Perron-Frobenius Theorem for Spectral Radius Analysis*

Jose Divasón, Ondej Kunar, René Thiemann and Akihisa Yamada

March 17, 2025

Abstract

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

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

Contents

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

1 Introduction

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

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

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

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

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

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

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

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

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

Eventually we want to combine both theorems to obtain:

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

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

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

2 Elimination of CARD('n)

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

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

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

lemma *n-zero-nonempty*: $n \neq 0 \Longrightarrow \{0 ..< n :: nat\} \neq \{\}$ ⟨*proof*⟩

lemma *type-impl-card-n*: **assumes** $\exists$(*Rep* :: *'a* $\Rightarrow$ *nat*) *Abs*. *type-definition Rep Abs* {*0* ..< *n* :: *nat*}
shows *class.finite* (*TYPE*(*'a*)) $\wedge$ *CARD*(*'a*) = *n*
⟨*proof*⟩

⟨*ML*⟩

end

3 Connecting HMA-matrices with JNF-matrices

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

3.1 Bijections between index types of HMA and natural numbers

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

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

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

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

definition *univ-list* :: *'a* :: *finite list* **where**
univ-list = (*SOME xs*. *distinct xs* $\wedge$ *set xs* = *UNIV*)

lemma *univ-list*: *distinct* (*univ-list* :: *'a list*) *set univ-list* = (*UNIV* :: *'a* :: *finite set*)
⟨*proof*⟩

definition *to-nat* :: *'a* :: *finite* $\Rightarrow$ *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*)
⟨*proof*⟩

lemma *to-nat-ex*: ∃! *i*. *univ-list* ! *i* = (*a* :: *'a* :: *finite*) ∧ *i* < *length* (*univ-list* :: *'a list*)
⟨*proof*⟩

lemma *to-nat-less-card*: *to-nat* (*a* :: *'a* :: *finite*) < *CARD*(*'a*)
⟨*proof*⟩

lemma *to-nat-from-nat-id*:
assumes *i*: *i* < *CARD*(*'a* :: *finite*)
shows *to-nat* (*from-nat i* :: *'a*) = *i*
⟨*proof*⟩

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*⟩

lemma *from-nat-to-nat-id*[*simp*]:
(*from-nat* (*to-nat a*)) = (*a*::*'a* :: *finite*)
⟨*proof*⟩

lemma *to-nat-inj*[*simp*]: **assumes** *to-nat a* = *to-nat b*
shows *a* = *b*
⟨*proof*⟩

lemma *range-to-nat*: *range* (*to-nat* :: *'a* :: *finite* ⇒ *nat*) = {*0* ..< *CARD*(*'a*)} (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *inj-to-nat*: *inj to-nat* ⟨*proof*⟩

lemma *bij-to-nat*: *bij-betw to-nat* (*UNIV* :: *'a* :: *finite set*) {*0* ..< *CARD*(*'a*)}
⟨*proof*⟩

lemma *numeral-nat*: (*numeral m1* :: *nat*) ∗ *numeral n1* ≡ *numeral* (*m1* ∗ *n1*)
(*numeral m1* :: *nat*) + *numeral n1* ≡ *numeral* (*m1* + *n1*) ⟨*proof*⟩

lemmas *card-num-simps* =
card-num1 card-bit0 card-bit1

mult-num-simps
add-num-simps
eq-num-simps
mult-Suc-right mult-0-right One-nat-def add.right-neutral
numeral-nat Suc-numeral

end

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

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

Prefer certain constants and lemmas without prefix.

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

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

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

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

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

lifting-forget *poly.lifting*

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

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

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

definition *similar-matrix-wit*
:: *′a* :: *semiring-1* ^ *′n* ^ *′n* ⇒ *′a* ^ *′n* ^ *′n* ⇒ *′a* ^ *′n* ^ *′n* ⇒ *′a* ^ *′n* ^ *′n* ⇒ *bool*
where

similar-matrix-wit A B P Q = (*P* ** *Q* = *mat 1* ∧ *Q* ** *P* = *mat 1* ∧ *A* = *P* ** *B* ** *Q*)

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

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

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

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

lemma *vec-elements-h-def′*: *vec-elements-h v* = {*v* $*h i* | *i*. *True*}
⟨*proof*⟩

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

lemma *elements-mat-h-def′*: *elements-mat-h A* = {*A* $*h i* $*h j* | *i j*. *True*}
⟨*proof*⟩

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

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

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

lemma *spectral-radius-ev-def*: *spectral-radius A* = *Max* (*norm* ‘ (*Collect* (*eigen-value A*)))
⟨*proof*⟩

lemma *elements-mat*: *elements-mat A* = {*A* $$ (*i*,*j*) | *i j*. *i* < *dim-row A* ∧ *j* < *dim-col A*}
⟨*proof*⟩

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

lemma *vec-elements*: *vec-elements v* = { *v* $ *i* | *i*. *i* < *dim-vec v*}
⟨*proof*⟩

context includes *vec.lifting*
begin
end

definition $\textit{from-hma}_v$ :: $'a$ ^ $'n$ ⇒ $'a$ *Matrix.vec* **where**
$\textit{from-hma}_v\ v = \textit{Matrix.vec}\ \textit{CARD}('n)\ (\lambda\ i.\ v\ \$h\ \textit{from-nat}\ i)$

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

definition $\textit{to-hma}_v$ :: $'a$ *Matrix.vec* ⇒ $'a$ ^ $'n$ **where**
$\textit{to-hma}_v\ v = (\chi\ i.\ v\ \$v\ \textit{to-nat}\ i)$

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

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

lemma $\textit{to-hma-from-hma}_v[\textit{simp}]$: $\textit{to-hma}_v\ (\textit{from-hma}_v\ v) = v$
⟨*proof*⟩

lemma $\textit{to-hma-from-hma}_m[\textit{simp}]$: $\textit{to-hma}_m\ (\textit{from-hma}_m\ v) = v$
⟨*proof*⟩

lemma $\textit{from-hma-to-hma}_v[\textit{simp}]$:
$v \in \textit{carrier-vec}\ (\textit{CARD}('n)) \Longrightarrow \textit{from-hma}_v\ (\textit{to-hma}_v\ v :: 'a\ \hat{}\ 'n) = v$
⟨*proof*⟩

lemma $\textit{from-hma-to-hma}_m[\textit{simp}]$:
$A \in \textit{carrier-mat}\ (\textit{CARD}('nr))\ (\textit{CARD}('nc)) \Longrightarrow \textit{from-hma}_m\ (\textit{to-hma}_m\ A :: 'a\ \hat{}\ 'nc\ \hat{}\ 'nr) = A$
⟨*proof*⟩

lemma $\textit{from-hma}_v\textit{-inj}[\textit{simp}]$: $\textit{from-hma}_v\ x = \textit{from-hma}_v\ y \longleftrightarrow x = y$
⟨*proof*⟩

lemma $\textit{from-hma}_m\textit{-inj}[\textit{simp}]$: $\textit{from-hma}_m\ x = \textit{from-hma}_m\ y \longleftrightarrow x = y$
⟨*proof*⟩

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

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

definition *HMA-I* :: *nat* ⇒ $'n$ :: *finite* ⇒ *bool* **where**
$\textit{HMA-I} = (\lambda\ i\ a.\ i = \textit{to-nat}\ a)$

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*)))
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *Domainp-HMA-I* [*transfer-domain-rule*]:
Domainp (*HMA-I* :: *nat* ⇒ *'n* :: *finite* ⇒ *bool*) = (λ *i*. *i* < *CARD*(*'n*)) (**is** *?l* = *?r*)
⟨*proof*⟩

lemma *bi-unique-HMA-V* [*transfer-rule*]: *bi-unique HMA-V left-unique HMA-V right-unique HMA-V*
⟨*proof*⟩

lemma *bi-unique-HMA-M* [*transfer-rule*]: *bi-unique HMA-M left-unique HMA-M right-unique HMA-M*
⟨*proof*⟩

lemma *bi-unique-HMA-I* [*transfer-rule*]: *bi-unique HMA-I left-unique HMA-I right-unique HMA-I*
⟨*proof*⟩

lemma *right-total-HMA-V* [*transfer-rule*]: *right-total HMA-V*
⟨*proof*⟩

lemma *right-total-HMA-M* [*transfer-rule*]: *right-total HMA-M*
⟨*proof*⟩

lemma *right-total-HMA-I* [*transfer-rule*]: *right-total HMA-I*
⟨*proof*⟩

lemma *HMA-V-index* [*transfer-rule*]: (*HMA-V* ===> *HMA-I* ===> (=)) ($*v*) ($*h*)
⟨*proof*⟩

