoly}] \)
\textbf{have} \( \text{poly} \ (\text{charpoly} \ A) \ ?s r = 0 \).
\textbf{hence} \( \text{order} \ ?s r \ (\text{charpoly} \ A) \neq 0 \) \textbf{unfolding} \( \text{order-root} \) \textbf{using} \( \text{nZ'} \) \textbf{by} \textit{auto}
\textbf{from} \( \text{order-pderiv} \ [\text{OF} \ \text{nZ'} \ this] \) \textbf{order-0I} [\text{OF} \ \text{nZ}]
\textbf{show} \( ?\text{thesis by simp} \)
\textbf{qed}

\textbf{lemma} \ \text{sr-spectral-radius}: \( \text{sr} = \text{spectral-radius} \ cA \)
\textbf{proof} --
\textbf{from} \( \text{eigen-vector-z-sr-c} \) \textbf{have} \( \text{eigen-value} \ cA \ (c \ ?s r) \)
\textbf{unfolding} \( \text{eigen-value-def} \) \textbf{by} \textit{auto}
\}

70
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 le-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-mult-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
      order.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 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 \(*\): \(sr \leq mu\).

define \(cc\) where \(cc = (mu + 1)^\cdot N\)

define \(n\) where \(n = N\)

from \(*\) \(sr\)-pos have \(mu; mu \geq 0\) \(mu > 0\) by auto

hence \(cc; cc > 0\) unfolding \(cc\)-def by simp

from \(y\) have \(pow-A-1\) \(y \in pow-A-1 \cdot X\) by auto

from \(Y\)-pos-main[of this] have lt: \(0 < (A1n \cdot v \cdot y) \cdot i\) for \(i\) by (simp add: pow-A-1-def)

have \(le\): \(le\)-vec \((A1n \cdot v \cdot y)\) \((cc \cdot s\cdot y)\) unfolding \(cc\)-def \(A1n\)-def \(n\)-def[symmetric]

proof (induct \(n\))

\[\text{case (Suc \(n\))}\]

have id': \(matpow (A + mat 1)\) \(n\)

let \(?An = matpow (A + mat 1)\) \(n\)

let \(?mu = (mu + 1)\)

have \(id'\): \(matpow (A + mat 1) (Suc n)\) \(\cdot v \cdot y = A \cdot v \cdot (?An \cdot v \cdot y) + (?An \cdot v \cdot y)\)

(is \(?a = (?b + ?c)\)

by (simp add: matrix-add-id-distrib matrix-mult-rid matrix-add-vec-distrib matpow-1-commute)

\[
\text{matrix-vec-mul-assoc[symmetric]}\]

have \(le\)-vec \(?b\) \((?mu \cdot n \cdot s\cdot (A \cdot v \cdot y))\)

using \(le\)-vec-mono-left[of \(OF\) nonneg \(Suc\)] by (simp add: algebra-simps vec.scale)

moreover have \(le\)-vec \((?mu \cdot n \cdot s\cdot (A \cdot v \cdot y))\) \((?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y))\)

using \(le\)-vec by auto

moreover have id: \(?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y) = (?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y))\) by simp

from \(le\)-vec-trans[of \(OF\) calculation[unfolded id]]

have le1: \(le\)-vec \(?b\) \((?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y))\).

from \(Suc\) have le2: \(le\)-vec \(?c\) \((mu + 1)\) \(\cdot n \cdot s\cdot y)\).

have le: \(le\)-vec \(?a\) \((?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y) + (?mu \cdot n \cdot s\cdot y))\)

\[
\text{unfolding id' using add-mono[of \(le\)-lib[take-vec-format \(le\)-lib[take-vec-format]] by auto}

have id’': \((?mu \cdot n \cdot s\cdot (mu \cdot s\cdot y) + (?mu \cdot n \cdot s\cdot y) = (?mu \cdot Suc n \cdot s\cdot y)\) by (simp add: algebra-simps)

\[
\text{show \(?case\ using \(le\) unfolding \(id''\).}
\]

qed (simp add: matrix-vec-mul-lid)

have \(lt\): \(0 < cc \cdot s\cdot y \cdot i\) 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 \(le\)-vec \(0\) \(by\) auto

assume \(\ast\): \(mu = sr \lor A \cdot v \cdot y = mu \cdot s\cdot y\)

\[
\text{assume \(A \cdot v \cdot y = mu \cdot s\cdot y\)}
\]

with \(y\) have \(eigen\)-vec-A \(mu\) unfolding \(X\)-def \(eigen\)-vec-def by auto

hence \(eigen\)-vec \(c\cdot A\) \((\text{map-vec c (y)}\) \((c \cdot mu)\) unfolding \(\text{of-real-hom.eigen\)-vec-hom\}

from \(eigen\)-vec-norm-prof[of this] \(*\) have \(mu = sr\) by auto

\[
\text{with \(\ast\) have \(mu-sr\) \((mu = sr)\) by auto}
\]

from eqlib[folded vec-mult-distrib]

have \(0\): \(?w \cdot v \cdot (sr \cdot s\cdot y - A \cdot v \cdot y) = 0\)

\[
\text{unfolding \(matvec\)-vec-right-distrib-diff by simp}
\]

have le: \(le\)-vec \(0\) \((sr \cdot s\cdot y - A \cdot v \cdot y)\) using \(assms(2)[\text{unfolded mu-sr}]\) by auto

have \(sr \cdot s\cdot y - A \cdot v \cdot y = 0\) using \(\text{pos-rowvec-mul-0[of \(OF\) \(w(2)\) \(0\) \(le\)]}\).
hence $ev\cdot y: A \cdot v \cdot y = sr \cdot s \cdot y$ by auto

show $mu = sr \land A \cdot v \cdot y = mu \cdot s \cdot y$ using $ev\cdot y \cdot mu\cdot sr$ by auto

qed

lemma nonnegative-eigenvector-has-ev-sr: assumes eigen-vector $A \cdot v \cdot mu$ and $le:$

shows $mu = sr$

proof –

from assms(1)[unfolded eigen-vector-def] have $v: v \neq 0$ and $ev: A \cdot v \cdot v = mu \cdot s \cdot v$ by auto

from $le \cdot v$ have $v: v \in X$ unfolding $X$-def by auto

from $ev$ have $le$-vec $(A \cdot v \cdot v) (mu \cdot s \cdot v)$ by auto

from $le$-vec-$A$-$mu$[OF $v$ this] ev show $?thesis$ by auto

qed

lemma similar-matrix-rotation: assumes $ev$: eigen-value $cA \cdot \alpha$ and $\alpha$: $cmod \cdot \alpha = sr$

shows similar-matrix $(cis \cdot (arg \cdot \alpha) \cdot k \cdot cA) \cdot cA$

proof –

from $ev$ obtain $y$ where $ev$: eigen-vector $cA \cdot y \cdot \alpha$ unfolding eigen-value-def by auto

let $?y = norm\cdot v \cdot y$

note maps = map-vector-def map-matrix-def

define $yp$ where $yp = norm\cdot v \cdot y$

let $?yp = map\cdot vector \cdot c \cdot yp$

have $yp$: $yp \in X$ unfolding $yp$-def by (rule $norm\cdot v\cdot X$[OF $ev$])

from $ev$[unfolded eigen-vector-def] have $ev\cdot y$: $cA \cdot v \cdot y = \alpha \cdot s \cdot y$ by auto

from $ev$-le-vec[OF $ev$, unfolded $\alpha$, folded $yp$-def]

have $1$: $le$-vec $(sr \cdot s \cdot yp) (A \cdot v \cdot yp)$ by simp

from rho-le-vec-$Ax$-imp-rho-le-rx[OF $yp$ 1] have $sr \leq r \cdot yp$ by auto

with ev-inequalities[OF $ev$, folded $yp$-def]

have $2$: $r \cdot yp = sr$ by auto

have $ev\cdot yp$: $A \cdot v \cdot yp = sr \cdot s \cdot yp$

and $pos\cdot yp$: $lt$-vec $0 \cdot yp$


define $D$ where $D = diagvector (\lambda j. cis (arg (y \cdot \$ \cdot j)))$

define $inv\cdot D$ where $inv\cdot D = diagvector (\lambda j. cis (- arg (y \cdot \$ \cdot j)))$

have $DD$: $inv\cdot D \cdot D = mat 1 \cdot D \cdot D$ by unfolding $D$-def $inv\cdot D$-def

by (auto simp add: diagvector-eq-mat cis-mult)

{ fix $i$
  have $(D \cdot v \cdot ?yp) \cdot i = cis (arg (y \cdot \$ \cdot i)) \cdot c (cmod (y \cdot \$ \cdot i))$
    unfolding $D$-def $yp$-def by (simp add: maps)
  also have $\ldots = y \cdot \$ \cdot i$ by (simp add: cis-mult-cmod-id)
  also note calculation
}

hence $y\cdot D$-$yp$: $y = D \cdot v \cdot ?yp$ by (auto simp: vec-eq-iff)

define $\varphi$ where $\varphi = arg \cdot \alpha$

73
let \( \varphi = \text{cis} (\varphi) \)

have [simp]: \( \text{cis} (\varphi) \) * \( \text{rcis} \) \( \varphi = \text{sr} \) unfolding \( \text{cis-rcis-eq} \ \text{rcis-mult} \) by simp

have \( \alpha \): \( \alpha = \text{rcis} \) \( \varphi \) unfolding \( \varphi\)-def \( \alpha\)-[symmetric] \( \text{rcis-cmod-arg} \) ..

define \( F \) where \( F = R \varphi \) \( k \) \( (\text{inv-D} \star \star \alpha \star \star D) \)

have \( \alpha \star \star V \) \( D \star \star ?y \) = \( \alpha \star \star Y \) unfolding \( \alpha\)-[symmetric] \( \alpha\)-[symmetric] \( \text{ev-y} \) by simp

also have \( \text{inv-D} \star \star V \) .. = \( \alpha \star \star ?y \)

unfolding \( \text{vector-smult-distrib} \ \text{y-D-yp} \) \( \text{matrix-vector-mult-assoc} \ \text{DD} \) \( \text{matrix-vector-mult-lid} \)

unfolding \( \text{maps} \ \text{matrix-vector-mult-def} \)

... have \( \varphi \star \star V \) .. = \( \text{sr} \star \star ?y \) unfolding \( \alpha \) by simp

also have .. = \( \text{cA} \star \star ?y \) unfolding \( \alpha\)-[symmetric] \( \text{vec-eq-iff} \) by (auto simp: \( \text{maps} \)

also have .. = \( \text{cA} \star \star V \) \( ?y \) unfolding \( \alpha\)-[symmetric] \( \text{vec-eq-iff} \) by (auto simp: \( \text{maps} \)

finally have \( F: F \star \star ?y \) = \( \alpha \star \star ?y \) unfolding \( \alpha\)-[symmetric] \( \alpha\)-[symmetric] \( \text{vec-eq-iff} \) by (auto simp: \( \text{maps} \)

unfolding \( \text{matrix-vector-mult-assoc} \ \text{symmetric} \) \( \text{vec-smult-distrib} \).

have prod: \( \text{inv-D} \star \star \alpha \star \star D = (\chi i. j. \text{cis} (-\text{arg} (y \$ i)) \star \alpha \star \star \$ i \$ j \star \text{cis} 

(arg (y \$ j))) \)

unfolding \( \text{inv-D} \star \star \alpha \star \star D \) = diagvector-mul-right diagvector-mul-left by simp

{ fix \( i j \)

have \( \text{cmod} \) \( F \star \star i \$ j \) = \( \text{cmod} \) \( \varphi \star \star \alpha \star \star \$ h i \$ h j \star \) \( \text{cis} \) (-\text{arg} (y \$ h i)) \star \( \text{cis} \) \( \) (arg (y \$ h j))

unfolding \( \alpha\)-[symmetric] \( \alpha\)-[symmetric] \( \text{vec-lambda-beta} \) \( \text{matrix-vector-mult-def} \)

by (simp only: ac-simps)

also have .. = \( \alpha \star \star \alpha \star \star \$ y \$ j \) unfolding \( \text{cis-mult} \) \( \text{unfolding} \) \( \text{norm-mult} \) by simp

also note calculation
}

hence \( FA: \text{map-matrix norm} \) \( F = \text{A} \) unfolding \( \text{maps} \) by auto

let \( \varphi = \text{map-matrix} \) \( \alpha \) (map-matrix \( \text{norm} \) \( F \))

let \( \text{Re} = \text{map-matrix} \) \( \text{Re} \)

from \( F[\text{folded FA}] \) have \( 0 : ?G \star \star \$ y \$ p = 0 \) unfolding \( \text{matrix-diff-vect-distrib} \)

by simp

have \( \text{Re} \) \( ?G \star \star \$ y \$ p = \text{map-matrix} \) \( \text{Re} \) \( ?G \star \star \$ y \$ p \)

unfolding \( \text{maps} \) \( \text{matrix-vector-mult-def} \) \( \text{vec-lambda-beta} \) \( \text{Re-sam} \) by auto

also have .. = \( 0 \) unfolding \( \text{Re} \) by (simp add: vec-eq-iff \( \text{maps} \)

finally have \( 0 : ?G \star \star \) \( \text{Re} \) \( ?G \star \star \$ y \$ p = 0 \).

have \( \text{Re} \) \( ?G = 0 \)

by (rule pos-matrix-mult-0) \( \text{OF} \) \( \text{pos-yp} \) \( 0 \), auto simp: \( \text{maps} \) \( \text{complex-Re-le-cmod} \)

hence \( \text{OF} \) \( F = F \) by (auto simp: \( \text{maps} \) \( \text{vec-eq-iff} \) \( \text{cmod-eq-\text{Re}} \))

with \( FA \) have \( \alpha \): \( \alpha \star \star \$ cA = F \) by simp

from \( \text{arg-cong} \) \( \text{OF} \) \( \text{this} \), \( \text{of} \ \alpha \). \( \alpha \) \( \varphi \) \( \star \star \$ k \) \( A \)

have sim: \( \text{cis} \) \( \varphi \) \( \star \star \$ k \) \( \alpha \star \star \$ \) \( \text{inv-D} \star \star \$ k \) \( \alpha \star \star \$ \) \( \text{D} \) unfolding \( \alpha\)-[symmetric] \( \alpha\)-[symmetric] \( \text{vec-smult-distrib} \)

by simp

have \( \text{similar-matrix} \) \( \alpha \) \( \varphi \) \( \star \star \$ D \alpha \) \( \text{unfolding} \) \( \text{similar-matrix-def} \) \( \text{similar-matrix-wit-def} \)

sim

by (rule exI) \( \text{of} \) \( \text{- inv-D} \), rule exI \( \text{of} \) \( \text{- D} \), auto simp: \( \text{DD} \)

thus \( ?\text{thesis} \) unfolding \( \alpha\)-[symmetric] .