We introduce the index function to have pointwise access to HMA-matrices by a constant. Otherwise, the transfer rule with λ*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* ===> (=)) (λ *A i j*. *A* $$ (*i*,*j*)) *index-hma*
⟨*proof*⟩

lemma *HMA-V-0* [*transfer-rule*]: *HMA-V* (0_v *CARD*('*n*)) (*0* :: '*a* :: *zero* ^ '*n*)
⟨*proof*⟩

lemma *HMA-M-0* [*transfer-rule*]:
HMA-M (0_m *CARD*('*nr*) *CARD*('*nc*)) (*0* :: '*a* :: *zero* ^ '*nc* ^ '*nr*)
⟨*proof*⟩

lemma *HMA-M-1*[*transfer-rule*]:
HMA-M (1_m (*CARD*('*n*))) (*mat 1* :: '*a*::{*zero*,*one*}^'*n*^'*n*)
⟨*proof*⟩

lemma *from-hma*$_v$*-add*: *from-hma*$_v$ *v* + *from-hma*$_v$ *w* = *from-hma*$_v$ (*v* + *w*)
⟨*proof*⟩

lemma *HMA-V-add* [*transfer-rule*]: (*HMA-V* ===> *HMA-V* ===> *HMA-V*) (+) (+)
⟨*proof*⟩

lemma *from-hma*$_v$*-diff*: *from-hma*$_v$ *v* − *from-hma*$_v$ *w* = *from-hma*$_v$ (*v* − *w*)
⟨*proof*⟩

lemma *HMA-V-diff* [*transfer-rule*]: (*HMA-V* ===> *HMA-V* ===> *HMA-V*) (−) (−)
⟨*proof*⟩

lemma *from-hma*$_m$*-add*: *from-hma*$_m$ *a* + *from-hma*$_m$ *b* = *from-hma*$_m$ (*a* + *b*)
⟨*proof*⟩

lemma *HMA-M-add* [*transfer-rule*]: (*HMA-M* ===> *HMA-M* ===> *HMA-M*) (+) (+)
⟨*proof*⟩

lemma *from-hma*$_m$*-diff*: *from-hma*$_m$ *a* − *from-hma*$_m$ *b* = *from-hma*$_m$ (*a* − *b*)
⟨*proof*⟩

lemma *HMA-M-diff* [*transfer-rule*]: (*HMA-M* ===> *HMA-M* ===> *HMA-M*) (−) (−)
⟨*proof*⟩

lemma *scalar-product*: **fixes** *v* :: '*a* :: *semiring-1* ^ '*n*
shows *scalar-prod* (*from-hma*$_v$ *v*) (*from-hma*$_v$ *w*) = *scalar-product v w*
⟨*proof*⟩

lemma [*simp*]:

from-hma$_m$ (*y* :: *'a* ^ *'nc* ^ *'nr*) ∈ *carrier-mat* (*CARD*(*'nr*)) (*CARD*(*'nc*))
dim-row (*from-hma*$_m$ (*y* :: *'a* ^ *'nc* ^ *'nr*)) = *CARD*(*'nr*)
dim-col (*from-hma*$_m$ (*y* :: *'a* ^ *'nc* ^ *'nr*)) = *CARD*(*'nc*)
⟨*proof*⟩

lemma [*simp*]:
from-hma$_v$ (*y* :: *'a* ^ *'n*) ∈ *carrier-vec* (*CARD*(*'n*))
dim-vec (*from-hma*$_v$ (*y* :: *'a* ^ *'n*)) = *CARD*(*'n*)
⟨*proof*⟩

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

lemma *HMA-scalar-prod* [*transfer-rule*]:
(*HMA-V* ===> *HMA-V* ===> (=)) *scalar-prod scalar-product*
⟨*proof*⟩

lemma *HMA-row* [*transfer-rule*]: (*HMA-I* ===> *HMA-M* ===> *HMA-V*) (λ *i* *a*. *Matrix.row a i*) *row*
⟨*proof*⟩

lemma *HMA-col* [*transfer-rule*]: (*HMA-I* ===> *HMA-M* ===> *HMA-V*) (λ *i* *a*. *col a i*) *column*
⟨*proof*⟩

definition *mk-mat* :: (*'i* ⇒ *'j* ⇒ *'c*) ⇒ *'c*^*'j*^*'i* **where**
mk-mat f = (χ *i j*. *f i j*)

definition *mk-vec* :: (*'i* ⇒ *'c*) ⇒ *'c*^*'i* **where**
mk-vec f = (χ *i*. *f i*)

lemma *HMA-M-mk-mat*[*transfer-rule*]: ((*HMA-I* ===> *HMA-I* ===> (=)) ===> *HMA-M*)
(λ *f*. *Matrix.mat* (*CARD*(*'nr*)) (*CARD*(*'nc*)) (λ (*i*,*j*). *f i j*))
(*mk-mat* :: ((*'nr* ⇒ *'nc* ⇒ *'a*) ⇒ *'a*^*'nc*^*'nr*))
⟨*proof*⟩

lemma *HMA-M-mk-vec*[*transfer-rule*]: ((*HMA-I* ===> (=)) ===> *HMA-V*)
(λ *f*. *Matrix.vec* (*CARD*(*'n*)) (λ *i*. *f i*))
(*mk-vec* :: ((*'n* ⇒ *'a*) ⇒ *'a*^*'n*))
⟨*proof*⟩

lemma *mat-mult-scalar*: *A* ** *B* = *mk-mat* (λ *i j*. *scalar-product* (*row i A*) (*column j B*))
⟨*proof*⟩

lemma *mult-mat-vec-scalar*: *A* *v *v* = *mk-vec* (λ *i*. *scalar-product* (*row i A*) *v*)
⟨*proof*⟩

lemma *dim-row-transfer-rule*:
HMA-M A $(A' :: 'a$ ^ $'nc$ ^ $'nr) \Longrightarrow (=)$ (*dim-row A*) (*CARD*($'nr$))
⟨*proof*⟩

lemma *dim-col-transfer-rule*:
HMA-M A $(A' :: 'a$ ^ $'nc$ ^ $'nr) \Longrightarrow (=)$ (*dim-col A*) (*CARD*($'nc$))
⟨*proof*⟩

lemma *HMA-M-mult* [*transfer-rule*]: (*HMA-M* ===> *HMA-M* ===> *HMA-M*) $((*))$ $((**))$
⟨*proof*⟩

lemma *HMA-V-smult* [*transfer-rule*]: ((=) ===> *HMA-V* ===> *HMA-V*) $(\cdot_v)$ $((*s))$
⟨*proof*⟩

lemma *HMA-M-mult-vec* [*transfer-rule*]: (*HMA-M* ===> *HMA-V* ===> *HMA-V*) $((*_v))$ $((*v))$
⟨*proof*⟩

lemma *HMA-det* [*transfer-rule*]: (*HMA-M* ===> (=)) *Determinant.det*
(*det* :: $'a$ :: *comm-ring-1* ^ $'n$ ^ $'n \Rightarrow 'a$)
⟨*proof*⟩

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

(*Finite-Cartesian-Product.mat* :: $'a$::*semiring-1* $\Rightarrow 'a$^$'n$^$'n$)
⟨*proof*⟩

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

(λ *A B*. *A* + *map-mat uminus B*) $((-)$ :: $'a$ :: *group-add* ^$'nc$^$'nr \Rightarrow 'a$^$'nc$^$'nr$ $\Rightarrow 'a$^$'nc$^$'nr$)
⟨*proof*⟩

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

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

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

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

definition *sum-UNIV-set* :: (*nat* ⇒ ′*a* :: *comm-monoid-add*) ⇒ *nat* ⇒ ′*a* **where**
sum-UNIV-set f n = *sum f* {..<*n*}

definition *HMA-T* :: *nat* ⇒ ′*n* :: *finite itself* ⇒ *bool* **where**
HMA-T n - = (*n* = *CARD*(′*n*))

lemma *HMA-mat2matofpoly*[*transfer-rule*]: (*HMA-M* ===> *HMA-M*) (λ*x*. *map-mat* (λ*a*. [:*a*:]) *x*) *mat2matofpoly*
⟨*proof*⟩

lemma *HMA-char-poly* [*transfer-rule*]:
((*HMA-M* :: (′*a*:: *comm-ring-1 mat* ⇒ ′*a*^′*n*^′*n* ⇒ *bool*)) ===> (=)) *char-poly charpoly*
⟨*proof*⟩

lemma *HMA-eigen-vector* [*transfer-rule*]: (*HMA-M* ===> *HMA-V* ===> (=)) *eigenvector eigen-vector*
⟨*proof*⟩

lemma *HMA-eigen-value* [*transfer-rule*]: (*HMA-M* ===> (=) ===> (=)) *eigenvalue eigen-value*
⟨*proof*⟩

lemma *HMA-spectral-radius* [*transfer-rule*]:
(*HMA-M* ===> (=)) *Spectral-Radius.spectral-radius spectral-radius*
⟨*proof*⟩

lemma *HMA-elements-mat*[*transfer-rule*]: ((*HMA-M* :: (′*a mat* ⇒ ′*a* ^ ′*nc* ^ ′*nr* ⇒ *bool*)) ===> (=))
elements-mat elements-mat-h
⟨*proof*⟩

lemma *HMA-vec-elements*[*transfer-rule*]: ((*HMA-V* :: (′*a Matrix.vec* ⇒ ′*a* ^ ′*n* ⇒ *bool*)) ===> (=))
vec-elements vec-elements-h
⟨*proof*⟩

lemma *norm-bound-elements-mat*: *norm-bound A b* = (∀ *x* ∈ *elements-mat A*. *norm x* ≤ *b*)
⟨*proof*⟩

lemma *HMA-normbound* [*transfer-rule*]:
((*HMA-M* :: ′*a* :: *real-normed-field mat* ⇒ ′*a* ^ ′*nc* ^ ′*nr* ⇒ *bool*) ===> (=) ===> (=))
norm-bound normbound
⟨*proof*⟩

lemma *HMA-map-matrix* [*transfer-rule*]:
((=) ===> *HMA-M* ===> *HMA-M*) *map-mat map-matrix*
⟨*proof*⟩

lemma *HMA-transpose-matrix* [*transfer-rule*]:
(*HMA-M* ===> *HMA-M*) *transpose-mat transpose*
⟨*proof*⟩

lemma *HMA-map-vector* [*transfer-rule*]:
((=) ===> *HMA-V* ===> *HMA-V*) *map-vec map-vector*
⟨*proof*⟩

lemma *HMA-similar-mat-wit* [*transfer-rule*]:
((*HMA-M* :: - ⇒ ′*a* :: *comm-ring-1* ^ ′*n* ^ ′*n* ⇒ -) ===> *HMA-M* ===> *HMA-M* ===> *HMA-M* ===> (=))
similar-mat-wit similar-matrix-wit
⟨*proof*⟩