74
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

75
lemma maximal-eigen-value-roots-of-unity-rotation:
assumes $M: M = \{ev :: complex. \text{eigen-value } cA ev \land \text{cmod } ev = sr\}$
and $kM: k = \text{card } M$
shows $k \neq 0$
\begin{align*}
k &\leq \text{CARD}(n) \\
\exists f. \text{charpoly } A = (\text{monom } 1 k - [sr^k]) * f \\
&\land (\forall x. \text{poly } (\text{map-poly } c f) x = 0 \rightarrow \text{cmod } x < sr) \\
M &= (\ast) (c sr) \cdot (\lambda i. (\text{cis } (\text{of-nat } i * 2 * \pi / k))) \cdot \{0 ..< k\} \\
M &= (\ast) (c sr) \cdot \{x :: \text{complex. } x \cdot k = 1\} \\
&M = (\ast) (\text{cis } (2 * \pi / k)) \cdot \text{Spectrum } cA = \text{Spectrum } cA
\end{align*}
unfolding $kM$
proof
let $?M = \text{card } M$
\begin{itemize}
  \item note $\text{fin} = \text{finite-spectrum}[\text{of } cA]$
  \item note $\text{char} = \text{degree-monic-charpoly}[\text{of } cA]$
  \item have $?M \leq \text{card } (\text{Collect } (\text{eigen-value } cA))$
    by (rule $\text{card mono}[\text{OF } \text{fin}], \text{unfold } M, \text{auto})$
  \item also have $\text{Collect } (\text{eigen-value } cA) = \{x. \text{poly } (\text{charpoly } cA) x = 0\}$
    unfolding $\text{eigen-value-root-charpoly}$ by auto
  \item also have $\text{card } ... \leq \text{degree } (\text{charpoly } cA)$
    by (rule $\text{poly-roots-degree}, \text{insert char}, \text{auto})$
  \item also have $... = \text{CARD}(n)$ using $\text{char}$ by simp
  \item finally show $?M \leq \text{CARD } (n)$.
  \item from $\text{finite-subset}[\text{OF } \text{- fin}, \text{of } M]$
    have $\text{finM: finite } M$ unfolding $M$ by blast
  \item from $\text{finite-distinct-list}[\text{OF this}]$
    obtain $m$ where $\text{Mm: } M = \text{set } m$ and $\text{dist: } \text{distinct } m$ by auto
  \item from $\text{Mm dist have card: } ?M = \text{length } m$ by (auto simp: $\text{distinct-card}$)
    have $sr: sr \in \text{set } m$ using $\text{eigen-value-sr-c sr-pos}$ unfolding $\text{Mm}[\text{symmetric}]$ $M$
    by auto
  \item define $s$ where $s = \text{sort-key arg } m$
  \item define $a$ where $a = \text{map arg } s$
  \item let $?k = \text{length } a$
  \item from $\text{dist Mm card } sr$ have $s: M = \text{set } s$ distinct $s$ $sr \in \text{set } s$
    and $\text{card: } ?M = ?k$
    and $\text{sorted: } \text{sorted } a$
    unfolding $\text{s-def a-def}$ by auto
  \item have $\text{map-s: } \text{map } ((\ast) (c sr)) (\text{map cis } a) = s$ unfolding $\text{map-map o-def a-def}$
  \item proof
    \begin{itemize}
      \item fix $x$
      \item assume $x \in \text{set } s$
      \item from $\text{this[folded s(1), unfolded } M]$
    \end{itemize}
\end{itemize}
have id: cmod x = sr by auto 

show sr * cis (arg x) = x 
by (subst (5) cis-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-split: a = aa ≠ a' unfolding a-def by (cases s, auto)
from arg-bounded have bounded: x ∈ set a → pi < x ∧ x ≤ pi for x 
unfolding a-def by auto 
from bounded[of aa, unfolded a-split] have aa: - pi < aa ∧ aa ≤ pi by auto
let ?aa = aa + 2 * pi

define args where args = a @ [?aa]
define ?aa where ?aa = Suc i - args ! i 

have bnd: x ∈ set a → x < ?aa for x using aa bounded[of x] by auto
hence aa-a: ?aa ∈ set a by fast 

have sorted: sorted args unfolding args-def using sorted unfolding sorted-append by (insert bnd, auto simp: order.strict-iff-order)

have dist: distinct args using a aa-a unfolding args-def distinct-append by auto
have sum: (\sum i < ?k. ?diff i) = 2 * pi

unfolding sum-lessThan-Telescope args-def a-split by simp

have k: ?k ≠ 0 unfolding a-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 \(?k + ?Min = (\sum i < ?k. ?Min) / ?k by (rule divide-right-mono[OF sum-mono[OF Min-le]], auto)
also have \(?k = 2 * pi / ?k 

finally have Min: Min ≤ 2 * pi / \(k unfolding Min-def by auto

have lt: i < ?k implies args ! i < args ! (Suc i) for i 
using sorted[unfolded sorted-iff-nth-mono, rule-format, of i Suc i]
dist[unfolded distinct-cone-nth, rule-format, of i Suc i] by (auto simp: args-def)

let ?c = λ i. rcis sr (args ! i)

have hda[simp]: hda a = aa unfolding a-split by simp
have Min0: Min > 0 using lt unfolding Min-def by (subst Min-gr-iff, insert k, auto)

have Min-A: Min ∈ ?A unfolding Min-def by (rule Min-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-def nth-append rcis-def)
also have \(i ∈ set (map (rcis sr) (a @ [hd a])) using i 

unfolding args-def set-map unfolding set-cone-nth by auto
also have \(i = rcis sr * set a unfolding a-split by auto
also have \(i = M unfolding s(1) map-s[symmetric] set-map image-image by (rule image-cong[OF refl], auto simp: rcis-def)

77
finally have \( c \in M \) by auto 

\[
\begin{align*}
\text{note} \quad ciM &= this \\
\{ \\
\text{fix} \quad i :: \text{nat} \\
\text{assume} \quad i < ?k \\
\text{hence} \quad i < \text{length args Suc} \quad \text{unfolding args-def by auto} \\
\text{from} \quad \text{maximal-eigen-values-group[OF M ciM[OF this(2)] ciM[OF this(1)]]} \\
\text{have} \quad \text{rcis sr (?diff i) \in M by simp} \\
\}
\end{align*}
\]

hence \( \text{Min-M: rcis sr Min \in M using Min-A by force} \)

have \( \text{rcisM: rcis sr (of-nat n \ast Min) \in M for n} \)

proof (induct n)

next

case (Suc n)

\[
\begin{align*}
\text{have} \quad \ast: \text{rcis sr (of-nat (Suc n) \ast Min) = rcis sr (of-nat n \ast Min) \ast cis Min} \\
\text{by} \quad \text{(simp add: rcis-mult ring-distrib add.commute)} \\
\text{from} \quad \text{maximal-eigen-values-group(1)[OF M Suc Min-M]} \\
\text{show} \quad \text{\?case unfolding \ast by simp} \\
\end{align*}
\]

qed

let \( \text{\?list = map (rcis sr) (map (\lambda i. of-nat i \ast Min) [0 ..< ?k])} \)

define list where list = \?list

have \( \text{sr-pos have sr0: sr \neq 0 by auto} \)

\[
\begin{align*}
\{ \\
\text{fix} \quad i \\
\text{assume} \quad i < ?k \\
\text{hence} \quad \ast: \text{0 \leq real i \ast Min using Min0 by auto} \\
\text{from} \quad \text{i have real i < real ?k by auto} \\
\text{from} \quad \text{mult-strict-right-mono[OF this Min0]} \\
\text{have real i \ast Min < real ?k \ast Min by simp} \\
\text{also have} \quad \ldots \leq \text{real \?k \ast (2 \ast pi / real \?k)} \\
\text{by} \quad \text{(rule mult-left-mono[OF Min], auto)} \\
\text{also have} \quad \ldots = 2 \ast pi \text{ using k by simp} \\
\text{finally have real i \ast Min < 2 \ast pi .} \\
\text{note \ast this} \\
\}
\text{note prod-pi = this} \\
\text{have dist: distinct ?list} \\
\text{unfolding distinct-map[of rcis sr]} \\
\text{proof (rule conj[OF inj-on-subset[OF rcis-inj-on[OF sr0]]])} \\
\text{show distinct (map (\lambda i. of-nat i \ast Min) [0 ..< ?k]) using Min0} \\
\text{by (auto simp: distinct-map inj-on-def)} \\
\text{show set (map (\lambda i. real i \ast Min) [0..<?k]) \subseteq \{0..<2 \ast pi\} using prod-pi by auto} \\
\text{qed} \\
\text{with len have card': card (set ?list) = \?M using distinct-card by fastforce} \\
\text{have listM: set ?list \subseteq M using rcisM by auto} \\
\text{from card-subset-eq[OF finM listM card']} 
\end{align*}
\]
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 ≤ ?k * Min ?k * Min < 2 * pi by auto  
  from rcisM[of ?k, unfolded M-list] have rcis sr (?k * Min) ∈ 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, unfolded M-list 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 ≠ 0 unfolding card using k by auto

note M-list

also have set ?list = ( * ) (c sr) ' (λ i. cis (of-nat i * 2 * pi / ?M))' {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 ^ 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 ≠ 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 ∈ ?L ⇒ x ∈ ?R for x using max(2)[of x] phi unfolding Spectrum-def by auto
  }

moreover

  {  
    assume x ∈ ?R
  }

79
hence eigen-value \( cA \) 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]
} note this *
} thus \( ? \)thesis by blast qed
from this[unfolded phi-def]
show ( \( * \) ) (cis (2 * pi / real (card M))) \( ^{\cdot} \) Spectrum \( cA = \) Spectrum \( cA \).
let \( ?p = \) monom 1 \( k - \) [:sr\(^{-k} \):]
let \( ?cp = \) monom 1 \( k - \) [:\((c \text{ sr})\)^{\cdot}k:]
let \( ?\text{one} = 1 :: \) complex
let \( ?\text{list} = \) map (rcis \( \text{sr} \)) (map (\( \lambda \) i. of-nat i * \( \text{?piM} \)) [0 ..< card M])
interpret c: field-hom c ..
interpret p: map-poly-inj-idom-divide-hom c ..
have \( \text{cp}: ?\text{cp} = \) map-poly c \( ?p \) by (simp add: hom-distribs)
have M-list: \( M = \) set \( ?\text{list} \) using M-list[unfolded Min card[symmetric]].
have dist: distinct \( ?\text{list} \) using dist[unfolded Min card[symmetric]].
have k\( 0: k \neq 0 \) using k[folded card] assms by auto
have \( ?\text{cp} = (\) monom 1 \( k + (- [:\((c \text{ sr})\)^{\cdot}k:]) \) by simp
also have degree .. = \( k \)
by (subst degree-add-eq-left, insert k\( 0 \), auto simp: degree-monom-eq)
finally have deg: degree \( ?\text{cp} = k \).
from deg k\( 0 \) have cp0: \( ?\text{cp} \neq 0 \) by auto
have \( \{x. \text{poly} \ ?\text{cp} x = 0\} = \{x. x^{\cdot}k = (c \text{ sr})^{\cdot}k\} \) unfolding poly-diff poly-monom
by simp
also have .. \( \subseteq M \)
proof –
\{ 
fix \( x \)
assume id: \( x^{\cdot}k = (c \text{ sr})^{\cdot}k \)
from sr-pos k\( 0 \) have \( (c \text{ sr})^{\cdot}k \neq 0 \) by auto
with arg-conq[OF id, of \( \lambda x . x / (c \text{ sr})^{\cdot}k \)]
have \( (x / c \text{ sr})^{\cdot}k = 1 \)
unfolding power-divide by auto
hence c sr \( \ast (x / c \text{ sr}) \in M \)
by (subst M-pow, unfold kM[symmetric], blast)
also have c sr \( \ast (x / c \text{ sr}) = x \) using sr-pos by auto
finally have \( x \in M \).
\}
thus \( ? \)thesis by auto qed
finally have cp-M: \( \{x. \text{poly} \ ?\text{cp} x = 0\} \subseteq M \).
have \( k = \) card (set \( ?\text{list} \)) unfolding distinct-card[OF dist] by (simp add: kM)
also have .. \( \subseteq \) card \( \{x. \text{poly} \ ?\text{cp} x = 0\} \)
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 * πM) 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 ⊆ {x. poly ?cp x = 0} by auto
qed

finally have k-card: k ≤ 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 dvd: ?cp dvd charpoly cA
proof (rule poly-roots-dvd[OF cp0 deg k-card])
  from cp-M show {x. poly ?cp x = 0} ⊆ {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 cA.
from this unfolded dvd-def obtain f where
decomp: charpoly cA = ?p * f by blast
let ?f = map-poly c f
  have decomp: 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 decomp p.hom-mult eigen-value-root-charpoly by auto
  hence le: cmod x ≤ 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 decomp power2-eq-square .
    from order-max[OF this 0] maximal-eigen-value-order-1[OF ev max]
    have False by auto
  }
}
81
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 \neq 0} \cap U \times U)

definition irreducible-mat where
  irreducible-mat A = (let n = dim-row A in
  (\forall i. i < n \rightarrow j < n \rightarrow (i,j) \in (graph-of-mat A)^-+))

definition nonneg-irreducible-mat A = (nonneg-mat A \land irreducible-mat A)

  Next, we have to install transfer rules

context
  includes lifting-syntax
begin

lemma HMA-irreducible[transfer-rule]: ((HMA-M \ :: \ - \ \Rightarrow \ ^n \ ^n \Rightarrow -) \ \Rightarrow 
(=))
  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
  $\exists\ k\ f.\ k\ ≠\ 0\ ∧\ k\ ≤\ n\ ∧\ char-poly\ A = (monom\ 1\ k - [sr \cdot k;]) \star f$ ∧
  (\ ∀\ x. poly (map-poly complex-of-real f) x = 0 −→ cmod x < sr)
proof (atomize (full), goal-cases)
We now need permutations on matrices to show that a matrix if a ma-
trix 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))

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

lemma permutation-mat-row[simp]: p permutes {..<n} =⇒ i < n =⇒
Matrix.row (permutation-mat n p) i = unit-vec n (inv p i)

lemma permutation-mat-col[simp]: p permutes {..<n} =⇒ i < n =⇒
Matrix.col (permutation-mat n p) i = unit-vec n (p i)

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} 
proof 
  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} 
proof 
  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 
proof 
  unfolding permutation-mat-left[OF mat-carrier p] 
  by ( subst permutation-mat-right[OF - permutes-inv[OF p], of - n], force, insert

85
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 \( \text{dim}[\text{simp}]: \text{dim-row} \ A_1 = n \ \text{dim-col} \ A_1 = n \)
\( \text{dim-row} \ A_4 = m \ \text{dim-col} \ A_4 = m \) by auto

show \?thesis

unfolding char-poly-matrix-def four-block-mat-def Let-def dim
by (rule eq-matI, insert \( A_2 \) \( A_3 \), auto)

qed

lemma char-poly-four-block-mat-lower-left-zero: fixes \( A :: 'a :: \text{idom mat} \)
assumes \( A :: A = \text{four-block-mat} B \ C (0 \ m \ n) \ D \)
and \( B :: B \in \text{carrier-mat} n \ n \)
and \( C :: C \in \text{carrier-mat} n \ m \)
and \( D :: D \in \text{carrier-mat} m \ m \)
shows \( \text{char-poly} A = \text{char-poly} B \times \text{char-poly} D \)

unfolding \( A \) char-poly-def
by (subst char-poly-matrix-four-block-mat \[ \text{OF} \ B \ C - D \], force,
rule det-four-block-mat-lower-left-zero \[ \text{OF} - n - m \], insert \( B \) \( C \) \( D \), auto)

lemma elements-mat-four-block-mat-supseteq: assumes \( A_1 :: A_1 \in \text{carrier-mat} n \ n \)
and \( A_2 :: A_2 \in \text{carrier-mat} n \ m \)
and \( A_3 :: A_3 \in \text{carrier-mat} m \ n \)
and \( A_4 :: A_4 \in \text{carrier-mat} m \ m \)
shows \( \text{elements-mat} (\text{four-block-mat} A_1 A_2 A_3 A_4) \supseteq \)
\( \text{elements-mat} A_1 \cup \text{elements-mat} A_2 \cup \text{elements-mat} A_3 \cup \text{elements-mat} A_4 \)

proof
let \( ?A = \text{four-block-mat} A_1 A_2 A_3 A_4 \)
have \( ?A \in \text{carrier-mat} (n + m) (n + m) \) using \( A_1 A_2 A_3 A_4 \) by simp
from $A1 \ A4$

have \( \text{dim[simp]}: \text{dim-row } A1 = n \ \text{dim-col } A1 = n \)
\( \text{dim-row } A4 = m \ \text{dim-col } A4 = m \) by auto

fix $x$

assume $x \in \text{elements-mat } A1 \cup \text{elements-mat } A2 \cup \text{elements-mat } A3 \cup \text{elements-mat } A4$

\{ 
  assume $x \in \text{elements-mat } A1$
  from this[unfolded elements-mat] $A1$ obtain $i \ j$ where $x = A1 (i, j)$
  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 $x \in \text{elements-mat } ?A$ by auto
\}

moreover
\{ 
  assume $x \in \text{elements-mat } A2$
  from this[unfolded elements-mat] $A2$ obtain $i \ j$ where $x = A2 (i, j)$
  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 $x \in \text{elements-mat } ?A$ by auto
\}

moreover
\{ 
  assume $x \in \text{elements-mat } A3$
  from this[unfolded elements-mat] $A3$ obtain $i \ j$ where $x = A3 (i, j)$
  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 $x \in \text{elements-mat } ?A$ by auto
\}

moreover
\{ 
  assume $x \in \text{elements-mat } A4$
  from this[unfolded elements-mat] $A4$ obtain $i \ j$ where $x = A4 (i, j)$
  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 $x \in \text{elements-mat } ?A$ by auto
\}

ultimately show $x \in \text{elements-mat } ?A$ using $x$ by blast

qed

lemma non-irreducible-mat-split:
fixes $A :: 'a :: 
 assumes $A :: A \in \text{carrier-mat } n \ n$
 and not: $\neg \text{irreducible-mat } A$
 and $n :: n > 1$
 shows $\exists n1 n2 B B1 B2 B4$. similar-mat $A B \land \text{elements-mat } A = \text{elements-mat } B$
 where $B = \text{four-block-mat } B1 B2 (0_m n2 n1) B4 \land$
 $B1 \in \text{carrier-mat } n1 n1 \land B2 \in \text{carrier-mat } n1 n2 \land B4 \in \text{carrier-mat } n2$
 $n2 \land$
 $0 < n1 \land n1 < n \land 0 < n2 \land n2 < n \land n1 + n2 = n$

proof -
from $A$ have [simp]: $\text{dim-row } A = n$ by auto
let $\exists G = \text{graph-of-mat } A$
let $\exists \text{reachp} = \lambda i j. (i,j) \in \exists G^*+$
let $\exists \text{reach} = \lambda i j. (i,j) \in \exists G^*$

have $\exists i j. i < n \land j < n \land \neg \exists \text{reach } i j$
proof (rule ccontr)
  assume $\neg \exists \text{thesis}$
  hence reach $\land i j. i < n \implies j < n \implies \exists \text{reach } i j$ by auto
from not[unfolded irreducible-mat-def Let-def]
  obtain $i j$ where $i :: i < n$ and $j :: j < n$ and \text{nreach: $\neg \exists \text{reachp } i j$ 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 \text{diff: $j \neq k$ by auto}
from reach[OF $O f j k$] diff reach[OF $O f k j$]
  have $\exists \text{reachp } j j$ 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 \text{nreach: $\neg \exists \text{reach } i j$ by auto}

define $I$ where $I :: \{k. k < n \land \neg \exists \text{reach } i k\}$

have $I :: i \in I$ unfolding I-def using nreach $i$ by auto

have $j :: j \notin I$ unfolding I-def using nreach $j$ by auto

define $f$ where $f :: (\lambda i. i f i \in I \then I \else 0 :: \text{nat})$

let $\exists xs :: [0 ..< n]$

from mset-eq-permutation[OF mset-sort, of $\exists xs f$] obtain $p$ where $p :: p \text{ permutes }$\{..< n\}
  and perm: \text{permute-list } p \exists xs = \text{sort-key } f \exists xs$ by auto
from $p$ have $\text{lt[simp]: } i :: i < n \implies i < n$ for $i$ by (rule permutes-lt)

let $p :: \text{inv } p$.

have $ip :: \forall p \text{ permutes }$\{..< n\} using permutes-inv[OF $p$] .

from ip have $\text{lt[simp]: } i :: i < n \implies \neg p i < n$ for $i$ by (rule permutes-lt)

let $B :: \text{Matrix}$.mat $n n (\lambda (i,j). A \$$ (p i, p j))$

define $B$ where $B :: B$

from permutation-similar-mat[OF $A B$] have sim: similar-mat $A B$ unfolding B-def .

let $\exists ys :: \text{permute-list } p \exists xs$

define $ys$ where $ys :: \exists ys$

have $\text{len-ys: } \text{length } ys = n$ unfolding $\exists ys$ by simp

let $\exists k :: \text{length } (\text{filter } (\lambda i. i f i = 0) \ exists)$

define $k$ where $k :: \exists k$

89
have kn: \( k \leq n \) unfolding \( k\)-def using \( \text{len-ys} \)

using \( \text{length-filter-le[of - ys]} \) by auto

have ys-p: \( i < n \implies \text{ys ! i} = p \) \( i \) for \( i \) unfolding \( \text{ys-def permute-list-def} \) by simp

have ys: \( ys = \text{map} (\lambda i. \text{ys ! i}) [0 ..< n] \) unfolding \( \text{len-ys}[\text{symmetric}] \)

by (simp add: \( \text{map-nth} \))

also have \( \ldots = \text{map} \ [0 ..< n] \)

by (rule \( \text{map-cong} \), insert \( \text{ys-p} \), auto)

also have \( [0 ..< n] = [0 ..< k] @ [k ..< n] \) using \( \text{kn} \)

using \( \text{le-Suc-ex upt-add-eq-append} \) by blast

finally have ys: \( ys = \text{map} \ [0 ..< k] @ \text{map} \ [k ..< n] \) by simp

\{

fix \( i \)

assume \( i: i < n \)

let \( ?g = (\lambda i. f i = 0) \)

let \( ?f = \text{filter} \ ?g \)

from \( i \) have \( p i: p i < n \) using \( p \) by simp

have k = length (\( ?f \) \( \text{ys} \)) by fact

also have \( ?f \text{ys} = ?f \ (\text{map} \ [0 ..< k]) \oplus ?f \ (\text{map} \ [k ..< n]) \) unfolding \( \text{ys} \)

by simp

also note \( k = \text{calculation} \)

finally have \( \text{True by blast} \)

from \( \text{perm}[\text{symmetric}, \text{folded \( \text{ys-def} \)]} \)

have \( \text{sorted} \ (\text{map} \ p \ \text{ys}) \) using \( \text{sorted-sort-key} \) by \( \text{metis} \)

from \( \text{this}[\text{unfolded \( \text{ys-map-append sorted-append set-map} \)]} \)

have sorted: \( \text{\( \forall x y. x < k \implies y \in \{k..<n\} \implies f \ (p x) \leq f \ (p y) \) by auto} \)

have 0: \( \forall i < k. f \ (p i) = 0 \)

proof (rule ccontr)

assume \( \neg \text{thesis} \)

then obtain \( i \) where \( i: i < k \) and \( \text{zero}: f \ (p i) \neq 0 \) by auto

hence \( f \ (p i) = 1 \) unfolding \( \text{f-def} \) by (auto split: if-splits)

from \( \text{sorted}[\text{OF} \ i, \ \text{unfolded this}] \) have 1: \( j \in \{k..<n\} \implies f \ (p j) \geq 1 \) for \( j \)

by auto

have le: \( j \in \{k ..< n\} \implies f \ (p j) = 1 \) for \( j \) unfolding \( \text{1[of j]} \) unfolding \( \text{f-def} \)

by (auto split: if-splits)

also have \( \neg f \ (\text{map} \ [k ..< n]) = [] \) using \( \text{le by auto} \)

from \( k[\text{unfolded this}] \) have length (\( \neg f \ (\text{map} \ [0..<k])) = k \) by simp

from \( \text{length-filter-less[of p i map p [0 ..< k]} \ ?g, \ \text{unfolded this]} \) i zero

show \( \text{False by auto} \)

qed

hence \( \neg f \ (\text{map} \ [0..<k]) = \text{map} \ [0..<k] \) by auto

from arg-cong[\text{OF} \ k[\text{unfolded this, simplified}, \ \text{of set}]]

have 1: \( \land i. \ i \in \{k ..< n\} \implies f \ (p i) \neq 0 \) by auto

have 1: \( i < n \implies \neg i < k \implies f \ (p i) \neq 0 \) for \( i \) using \( 1[\text{of \ i}] \) by auto

have 0: \( i < n \implies f \ (p i) = 0 \) = (\( i < k \)) for \( i \) using \( 1[\text{of \ i}] \) 0[\text{rule-format, of \ i}] by blast

have main: \( f i = 0 \) = (\( \neg p i \ < k \)) using \( 0[\text{of \ ?p \ i}] \) i p

by (auto simp: \( \text{permutations-inverses} \))

have i \in \( I \iff f i \neq 0 \) unfolding \( \text{f-def} \) by simp

90
also have \((f i = 0) \iff ?p i < k\) using main by auto

finally have \(i \in I \iff ?p i \geq k\) 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 \(i\) \(k\) \(p\), auto simp: permutes-inverses)

from \(i j\) have \(B \$(i, j) = A \$(p i, p j)\) unfolding B-def by auto

also have \(\ldots = 0\)

proof (rule ccontr)

assume \(A \$(p i, p j) \neq 0\)

hence \((p i, p j) \in ?G\) unfolding graph-of-mat-def Let-def using \(i\) \(p\) by auto

with \(ilI\) \(j\) have \(p j \in I\) unfolding I-def by auto

with \(jI\) show False by simp

qed

finally have \(B \$(i, j) = 0\).

\}

note zero = this

have \(\text{dimB[simp]}: \text{dim-row } B = n \text{ dim-col } B = n\) unfolding B-def by auto

have \(\text{dim}: \text{dim-row } B = k + (n - k) \text{ dim-col } B = k + (n - k)\) using \(kn\) by auto

obtain \(B1 B2 B3 B4\) where \(\text{spl: split-block } B k k = (B1, B2, B3, B4)\) (is \(?tmp = -1\)) by (cases \(?tmp\), auto)

from \(\text{split-block[OF this dim]}\) have \(\text{Bs}: B1 \in \text{carrier-mat } k k \text{ B2 } \in \text{carrier-mat } k (n - k) \text{ B3 } \in \text{carrier-mat } (n - k) k \text{ B4 } \in \text{carrier-mat } (n - k) (n - k)\)

and \(\text{B}: B = \text{four-block-mat } B1 B2 B3 B4\) by auto

have \(B3: B3 = 0_{m} (n - k) k\) unfolding arg-cong[OF \(\text{spl}[\text{symmetric}]\), of \(\lambda (\cdot,\cdot,\cdot).\) \(B\), unfolded split]

unfolding split-block-def Let-def split

by (rule eq-matI, auto simp: \(kn\) zero)

from \(\text{elements-mat-permutes[OF } p A\text{ B-def]}\)

have \(\text{elem}: \text{elements-mat } A = \text{elements-mat } B\).

show ?thesis

by (intro exI conjI, rule sim, rule elem, rule \(B[\text{unfolded } B3]\), insert \(Bs\) \(k0\) \(kn\), auto)

qed

lemma non-irreducible-nonneg-mat-split:

fixes \(A::'a::\text{linordered-idom mat}\)

assumes \(A: A \in \text{carrier-mat } n n\)
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
and nonneg-mat A1 ∧ nonneg-mat A2
∧ A1 ∈ carrier-mat n1 n1 ∧ A2 ∈ carrier-mat n2 n2
∧ 0 < n1 ∧ n1 < n ∧ 0 < n2 ∧ n2 < n ∧ 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 ∈ carrier-mat n1 n1 B2 ∈ carrier-mat n1 n2 B4 ∈ 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 ∈ carrier-mat n n and pos: nonneg-mat A and n: n ≠ 0
**shows** ∃ sr ks f. sr ≥ 0 ∧ 0 ∉ set ks ∧ ks ≠ [] ∧ char-poly 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)
**proof –
define p where p = (λ sr k. monom 1 k - [: sr :: real ^ k ::])
let ?small = λ f sr. (∀ x. poly (map-poly complex-of-real f) x = 0 → cmod x < sr)
let ?wit = λ A sr ks f. sr ≥ 0 ∧ 0 ∉ set ks ∧ ks ≠ [] ∧ char-poly A = prod-list (map (p sr) ks) * f ∧ ?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-distrib)
{
  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 n1: nonneg-mat A1 nonneg-mat A2
      and A: A1 ∈ carrier-mat n1 n1 A2 ∈ carrier-mat n2 n2
      and n0: n1 < n n1 < 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-mul)
  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-mul 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, I :] 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

case 2
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.
\[ sr \geq 0 \land 0 \notin \text{set } ks \land ks \neq [] \land \]
\[ \text{charpoly } A = \text{prod-list } (\text{map } (\lambda k. \text{monom } 1 k - [:sr ^ k:]]) ks) \ast f \land \]
\[ (\forall x. \text{poly } (\text{map-poly complex-of-real } f) x = 0 \rightarrow \text{cmod } x < sr) \]
using \text{pos}

proof (\text{transfer}, \text{goal-cases})
case (1 A)
from \text{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 at 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 \text{decompose-prod-root-unity}, an algorithm which computes this factorization in an efficient way.

lemma \text{perron-frobenius-for-complexity}: fixes \( A :: \text{real} \rightarrow 'n \rightarrow 'n \) and \( f :: \text{real } \text{poly} \)

\begin{align*}
\text{defines } cA &\equiv \text{map-matrix complex-of-real } A \\
\text{defines } cf &\equiv \text{map-poly complex-of-real } f \\
\text{assumes } \text{pos}: \text{non-neg-mat } A \\
\text{and } sr &\land \text{charpoly } A = \text{prod-root-unity } ks \ast f \\
\text{and } \text{decomp}: \text{decompose-prod-root-unity } (\text{charpoly } A) = (ks, f) \\
\text{shows } 0 \notin \text{set } ks \\
\text{charpoly } A &\equiv \text{prod-root-unity } ks \ast f \\
\text{charpoly } cA &\equiv \text{prod-root-unity } ks \ast cf \\
(\land x. \text{poly } (\text{charpoly } cA) x = 0 \rightarrow \text{cmod } x \leq 1) \\
(\land x. \text{poly } cf x = 0 \rightarrow \text{cmod } x < 1) \\
(\land x. \text{cmod } x = 1 \rightarrow \text{order } x (\text{charpoly } cA) = \text{length } (ks \rightarrow x \rightarrow k = 1) \\
(\land x. \text{cmod } x = 1 \rightarrow \text{poly } (\text{charpoly } cA) x = 0 \rightarrow \exists k \in \text{set } ks. x \rightarrow k = 1) \\
\text{unfolding } cf-def \text{cA-def}
\end{align*}

proof (\text{atomize(full)}, \text{goal-cases})
case 1
let \(?c = \text{complex-of-real} \) 
let \(?cp = \text{map-poly } ?c \)
let \(?A = \text{map-matrix } ?c A \)
let \(?wit = \lambda ks f. \ 0 \notin \text{set } ks \land \)
\begin{align*}
\text{charpoly } A &\equiv \text{prod-root-unity } ks \ast f \\
\text{charpoly } A &\equiv \text{prod-root-unity } ks \ast \text{map-poly of-real } f \\
(\forall x. \text{poly } (\text{charpoly } ?A) x = 0 \rightarrow \text{cmod } x \leq 1) \\
(\forall x. \text{poly } (?cp f) x = 0 \rightarrow \text{cmod } x < 1) \\
\text{interpret } \text{field-hom } ?c.. \\
\text{interpret } p: \text{map-poly-inj-idom-divide-hom } ?c.. \\
\end{align*}
}\{ 
from \text{perron-frobenius-nonneg}[OF pos] \text{obtain } sr ks f 
where \( *: sr \geq 0 \ 0 \notin \text{set } ks ks \neq [] \) 
and cp: \text{charpoly } A = \text{prod-list } (\text{map } (\lambda k. \text{monom } 1 k - [:sr ^ k:])]) ks) \ast f 
\}
and small: \( \land x. \text{poly (?cp f)} \) \( x = 0 \implies \text{cmod} x < \text{sr} \) by blast

from arg-cong[of cp, of map-poly ?c]
have cpc: \( \text{charpoly ?A} = \text{prod-list} (\text{map} (\lambda k. \text{monom} 1 k - [\text{cp sr} \wedge k]) \text{ks}) \)
* map-poly (?c f)
  by (simp add: charpoly-of-real hom-distribs p.prod-list-map-hom[symmetric]
    o-def)
have sr-le-1: \( \text{sr} \leq 1 \)
  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) \( \text{cmod} x < \text{sr} \) where \( k \in \text{set} \text{ks} \wedge k = (\text{cp sr})^k \)

by force

  hence \( \text{cmod} x \leq \text{sr} \)
  proof (cases)
    case (mem k)
    with * have \( k \neq 0 \) by metis
    with arg-cong[of mem(2), of \( \lambda x. \text{root} k \) (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 \( \exists \text{ks f. \( ?\text{wit} \text{ks f} \)} \)
  proof (cases \( \text{sr} = 1 \))
    case False
    with sr-le-1 have *: \( \text{cmod} x \leq \text{sr} \implies \text{cmod} x < 1 \text{ cmod} x \leq \text{sr} \implies \text{cmod} x \leq 1 \) 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: \( \text{wit} Ks F \) by auto
  have cA0: \( \text{charpoly ?A} \neq 0 \) using degree-monic-charpoly[of ?A] by auto
  from wit have id: \( \text{charpoly ?A} = \text{prod-root-unity} Ks \ast \text{cp f} \) by auto
  from of-real-hom.hom-compose-charpoly[of charpoly A, unfolded decomp]
  have decompc: decompose-charpoly[of charpoly ?A] = (ks, cp f)
    by (auto simp: charpoly-of-real)
  from wit have small: \( \text{cmod} x = 1 \implies \text{poly (?cp f)} x \neq 0 \) for \( x \) by auto
  from decompose-charpoly[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
.. 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 \textit{spectral-radius} \textbf{where} \textit{spectral-radius} \equiv \text{Spectral-Radius.spectral-radius}

hide-const \textbf{(open)} \text{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. \[ ?A \in \text{carrier-mat} \ ?n \ ?n; \text{Spectral-Radius-Theory.spectral-radius} \ ?A \leq 1; \wedge \text{ev k.} \]
\[ \text{poly} \ (\text{char-poly} \ ?A) \text{ ev} = 0; \text{cmod} \ 	ext{ev} = 1 \implies \text{order} \ 	ext{ev} \ (\text{char-poly} \ ?A) \leq \ ?d \implies \exists c1 \ c2. \forall k. \text{norm-bound} \ (\ ?A \ ^m \ k) \ (c1 + c2 \ast \text{(real k)}^{?d - 1}). \]