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

similar-mat similar-matrix
⟨*proof*⟩

lemma *HMA-spectrum*[*transfer-rule*]: (*HMA-M* ===> (=)) *spectrum Spectrum*
⟨*proof*⟩

lemma *HMA-M-erase-mat*[*transfer-rule*]: (*HMA-M* ===> *HMA-I* ===> *HMA-I* ===> *HMA-M*) *mat-erase erase-mat*
⟨*proof*⟩

lemma *HMA-M-sum-UNIV*[*transfer-rule*]:
((*HMA-I* ===> (=)) ===> *HMA-T* ===> (=)) *sum-UNIV-set sum-UNIV-type*
⟨*proof*⟩
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*
⟨*proof*⟩

lemma *matrix-vector-right-distrib*: $M *v (v + w) = M *v\ v + M *v\ w$
⟨*proof*⟩

lemma *matrix-vector-right-distrib-diff*: $(M :: 'a :: ring\text{-}1\ \hat{}\ 'nr\ \hat{}\ 'nc) *v (v - w) = M *v\ v - M *v\ w$
⟨*proof*⟩

lemma *eigen-value-root-charpoly*:
eigen-value $A\ k \longleftrightarrow$ *poly* (*charpoly* $(A :: 'a :: field\ \hat{}\ 'n\ \hat{}\ 'n)$) $k = 0$
⟨*proof*⟩

lemma *finite-spectrum*: **fixes** $A :: 'a :: field\ \hat{}\ 'n\ \hat{}\ 'n$
shows *finite* (*Collect* (*eigen-value* A))
⟨*proof*⟩

lemma *non-empty-spectrum*: **fixes** $A :: complex\ \hat{}\ 'n\ \hat{}\ 'n$
shows *Collect* (*eigen-value* A) $\neq \{\}$
⟨*proof*⟩

lemma *charpoly-transpose*: *charpoly* (*transpose* $A :: 'a :: field\ \hat{}\ 'n\ \hat{}\ 'n$) = *charpoly* A
⟨*proof*⟩

lemma *eigen-value-transpose*: *eigen-value* (*transpose* $A :: 'a :: field\ \hat{}\ 'n\ \hat{}\ 'n$) v = *eigen-value* $A\ v$
⟨*proof*⟩

lemma *matrix-diff-vect-distrib*: $(A - B) *v\ v = A *v\ v - B *v (v :: 'a :: ring\text{-}1\ \hat{}\ 'n)$
⟨*proof*⟩

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

⟨*proof*⟩

lemma *pderiv-char-poly-erase-mat*: **fixes** $A :: 'a :: idom\ \hat{}\ 'n\ \hat{}\ 'n$
shows *monom* 1 1 * *pderiv* (*charpoly* A) = *sum* (λ *i*. *charpoly* (*erase-mat* A *i* *i*)) *UNIV*
⟨*proof*⟩

lemma *degree-monic-charpoly*: **fixes** $A :: 'a :: comm\text{-}ring\text{-}1\ \hat{}\ 'n\ \hat{}\ 'n$
shows *degree* (*charpoly* A) = *CARD*('n) $\wedge$ *monic* (*charpoly* A)
⟨*proof*⟩

end

4 Perron-Frobenius Theorem

4.1 Auxiliary Notions

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

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

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

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

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

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

lemma *real-non-neg-matD*: **assumes** *real-non-neg-mat A*
shows *A $h i $h j* ∈ ℝ *Re* (*A $h i $h j*) ≥ *0*
⟨*proof*⟩

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

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

context includes *lifting-syntax*
begin

lemma *HMA-real-non-neg-mat* [*transfer-rule*]:
((*HMA-M* :: *complex mat* ⇒ *complex* ^ *'nc* ^ *'nr* ⇒ *bool*) ===> (=))
real-nonneg-mat real-non-neg-mat
⟨*proof*⟩

lemma *HMA-real-non-neg-vec* [*transfer-rule*]:
((*HMA-V* :: *complex Matrix.vec* ⇒ *complex* ^ *'n* ⇒ *bool*) ===> (=))
real-nonneg-vec real-non-neg-vec
⟨*proof*⟩

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

⟨*proof*⟩

end

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

context includes *lifting-syntax*
begin
lemma *HMA-pow-mat*[*transfer-rule*]:
((*HMA-M* ::*'a::{semiring-1} mat ⇒ 'a^'n^'n ⇒ bool*) ===> (=) ===> *HMA-M*)
pow-mat matpow
⟨*proof*⟩
end

lemma *trancl-image*:
$(i,j) \in R^+ \Longrightarrow (f\ i,\ f\ j) \in (map\text{-}prod\ f\ f\ `\ R)^+$
⟨*proof*⟩

lemma *inj-trancl-image*: **assumes** *inj*: *inj f*
shows $(f\ i,\ f\ j) \in (map\text{-}prod\ f\ f\ `\ R)^+ = ((i,j) \in R^+)$ (**is** *?l = ?r*)
⟨*proof*⟩

lemma *matrix-add-rdistrib*: ((*B* + *C*) ** *A*) = (*B* ** *A*) + (*C* ** *A*)
⟨*proof*⟩

lemma *norm-smult*: *norm* ((*a* :: *real*) *s *x*) = *abs a* * *norm x*
⟨*proof*⟩

lemma *nonneg-mat-mult*:
nonneg-mat A ⟹ *nonneg-mat B* ⟹ *A* ∈ *carrier-mat nr n*
⟹ *B* ∈ *carrier-mat n nc* ⟹ *nonneg-mat* (*A* * *B*)
⟨*proof*⟩

lemma *nonneg-mat-power*: **assumes** *A* ∈ *carrier-mat n n nonneg-mat A*
shows *nonneg-mat* ($A\ \widehat{}_m\ k$)
⟨*proof*⟩

lemma *nonneg-matD*: **assumes** *nonneg-mat A*
and *i* < *dim-row A* **and** *j* < *dim-col A*
shows *A* $$ (*i,j*) ≥ *0*
⟨*proof*⟩

lemma (**in** *comm-ring-hom*) *similar-mat-wit-hom*: **assumes**
similar-mat-wit A B C D
shows *similar-mat-wit* (mat_h *A*) (mat_h *B*) (mat_h *C*) (mat_h *D*)
⟨*proof*⟩

lemma (**in** *comm-ring-hom*) *similar-mat-hom*:
  similar-mat A B $\Longrightarrow$ *similar-mat* (mat_h *A*) (mat_h *B*)
  ⟨*proof*⟩

lemma *det-dim-1*: **assumes** *A*: $A \in$ *carrier-mat n n*
  and *n*: $n = 1$
shows *Determinant.det A* = *A* $$ (*0*,*0*)
  ⟨*proof*⟩

lemma *det-dim-2*: **assumes** *A*: $A \in$ *carrier-mat n n*
  and *n*: $n = 2$
shows *Determinant.det A* = *A* $$ (*0*,*0*) ∗ *A* $$ (*1*,*1*) − *A* $$ (*0*,*1*) ∗ *A* $$ (*1*,*0*)
⟨*proof*⟩

lemma *jordan-nf-root-char-poly*: **fixes** *A* :: *'a* :: {*semiring-no-zero-divisors*, *idom*} *mat*
  assumes *jordan-nf A n-as*
  and (*m*, *lam*) ∈ *set n-as*
shows *poly* (*char-poly A*) *lam* = *0*
⟨*proof*⟩

lemma *inverse-power-tendsto-zero*:
  (λ*x*. *inverse* ((*of-nat x* :: *'a* :: *real-normed-div-algebra*) ^ *Suc d*)) ⟶ *0*
⟨*proof*⟩

lemma *inverse-of-nat-tendsto-zero*:
  (λ*x*. *inverse* (*of-nat x* :: *'a* :: *real-normed-div-algebra*)) ⟶ *0*
  ⟨*proof*⟩

lemma *poly-times-exp-tendsto-zero*: **assumes** *b*: *norm* (*b* :: *'a* :: *real-normed-field*) < *1*
  shows (λ *x*. *of-nat x* ^ *k* ∗ *b* ^ *x*) ⟶ *0*
⟨*proof*⟩

lemma (**in** *linorder-topology*) *tendsto-Min*: **assumes** *I*: *I* ≠ {} **and** *fin*: *finite I*
  shows (⋀*i*. *i* ∈ *I* ⟹ (*f i* ⟶ *a i*) *F*) ⟹ ((λ*x*. *Min* ((λ *i*. *f i x*) ‘ *I*)) ⟶
    (*Min* (*a* ‘ *I*) :: *'a*)) *F*
  ⟨*proof*⟩

lemma *tendsto-mat-mult* [*tendsto-intros*]:
  (*f* ⟶ *a*) *F* ⟹ (*g* ⟶ *b*) *F* ⟹ ((λ*x*. *f x* ∗∗ *g x*) ⟶ *a* ∗∗ *b*) *F*
  for *f* :: *'a* ⇒ *'b* :: {*semiring-1*, *real-normed-algebra*} ^ *'n1* ^ *'n2*
  ⟨*proof*⟩

lemma *tendsto-matpower* [*tendsto-intros*]: (*f* ⟶ *a*) *F* ⟹ ((λ*x*. *matpow* (*f x*) *n*) ⟶ *matpow a n*) *F*
  for *f* :: *'a* ⇒ *'b* :: {*semiring-1*, *real-normed-algebra*} ^ *'n* ^ *'n*