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.

define \textbf{lemma} \textit{perron-frobenius-spectral-radius-complex:} \textit{fixes} \ A :: \text{complex mat}
\textit{assumes} \ A \ : \ A \in \text{carrier-mat} \ n \ n \\
\text{and} \ \text{real-nonneg:} \ \text{real-nonneg-mat} \ A \\
\text{and} \ \text{ev-le-1:} \ \wedge \ x. \ \text{poly} \ (\text{char-poly} \ (\text{map-mat} \ \text{Re} \ A)) \ x = 0 \implies x \leq 1 \\
\text{and} \ \text{ev-order:} \ \wedge \ x. \ \text{norm} \ x = 1 \implies \text{order} \ x \ (\text{char-poly} \ A) \leq d \\
\text{shows} \ \exists c1 \ c2. \forall k. \ \text{norm-bound} \ (\ A ^m \ k) \ (c1 + c2 \ast \text{(real k)} ^{(d - 1)})
\textbf{proof} \ (\text{cases} \ n = 0) \\
\textbf{case} \ False \\
hence \ n: \ n > 0 \ n \neq 0 \ \textbf{by} \ \text{auto} \\
define \ ?sr \ \textbf{where} \ ?sr = \text{spectral-radius} \ A \\
\textit{note} \ ?sr = \text{spectral-radius-mem-max}[\text{OF} \ A \ n(1), \ \text{folded} \ \text{sr-def}] \\
\textit{show} \ \textit{?thesis} \\
\textbf{proof} \ (\text{rule} \ \text{spectral-radius-poly-bound}[\text{OF} \ A], \ \text{unfold} \ \text{sr-def}[\text{symmetric}]) \\
\textbf{let} \ ?\text{cr} = \text{complex-of-real} \\
\textit{here is the transition from type-based perron-frobenius to set-based} \\
\textit{from} \ \textit{perron-frobenius[untransferred, cancel-card-constraint, OF} \ A \ \text{real-nonneg n(2)]} \\
\textbf{obtain} \ v \ \textbf{where} \ v: \ v \in \text{carrier-vec} \ n \ \text{and} \ \text{ev:} \ \text{eigenvector} \ A \ v \ (\ ?\text{cr} \ ?sr) \ \text{and} \\
\textit{run:} \ \text{real-nonneg-vec} \ v \ \text{unfolding} \ \text{sr-def} \ \textbf{by} \ \text{auto} \\
\textbf{define} \ B \ \textbf{where} \ B = \text{map-mat} \ \text{Re} \ A \\
\textbf{let} \ ?A = \text{map-mat} \ ?\text{cr} \ B \\
\textbf{have} \ AB: \ A = ?A \ \text{unfolding} \ B\text{-def} \\
\textit{by} \ (\text{rule} \ \text{eq-matI}, \ \text{insert} \ \text{real-nonneg}[\text{unfolded} \ \text{real-nonneg-mat-def} \ \text{elements-mat-def}], \ \text{auto})

98
define \( w \) where \( w = \text{map-vec Re } v \)
let \( ?v = \text{map-vec } ?cr \ w \)
have \( \text{evw: } v = ?v \) unfolding \( w \)-def
  by (rule eq-vecI, insert \( \text{rnn[unfolded real-nonneg-vec-def vec-elements-def]} \), auto)
have \( \text{B: } B \in \text{carrier-mat } n \ n \) unfolding \( B \)-def using \( A \) by auto
from \( \text{AB \ ev \ v \ have: eigenvector } ?A \ ?v \ (?cr \ sr) \) by simp
have \( \text{ev: eigenvector } B \ w \ sr \) by (rule of-real-hom.eigenvector-hom-rev[\( \text{OF } B \ ev \)], auto)

hence \( \text{eigenvalue } B \ sr \) unfolding \( \text{eigenvalue-def} \) by blast
from \( \text{ev-le-1[folded } B \)-def, OF this[unfolded eigenvalue-root-char-poly[\( \text{OF } B \)]\]
show \( \text{sr } \leq 1 \).

next
fix \( \text{ev} \)
assume \( \text{cmod ev } = 1 \)
thus \( \text{order ev } (\text{char-poly } A) \leq d \) by (rule ev-order)
qed
next
\( \text{case True} \)
with \( \text{A show } ?\text{thesis} \)
by (intro \( \text{exI[of - 0] , auto simp: norm-bound-def} \))
qed

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

\textbf{lemma} \( \text{perron-frobenius-spectral-radius: fixes } A :: \text{real mat} \)
\textbf{assumes} \( A: A \in \text{carrier-mat } n \ n \)
\textbf{and} \( \text{nonneg: nonneg-mat } A \)
\textbf{and} \( \text{ev-le-1: } \forall x. \text{poly } (\text{char-poly } A) \ x = 0 \implies x \leq 1 \)
\textbf{and} \( \text{ev-order: } \forall x :: \text{complex. norm } x = 1 \implies \text{order } x (\text{map-pol}\text{y of-real (char-poly } A) \leq d \)
shows \( \exists c1 \ c2. \forall k a. a \in \text{elements-mat } (A ^ \_ m k) \implies \text{abs a } \leq (c1 + c2 * \text{real } k ^ (d - 1)) \)
\textbf{proof} –
let \( \text{?cr = complex-of-real} \)
let \( \text{?B = map-mat } ?cr \ A \)
have \( \text{rnn: real-nonneg-mat } ?B \) using \( \text{nonneg unfolding real-nonneg-mat-def} \)
nonneg-mat-def
  by (auto simp: elements-mat-def)
have id: \( \text{map-mat Re } ?B = A \)
  by (rule eq-matI, auto)
have \( \exists c1 \ c2. \forall k. \text{norm-bound } (?B ^ \_ m k) (c1 + c2 * \text{real } k ^ (d - 1)) \)
  by (rule perron-frobenius-spectral-radius-complex[\( \text{OF } \text{B \ rnn} \)], unfold id, insert ev-le-1 ev-order, auto simp: of-real-hom.char-poly-hom[\( \text{OF } A \)])
then obtain \( c1 \ c2 \) where \( \exists k. \text{norm-bound } (?B ^ \_ m k) (c1 + c2 * \text{real } k ^ \_ m) \)
\[(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 \((A \^\_ m k)\)
  with pow-carrier-mat[of A] obtain i j where \(a = (A \^\_ m k) \$$ (i,j)\) and
  \(ij: i < n j < n\)
  unfolding elements-mat by force
  from \(ij \; nb[of k]\) \(A\) have norm \((\?B \^\_ m k) \$$ (i,j)\) \(\leq c1 + c2 * \text{real k} \^\_ \(d - 1\)\)
  unfolding norm-bound-def by auto

also have \((\?B \^\_ m k) \$$ (i,j)\) = \(?cr a\)
  unfolding of-real-hom.mat-hom-pow[of A, symmetric] a using \(ij A\) by auto

also have \(\text{norm} (\?cr a) = \text{abs a}\) by auto

finally show \(\text{abs a} \leq (c1 + c2 * \text{real k} \^\_ \(d - 1\))\).

qed

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

lemma perron-frobenius-spectral-type-based:
  assumes non-neg-mat \((A :: \text{real} \^\_ n \^\_ n)\)
  and \(\forall x. \; \text{poly} (\text{charpoly} \; A) x = 0 \longrightarrow x \leq 1\)
  and \(\forall x :: \text{complex}. \; \text{norm} x = 1 \longrightarrow \text{order} x (\text{map-poly of-real} (\text{charpoly} \; A)) \leq \; \?d\)
  shows \(\exists c1 c2. \; \forall k a. \; a \in \text{elements-mat-h} (\text{matpow} A k) \longrightarrow \text{abs} a \leq (c1 + c2 * \text{real k} \^\_ \(d - 1\))\)
  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 \(A \in \text{carrier-mat} \; n \; n\)
  and \(\text{nonneg-mat} A\)
  and \(\forall x. \; \text{poly} (\text{charpoly} \; A) x = 0 \longrightarrow x \leq 1\)
  and \(\forall x :: \text{complex}. \; \text{norm} x = 1 \longrightarrow \text{order} x (\text{map-poly of-real} (\text{charpoly} \; A)) \leq \; \?d\)
  and \(\; n \neq 0\)
  shows \(\exists c1 c2. \forall k a. \; a \in \text{elements-mat} \((A \^\_ m k)\) \longrightarrow \text{abs} a \leq (c1 + c2 * \text{real k} \^\_ \(d - 1\))\)

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 $f_1 \cdot \ldots \cdot f_d$ where each $f_i$ has precisely the roots of order $i$.

**context**

fixes $A :: \text{real mat}$ and $c :: \text{real}$ and $fis$ and $n :: \text{nat}$

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

and nonneg : $\text{nonneg-mat} \; A$

and yun : $\text{yun-factorization} \text{gcd} (\text{char-poly} \; A) = (c, fis)$

and ev-le-1 : $\text{card} \{x . \text{poly} (\text{char-poly} \; A) \; x = 0 \land x > 1\} = 0$

begin

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

**lemma** perron-frobenius-spectral-radius-yun:

assumes $bnd : \bigwedge f_i . (f_i, i) \in \text{set fis}$

implies $\exists x :: \text{complex. poly (map-polys of-real f_i)} \; x = 0 \land \text{norm} \; x = 1$

implies $\text{Suc} \; i \leq d$

shows $\exists c1 \; c2. \forall k \; a . a \in \text{elements-mat} \; (A \downarrow m \; k) \rightarrow \text{abs} \; a \leq (c1 + c2 \cdot \text{real} \; k \cdot (d - 1))$

**proof** (rule perron-frobenius-spectral-radius[OF A nonneg]; intro allI impI)

let $?cr = \text{complex-of-real}$

let $?cp = \text{map-polys} \; \text{?cr} \; (\text{char-poly} \; A)$

fix $x :: \text{complex}$

assume $x : \text{norm} \; x = 1$

have $A0 : \text{char-poly} \; A \neq 0$ using $\text{degree-monic-char-poly}[OF A]$ by auto

interpret field-hom-0'[?cr by (standard, auto)]

from $A0$ have $cp0 : ?cp \neq 0$ by auto

obtain $ox$ where $ox : \text{order} \; x \; ?cp = ox$ by blast

note $\text{sff} = \text{square-free-factorization-order-root}[\text{OF yun-factorization}(1)][\text{OF yun-factorization-hom}[\text{of char-poly} \; A, \text{unfolded yun map-polys-def split}] cp0, of x \; ox, \text{unfolded} \; ox$

show $\text{order} \; x \; ?cp \leq d \; \text{unfolding} \; ox$

proof (cases $ox$

with $\text{sff}$ obtain $f_i$ where $\text{mem} : (f_i, oo) \in \text{set fis}$ and $rt : \text{poly} (\text{map-polys} \; \text{?cr} \; f_i) \; x = 0$ by auto

from $bnd[\text{OF mem exI[of - x]}, \text{OF conjI[OF rt x]}]$ show $ox \leq d \; \text{unfolding} \; \text{Suc} \; .$

qed auto

next

let $?L = \{x . \text{poly} (\text{char-poly} \; A) \; x = 0 \land x > 1\}$

fix $x :: \text{real}$

assume $rt : \text{poly} (\text{char-poly} \; A) \; x = 0$

have finite $?L$

by (rule finite-subset[\text{OF - poly-roots-finite[of char-poly A]],

insert degree-monic-char-poly[\text{OF A}], auto)

with ev-le-1 have $?L = \{\}$ by simp

with $rt$ show $x \leq 1$ by auto

qed
Note that the only remaining problem in applying \(\bigwedge_i f_i \in \text{set} \ f_i; \exists x. \ \text{poly} (\text{map-poly complex-of-real} f_i) x = 0 \land \text{cmd} x = 1\) \(\implies\) \(\text{Suc} \ i \leq ?d\) \(\implies\) \(\exists c1 \ 2. \ \forall \ a. \ a \in \text{elements-mat} (A \ ^m \ k) \implies |a| \leq c1 + c2 \ast (\text{real} \ k)^d - 1\) is to check the condition \(\exists x. \ \text{poly} (\text{map-poly complex-of-real} f_i) x = 0 \land \text{cmd} x = 1\). 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** \(\text{perron-frobenius-spectral-radius-yun-real-roots}\):

**assumes** \(\text{bnd: } \bigwedge_i \ \text{f}_i \in \text{set} \ \ f_i; \ \exists x. \ \text{poly} (\text{map-poly complex-of-real} f_i) x = 0 \land \text{cmd} x = 1\) \(\implies\) \(\text{Suc} \ i \leq ?d\) \(\implies\) \(\exists c1 \ 2. \ \forall \ a. \ a \in \text{elements-mat} (A \ ^m \ k) \implies |a| \leq c1 + c2 \ast (\text{real} \ k)^d - 1\)

**shows** \(\exists c1 \ 2. \ \forall \ a. \ a \in \text{elements-mat} (A \ ^m \ k) \implies |a| \leq (c1 + \ast \text{real} \ k \ast (\text{real} \ k)^d - 1)\)

**proof** (rule \(\text{perron-frobenius-spectral-radius-yun}\))

**fix** \(f_i \ i\)

**let** \(?cr = \text{complex-of-real}\)

**let** \(?cp = \text{map-poly} \ ?cr\)

**assume** \(\text{fi: } (f_i, i) \in \text{set} \ f_i\)

**and** \(\exists x. \ \text{poly} (\text{map-poly} \ ?cr \ f_i) x = 0 \land \text{norm} x = 1\)

**then obtain** \(x\) **where** \(\text{rt: } \text{poly} (\ ?cp \ f_i) x = 0\) **and** \(x: \text{norm} x = 1\) **by auto**

**show** \(\text{Suc} i \leq d\)

**proof** (rule \(\text{bnd[OF fi]}\))

**consider** \((c) \ x \not\in \mathbb{R} \ | \ (1) \ x = 1 \ | \ (m1) \ x = -1 \ | \ (r) \ x \in \mathbb{R} \ x \not\in \{1, -1\}\)

**by** (cases \(x \in \mathbb{R}; \ \text{auto}\))

**thus** \(\text{card} \ \{x. \ \text{poly} \ f_i x = 0\} \neq \text{degree} f_i \land \text{poly} \ f_i 1 = 0 \land \text{poly} f_i (-1) = 0\)

**proof** (cases)

**case** \(1\)

**from** \(\text{rt}\) **have** \(\text{poly} f_i 1 = 0\)

**unfolding** \(1\) **by** \(\text{simp}\)

**thus** \(\text{thesis}\) **by** \(\text{simp}\)

**next**

**case** \(m1\)

**have id: -1 = ?cr (-1)\) **by** \(\text{simp}\)
from rt have \( \text{poly } f(-1) = 0 \)

unfolding \( m1 \text{id of-real-hom.hom-zero} \) \( \text{where } 'a=\text{complex,symmetric} \)
of-real-hom.poly-map-poly by simp

thus \(?\text{thesis by simp} \)

next
case \( r \)
then obtain \( y \) where \( xy \colon x = \text{of-real } y \) unfolding Reals-def by auto
from \( r(2)[\text{unfolded } xy] \) have \( y \notin \{1,-1\} \) by auto
from \( x[\text{unfolded } xy] \) have \( \text{abs } y = 1 \) by auto
with \( y \) have \( \text{False by auto} \)
thus \(?\text{thesis} .. \)

next
case \( c \)
from \( \text{yun-factorization}(2)[OF } \text{yun} ] \) \( f \) have \( \text{monic } f \) by auto
hence \( fi' \colon ?cp f' \neq 0 \) by auto
hence \( \text{fin: finite } \{x. \text{poly } (?cp f') x = 0\} \) by (rule poly-roots-finite)
have \( ?cr' \colon \{x. \text{poly } (?cp f') (?cr x) = 0\} \subset \{x. \text{poly } (?cp f') x = 0\} \) (is \( ?l \subset ?r \))

proof (rule, force)
  have \( x \in ?r \) using \( rt \) by auto
  moreover have \( x \notin ?l \) using \( c \) unfolding Reals-def by auto
  ultimately show \( ?l \neq ?r \) by blast
qed
from \( \text{psubset-card-mono}[OF fin this] \) have \( \text{card } ?l < \text{card } ?r \).
also have \( \ldots \leq \text{degree } (?cp f') \) by (rule poly-roots-degree[OF \( f' \)]
also have \( \ldots = \text{degree } f' \) by simp
also have \( ?l = ?cr' \colon \{x. \text{poly } f x = 0\} \) by auto
also have \( \text{card } \ldots = \text{card } \{x. \text{poly } f x = 0\} \)
  by (rule card-image, auto simp: inj-on-def)
finally have \( \text{card } \{x. \text{poly } f x = 0\} \neq \text{degree } f' \) by simp
thus \(?\text{thesis by auto} \)

qed

qed

end

thm perron-frobenius-spectral-radius-yun-real-roots

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
begin

lemma sum-root-unity: fixes x :: 'a :: {comm-ring,division-ring}
  assumes x' * n = 1
  shows (\sum (\lambda i. x' * i)\{..< n\} = (if x = 1 then of-nat n else 0)
proof (cases x = 1 \lor n = 0)
case False
from x obtain m where 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: \( \forall k. \text{Re} (\sum (\lambda i. b * x i \cdot k)) > 0 \)
  and root-unity: \( \bigwedge i. i \in I \Longrightarrow \exists d. d \neq 0 \land x i \cdot d = 1 \)
  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 \( \forall i. \exists d. i \in I \Longrightarrow d \neq 0 \land x i \cdot d = 1 \) using root-unity by auto
  from choice[OF this] obtain d where d: \( \bigwedge i. i \in I \Longrightarrow d i \neq 0 \land x i \cdot (d i) = 1 \) 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. \text{Re} (\sum (\lambda i. b * x i \cdot k) I))\{..< D\}
    by (rule sum-pos[OF - sumpos], insert D0, auto)
  also have \ldots = \text{Re} (\sum (\lambda k. \sum (\lambda i. b * x i \cdot k) I)\{..< D\}) by auto
  also have \ldots = \sum (\lambda k. b * x i \cdot k)\{..< D\} by (rule sum.swap)
  also have \ldots = \sum (\lambda i. b * x \cdot (\lambda k. x i \cdot 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 \cdot d i = 1
    and x: x i \neq 1 by auto
  have \exists C. D = d i \cdot C unfolding D-def
    by (subst prod.remove[of - i], insert i I, auto)
  then obtain C where D: D = d i \cdot 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 \cdot c)
  have \{..< D\} = \?g \cdot (\lambda j. (j mod d i, j div d i)) \cdot \{..< d i \cdot C\}
}

104
unfolding image-image split \(D[\text{symmetric}]\) by (rule image, insert \(d\) mod-mult-div-eq, blast)
also have \((\lambda j. (j \mod d \ i, j \div d \ i)) \cdot \{..<d \ i \ast C\} = \{..<d \ i\} \times \{..<C\}\)
(is \(\not\emptyset f \cdot \not\emptyset A = \not\emptyset B\))
proof -
{
  fix \(x\)
  assume \(x \in \not\emptyset B\) then obtain \(a\ c\) where \(x = (a,c)\) and \(a : a < d \ i\) and \(c : c < C\) by auto
  hence \(a + c \ast d \ i < d \ i \ast (1 + c)\) by simp
also have \(\ldots \leq d \ i \ast C\) by (rule mult-le-mono2, insert \(c\), auto)
finally have \(a + c \ast d \ i \leq \not\emptyset A\) by auto
  hence \(\not\emptyset f (a + c \ast d \ i) \in \not\emptyset f \cdot \not\emptyset A\) by blast
also have \(\not\emptyset f (a + c \ast d \ i) = x\) unfolding \(x\) using \(a\) by auto
finally have \(x \in \not\emptyset f \cdot \not\emptyset A\).
}
thus \(\not\emptyset \text{thesis}\) using \(d\) by (auto simp: div-it-nat)
qed

finally have \(D: \{..<D\} = (\lambda (a,c). a + d \ i \ast c) \cdot \not\emptyset B\) by auto
have inj: inj-on \(?g\ ?B\)
proof -
{
  fix \(a1\ a2\ c1\ c2\)
  assume id: \(?g (a1,c1) = \not\emptyset g (a2,c2)\) and \(*: (a1,c1) \in \not\emptyset B (a2,c2) \in \not\emptyset B\)
  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 \(\not\emptyset \text{thesis}\) by (smt SigmaE inj-onI)
qed

have \(\text{sum} (\lambda k. x i \ast k) \{..<D\} = \text{sum} (\lambda (a,c). x i \ast (a + d \ i \ast c)) \not\emptyset B\)
unfolding \(D\) by (subst sum.reindex, rule inj, auto intro!: sum.cong)
also have \(\ldots \leq \text{sum} (\lambda (a,c). x i \ast a) \not\emptyset B\)
  by (rule sum.cong, auto simp: power-add power-mult rt-unity)
also have \(\ldots = 0\) unfolding sum.cartesian-product[\(\text{symmetric}\)] sum.swap[of - \{..<C\}]
  by (rule sum.neutral, intro ballI, subst sum-root-unity[OF rt-unity], insert \(x\), auto)
finally show \(b \ i \ast \text{sum} (\lambda k. x i \ast k) \{..<D\} = 0\) by simp
qed

finally show False by simp
qed

fun \(j\to\j b\)-index :: \((\text{nat} \times \ 'a\)\)list \(\Rightarrow \text{nat} \Rightarrow \text{nat} \times \text{nat}\) where
\(j\to\j b\)-index \((n,a) \# n\-as\) \(i = (\text{if } i < n\ \text{then } (0,i)\ \text{else}\)
let rec = \(j\to\j b\)-index n-as \((i - n)\) in \((\Suc (\fst \text{rec}), \snd \text{rec})\)

fun \(j\to\j b\)-index :: \((\text{nat} \times \ 'a\)\)list \(\Rightarrow \text{nat} \times \text{nat} \Rightarrow \text{nat}\) where
\[
\text{jb-to-j-index n-as (0,j) = j}
\]
| \text{jb-to-j-index ((n,\# n-as) (Suc i, j) = n + jb-to-j-index n-as (i,j)}

\text{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) \(^\sim\) r) \(\bowtie\) (i,j) = (if bi = bj then ((jordan-block n a) 
\(\sim\) r) \(\bowtie\) (li, lj) else 0) 
\(\land\) (bi = bj \(\Rightarrow\) li < n \(\land\) lj < n \(\land\) bj < length n-as \(\land\) (n,a) \(\in\) set n-as) unfolding jordan-matrix-pow using assms

\text{proof (induct n-as arbitrary: i j bi bj)}
\text{case (Cons mb n-as i j bi bj) obtain m b where mb: mb = (m,b) by force}
\text{note Cons = Cons[unfolded mb] have [simp]: dim-col (case x of (n, a) => I_m n) = fst x for x by (cases x, auto) have [simp]: dim-row (case x of (n, a) => I_m n) = fst x for x by (cases x, auto) have [simp]: dim-col (case x of (n, a) => jordan-block n a \(\sim\) r) = fst x for x by (cases x, auto) have [simp]: dim-row (case x of (n, a) => jordan-block n a \(\sim\) r) = fst x for x by (cases x, auto) consider (UL) i < m j < m \(\land\) (UR) i < m j \(\geq\) m \(\land\) (LL) i \(\geq\) m j < m thus ?case
\text{proof cases}
\text{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 \(\neq\) 0 by (auto simp: Let-def) from LR Cons(2\(-\)) have bj: j-to-jb-index n-as ?j = (bj - 1, lj) bj \(\neq\) 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 "m r) (mb # n-as)) $$$ (i, j)

= diag-block-mat (map (λa. case a of (n, a) ⇒ jordan-block n a "m r) n-as)

$$ (\tilde{i}, \tilde{j}) $$

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: \( i < \text{sum-list} \ (\text{map} \ \text{fst} \ n-as) \)

and k: \( k \leq \tilde{i} \)

shows li ≤ i ∧ j-to-jb-index n-as (i - k) = (bi, li - k) ∧ (j-to-jb-index n-as j = (bi, li - k) \( \rightarrow \) j < sum-list (map fst n-as) \( \rightarrow \) 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 \( \tilde{i} = i - m \)

have \( \tilde{i} < \text{sum-list} \ (\text{map} \ \text{fst} \ n-as) \) using Cons(2−) i-large by auto

from Cons(2−) i-large have j: j-to-jb-index n-as \( \tilde{i} = (bi - 1, \tilde{li}) \)

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 \( li \leq i - m \) by auto

from li have aim1: \( li \leq i \) by auto

from li k i-large have \( i - k \geq m \) 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 \( \tilde{j} = j - m \)

from j * have \( \tilde{j} < \text{sum-list} \ (\text{map} \ \text{fst} \ n-as) \) 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 \( \tilde{j} \)] \( \tilde{j} \) ** have j - m = i - m - k by auto

with \( j \) i-large \( k \) have j = i - k using \( m \leq i - k \) by linarith
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] .. 

  definition c = (∏ ia = 0..<m−1. (of-nat m :: real) − 1 − of-nat ia) 
  lemma c-gt-0 : c > 0 unfolding c-def by (rule prod-pos, auto) 
  lemma c-bold : c ≠ 0 using c-gt-0 by auto 
  lemma c-int-def : c = (∏ ia = 0..<m−1. (of-nat m :: int) − 1 − of-nat ia) 
                   unfolding c-def by auto 
  lemma c-int : c ∈ ℤ unfolding c-int-def Ints-of-int by metis 
  lemma c-ge-1 : c ≥ 1 using c-gt-0 unfolding c-int-def by presburger 

  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 = I_m n iP * P = I_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 iP 
     where similar-mat-wit cA J Q iP 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 I :: nat set
I = {i | i bi li nn la. i < n ∧ j-to-jb-index n-as i = (bi, li) ∧ n-as ! bi = (nn,la) ∧ cmod la = 1 ∧ nn = m ∧ li = nn − 1}

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

lemma I-nonempty: I ≠ {}
proof –
  from split-list[OF mem] obtain as bs where n-as: n-as = as @ (m,la) ≠ 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 ∈ I unfolding I-def unfolding sumlist-nf[symmetric] n-as using m0 by auto
  thus ?thesis by blast
qed

lemma I-n: I ⊆ {..<n} unfolding I-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
definition \( d \) where \( d = (\text{SOME} \ d. \ \forall \ la. \ \text{la} \in \text{snd} \ \Rightarrow \ \text{set n-as} \ \longrightarrow \ \text{cmod la} = 1 \) 
\[ \rightarrow \ d \ \text{la} \neq 0 \land \ \text{la} \ ^* (d \ \text{la}) = 1 \) 

lemma \( d \): assumes \((k, \text{la}) \in \text{set n-as} \ cmod \ \text{la} = 1 \) 
shows \( \text{la} \ ^* (d \ \text{la}) = 1 \ \land \ d \ \text{la} \neq 0 \) 
proof – 
let \(?P = \lambda d. \ \forall \ \text{la}. \ \text{la} \in \text{snd} \ \Rightarrow \ \text{set n-as} \ \longrightarrow \ \text{cmod la} = 1 \ ) 
\rightarrow \ d \ \text{la} \neq 0 \land \ \text{la} \ ^* (d \ \text{la}) = 1 \ 
from \text{root-unity-cmod-1} \ have \ \forall \ \text{d} \ \exists \ \text{la} \ \in \ \text{snd} \ \Rightarrow \ \text{set n-as} \ \longrightarrow \ \text{cmod la} = 1 
\rightarrow \ d \ \neq 0 \land \ \text{la} \ ^* d = 1 \ \text{by blast} 
\from \text{choice}[\text{OF this}] \ have \ \exists \ \text{d} \ ?P \ d \ . 
\from \text{some-ex}[\text{OF this}] \ have \ ?P \ d \ unfolding \ \text{d-def} \ . 
\from \text{this}[\text{rule-format}, \ \text{of la}, \ \text{OF - assms}(2)] \ \text{assms}(1) \ show \ ?thesis \ by \ force 
qed 

definition \( D \) where \( D = \text{prod-list} \ (\lambda \ \text{na}. \ \text{if cmod} \ (\text{snd na}) = 1 \ \text{then} \ d \ (\text{snd na}) \ else \ 1) \ n\text{-as} \) 

lemma \( D0; \ D \neq 0 \ unfolding \ \text{D-def} \) 
by \( \text{unfold} \ \text{prod-list-zero-iff}, \ \text{insert} \ d, \ \text{force} \) 

definition \( K \) where \( \text{K off k} = D \ast k + (m-1) + \text{off} \) 

definition \( C \) where \( \text{C off k} = (c / \text{real} \ (\text{K off k}) \ ^* (m - 1)) \) 

lemma \( \text{limit-jordan-block}: \ \text{assumes} \ \text{kla}: \ (k, \ \text{la}) \in \text{set n-as} \) 
and \( \forall i : j < k \) 
shows \( (\lambda N. \ (\text{Jordan-block k la \ } ^* _m (\text{K off N})) \ $$ (i, j) * C \ \text{off N}) \) 
\[ \rightarrow \ (i = 0 \land \ j = k - 1 \land \text{cmod la} = 1 \land k = m \ \text{then} \ \text{la} \ ^\ast \text{off} \ \text{else} \ 0) \) 
proof – 
let \(?c = \text{of-nat} :: \ \text{nat} \Rightarrow \ \text{complex} \) 
let \(?r = \text{of-nat} :: \ \text{nat} \Rightarrow \ \text{real} \) 
let \(?cr = \text{complex-of-real} \) 
from \( i j \ have \ k0; \ k \neq 0 \ by \ \text{auto} \) 
from \text{Jordan-nf-char-poly}[\text{OF jnf}] \ have \ \text{cA}: \ \text{char-poly} \ \text{cA} = (\prod \ (n, a)\leftarrow \text{n-as}. \ [-]: \ a, 1:] \ ^* n) \ . 
from \text{degreemonic-char-poly}[\text{OF A}] \ have \ \text{degree} \ (\text{char-poly} \ A) = n \ by \ \text{auto} 

have \( \text{deg}: \ \text{degree} \ (\text{char-poly} \ A) = n \ \text{using} \ A \ by \ \text{simp add: degreemonicchar-poly} \) 
from \text{this}[\text{unfolded} \ \text{cA}] \ have \ n = \text{degree} \ (\prod \ (n, a)\leftarrow \\text{n-as}. \ [-]: \ a, 1:] \ ^* n) \ by \ \text{auto} 
also \ have \ ... = \ \text{sum-list} \ (\text{map} \ \text{degree} \ (\text{map} \ (\lambda(n, a). \ [-]: \ a, 1:] \ ^* n) \ \text{n-as})) 
\by \ (\text{subst degreeprodlisteq, auto}) 
also \ have \ ... = \ \text{sum-list} \ (\text{map} \ \text{fst} \ \text{n-as}) 
\by \ (\text{rule} \ \text{argcong}[\text{of - sum-list}], \ \text{auto} \ \text{simp: degreelinearpower}) 