⟨*proof*⟩

lemma *continuous-matpow*: *continuous-on R* (λ *A* :: *'a* :: {*semiring-1*, *real-normed-algebra-1*} ^ *'n* ^ *'n*. *matpow A n*)
⟨*proof*⟩

lemma *vector-smult-distrib*: (*A* *v ((*a* :: *'a* :: *comm-ring-1*) *s *x*)) = *a* *s ((*A* *v *x*))
⟨*proof*⟩

instance *real* :: *ordered-semiring-strict*
⟨*proof*⟩

lemma *poly-tendsto-pinfty*: **fixes** *p* :: *real poly*
assumes *lead-coeff p* > *0 degree p* ≠ *0*
shows *poly p* ⟶ ∞
⟨*proof*⟩

lemma *div-lt-nat*: (*j* :: *nat*) < *x* ∗ *y* ⟹ *j div x* < *y*
⟨*proof*⟩

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

lemma *diagvector-mult-vector*[*simp*]: *diagvector x* *v *y* = (χ *i*. *x i* ∗ *y* $ *i*)
⟨*proof*⟩

lemma *diagvector-mult-left*: *diagvector x* ∗∗ *A* = (χ *i j*. *x i* ∗ *A* $ *i* $ *j*) (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *diagvector-mult-right*: *A* ∗∗ *diagvector x* = (χ *i j*. *A* $ *i* $ *j* ∗ *x j*) (**is** *?A* = *?B*)
⟨*proof*⟩

lemma *diagvector-mult*[*simp*]: *diagvector x* ∗∗ *diagvector y* = *diagvector* (λ *i*. *x i* ∗ *y i*)
⟨*proof*⟩

lemma *diagvector-const*[*simp*]: *diagvector* (λ *x*. *k*) = *mat k*
⟨*proof*⟩

lemma *diagvector-eq-mat*: *diagvector x* = *mat a* ⟷ *x* = (λ *x*. *a*)
⟨*proof*⟩

lemma *cmod-eq-Re*: **assumes** *cmod x* = *Re x*
shows *of-real* (*Re x*) = *x*
⟨*proof*⟩

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

no-notation
vec-index (**infixl** ‹$› *100*)

lemma *spectral-radius-ev*:
$\exists$ *ev v. eigen-vector A v ev* $\wedge$ *norm ev* = *spectral-radius A*
⟨*proof*⟩

lemma *spectral-radius-max*: **assumes** *eigen-value A v*
shows *norm v* $\leq$ *spectral-radius A*
⟨*proof*⟩

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

definition *norm1* :: *'a* :: *real-normed-field* ^ *'n* $\Rightarrow$ *real* **where**
norm1 v = ($\sum$ *i*$\in$*UNIV*. *norm* (*v* $ *i*))

lemma *norm1-ge-0*: *norm1 v* $\geq$ *0* ⟨*proof*⟩

lemma *norm1-0*[*simp*]: *norm1 0* = *0* ⟨*proof*⟩

lemma *norm1-nonzero*: **assumes** *v* $\neq$ *0*
shows *norm1 v* > *0*
⟨*proof*⟩

lemma *norm1-0-iff*[*simp*]: (*norm1 v* = *0*) = (*v* = *0*)
⟨*proof*⟩

lemma *norm1-scaleR*[*simp*]: *norm1* (*r* $*_R$ *v*) = *abs r* * *norm1 v* ⟨*proof*⟩

lemma *abs-norm1*[*simp*]: *abs* (*norm1 v*) = *norm1 v* ⟨*proof*⟩

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*) = *1*
⟨*proof*⟩

lemma *norm1-cont*[*simp*]: *isCont norm1 v* ⟨*proof*⟩

lemma *norm1-ge-norm*: *norm1 v* $\geq$ *norm v* ⟨*proof*⟩

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

lemma *tendsto-matrix-vector-mult*[*tendsto-intros*]:
((*v) (*A* :: *'a* :: *real-normed-algebra-1* ^ *'n* ^ *'k*) $\longrightarrow$ *A* *v *v*) (*at v within S*)
⟨*proof*⟩

lemma *tendsto-matrix-matrix-mult*[*tendsto-intros*]:
((**) (*A* :: *'a* :: *real-normed-algebra-1* ^ *'n* ^ *'k*) ⟶ *A* ** *B*) (*at B within S*)
⟨*proof*⟩

lemma *matrix-vect-scaleR*: (*A* :: *'a* :: *real-normed-algebra-1* ^ *'n* ^ *'k*) *v (*a* $*_R$ *v*) = *a* $*_R$ (*A* *v *v*)
⟨*proof*⟩

lemma (**in** *inj-semiring-hom*) *map-vector-0*: (*map-vector hom v* = *0*) = (*v* = *0*)
⟨*proof*⟩

lemma (**in** *inj-semiring-hom*) *map-vector-inj*: (*map-vector hom v* = *map-vector hom w*) = (*v* = *w*)
⟨*proof*⟩

lemma (**in** *semiring-hom*) *matrix-vector-mult-hom*:
(*map-matrix hom A*) *v (*map-vector hom v*) = *map-vector hom* (*A* *v *v*)
⟨*proof*⟩

lemma (**in** *semiring-hom*) *vector-smult-hom*:
hom x *s (*map-vector hom v*) = *map-vector hom* (*x* *s *v*)
⟨*proof*⟩

lemma (**in** *inj-comm-ring-hom*) *eigen-vector-hom*:
eigen-vector (*map-matrix hom A*) (*map-vector hom v*) (*hom x*) = *eigen-vector A v x*
⟨*proof*⟩

end

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

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

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

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

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

private definition *max-v-ev* :: (*complex*^*'n*) × *complex* **where**

$max\text{-}v\text{-}ev = (SOME\ v\text{-}ev.\ eigen\text{-}vector\ A\ (fst\ v\text{-}ev)\ (snd\ v\text{-}ev)$
$\wedge\ norm\ (snd\ v\text{-}ev) = sr)$

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

private lemma *max-v-ev*:
eigen-vector A max-v max-ev
$norm\ max\text{-}ev = sr$
$norm1\ max\text{-}v = 1$
⟨*proof*⟩

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

private lemma *closedS*: *closed S*
⟨*proof*⟩ **lemma** *boundedS*: *bounded S*
⟨*proof*⟩ **lemma** *compactS*: *compact S*
⟨*proof*⟩ **lemmas** *rnn* = *real-non-neg-matD*[*OF rnnA*]

lemma *B-norm*: $B\ \$\ i\ \$\ j = norm\ (A\ \$\ i\ \$\ j)$
⟨*proof*⟩

lemma *mult-B-mono*: **assumes** $\bigwedge\ i.\ v\ \$\ i \geq w\ \$\ i$
shows $(B *v\ v)\ \$\ i \geq (B *v\ w)\ \$\ i$ ⟨*proof*⟩ **lemma** *non-emptyS*: $S \neq \{\}$
⟨*proof*⟩ **lemma** *convexS*: *convex S*
⟨*proof*⟩ **abbreviation** (*input*) $r :: real \Rightarrow complex$ **where**
$r \equiv of\text{-}real$

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

private lemma *rv-0*: $(rv\ v = 0) = (v = 0)$
⟨*proof*⟩ **lemma** *rv-mult*: $A *v\ rv\ v = rv\ (B *v\ v)$
⟨*proof*⟩

context
assumes *zero-no-ev*: $\bigwedge\ v.\ v \in S \Longrightarrow A *v\ rv\ v \neq 0$
begin
private lemma *normB-S*: **assumes** v: $v \in S$
shows $norm1\ (B *v\ v) \neq 0$
⟨*proof*⟩ **lemma** *image-f*: $f \in S \rightarrow S$