110
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 f where decomp: decompose-prod-root-unity (char-poly A) = (ks, f)
by force
note pf = perron-frobenius-for-complexity-jmf[OF A n0 nonneg sr1 decomp]
define ji where ji = j - i
let ?f = λ N. (c / (?r N) *(m-1))
let ?jb = λ N. (jordan-block k la ^ m N) $$ (i,j)$$
let ?jbc = λ N. (jordan-block k la ^ 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. (∏ ia = 0..<ji. (?c N - ?c ia) / ?c (ji - ia)) * la ^ (N - ji)
let ?e2 = λ N. (∏ ia = 0..<ji. (?c N - ?c ia) / ?c (ji - ia)) * la ^ (N - ji)
let ?e3 = λ N. (((∏ ia = 0..<ji. (?c N - ?c ia) / ?c (ji - ia))) * la ^ (N - ji)) * ?f N
{ 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 iji' by auto
also have ... = ?e1 N unfolding ji-def
unfolding binomial-altdef-of-nat[OF ji] id ji-def
proof (rule arg-cong[of - - λ x. x * -], rule prod.cong[OF refl], goal-cases)
case (1 x)
  hence x ≤ N using (N ≥ k) ij by auto
  thus ?case by (simp add: of-nat-diff)
qed
finally have id: ?jb N = ?e1 N .
have ?jbc N = e2 N
  unfolding id e2-def using c-gt-0 by (simp add: norm-mult norm-divide
norm-power)
} note jbc = this
have e23: ?e2 N = ?e3 N for N using c-gt-0 by auto
have (e2 o K 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
proof (intro exI[of - Suc ji] impl allI, goal-cases)
case (1 r n) thus ?case by (cases n - ji, auto)
qed
next
case small
have e0: e = 0 * (of-real (if m - 1 = 0 then c else 0)) using small unfolding e-def by auto
show ?thesis unfolding e0 unfolding e23 e2-def
proof (rule tendsto-mul[OF tendsto-of-real])
show (λx. c / real x :: (m - 1)) → (if m - 1 = 0 then c else 0) (is _ → _)
proof (cases m - 1, force)
case (Suc k)
hence f: ?f = 0 by auto
have inv:c / real x :: Suc k = inverse (inverse c * real x :: Suc k)
  for x using divide-real-def by auto
show ?thesis unfolding f unfolding Suc inv
proof (rule LIMSEQ-inverse-zero, standard, standard, standard)
fix r::real fix x::nat
let ?v = ceiling c * ceiling (abs r)
have inv-pos:inverse c > 0 using c-gt-0 by simp
have c-int':real-of-int ⌈c⌉ = c using Ints-cases[OF c-int] by fastforce
assume nat ?v + 1 ≤ x
hence vr:?v < real x and real x ≥ 1 by linarith+
hence real x ≥ k ≥ 1 real x ≥ 1 using one-le-power by blast+
hence x:inverse c * real x ≤ inverse c * real x * real x :: k
  using inv-pos by simp
have inverse c * c * |r| ≤ inverse c * real-of-int (⌈c⌉ * |r|)
  using inv-pos c-int'
by (metis c0 le-of-int-ceiling left-inverse mult.assoc mult-cancel-right2 of-int-mul)
  with mult-strict-left-mono[OF vr inv-pos]
have inverse c * c * (abs r) < inverse c * real x by argo
hence r < inverse c * real x using c-gt-0 by simp
thus r < inverse c * real x :: Suc k using x by simp
qed
qed
let ?laji = inverse (la ^ ji)
let ?f = (λx. (Π i≤ja. (?c x - ?c ia) / ?c (ji - ia))) * la ^ (x - ji))
let ?g = λx. (Π i≤ja. (?c x - ?c ia) / ?c (ji - ia)) * (((?c x) ^ ji * la ^ x) * ?laji)
  have fy: ∀ f x in sequentially. ?f x = ?g x
  apply (rule eventually-sequentiallyI[of Suc ji])
unfolding prod-pow[symmetric] prod.distrib[symmetric] mult.assoc[symmetric]
unfolding prod-pow mult.assoc
by (rule arg_cong2[of - - - - ( * )], rule prod.cong, auto simp: ring-distrib,
insert small, subst power-diff, auto simp: divide-inverse

have 0: 0 = (∏ ia = 0..<ji. (1 - of-nat ia * 0) / of-nat (ji - ia)) * (0
* ?laji) by simp

show \( \exists f \longrightarrow 0 \) unfolding tendsto-cong[OF \( f \)]

proof (subt 0, rule tendsto-mult[OF tendsto-prod tendsto-mult[OF -
tendsto-const]],

to tendsto-intros inverse-of-nat-tendsto-zero)

show (ax. of-nat x * ji * la \~ x) \longrightarrow 0

by (rule poly-times-exp-tendsto-zero, insert small, auto)

qed auto

next

case medium

with max-block[OF kla] have k ≤ m and 1: \( \land \ x. \ \text{cmod la \~ x} = 1 \) by auto

with \( \text{ij medium have ji} < m - 1 \) unfolding ji-def by linarith

then obtain d where \( m1: m - 1 = \text{Suc} \ d + ji \) using less-iff-Suc-add by

auto

have e0: \( e = 0 \) using medium unfolding e-def by auto

have 0: 0 = (∏ ia = 0..<ji. (1 - ?c ia * 0) / ?c (ji - ia)) * (of-real c) *

0 by simp

let \( ?e = \lambda \ ia. \ \text{if} \ N = 0 \ \text{then 0 else} (1 - ?c ia / ?c N) / ?c (ji - ia) \)

let \( ?f = \lambda \ ia. (1 - ?c ia * (1 / ?c N)) / ?c (ji - ia) \)

\{

fix \( N \)

have c2 N = (∏ ia = 0..<ji. (?c N - ?c ia) / ?c (ji - ia)) / ?c N \~

ji) * la \~ (N - ji) * (of-real c / ?c N \sim \text{Suc} \ d)

unfolding medium m1 power-add e2-def by simp

also have (∏ ia = 0..<ji. (?c N - ?c ia) / ?c (ji - ia)) / ?c N \~ ji

= (∏ ia = 0..<ji. ?c ia N) unfolding prod-pow[symmetric] prod-divide[symmetric]

by (cases \( ?c N = 0, \) auto simp add: field-simps)

finally have c2 N = (∏ ia = 0..<ji. ?e ia N) * of-real c * inverse (?c N

\sim \text{Suc} \ d) * la \~ (N - ji)

by (simp add: divide-inverse)

also have \( \text{cmod . . . cmod} \ (\prod ia = 0..<ji. ?e ia N) * \text{of-real c} *

(inverse (?c N \sim \text{Suc} \ d)))

unfolding norm-mult norm-power 1 by simp

finally have \( \text{cmod} \ (c2 N) = \text{cmod} \ (\prod ia = 0..<ji. ?e ia N) * \text{of-real c}

\sim \text{Suc} \ d)) \sim \text{Suc} \ d)) \ by simp

\} note e2 = this

show \( \text{thesis unfolding e0} \)

apply (rule tendsto-norm-zero-cancel, unfold e2, rule tendsto-norm-zero)

apply (subst (2) 0)

apply (rule tendsto-mult[OF tendsto-mult[OF tendsto-prod tendsto-const]

inverse-power-tendsto-zero], goal-cases)

proof –

case (1 i)

let \( ?g = \lambda \ x. (1 - ?c i * (1 / \text{of-nat} \ x)) / \text{of-nat} (ji - i) \)

have eq: \( \forall \ x \in \text{sequentially}. \ ?e i x = ?g x \)
by (rule eventually-sequentially[of 1], auto)

show \( \exists c \; i \rightarrow (1 - \exists c \; i \ast 0) / \exists c \; (ji - i) \)

unfolding tendsto-cong[OF eq] using 1

by (intro tendsto-intros lim-1-over-n, auto)

qed

qed

show \((c2 \; o \; K \; off) \rightarrow e\)

by (rule LIMSEQ-subseq-LIMSEQ[OF main mono-K])

next

case True

hence large: \( \text{cmod} \; la = 1 \; k = m \; i = 0 \; j = k - 1 \) by auto

hence e: \( e = la' \; off\) unfolding e-def by auto

from large k0 have m0: \( m \geq 1 \) by auto

define m1 where \( m1 = m - 1 \)

have id: \( \text{real} \; (m - 1) - \text{real} \; ia = \?r \; m - 1 - \?r \; ia \; \text{for} \; \text{ia} \; \text{using} \; m0\)

unfolding m1-def by auto

let \( \forall e4 = \lambda x. (\prod ia = 0..<m1. 1 - \?cr \; (\?r \; ia / x)) \)

\{ fix x :: nat

assume x: \( x \neq 0\)

have \( \exists e2 \; x = (\prod ia = 0..<m1. (\exists c \; x - \?c \; ia) / \exists c \; (m1 - ia)) \ast\)

\((\prod ia = 0..<m1. \?cr \; (\text{real} \; m1 - \text{real} \; ia)) \ast\)

\((\prod i = 0..<m1. \?c \; x) \ast \text{la}'' \; (x - (m - 1)) \; (\text{is} - \?r \; \text{A} / \?B \ast \?C)\)

unfolding m1-def \( ?i \)-def \( \text{large} \; \text{c-def} \; \text{prod-pow}[\text{symmetric} \; \text{id} \; \text{by} \; \text{simpl}] \)

also have \( \?A = (\prod ia = 0..<m1. (\?cr \; x - \?c \; ia)) \; (\text{is} - \?r \; \text{A})\)

unfolding \( \text{prod-distrib}[\text{symmetric}] \; \text{by} \; (\text{rule} \; \text{prod.cong[OF refl]}, \text{substitution})\)

also have \( \?A / \?B = (\prod ia = 0..<m1. 1 - \?cr \; (\?r \; ia / x))\)

unfolding \( \text{prod-divide}[\text{symmetric}] \; \text{by} \; (\text{rule} \; \text{prod.cong[OF refl]}, \text{insert} \; x, \text{auto simp: field-simps})\)

finally have \( \exists e2 \; x = \exists e4 \; x \ast \?C . \)

\} note main = this

from d[of kla large(1)] have 1: \( \text{la}'' \; d \; \text{la} = 1 \) 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: \( D = d \; \text{la} * C \) unfolding D-def unfolding n-as using

large by auto

have \( (\lambda x. \?e4 \; x \ast e) \rightarrow (\prod ia = 0..<m1. 1 - \?cr \; 0) \ast e\)

by (intro tendsto-intros real-tendsto-divide-at-top, auto simp: filterlim-real-sequentially)

also have \( (\prod ia = 0..<m1. 1 - \?cr \; 0) = 1 \) unfolding e by simp

finally have \( (\lambda x. \?e4 \; x \ast e) \rightarrow e \) by auto

from LIMSEQ-subseq-LIMSEQ[of this mono-K]

have \( e4: (\lambda k. (\?e4 \; o \; K \; off)) \; k \ast e \rightarrow e \; (\text{is} \; ?A \rightarrow e)\)

by (auto simp: o-def)

\{

fix k :: nat

assume k: \( k \neq 0\)

hence 0: \( K \; off \; k \neq 0 \) unfolding K-def using D0 by auto

have \( \exists e2 \; (K \; off \; k) = \exists e4 \; (K \; off \; k) \ast la'' (K \; off \; k - (m - 1)) \) unfolding

114
main[of 0] ..
also have K off k = D * k + off unfolding K-def by simp
also have la ... = e unfolding e power-add D power-mult 1 by auto
finally have e2 (K off k) = (e4 o K off) k * e unfolding o-def e2-def .
} note main = this
have id: (?A ---\rightarrow e) = ((e2 o K off) ---\rightarrow e)
by (rule tendsto-cong, unfold eventually-at-top-linorder,
rule exI[of - 1], insert main, auto)
from e4 unfolded id show ?thesis .
qed
also have ((e2 o K off) ---\rightarrow e) = ((?jbc o K off) ---\rightarrow e)
proof (rule tendsto-cong, unfold eventually-at-top-linorder, rule exI[of - k],
intro allI impI, goal-cases)
case (1 n)
from mono-K[of off] I have K 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 K off) ---\rightarrow e) .
} note part1 = this
{
assume i > j \lor la = 0
hence e; e = 0 and jbn: N \geq k \implies ?jbc N = 0 for N
unfolding jordan-block-pow e-def using ij by auto
have ?jbc ---\rightarrow e unfolding e LIMSEQ-iff by (intro exI[of - k] allI impI,
subst jbn, auto)
from LIMSEQ-subseq-LIMSEQ[OF this mono-K] have (?jbc o K off) ---\rightarrow e .
} note part2 = this
from part1 part2 have (?jbc o K off) ---\rightarrow e by linarith
thus ?thesis unfolding e-def o-def C-def .
qed

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

lemma cmod-Lam: i \in I \implies cmod (Lam i) = 1
unfolding I-def Lam-def by auto

lemma I-Lam: assumes i: i \in I
shows (m, Lam i) \in set n-as
proof –
from i[unfolded I-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 \& li = m - 1 by auto
hence lam: Lam i = la unfolding Lam-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 \((\lambda N. (J \sim m (K \circ off N))) \circ\ (i, j) * C \circ off N)\)
\[\text{proof -}\]
\begin{itemize}
  \item obtain \(bi li\) where \(bi: j\text{-to-}jb\text{-index} \ n-as\ i = (bi, li)\) by force
  \item obtain \(bj lj\) where \(bj: j\text{-to-}jb\text{-index} n-as\ j = (bj, lj)\) by force
  \item define \(la\) where \(la = \text{snd (n-as)!fst (j\text{-to-}jb\text{-index} n-as\ j)}\)
  \item obtain \(nn\) where \(nbj: n-as! \ bj = (nn, la)\) unfolding \(la\text{-def}\ bj\text{fst-conv}\) by (metis prod.collapse)
\end{itemize}
\begin{itemize}
  \item from \(j\text{-to-}jb\text{-index}[OF ij\text{folded sumlist-nf}]\ bi bj nbj\)
  \item have \(eq: bi = bj \implies li < nn \land lj < nn \land bj < length n-as \land (nn, la) \in set\)
  \item index: \((J \sim m r) \circ (i, j) = (\text{if } bi = bj \text{ then } (\text{jordan-block nn la } \sim m r) \circ (li, lj) \text{ else 0})\) for \(r\) by auto
  \item note \(index\text{-rev} = j\text{-to-}jb\text{-index}\text{-rev}[OF bj, unfolded sumlist-nf, OF ij(2) le-refl]\)
  \item show \(?thesis\)
\end{itemize}
\begin{itemize}
  \item proof (cases \(bi = bj\))
  \item case False
    \begin{itemize}
      \item assume \(j \in I\ i = j - (m - 1)\)
      \item from \(this[unfolded I\text{-def}] bj nbj\) have \(i = j - lj\) by auto
    \end{itemize}
  \item thus \(?thesis\) unfolding index id if-False by auto
\end{itemize}
\begin{itemize}
  \item next
    \item case True
      \begin{itemize}
        \item hence \(id: (bi = bj) = True\) by auto
      \end{itemize}
  \item next \(n-as\) by auto
    \begin{itemize}
      \item have \((\lambda N. (J \sim m (K \circ off N))) \circ (i, j) * C \circ off N)\)
        \[\text{proof}\]
        \begin{itemize}
          \item assume \(?r\)
            \begin{itemize}
              \item hence \(j \in I\)
              \item from \(this[unfolded I\text{-def}] bj nbj\)
              \item have \(*: nn = m \circ mod la = 1 lj = nn = m\) by auto
              \item with \(*\) have \(li = 0\) using \(index\text{-rev}\ bj\ by\ auto\)
            \end{itemize}
        \end{itemize}
    \end{itemize}
  \item next
    \begin{itemize}
      \item assume \(?l\)
        \begin{itemize}
          \item hence \(j\) in using \(bj nbj ij\) by (auto simp: I\text{-def})
        \end{itemize}
    \end{itemize}
  \item with \(index\text{-rev}[of\ i]\ bi ij(1)\) (\(?l\) True
have \( i = j - (m - 1) \) by auto
with \( j \) show \(?r\) by auto
qed
finally show \(?thesis\ unfolding\ la-def\ Lam-def\).
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 \( \ldots = cA^m k * (P * J * iP) * P \) using JNF by simp
also have \( \ldots = (cA^m k * P) * (J * (iP * P)) \)
using A JNF(1-3) by (simp add: assoc-mult-mat[of - n n - n - n])
also have \( J * (iP * P) = J\ unfolding\ JNF\ by\ auto\)
finally show \(?case\ unfolding\ Suc\)
using A JNF(1-3) by (simp add: assoc-mult-mat[of - n n - n - n])
qed

lemma C-nonneg: \( C_{off k} \geq 0\ unfolding\ C-def\ using\ c-gt-0\ by\ auto\)