⟨*proof*⟩ **lemma** *cont-f*: *continuous-on S f*
⟨*proof*⟩ **lemma** *perron-frobenius-positive-ev*:
∃ *v. eigen-vector A v* (*r sr*) ∧ *real-non-neg-vec v*
⟨*proof*⟩
end

qualified lemma *perron-frobenius-both*:
∃ *v. eigen-vector A v* (*r sr*) ∧ *real-non-neg-vec v*
⟨*proof*⟩
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 ∃ *v. eigen-vector A v* (*of-real* (*spectral-radius A*)) ∧ *real-non-neg-vec v*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *rcis-mult-cis*[*simp*]: *rcis n a* ∗ *cis b* = *rcis n* (*a* + *b*) ⟨*proof*⟩
lemma *rcis-div-cis*[*simp*]: *rcis n a* / *cis b* = *rcis n* (*a* − *b*) ⟨*proof*⟩

lemma *cis-plus-2pi*[*simp*]: *cis* (*x* + *2* ∗ *pi*) = *cis x* ⟨*proof*⟩
lemma *cis-plus-2pi-neq-1*: **assumes** *x*: *0* < *x x* < *2* ∗ *pi*
shows *cis x* ≠ *1*
⟨*proof*⟩

lemma *cis-times-2pi*[*simp*]: *cis* (*of-nat n* ∗ *2* ∗ *pi*) = *1*
⟨*proof*⟩

lemma *cis-add-pi*[*simp*]: *cis* (*pi* + *x*) = − *cis x*
⟨*proof*⟩

lemma *cis-3-pi-2*[*simp*]: *cis* (*pi* ∗ *3* / *2*) = − i
⟨*proof*⟩

lemma *rcis-plus-2pi*[*simp*]: *rcis y* (*x* + *2* ∗ *pi*) = *rcis y x* ⟨*proof*⟩
lemma *rcis-times-2pi*[*simp*]: *rcis r* (*of-nat n* ∗ *2* ∗ *pi*) = *of-real r*
⟨*proof*⟩

lemma *arg-rcis-cis*: **assumes** *n*: *n* > *0* **shows** *Arg* (*rcis n x*) = *Arg* (*cis x*)
⟨*proof*⟩

lemma *arg-eqD*: **assumes** *Arg* (*cis x*) = *Arg* (*cis y*) −*pi* < *x x* ≤ *pi* −*pi* < *y y* ≤ *pi*
shows *x* = *y*
⟨*proof*⟩

lemma *rcis-inj-on*: **assumes** *r*: *r* ≠ *0* **shows** *inj-on* (*rcis r*) {*0* ..< *2* ∗ *pi*}
⟨*proof*⟩

lemma *cis-inj-on*: *inj-on cis* {*0* ..< *2* ∗ *pi*}
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *degree-root-unity*[*simp*]: *degree* (*root-unity n*) = *n* (**is** *degree ?p* = -)
⟨*proof*⟩

lemma *zero-root-unit*[*simp*]: *root-unity n* = *0* ⟷ *n* = *0* (**is** *?p* = *0* ⟷ -)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *degree-prod-root-unity*[*simp*]: *0* ∉ *set ns* ⟹ *degree* (*prod-root-unity ns*) = *sum-list ns*
⟨*proof*⟩

lemma *zero-prod-root-unit*[*simp*]: *prod-root-unity ns* = *0* ⟷ *0* ∈ *set ns*

⟨*proof*⟩

lemma *roots-of-unity*: **assumes** *n*: $n \neq 0$
shows $(\lambda\ i.\ (cis\ (of\text{-}nat\ i * 2 * pi\ /\ n)))$ ‘ $\{0\ ..<\ n\} = \{\ x :: complex.\ x\ \hat{}\ n = 1\}$ (**is** *?prod = ?Roots*)
$\{x.\ poly\ (root\text{-}unity\ n)\ x = 0\} = \{\ x :: complex.\ x\ \hat{}\ n = 1\}$
$card\ \{\ x :: complex.\ x\ \hat{}\ n = 1\} = n$
⟨*proof*⟩

lemma *poly-roots-dvd*: **fixes** $p :: 'a :: field\ poly$
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*: $n \neq 0$
shows *root-unity n* =
prod-list $(map\ (\lambda\ i.\ [:-cis\ (of\text{-}nat\ i * 2 * pi\ /\ n),\ 1:])\ [0\ ..<\ n])$ (**is** *?u = ?p*)
⟨*proof*⟩

lemma *order-monic-linear*: $order\ x\ [:y,1:] = (if\ y + x = 0\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *order-root-unity*: **fixes** *x* :: *complex* **assumes** *n*: $n \neq 0$
shows $order\ x\ (root\text{-}unity\ n) = (if\ x\hat{}n = 1\ then\ 1\ else\ 0)$
(**is** *order - ?u = -*)
⟨*proof*⟩

lemma *order-prod-root-unity*: **assumes** *0*: $0 \notin set\ ks$
shows $order\ (x :: complex)\ (prod\text{-}root\text{-}unity\ ks) = length\ (filter\ (\lambda\ k.\ x\hat{}k = 1)\ ks)$
⟨*proof*⟩

lemma *root-unity-witness*: **fixes** *xs* :: *complex list*
assumes $prod\text{-}list\ (map\ (\lambda\ x.\ [:-x,1:])\ xs) = monom\ 1\ n - 1$
shows $x\hat{}n = 1 \longleftrightarrow x \in set\ xs$
⟨*proof*⟩

lemma *root-unity-explicit*: **fixes** *x* :: *complex*
shows
$(x\ \hat{}\ 1 = 1) \longleftrightarrow x = 1$
$(x\ \hat{}\ 2 = 1) \longleftrightarrow (x \in \{1,\ -1\})$
$(x\ \hat{}\ 3 = 1) \longleftrightarrow (x \in \{1,\ Complex\ (-1/2)\ (sqrt\ 3\ /\ 2),\ Complex\ (-1/2)\ (-sqrt\ 3\ /\ 2)\})$
$(x\ \hat{}\ 4 = 1) \longleftrightarrow (x \in \{1,\ -1,\ \mathrm{i},\ -\mathrm{i}\})$
⟨*proof*⟩

definition *primitive-root-unity* :: $nat \Rightarrow 'a :: power \Rightarrow bool$ **where**
$primitive\text{-}root\text{-}unity\ k\ x = (k \neq 0 \land x\hat{}k = 1 \land (\forall\ k' < k.\ k' \neq 0 \longrightarrow x\hat{}k' \neq$

1))

lemma *primitive-root-unityD*: **assumes** *primitive-root-unity k x*
shows $k \neq 0$ $x\hat{\ }k = 1$ $k' \neq 0 \Longrightarrow x\hat{\ }k' = 1 \Longrightarrow k \leq k'$
⟨*proof*⟩

lemma *primitive-root-unity-exists*: **assumes** $k \neq 0$ $x \hat{\ } k = 1$
shows $\exists\ k'.\ k' \leq k \wedge$ *primitive-root-unity* k' x
⟨*proof*⟩

lemma *primitive-root-unity-dvd*: **fixes** *x* :: *complex*
assumes *k*: *primitive-root-unity k x*
shows $x \hat{\ } n = 1 \longleftrightarrow k$ *dvd* n
⟨*proof*⟩

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

lemma *primitive-root-unity-explicit*: **fixes** *x* :: *complex*
shows *primitive-root-unity 1 x* $\longleftrightarrow x = 1$
primitive-root-unity 2 x $\longleftrightarrow x = -1$
primitive-root-unity 3 x $\longleftrightarrow$ ($x \in$ {*Complex* $(-1/2)$ (*sqrt* 3 / 2), *Complex* $(-1/2)$ ($-$ *sqrt* 3 / 2)})
primitive-root-unity 4 x $\longleftrightarrow$ ($x \in$ {i, $-$ i})
⟨*proof*⟩

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

termination ⟨*proof*⟩

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*: $\bigwedge$ *x*. *cmod x* = *1* ⟹ *poly f x* ≠ *0*
and *k*: $\bigwedge$ *k'*. $k' > k$ ⟹ ¬ *root-unity k' dvd p*
shows *p* = *prod-root-unity ks'* ∗ *f* ∧ *f* = *g* ∧ *set ks* = *set ks'*
⟨*proof*⟩

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*⟩

lemma (**in** *comm-ring-hom*) *hom-root-unity*: *map-poly hom* (*root-unity n*) = *root-unity n*
⟨*proof*⟩

lemma (**in** *idom-hom*) *hom-prod-root-unity*: *map-poly hom* (*prod-root-unity n*) = *prod-root-unity n*
⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

end

5.1 The Perron Frobenius Theorem for Irreducible Matrices

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

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

lemma *charpoly-of-real*: *charpoly* (*map-matrix complex-of-real A*) = *map-poly of-real* (*charpoly A*)
  ⟨*proof*⟩

context includes *lifting-syntax*
begin
lemma *HMA-M-smult*[*transfer-rule*]: ((=) ===> *HMA-M* ===> *HMA-M*) ($\cdot_m$) ((*$*k$*))
⟨*proof*⟩
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*)
⟨*proof*⟩

lemma *smult-eigen-vector*: **fixes** *a* :: *'a* :: *field*
assumes *eigen-vector A v x*
shows *eigen-vector* (*a* ∗*k A*) *v* (*a* ∗ *x*)
⟨*proof*⟩

lemma *smult-eigen-value*: **fixes** *a* :: *'a* :: *field*
assumes *eigen-value A x*
shows *eigen-value* (*a* ∗*k A*) (*a* ∗ *x*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *G-transpose-trancl*:
(*fixed-mat.G* (*transpose A*))^+ = ((*fixed-mat.G A*)^+)^−*1*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *nonneg-matpow*: *matpow A n* $ *i* $ *j* ≥ *0*

⟨*proof*⟩

lemma *G-relpow-matpow-pos*: (*i*,*j*) ∈ *G* ^^ *n* ⟹ *matpow A n* $ *i* $ *j* > *0*
⟨*proof*⟩

lemma *matpow-mono*: **assumes** *B*: ⋀ *i j*. *B* $ *i* $ *j* ≥ *A* $ *i* $ *j*
shows *matpow B n* $ *i* $ *j* ≥ *matpow A n* $ *i* $ *j*
⟨*proof*⟩

lemma *matpow-sum-one-mono*: *matpow* (*A* + *mat 1*) (*n* + *k*) $ *i* $ *j* ≥ *matpow* (*A* + *mat 1*) *n* $ *i* $ *j*
⟨*proof*⟩

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

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

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

lemma *N*: ∃ *n* ≤ *N*. *ij* ∈ *G* ^^ *n*
⟨*proof*⟩

lemma *irreducible-matpow-pos*: **assumes** *irreducible*
shows *matpow* (*A* + *mat 1*) *N* $ *i* $ *j* > *0*
⟨*proof*⟩

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

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

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

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

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

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

definition X :: $(real$ ^ $'n\ )set$ **where**
  $X = \{\ x\ .\ le\text{-}vec\ 0\ x \wedge x \neq 0\ \}$

lemma *nonneg-Ax*: $x \in X \Longrightarrow le\text{-}vec\ 0\ (A *v\ x)$
  ⟨*proof*⟩

lemma *A-nonzero-fixed-i*: $\exists\ j.\ A\ \$\ i\ \$\ j \neq 0$
⟨*proof*⟩

lemma *A-nonzero-fixed-j*: $\exists\ i.\ A\ \$\ i\ \$\ j \neq 0$
⟨*proof*⟩

lemma *Ax-pos*: **assumes** x: *lt-vec 0 x*
  shows *lt-vec 0* $(A *v\ x)$
⟨*proof*⟩

lemma *nonzero-Ax*: **assumes** x: $x \in X$
  shows $A *v\ x \neq 0$
⟨*proof*⟩

lemma *r-witness*: **assumes** x: $x \in X$
  shows $\exists\ j.\ x\ \$\ j > 0 \wedge r\ x = (A *v\ x)\ \$\ j\ /\ x\ \$\ j$
⟨*proof*⟩

lemma *rx-nonneg*: **assumes** x: $x \in X$
  shows $r\ x \geq 0$
⟨*proof*⟩

lemma *rx-pos*: **assumes** x: *lt-vec 0 x*
  shows $r\ x > 0$
⟨*proof*⟩

lemma *rx-le-Ax*: **assumes** x: $x \in X$
  shows *le-vec* $(r\ x *s\ x)\ (A *v\ x)$
⟨*proof*⟩

lemma *rho-le-x-Ax-imp-rho-le-rx*: **assumes** x: $x \in X$
  and ϱ: *le-vec* $(\varrho *s\ x)\ (A *v\ x)$
shows $\varrho \leq r\ x$
⟨*proof*⟩

lemma *rx-Max*: **assumes** x: $x \in X$
  shows $r\ x = Sup\ \{\ \varrho\ .\ le\text{-}vec\ (\varrho *s\ x)\ (A *v\ x)\ \}$ (**is** - = *Sup ?S*)
⟨*proof*⟩

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

⟨*proof*⟩

definition *X1* = (*X* ∩ {*x*. *norm x* = *1*})

lemma *bounded-X1*: *bounded X1* ⟨*proof*⟩

lemma *closed-X1*: *closed X1*
⟨*proof*⟩

lemma *compact-X1*: *compact X1* ⟨*proof*⟩

definition *pow-A-1 x* = *A1n* ∗*v x*

lemma *continuous-pow-A-1*: *continuous-on R pow-A-1*
⟨*proof*⟩

definition *Y* = *pow-A-1* ‘ *X1*

lemma *compact-Y*: *compact Y*
⟨*proof*⟩

lemma *Y-pos-main*: **assumes** *y*: *y* ∈ *pow-A-1* ‘ *X*
shows *y* $ *i* > *0*
⟨*proof*⟩

lemma *Y-pos*: **assumes** *y*: *y* ∈ *Y*
shows *y* $ *i* > *0*
⟨*proof*⟩

lemma *Y-nonzero*: **assumes** *y*: *y* ∈ *Y*
shows *y* $ *i* ≠ *0*
⟨*proof*⟩

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

lemma *r′-r*: **assumes** *x*: *x* ∈ *Y* **shows** *r′ x* = *r x*
⟨*proof*⟩

lemma *continuous-Y-r*: *continuous-on Y r*
⟨*proof*⟩

lemma *X1-nonempty*: *X1* ≠ {}
⟨*proof*⟩

lemma *Y-nonempty*: *Y* ≠ {}
⟨*proof*⟩

definition *z* **where** $z = (SOME\ z.\ z \in Y \wedge (\forall\ y \in Y.\ r\ y \leq r\ z))$

abbreviation $sr \equiv r\ z$

lemma *z*: $z \in Y$ **and** *sr-max-Y*: $\bigwedge y.\ y \in Y \Longrightarrow r\ y \leq sr$
⟨*proof*⟩

lemma *Y-subset-X*: $Y \subseteq X$
⟨*proof*⟩

lemma *zX*: $z \in X$
⟨*proof*⟩

lemma *le-vec-mono-left*: **assumes** *B*: $\bigwedge i\ j.\ B\ \$\ i\ \$\ j \geq 0$
and *le-vec x y*
shows *le-vec* $(B *v\ x)$ $(B *v\ y)$
⟨*proof*⟩

lemma *matpow-1-commute*: $matpow\ (A + mat\ 1)\ n ** A = A ** matpow\ (A + mat\ 1)\ n$
⟨*proof*⟩

lemma *A1n-commute*: $A1n ** A = A ** A1n$
⟨*proof*⟩

lemma *le-vec-pow-A-1*: **assumes** *le*: *le-vec* $(rho *s\ x)$ $(A *v\ x)$
shows *le-vec* $(rho *s\ pow\text{-}A\text{-}1\ x)$ $(A *v\ pow\text{-}A\text{-}1\ x)$
⟨*proof*⟩

lemma *r-pow-A-1*: **assumes** *x*: $x \in X$
shows $r\ x \leq r\ (pow\text{-}A\text{-}1\ x)$
⟨*proof*⟩

lemma *sr-max*: **assumes** *x*: $x \in X$
shows $r\ x \leq sr$
⟨*proof*⟩

lemma *z-pos*: $z\ \$\ i > 0$
⟨*proof*⟩

lemma *sr-pos*: $sr > 0$
⟨*proof*⟩

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

lemma *sr-imp-eigen-vector-main*: *sr* ∗*s u* = *A* ∗*v u*
⟨*proof*⟩

lemma *sr-imp-eigen-vector*: *eigen-vector A u sr*
⟨*proof*⟩

lemma *sr-u-pos*: *lt-vec 0 u*
⟨*proof*⟩
end

lemma *eigen-vector-z-sr*: *eigen-vector A z sr*
⟨*proof*⟩

lemma *eigen-value-sr*: *eigen-value A sr*
⟨*proof*⟩

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*) ⟨*proof*⟩
lemma *norm-v-eq-0*: *norm-v v* = *0* ⟷ *v* = *0* ⟨*proof*⟩

lemma *cA-index*: *cA* $ *i* $ *j* = *c* (*A* $ *i* $ *j*)
⟨*proof*⟩

lemma *norm-cA*[*simp*]: *norm* (*cA* $ *i* $ *j*) = *A* $ *i* $ *j*
⟨*proof*⟩

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

lemma *evD*: *α* ∗*s v* = *cA* ∗*v v v* ≠ *0*
⟨*proof*⟩

lemma *ev-alpha-norm-v*: *norm-v* (*α* ∗*s v*) = (*norm α* ∗*s norm-v v*)
⟨*proof*⟩

lemma *ev-A-norm-v*: *norm-v* (*cA* ∗*v v*) $ *j* ≤ (*A* ∗*v norm-v v*) $ *j*
⟨*proof*⟩

lemma *ev-le-vec*: *le-vec* (*norm α* ∗*s norm-v v*) (*A* ∗*v norm-v v*)
⟨*proof*⟩

lemma *norm-v-X*: *norm-v v* ∈ *X*
⟨*proof*⟩

lemma *ev-inequalities*: *norm α* ≤ *r* (*norm-v v*) *r* (*norm-v v*) ≤ *sr*

⟨*proof*⟩

lemma *eigen-vector-norm-sr*: *norm* $\alpha \leq$ *sr* ⟨*proof*⟩
end

lemma *eigen-value-norm-sr*: **assumes** *eigen-value cA* α
shows *norm* $\alpha \leq$ *sr*
⟨*proof*⟩

lemma *le-vec-trans*: *le-vec x y* $\Longrightarrow$ *le-vec y u* $\Longrightarrow$ *le-vec x u*
⟨*proof*⟩

lemma *eigen-vector-z-sr-c*: *eigen-vector cA* (*map-vector c z*) (*c sr*)
⟨*proof*⟩

lemma *eigen-value-sr-c*: *eigen-value cA* (*c sr*)
⟨*proof*⟩

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*⟩

lemma *c-cmod-id*: $a \in \mathbb{R} \Longrightarrow$ *Re a* $\geq 0 \Longrightarrow$ *c* (*cmod a*) = *a* ⟨*proof*⟩