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

definition \( i\ where\ i = (SOME\ i.\ i \in I)\)

lemma \( i: i \in I\ unfolding\ i-def\ using\ I-nonempty\ some-in-eq\ by\ blast\)

lemma \( i-n: i < n\ using\ i\ unfolding\ I-def\ by\ auto\)

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

lemma \( j: j < n\ P \$(j, i - (m - 1)) \neq 0\)
proof
from \( i-n\ have \( lt: i - (m - 1) < n\) by\ auto\)
show \( j < n \) \( P \, \langle j, i - (m - 1) \rangle \neq 0 \)

unfolding \( j \)-def using some I-ex[OF P-nonzero-entry[OF lt]] by auto

qed

definition \( B = \text{cmod} \, (P \, \langle j, i - (m - 1) \rangle) / 2 \)

lemma \( 0 < B \) unfolding \( B \)-def using \( j \) by auto

definition \( w = P \ast_v \text{unit-vec} \, n \, i \)

lemma \( w \in \text{carrier-vec} \, n \) using JNF unfolding \( w \)-def by auto

definition \( v = \text{map-vec} \, \text{cmod} \, w \)

lemma \( v \in \text{carrier-vec} \, n \) unfolding \( v \)-def using \( w \) by auto
also have cAk *v w = (cAk * P) *v unit-vec n i
unfolding w-def i-def using JNF cAk by simp
also have \( \ldots = P *v (J \hat{m} k *v unit-vec n i) \) unfolding cAk-def A-power-P
using JNF by (subst assoc-mult-mat-vec[of - n n - n], auto)
also have \( J \hat{m} k *v unit-vec n i = \text{col} (J \hat{m} k) i \)
by (rule eq-vecI, insert i, auto)
also have \( (P *v (\text{col} (J \hat{m} k) i)) \) $j = \text{Matrix}.row P j \cdot \text{col} (J \hat{m} k) i$
by (subst index-mult-mat-vec, insert j JNF, auto)
also have \( \ldots = \text{sum} (\lambda \i. P \$(j, \i) * (J \hat{m} k) \$(\i, \i)) \{..<n\} \)
unfolding scalar-prod-def by (rule sum.cong, insert j JNF(1), auto)
finally have \( (A \hat{m} k *v v) \$_v j \geq \text{cmod} (?\exp k) \).
from mult-right-mono[OF this C-nonneg]
have \( (A \hat{m} k *v v) \$_v j * C \text{ off} \ k k \geq \text{cmod} (?\exp k * C \text{ off} \ k k) \) unfolding norm-mult
using C-nonneg by auto

} hence gc: \( (A \hat{m} k \text{ off} \ k *v v) \$_v j * C \text{ off} \ k \geq M k \) for \( k \) unfolding M-def
by auto
from i have mem: \( i - (m - 1) \in \{..<n\} \) by auto
have \( \lambda k. \text{?exp} (K \text{ off} \ k) * C \text{ off} \ k \)
(\( \text{=} \sum i < n. P \$(j, \i) * (\text{if} \ i \in I \wedge \i = i - (m - 1) \text{ then} \langle \text{Lam} \ i \rangle \) off else 0)\)
(is - \( \text{=} \sum)\)
unfolding sum-distrib-right mult.assoc
by (rule tendsto-sum, rule tendsto-mult, force, rule limit-jordan-matrix[OF -i], auto)
also have \( \text{?sum} = P \$(j, i - (m - 1)) \) Lam i \hat{\text{ off}}
by (subst sum.remove[OF OF - mem], force, subst sum.neutral, insert i, auto)
finally have tend1: \( \lambda k. \text{?exp} (K \text{ off} \ k) * C \text{ off} \ k \)
(\( \rightarrow \) P \$(j, i - (m - 1)) \) Lam i \hat{\text{ off}}
have tend2: \( M \rightarrow \text{cmod} (P \$(j, i - (m - 1)) \) Lam i \hat{\text{ off}} \) unfolding M-def
by (rule tendsto-norm, rule tend1)
\(
)\)
from j have 0: \( P \$(j, i - (m - 1)) \) \( \neq 0 \) by auto
define E where E = \( \text{cmod} (P \$(j, i - (m - 1)) \) Lam i \hat{\text{ off}} \)
from cmod-Lam[OF iI] 0 have E: \( E / 2 > 0 \) unfolding E-def by auto
from tend2[folded E-def] have tend2: \( M \rightarrow E \).
from gc have gc: \( G k \geq M k \) for \( k \) unfolding G-def.
from tend2[unfolded LMSEQ-iff] rule-format, OF E]
obtain k' where diff: \( \lambda k. k \geq k' \rightarrow \text{norm} (M (k - E) < E / 2) \) by auto
\{fix k
assume k \( \geq k' \)
from diff[OF this] have norm: \( \text{norm} (M k - E) < E / 2 \).
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 gc[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

119
cmod-Lam[of i1] by auto
finally have G k ≥ B .
}

hence ∃ k'. ∀ k. k ≥ k' → G k ≥ B by auto

hence Bound: ∃ k'. ∀ k ≥ k'. B ≤ G k by auto

{ fix kk
define k where k = K off kk
have (\( (A \sim m k) * v \) $ j * C off kk = Re (?c ((A \sim m k) * v) $ j * C off kk)) by simp
also have ?c ((A \sim m k) * v) $ j * C off kk = ?cv ((A \sim m k) * v) $ j * ?c (C off kk)
using j A by simp
also have ?cv ((A \sim m k) * v) = (?cm (A \sim m k) * v) using A
by (subst of-real-hom.mult-mat-vec-hom[of - v], auto)
also have ... = (cA \sim m k * v ?v)
by (simp add: of-real-hom.mat-hom-pow[of A])
also have ... = (cA \sim m k * v ((P * iP) * v ?v)) unfolding JNF using v by auto
also have ... = (cA \sim m k * v (P * v u)) unfolding u-def
by (subst assoc-mult-mat-vec, insert JNF v, auto)
also have ... = (P * J \sim m k * v u) unfolding A-power-P[symmetric]
by (subst assoc-mult-mat-vec, insert u JNF(1) A, auto)
also have ... = (P * (J \sim m k * v u))
by (rule assoc-mult-mat-vec, insert u JNF(1) A, auto)
finally have (A \sim m k * v) $v j * C off kk = Re ((P * v (J \sim m k * v u)) $ j
* C off kk) by simp
also have ... = Re (\( \sum i = 0..<n. P $$(j, i) * (\sum ia = 0..<n. (J \sim m k) $$(i, ia) * (i, ia)) * C off kk)
by (subst index-mult-mat-vec, insert JNF(1) j u, auto simp: scalar-prod-def
sum-distrib-right[symmetric]
mult.assoc[symmetric])
finally have (A \sim m k * v) $v j * C off kk =
Re (\( \sum i = 0..<n. P $$(j, i) * (\sum ia = 0..<n. (J \sim m k) $$(i, ia) * C off kk)
* u $v ia))
unfolding k-def
by (simp only: ac-simps)
} note A-to-u = this
define F where F = (\( \sum ia \in I. a ia * Lam ia \sim off \))
have G →→ Re (\( \sum i = 0..<n. P $$(j, i) *
(\sum ia = 0..<n. (if ia \in I \land ia = (m - 1) then (Lam ia) \sim off else 0)
* u $v ia))
unfolding A-to-u G-def
by (rule tendsto-Re[OF tendsto-sum[OF tendsto-mult[of tendsto-sum[of tendsto-mult[of limit-jordan-matrix]]]], auto)
also have (\( \sum i = 0..<n. P $$(j, i) *
(\sum ia = 0..<n. (if ia \in I \land ia = (m - 1) then (Lam ia) \sim off else 0)

120
\[\sum_{i=0}^{n} (\sum_{j=0}^{m-1} i = a - (m - 1)) \text{ then } P \]
proof

note \(\text{jnf}' = \text{jnf}[\text{unfolded jordan-nf-def}]\)
from \(\text{jnf}'\) similar-matD[\(\text{OF jnf}' [\text{THEN conjunct2}]\)] obtain \(n\)
where \(A: \ A \in \text{carrier-mat n n and n-as}: 0 \notin \text{fst} ' \text{set n-as by auto}\)
let \(?M = \{ \ m, \exists \ lam. (m,lam) \in \text{set n-as} \land \text{cmod lam} = 1 \}\)
have \(m: m \in {?M}\) using mem lam1 by auto
have \(\text{fin: finite {?M}}\)

define \(\text{M where} \quad \text{M} = \{ m \in {?}M \implies \text{cmod lam} = 1 \}\)
have \(\text{M} \in {?}M\) using fin m unfolding M-def using Max-in by blast
then obtain \(\text{Lam}\) where \(\text{M} = (\text{M,Lam}) \in \text{set n-as cmod Lam} = 1\) by auto
from \(m\) fin have \(\text{mM}\) : \(m \leq \text{M}\) unfolding \(\text{M-def}\)
interpret \(\text{spectral-radius-1-jnf-max A n M Lam}\)
proof (unfold-locales, rule A, rule nonneg, rule jnf, rule M, rule M, rule sr1)
fresh \(k\) \(\text{la}\)
assume \(\text{kla}\) : \(k, \text{la} \in \text{set n-as}\)
with \(\text{fin}\) have \(1\) : \(\text{cmod la} = 1 \implies k \leq \text{M}\)
by blast
obtain \(\text{ks f}\) where \(\text{decomp: decompose-prod-root-unity (char-poly A) = (ks, f)}\)
by force
from \(\text{n-as}\) \(\text{kl}\) have \(\text{k0}\) : \(k \neq 0\) by force
let \(\text{cA = map-mat complex-of-real A}\)
from \(\text{split-list[\{OF kla\]}\ obtain as bs where nas: n-as = as @ (k,\text{la}) # bs\)
by auto
have \(\text{rt: poly (char-poly cA) la = 0 using k0}\)
unfolding jordan-nf-char-poly[\(\text{OF jnf}\)] nas poly-prod-list prod-list-zero-iff by auto
have \(\text{sumlist-nf: sum-list (map fst n-as) = n}\)
proof –
have \(\text{sum-list (map fst n-as) = dim-row (jordan-matrix n-as) by simp}\)
also have \(\ldots = \text{dim-row cA using similar-matD[\{OF jnf'[\text{THEN conjunct2}]\]}\)
by auto
finally show \(\text{thesis using A by auto}\)
qed
from this[unfolded nas] \(\text{k0}\) have \(\text{n0}\) : \(n \neq 0\) by auto
from perron-frobenius-for-complexity-jnf[4][\(\text{OF A n0 nonneg sr1 decomp rt}\)]
have \(\text{cmod la} \leq 1\).
with \(\text{1}\) show \(\text{cmod la} \leq 1 \land (\text{cmod la} = 1 \implies k \leq M)\) by auto
qed
from main-theorem
show \(\text{thesis using mM by auto}\)
qed
hide-const(open) spectral-radius

lemma (in ring-hom) \(\text{hom-smult-mat: mat_h (a \cdot_m A) = hom a \cdot_m \text{mat_h A}}\)
by (rule eq-matI, auto simp: hom-mul)

lemma \(\text{root-char-poly-smult: fixes A :: complex mat}\)
assumes \(\text{A: A \in carrier-mat n n}\)

122
and \( k: k \neq 0 \) shows \((\text{poly} (\text{char-poly} (k \cdot_m A)) x = 0) = (\text{poly} (\text{char-poly} A) (x / k) = 0)\)

using \( \text{order-char-poly-smult}[\text{OF} A k, of x] \)
by \( \text{metis A degree-0 degree-monic-char-poly monic-degree-0 order-root smult-carrier-mat} \)

**Theorem** \( \text{real-nonneg-mat-spectral-radius-largest-jordan-block} \):
assumes \( \text{real-nonneg-mat} A \)
and \( \text{jordan-nf} A n-as \)
and \( (m, \lambda m) \in \text{set} n-as \)
and \( \text{cmod} \lambda m = \text{spectral-radius} A \)
shows \( \exists M \geq m. (M, \text{of-real} (\text{spectral-radius} A)) \in \text{set} n-as \)

**Proof** –
from \( \text{similar-matD}[\text{OF assms} (2)][\text{unfolded jordan-nf-def}] \)
\[ \text{obtain} \ n \text{ where} \]
\( A: A \in \text{carrier-mat} n n \) by \( \text{auto} \)
let \(?c = \text{complex-of-real} \)
define \( B \) where \( B = \text{map-mat} \Re A \)
have \( AB: A = \text{map-mat} (?c B) \) unfolding \( B-def \)
by \( \text{auto simp: real-nonneg-mat-def} \)
have \( \text{nonneg}: \text{nonneg-mat} B \) \( \text{using} \) \( \text{assms}(1) \)
unfolding \( AB \)
by \( \text{auto simp: nonneg-mat-def} \)
let \(?sr = \text{spectral-radius} A \)
show \( ?\text{thesis} \)
proof
(cases \(?sr = 0 \))
case False

define \( \lambda isr \) where \( \lambda \) \( = \text{inverse} \ ?sr \)
let \(?nas = \text{map} \ (\lambda(n, a). (n, ?c \ \lambda m * a)) \) \( n-as \)
from False have \( \lambda isr0: \lambda isr \neq 0 \) unfolding \( \lambda-def \)
by \( \text{auto} \)

define \( C \) where \( C = \lambda isr \cdot_m B \)

have \( BC: C \in \text{carrier-mat} n n \) using \( B \) unfolding \( C-def \)
by \( \text{auto} \)

have \( \text{nonneg}: \text{nonneg-mat} C \) unfolding \( C-def \)
unfolding \( \text{isr-pos} \)
by \( \text{auto} \)

from \( \text{jordan-nf-smult}[\text{OF assms}(2)][\text{unfolded AB}] \)
have \( \text{jnf}: \text{jordan-nf} \ (\text{map-mat} (?c C)) \) \( \text{nonas} \) unfolding \( C-def \)
by \( \text{auto simp: of-real-hom-hom-smult-mat} \)

from \( \text{assms}(3) \)
have mem: \( (m, ?c \ \lambda m * \lambda m) \in \text{set} \) \( ?nas \)
by \( \text{auto} \)

have \( 1: \text{cmod} \ (?c ism * \lambda m) = 1 \)
using False unfolding \( \text{isr-pos} \)

norm-mult \( \text{assms}(4) \)
by \( \text{smt norm-ge-zero} \)

\( \{ \)