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*⟩

lemma *pos-matrix-mult-0*: **assumes** *le*: $\bigwedge$ *i j*. *B* \$ *i* \$ *j* $\geq$ *0*
and *lt*: *lt-vec 0 x*
and *0*: *B* *$*v$ *x* = *0*
shows *B* = *0*
⟨*proof*⟩

lemma *eigen-value-smaller-matrix*: **assumes** *B*: $\bigwedge$ *i j*. *0* $\leq$ *B* \$ *i* \$ *j* $\wedge$ *B* \$ *i* \$ *j* $\leq$ *A* \$ *i* \$ *j*
and *AB*: $A \neq B$
and *ev*: *eigen-value* (*map-matrix c B*) *sigma*
shows *cmod sigma* < *sr*
⟨*proof*⟩

lemma *charpoly-erase-mat-sr*: *0* < *poly* (*charpoly* (*erase-mat A i i*)) *sr*
⟨*proof*⟩

lemma *multiplicity-sr-1*: *order sr* (*charpoly A*) = *1*
⟨*proof*⟩

lemma *sr-spectral-radius*: *sr* = *spectral-radius cA*
⟨*proof*⟩

lemma *le-vec-A-mu*: **assumes** *y*: *y* ∈ *X* **and** *le*: *le-vec* (*A* ∗*v y*) (*mu* ∗*s y*)
shows *sr* ≤ *mu lt-vec 0 y*
mu = *sr* ∨ *A* ∗*v y* = *mu* ∗*s y* ⟹ *mu* = *sr* ∧ *A* ∗*v y* = *mu* ∗*s y*
⟨*proof*⟩

lemma *nonnegative-eigenvector-has-ev-sr*: **assumes** *eigen-vector A v mu* **and** *le*: *le-vec 0 v*
shows *mu* = *sr*
⟨*proof*⟩

lemma *similar-matrix-rotation*: **assumes** *ev*: *eigen-value cA* α **and** α: *cmod* α = *sr*
shows *similar-matrix* (*cis* (*Arg* α) ∗*k cA*) *cA*
⟨*proof*⟩

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*⟩

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*⟩

lemma *maximal-eigen-value-roots-of-unity-rotation*:
assumes *M*: *M* = {*ev* :: *complex*. *eigen-value cA ev* ∧ *cmod ev* = *sr*}
and *kM*: *k* = *card M*
shows *k* ≠ *0*
k ≤ *CARD*(*'n*)
∃ *f*. *charpoly A* = (*monom 1 k* − [:*sr*^*k*:]) ∗ *f*
∧ (∀ *x*. *poly* (*map-poly c f*) *x* = *0* ⟶ *cmod x* < *sr*)
M = (∗) (*c sr*) ‘ (λ *i*. (*cis* (*of-nat i* ∗ *2* ∗ *pi* / *k*))) ‘ {*0* ..< *k*}
M = (∗) (*c sr*) ‘ { *x* :: *complex*. *x* ^ *k* = *1*}
(∗) (*cis* (*2* ∗ *pi* / *k*)) ‘ *Spectrum cA* = *Spectrum cA*
⟨*proof*⟩

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

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

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

end

5.2 Handling Non-Irreducible Matrices as Well

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

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

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

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

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

Next, we have to install transfer rules

context
includes *lifting-syntax*
begin
lemma *HMA-irreducible*[*transfer-rule*]: ((*HMA-M* :: - ⇒ - ^ ′*n* ^ ′*n* ⇒ -) ===> (=))
irreducible-mat fixed-mat.irreducible
⟨*proof*⟩

lemma *HMA-nonneg-irreducible-mat*[*transfer-rule*]: (*HMA-M* ===> (=)) *nonneg-irreducible-mat perron-frobenius*
⟨*proof*⟩
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*: $A \in$ *carrier-mat n n* **and** *n*: $n \neq 0$ **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 $\alpha \Longrightarrow$ *cmod* $\alpha \leq$ *sr*
eigenvalue cA $\alpha \Longrightarrow$ *cmod* α = *sr* $\Longrightarrow$ *order* α (*char-poly cA*) = *1*
$\exists$ *k f*. $k \neq 0 \wedge k \leq n \wedge$ *char-poly A* = (*monom 1 k* − [:*sr* ^ *k*:]) ∗ *f* ∧
($\forall$ *x*. *poly* (*map-poly complex-of-real f*) *x* = *0* ⟶ *cmod x* < *sr*)
⟨*proof*⟩

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

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

unbundle *no m-inv-syntax*

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*
⟨*proof*⟩

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

lemma *permutation-mat-col*[*simp*]: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹
Matrix.col (*permutation-mat n p*) *i* = *unit-vec n* (*p i*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *permutes-lt*: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹ *p i* < *n*
⟨*proof*⟩

lemma *permutes-iff*: *p permutes* {..<*n*} ⟹ *i* < *n* ⟹ *j* < *n* ⟹ *p i* = *p j* ⟷ *i* = *j*
⟨*proof*⟩

lemma *permutation-mat-id-1*: **assumes** *p*: *p permutes* {..<*n*}
shows *permutation-mat n p* ∗ *permutation-mat n* (*inv p*) = *1*$_m$ *n*
⟨*proof*⟩

lemma *permutation-mat-id-2*: **assumes** *p*: *p permutes* {..<*n*}
shows *permutation-mat n* (*inv p*) ∗ *permutation-mat n p* = *1*$_m$ *n*
⟨*proof*⟩

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*⟩

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*)))
⟨*proof*⟩

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*⟩

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*⟩

lemma *char-poly-four-block-mat-lower-left-zero*: **fixes** *A* :: *'a* :: *idom mat*
assumes *A*: *A* = *four-block-mat B C* (*0*$_m$ *m n*) *D*
and *B*: *B* ∈ *carrier-mat n n*
and *C*: *C* ∈ *carrier-mat n m*
and *D*: *D* ∈ *carrier-mat m m*
shows *char-poly A* = *char-poly B* ∗ *char-poly D*
⟨*proof*⟩

lemma *elements-mat-permutes*: **assumes** *p*: *p permutes* {..< *n*}
  and *A*: $A \in$ *carrier-mat n n*
  and *B*: *B* = *Matrix.mat n n* (λ (*i*,*j*). *A* $$ (*p i*, *p j*))
shows *elements-mat A* = *elements-mat B*
⟨*proof*⟩

lemma *elements-mat-four-block-mat-supseteq*:
  assumes *A1*: *A1* ∈ *carrier-mat n n*
  and *A2*: *A2* ∈ *carrier-mat n m*
  and *A3*: *A3* ∈ *carrier-mat m n*
  and *A4*: *A4* ∈ *carrier-mat m m*
shows *elements-mat* (*four-block-mat A1 A2 A3 A4*) ⊇
  (*elements-mat A1* ∪ *elements-mat A2* ∪ *elements-mat A3* ∪ *elements-mat A4*)
⟨*proof*⟩

lemma *non-irreducible-mat-split*:
  fixes *A* :: *'a* :: *idom mat*
  assumes *A*: *A* ∈ *carrier-mat n n*
  and *not*: ¬ *irreducible-mat A*
  and *n*: *n* > *1*
shows ∃ *n1 n2 B B1 B2 B4*. *similar-mat A B* ∧ *elements-mat A* = *elements-mat B* ∧
  B = *four-block-mat B1 B2* (0_m *n2 n1*) *B4* ∧
  B1 ∈ *carrier-mat n1 n1* ∧ *B2* ∈ *carrier-mat n1 n2* ∧ *B4* ∈ *carrier-mat n2 n2* ∧
  0 < *n1* ∧ *n1* < *n* ∧ *0* < *n2* ∧ *n2* < *n* ∧ *n1* + *n2* = *n*
⟨*proof*⟩

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

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*⟩

And back to HMA world via transfer.

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

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

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

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

defines *cA* ≡ *map-matrix complex-of-real A*
defines *cf* ≡ *map-poly complex-of-real f*
assumes *pos*: *non-neg-mat A*
and *sr*: ⋀ *x*. *poly* (*charpoly A*) *x* = *0* ⟹ *x* ≤ *1*
and *decomp*: *decompose-prod-root-unity* (*charpoly A*) = (*ks*, *f*)
shows *0* ∉ *set ks*
charpoly A = *prod-root-unity ks* ∗ *f*
charpoly cA = *prod-root-unity ks* ∗ *cf*
⋀ *x*. *poly* (*charpoly cA*) *x* = *0* ⟹ *cmod x* ≤ *1*
⋀ *x*. *poly cf x* = *0* ⟹ *cmod x* < *1*
⋀ *x*. *cmod x* = *1* ⟹ *order x* (*charpoly cA*) = *length* [*k*←*ks* . *x* ^ *k* = *1*]
⋀ *x*. *cmod x* = *1* ⟹ *poly* (*charpoly cA*) *x* = *0* ⟹ ∃ *k* ∈ *set ks*. *x*^*k* = *1*
⟨*proof*⟩

and convert to JNF-world

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

end

6 Combining Spectral Radius Theory with Perron Frobenius theorem

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

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

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

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

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

lemma *perron-frobenius-spectral-radius-complex*: **fixes** *A* :: *complex mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *real-nonneg*: *real-nonneg-mat A*
and *ev-le-1*: ⋀ *x*. *poly* (*char-poly* (*map-mat Re A*)) *x* = *0* ⟹ *x* ≤ *1*
and *ev-order*: ⋀ *x*. *norm x* = *1* ⟹ *order x* (*char-poly A*) ≤ *d*
shows ∃ *c1 c2*. ∀ *k*. *norm-bound* (*A* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
⟨*proof*⟩

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

lemma *perron-frobenius-spectral-radius*: **fixes** *A* :: *real mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *ev-le-1*: ∀ *x*. *poly* (*char-poly A*) *x* = *0* ⟶ *x* ≤ *1*
and *ev-order*: ∀ *x* :: *complex*. *norm x* = *1* ⟶ *order x* (*map-poly of-real* (*char-poly A*)) ≤ *d*
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (*A* ^*m* *k*) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
⟨*proof*⟩