\( \text{fix} x \)

have \( B': \text{map-mat} (?c B) \in \text{carrier-mat} n n \)
using \( B \) \( \text{by auto} \)
assume \( \text{poly} (\text{char-poly} C) x = 0 \)

hence \( \text{poly} (\text{char-poly} (\text{map-mat} (?c C))) (\?c x) = 0 \)
unfolding of-real-hom-char-poly-hom[\text{OF}]
\]
\[ C \text{ by auto} \]

\[ \text{hence } \text{poly (char-poly } A \text{) (} ?c x / ?c \text{ isr)} = 0 \text{ unfolding } C\text{-def of-real hom smult mat} \]

\[ AB \]

\[ \text{unfolding root-char-poly-smult[OF } B' \text{ cisr0}. \text{]} \]

\[ \text{hence } \text{eigenvalue } A \text{ (} ?c x / ?c \text{ isr)} \text{ unfolding eigenvalue-root-char-poly[OF } A \text{].} \]

\[ \text{hence mem: cmod (} ?c x / ?c \text{ isr)} \in \text{ cmod ' spectrum } A \text{ unfolding spectrum-def} \]

by auto

\[ \text{from Max-ge[OF finite-imageI this]} \]

\[ \text{have cmod (} ?c x / ?c \text{ isr)} \leq ?sr \text{ unfolding Spectral-Radius.spectral-radius-def} \]

\[ \text{using } A \text{ card-finite-spectrum(1) by blast} \]

\[ \text{hence cmod (} ?c x \text{)} \leq 1 \text{ using isr0 isr-pos unfolding isr-def} \]

by (auto simp: field-simps norm-divide norm-mult)

\[ \text{hence } x \leq 1 \text{ by auto} \]

\[ \text{note } sr = \text{ this} \]

\[ \text{from nonneg-sr-1-largest-jb[OF nonneg jnf mem 1 sr]} \text{ obtain } M \text{ where} \]

\[ M: M \geq m (M,1) \in \text{ set } \text{nas by blast} \]

\[ \text{from } M(2) \text{ obtain } a \text{ where mem: (} M,a \text{) } \in \text{ set n-as and } 1 = ?c \text{ isr } * a \text{ by auto} \]

\[ \text{from this(2) have } a: a = ?c \text{ ?sr using isr0 unfolding isr-def by (auto simp: field-simps)} \]

\[ \text{show } ?\text{thesis} \]

\[ \text{by (intro exI[of - M], insert mem a M(1), auto)} \]

next case True

\[ \text{from jordan-nf-root-char-poly[OF assms(2,3)]} \]

\[ \text{have eigenvalue } A \text{ lam unfolding eigenvalue-root-char-poly[OF } A \text{].} \]

\[ \text{hence cmod lam} \in\text{ cmod ' spectrum } A \text{ unfolding spectrum-def by auto} \]

\[ \text{from Max-ge[OF finite-imageI this]} \]

\[ \text{have cmod lam} \leq ?sr \text{ unfolding Spectral-Radius.spectral-radius-def} \]

\[ \text{using } A \text{ card-finite-spectrum(1) by blast} \]

\[ \text{from this[unfolded True] have } lam0: lam = 0 \text{ by auto} \]

\[ \text{show } ?\text{thesis unfolding True using assms(3)[unfolded lam0]} \text{ by auto} \]

qed

\[ \text{qed} \]

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

\[ \text{similar-mat-wit } A \text{ B C D} \]

124
shows \(\text{similar-mat-wit} \ (\mat_h \ A) \ (\mat_h \ B) \ (\mat_h \ C) \ (\mat_h \ D)\)
proof –
  obtain \(n\) where \(n = \dim\row A\) by auto
note * = \(\text{similar-mat-witD}[OF \ n \\ assms]\)
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 ** = \((1 - 3)[\text{THEN arg-cong}[OF - \ mat_h], \ \text{unfolded id}]\)
note \(\mult = \mat\hom\mult[\text{of - n n}]\)
note \(\text{hom-mult} = \mat\hom\mult[\text{of - n n - n}]\)
  show \(?\thesis\) unfolding \(\text{similar-mat-wit}\) \(\text{Let-def}\) \(\text{similar-mat-def}\)
  unfolding **(3) using **(1,2)
    by \((\text{auto simp: n}[\text{symmetric}] \ \text{hom-mult simp: *(- mult)]}\)
qed

lemma \(\text{in comm-ring-hom}\) \(\text{similar-mat-hom}:\)
\(\text{similar-mat} \ A \ B \implies \text{similar-mat} \ (\mat_h \ A) \ (\mat_h \ B)\)
using \(\text{similar-mat-wit-hom}[\text{of} \ A \ B \ C \ D \ \text{for} \ C \ D]\)
by \((\text{smt \ similar-mat-def})\)

context \(\text{field-hom}\)
begin

lemma \(\text{hom-swaprows}: i < \dim\row A \implies j < \dim\row A \implies\)
\(\text{swaprows} \ i \ j \ (\mat_h \ A) = \mat_h \ (\text{swaprows} \ i \ j \ A)\)
unfolding \(\text{mat-swaprows-def}\) by \((\text{rule eq-matI, auto})\)

lemma \(\text{hom-gauss-jordan-main}: A \in \carrier\mat \ nr \ nc \implies B \in \carrier\mat \ nr \ nc2 \implies\)
\(\text{gauss-jordan-main} \ (\mat_h \ A) \ (\mat_h \ B) \ i \ j =\)
\(\text{map-prod} \ mat_h \ mat_h \ (\text{gauss-jordan-main} \ A \ B \ i \ j)\)
proof \((\text{induct} A \ B \ i \ j \ \text{rule: gauss-jordan-main.induct})\)
  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 \(?hp = \text{map-prod} \ mat_h \ mat_h\)
show \(?\case\ unfolding \text{gauss-jordan-main.simps}[of \ A \ B \ i \ j]\)
\(\text{gauss-jordan-main.simps}[of \ ?h \ A - i \ j]\)
index-map-mat \(\text{Let-def if-distrib}[of \ ?hp]\) \(\dim\)
proof \((\text{rule if-cong}[OF \ refl], \ \text{goal-cases})\)
  case \(1\)
  note \(IH = IH[OF \ dim[\text{symmetric}] \ \text{1 refl}]\)
  from \(1\) have \(ij: i < nr \ j < nc\) by auto
  hence \(\text{hij}: (\text{?h} \ A) \ (i, j) = \text{hom} \ (A \ (i, j))\) \(\text{using AB by auto}\)
define \(\text{idx where}\) \(\text{idx = concat} \ (\text{map } (\lambda i'. \ i', j) \?= 0 \ \text{then} [i'] \ \text{else} []))))\)
[Suc i..<nr] = \(\text{map} \ (\lambda i'. \ \text{if} \ A \ (i', j) \?= 0 \ \text{then} [i'] \ \text{else} []))\)
[Suc i..<nr] = \(\text{map} \ (\lambda i'. \ \text{if} \ A \ (i', j) \?= 0 \ \text{then} [i'] \ \text{else} []))\)
[Suc i..<nr]
by (rule map-cong[OF refl], insert ij AB, auto)

show ?case unfolding bij 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 hom-0-iff hom-1-iff id ixs-def
      by (rule if-cong[OF refl - if-cong[OF refl]], auto)
  next
    case (Cons i x)
    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(1)[OF Nil AB] by auto
  qed
  qed

next
  case 2
  from AB have elim: eliminate-entries (λi. A $$ (i, j)) A i j ∈ carrier-mat nr nc
    unfolding eliminate-entries (λi. A $$ (i, j)) B i j ∈ carrier-mat nr nc2
    by auto
  show ?case unfolding IH(3)[OF 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)
  qed
  qed

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 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

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.
  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 simp

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 ∈ 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 ∈ 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

127
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 ∈ carrier-mat n n
  and sr: ∀ x. poly (char-poly A) x = 0 ⇒ cmod x ≤ 1
  and 1: ∀ x. poly (char-poly A) x = 0 ⇒ cmod x = 1 ⇒
       order x (char-poly A) > d + 1 ⇒
       (∀ bsize ∈ fst (set (compute-set-of-jordan-blocks A x)). bsize ≤ d + 1)
shows ∃ c1 c2. ∀ k. norm-bound (A ^ m k) (c1 + c2 * of-nat k ^ d)
proof –
  from char-poly-factorized[OF A] obtain as where cA: char-poly A = (Π 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 + 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 $A$: $A \in \text{carrier-mat } n n$
  and $\text{bnd}: \exists c1 c2. \forall k. \text{norm-bound } ((\text{map-mat complex-of-real } A) \ ^{\sim} m k) (c1 + c2 \ast \text{of-nat } k \ ^{\sim} d)$
  shows $\exists c1 c2. \forall k a. a \in \text{elements-mat } (A \ ^{\sim} m k) \longrightarrow \text{abs } a \leq (c1 + c2 \ast \text{of-nat } k \ ^{\sim} d)$
proof --
  let $?B = \text{map-mat complex-of-real } A$
  from $\text{bnd}$ obtain $c1 c2$ where $\text{nb}: \bigwedge k. \text{norm-bound } (?B \ ^{\sim} m k) (c1 + c2 \ast \text{real } k \ ^{\sim} d)$
  show $?\text{thesis}$
  proof (rule $\text{exI}[\text{of - c1}])$
    assume $a \in \text{elements-mat } (A \ ^{\sim} m k)$
    with $\text{pow-carrier-mat}[\text{OF } A]$ obtain $i j$ where $a = (A \ ^{\sim} m k) \ \langle i, j \rangle$ and $ij: i < n \land j < n$
    unfolding elements-mat by force
    from $ij \ \text{nb}[\text{of } k] A$ have $\text{norm } ((?B \ ^{\sim} m k) \ \langle i, j \rangle) \leq c1 + c2 \ast \text{real } k \ ^{\sim} d$
    unfolding norm-bound-def by auto
    also have $?(?B \ ^{\sim} m k) \ \langle i, j \rangle = \text{of-real } a$
    unfolding of-real-hom. mat-hom-pow[\text{OF } A, \text{symmetric}] a using $ij A$ by auto
    also have $\text{norm } (\text{complex-of-real } a) = \text{abs } a$ by auto
    finally show $\text{abs } a \leq (c1 + c2 \ast \text{real } k \ ^{\sim} d)$.
  qed
qed

lemma dim-gen-eigenspace-max-jordan-block: assumes $\text{jnf}$: jordan-nf $A$ n-as
  shows $\text{dim-gen-eigenspace } A l d = \text{order } l (\text{char-poly } A) \longleftrightarrow$
  $(\forall n. (n, l) \in \text{set } n\text{-as } \longrightarrow n \leq d)$
proof --
  let $?\text{list} = [(n, e)\leftarrow n\text{-as } . \ e = l]$
  define list where list = [na\leftarrow n-as . snd na = l]
  have list: $?\text{list} = \text{list}$ unfolding list-def by (induct n-as, force+)
  have id: $(\forall n. (n, l) \in \text{set } n\text{-as } \longrightarrow n \leq d) = (\forall n \in \text{set } (\text{map } \text{fst } \text{list}). n \leq d)$
    unfolding list-def by auto
  define ns where ns = $\text{map } \text{fst } \text{list}$
  show $?\text{thesis}$
    unfolding $\text{dim-gen-eigenspace}[\text{OF } \text{jnf}] \text{ jordan-af-order }[\text{OF } \text{jnf}] \text{ list-def}[\text{symmetric}]$
    id
    unfolding ns-def[\text{symmetric}]$\text{proof}$ (induct ns)
    case (Cons n ns)
    show $?\text{case}$
      proof (cases $n \leq d$)
        case True
        thus $?\text{thesis}$ using Cons by auto
      next
        case False
hence $n > d$ by auto

moreover have sum-list (map (min d) ns) $\leq$ 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 $\in$ carrier-mat n n
and nonneg: real-nonneg-mat A
and 1: poly (char-poly A) 1 = 0 $\implies$
order 1 (char-poly A) $> d + 1$ $\implies$
dim-gen-eigenspace A 1 (d+1) = order 1 (char-poly A)
shows $\exists c1 c2. \forall k. \text{norm-bound} (A ^\wedge m ^k) (c1 + c2 \ast of-nat k ^ d)$
proof
from char-poly-factorized[OF A] obtain as where cA: char-poly A = ($\prod a \leftarrow as. [-a, 1]$)
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: $x ^\wedge d = x ^\wedge ((d + 1) - 1)$ 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) $\in$ 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 $\leq$ 1.
assumee a: cmod a = 1
from rt have a $\in$ spectrum A using A spectrum-root-char-poly by auto
hence 11: 1 $\in$ cmod $^1$ spectrum A using a by auto
note spec = spectral-radius-mem-max[OF A n0]
from spec(2)[OF 11] have le: 1 $\leq$ spectral-radius A.
from spec(1)[unfolded spectrum-root-char-poly[OF A]] sr have spectral-radius A $\leq$ 1 by auto
with le have sr: spectral-radius A = 1 by auto
show n $\leq$ 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 $\geq$ n and mem: (N, 1) $\in$ 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 \(N \leq \text{order 1}\)
\((\text{char-poly } A)\).

with \(N \text{nd} \) have \(d + 1 < \text{order 1}\) (\(\text{char-poly } A\)) by simp

from \(1\)\{OF rt this, unfolded \(\text{dim-gen-eigenspace-max-jordan-block}[OF jnf]\)\]
mem \(N \text{nd}

show False by force

qed

lemma jnf-complexity-1-real: fixes \(A::\text{real mat}\)

assumes \(A: A \in \text{carrier-mat } n\ n\)

and \(\text{nonneg}: \text{nonneg-mat } A\)

and \(\text{sr}: \bigwedge x. \text{poly } (\text{char-poly } A) x = 0 \Rightarrow x \leq 1\)

and \(\text{jb}: \text{poly } (\text{char-poly } A) 1 = 0 \Rightarrow\)

order 1 (\(\text{char-poly } A\)) > \(d + 1 \Rightarrow\)

\(\text{dim-gen-eigenspace } A 1\ (d+1) = \text{order 1}\) (\(\text{char-poly } A\))

shows \(\exists c_1 c_2. \forall k. a \in \text{elements-mat } (A \hat{~} m k) \rightarrow |a| \leq c_1 + c_2 \times \text{real } k \times d\)

proof --

let \(?c= \text{complex-of-real}\)

let \(?A= \text{map-mat } ?c A\)

have \(A': \ ?A \in \text{carrier-mat } n\ n\ using A\ by \text{auto}\)

have \(\text{nn}: \ ?A \in \text{real-nonneg-mat } ?A using \text{nonneg } A\ \text{unfolding}\ \text{nonneg-mat-def}\ \text{real-nonneg-mat-def}\)

by \((\text{force simp: elements-mat})\)

have \(1: 1 = ?c 1\ by \text{auto}\)

note \(cp = \text{of-real-hom}\ . \text{char-poly-hom}[OF } A\)

have \(\text{hom}: \text{map-poly-inj-idom-divide-hom complex-of-real ..}\)

show \(?thesis\)

proof \((\text{rule norm-bound-complex-to-real}[OF } A\ \text{jnf-complexity-1-complex}[OF } A'\ \text{nn}])\)

\(\text{unfold cp of-real-hom.poly-map-poly-1, unfold } 1\)
\(\text{of-real-hom.hom-dim-gen-eigenspace}[OF } A\)

\(\text{map-poly-inj-idom-divide-hom.order-hom}[OF } \text{hom}, \text{goal-cases})\)

\(\text{case } 2\)

thus \(?case using } \text{jb by auto}\)

next

\(\text{case } (1 \ x)\)

let \(?cp= \text{char-poly } A\)

assume \(rt: \text{poly } (\text{map-poly } \ ?c \ ?cp) x = 0\)

with \(\text{degree-monic-char-poly}[OF } A', \text{unfolded } cp\) have \(\text{n0: } n \neq 0\)

using \(\text{degree0-coeffs[of } ?cp\] by \((\text{cases } n, \text{auto})\)

from \(\text{perron-frobenius-nonneg}[OF } A\ \text{nonneg } \text{n0}\)

obtain \(sr\ \text{ks } f\) where \(sr0: 0 \leq sr\ and \ \text{ks: } 0 \notin \text{set } \text{ks } \text{ks} \neq []\)

and \(cp: \ ?cp = (\prod k \sim \text{ks. monom } 1 k - [sr \hat{~} k]) \ast f\)

and \(rtf: \text{poly } (\text{map-poly } \ ?c \ f) x = 0 \Rightarrow \text{cmod } x < sr\ by \text{auto}\)

have \(sr-rt: \text{poly } ?cp sr = 0\ unfolding cp\ \text{poly-prod-list-zero-iff poly-mult-zero-iff}\)

using \(\text{ks}\)

by \((\text{cases } \text{ks, auto simp: poly-monom})\)
from sr | OF sr-rt | have sr1: sr ≤ 1 .
interpret c: map-poly-comm-ring-hom ?c ..
from rt | unfolded cp e.hom-mul c e.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 e.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)) °m (d + 1)) = ord)))

lemma check-matrix-complexity: assumes A: A ∈ carrier-mat n n and nn: nonneg-mat A
  and check: check-matrix-complexity A (char-poly A) d
shows ∃ c1 c2. ∀ k a. a ∈ elements-mat (A °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)) °m (d + 1))
    unfolding dim-gen-eigenspace-def

132
by (rule arg-cong[of - - λ x. kernel-dim (x ⊕ 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