We can also convert the set-based lemma ⟦*?A* ∈ *carrier-mat ?n ?n*; *nonneg-mat ?A*; ∀ *x*. *poly* (*char-poly ?A*) *x* = *0* ⟶ *x* ≤ *1*; ∀ *x*. *cmod x* = *1* ⟶ *order x* (*map-poly complex-of-real* (*char-poly ?A*)) ≤ *?d*⟧ ⟹ ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (*?A* ^*m* *k*) ⟶ $|a|$ ≤ *c1* + *c2* ∗ $(real\ k)^{?d\ -\ 1}$ to a type-based version.

lemma *perron-frobenius-spectral-type-based*:
assumes *non-neg-mat* (*A* :: *real* ^ *'n* ^ *'n*)
and ∀ *x*. *poly* (*charpoly A*) *x* = *0* ⟶ *x* ≤ *1*
and ∀ *x* :: *complex*. *norm x* = *1* ⟶ *order x* (*map-poly of-real* (*charpoly A*)) ≤ *d*
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat-h* (*matpow A k*) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
⟨*proof*⟩

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* ∈ *carrier-mat n n*
and *nonneg-mat A*
and ∀ *x*. *poly* (*char-poly A*) *x* = *0* ⟶ *x* ≤ *1*
and ∀ *x* :: *complex*. *norm x* = *1* ⟶ *order x* (*map-poly of-real* (*char-poly A*)) ≤ *d*
and *n* ≠ *0*
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (*A* ^*m* *k*) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *real k* ^ (*d* − *1*))
⟨*proof*⟩

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

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

begin

lemma *perron-frobenius-spectral-radius-yun*:
  assumes *bnd*: $\bigwedge f_i\ i.\ (f_i,i) \in set\ fis$
    $\Longrightarrow (\exists\ x :: complex.\ poly\ (map\text{-}poly\ of\text{-}real\ f_i)\ x = 0 \wedge norm\ x = 1)$
    $\Longrightarrow i \leq d$
  shows $\exists\, c1\ c2.\ \forall\, k\ a.\ a \in elements\text{-}mat\ (A\ \widehat{\ }_m\ k) \longrightarrow abs\ a \leq (c1 + c2 * real\ k\ \widehat{\ }(d - 1))$
⟨*proof*⟩

Note that the only remaining problem in applying $(\bigwedge f_i\ i.\ [\![(f_i,\ i) \in set\ fis;\ \exists\, x.\ poly\ (map\text{-}poly\ complex\text{-}of\text{-}real\ f_i)\ x = 0 \wedge cmod\ x = 1]\!] \Longrightarrow i \leq ?d) \Longrightarrow \exists\, c1\ c2.\ \forall\, k\ a.\ a \in elements\text{-}mat\ (A\ \widehat{\ }_m\ k) \longrightarrow |a| \leq c1 + c2 * (real\ k)^{?d - 1}$ is to check the condition $\exists\, x.\ poly\ (map\text{-}poly\ complex\text{-}of\text{-}real\ f_i)\ x = 0 \wedge cmod\ x = 1$. Here, there are at least three possibilities. First, one can just ignore this precondition and weaken the statement. Second, one can apply Sturm's theorem to determine whether all roots are real. This can be done by comparing the number of distinct real roots with the degree of f_i, since f_i is square-free. If all roots are real, then one can decide the criterion by checking the only two possible real roots with norm equal to 1, namely 1 and -1. If on the other hand there are complex roots, then we loose precision at this point. Third, one uses a factorization algorithm (e.g., via complex algebraic numbers) to precisely determine the complex roots and decide the condition.

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

lemma *perron-frobenius-spectral-radius-yun-real-roots*:
  assumes *bnd*: $\bigwedge f_i\ i.\ (f_i,i) \in set\ fis$
    $\Longrightarrow card\ \{\ x.\ poly\ f_i\ x = 0\} \neq degree\ f_i \vee poly\ f_i\ 1 = 0 \vee poly\ f_i\ (-1) = 0$
    $\Longrightarrow i \leq d$
  shows $\exists\, c1\ c2.\ \forall\, k\ a.\ a \in elements\text{-}mat\ (A\ \widehat{\ }_m\ k) \longrightarrow abs\ a \leq (c1 + c2 * real\ k\ \widehat{\ }(d - 1))$
⟨*proof*⟩

end

end

7 The Jordan Blocks of the Spectral Radius are Largest

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

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

lemma *poly-asymp-equiv*: $(\lambda x.\ poly\ p\ (real\ x)) \sim[at\text{-}top]\ (\lambda x.\ lead\text{-}coeff\ p * real\ x\ \hat{}\ (degree\ p))$
⟨*proof*⟩

lemma *sum-root-unity*: **fixes** $x :: 'a :: \{comm\text{-}ring, division\text{-}ring\}$
  assumes $x\hat{}n = 1$
  shows $sum\ (\lambda\ i.\ x\hat{}i)\ \{..< n\} = (if\ x = 1\ then\ of\text{-}nat\ n\ else\ 0)$
⟨*proof*⟩

lemma *sum-root-unity-power-pos-implies-1*:
  assumes *sumpos*: $\bigwedge\ k.\ Re\ (sum\ (\lambda\ i.\ b\ i * x\ i\ \hat{}\ k)\ I) > 0$
  and *root-unity*: $\bigwedge\ i.\ i \in I \Longrightarrow \exists\ d.\ d \neq 0 \wedge x\ i\ \hat{}\ d = 1$
shows $1 \in x\ \text{`}\ I$
⟨*proof*⟩

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

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

lemma *j-to-jb-index*: **assumes** $i < sum\text{-}list\ (map\ fst\ n\text{-}as)$
  and $j < sum\text{-}list\ (map\ fst\ n\text{-}as)$
  and $j\text{-}to\text{-}jb\text{-}index\ n\text{-}as\ i = (bi,\ li)$
  and $j\text{-}to\text{-}jb\text{-}index\ n\text{-}as\ j = (bj,\ lj)$
  and $n\text{-}as\ !\ bj = (n,\ a)$
shows $((jordan\text{-}matrix\ n\text{-}as)\ \hat{}_m\ r)\ \$\$\ (i,j) = (if\ bi = bj\ then\ ((jordan\text{-}block\ n\ a)\ \hat{}_m\ r)\ \$\$\ (li,\ lj)\ else\ 0)$
  $\wedge\ (bi = bj \longrightarrow li < n \wedge lj < n \wedge bj < length\ n\text{-}as \wedge (n,a) \in set\ n\text{-}as)$
  ⟨*proof*⟩

lemma *j-to-jb-index-rev*: **assumes** *j*: $j\text{-}to\text{-}jb\text{-}index\ n\text{-}as\ i = (bi,\ li)$
  and *i*: $i < sum\text{-}list\ (map\ fst\ n\text{-}as)$

and *k*: $k \leq li$
shows $li \leq i \wedge$ *j-to-jb-index n-as* $(i - k) = (bi,\ li - k) \wedge ($
j-to-jb-index n-as $j = (bi,li - k) \longrightarrow j <$ *sum-list* (*map fst n-as*) $\longrightarrow j = i - k)$
⟨*proof*⟩

locale *spectral-radius-1-jnf-max* =
fixes *A* :: *real mat* **and** *n m* :: *nat* **and** *lam* :: *complex* **and** *n-as*
assumes *A*: $A \in$ *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* ⟹ $x \leq 1$
and *max-block*: ⋀ *k la*. (*k*,*la*) ∈ *set n-as* ⟹ *cmod la* ≤ *1* ∧ (*cmod la* = *1* ⟶ $k \leq m$)
begin

lemma *n-as0*: *0* ∉ *fst* ‘ *set n-as*
⟨*proof*⟩

lemma *m0*: $m \neq 0$ ⟨*proof*⟩

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*
⟨*proof*⟩

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

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

lemma *p-binom*:
assumes $s \leq k$
shows *of-nat* (*k choose s*) = *poly* (*p s*) (*of-nat k*)
⟨*proof*⟩

lemma *p-binom-complex*: **assumes** *sk*: $s \leq k$
shows *of-nat* (*k choose s*) = *complex-of-real* (*poly* (*p s*) (*of-nat k*))
⟨*proof*⟩

lemma *deg-p*: *degree* (*p s*) = *s* ⟨*proof*⟩

lemma *lead-coeff-p*: *lead-coeff* (*p s*) = (∏ *i* = *0*..<*s*. *1* / (*of-nat s* − *of-nat i*))
⟨*proof*⟩

lemma *lead-coeff-p-gt-0*: *lead-coeff* (*p s*) > *0* ⟨*proof*⟩

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

lemma *c-gt-0*: *c* > *0* ⟨*proof*⟩
lemma *c0*: *c* ≠ *0* ⟨*proof*⟩

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

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

lemma *JNF*: *P* ∈ *carrier-mat n n iP* ∈ *carrier-mat n n J* ∈ *carrier-mat n n*
P ∗ *iP* = 1_m *n iP* ∗ *P* = 1_m *n cA* = *P* ∗ *J* ∗ *iP*
⟨*proof*⟩

definition *C* :: *nat set* **where**
C = {*j* | *j bj lj nn la*. *j* < *n* ∧ *j-to-jb-index n-as j* = (*bj*, *lj*)
∧ *n-as* ! *bj* = (*nn*,*la*) ∧ *cmod la* = *1* ∧ *nn* = *m* ∧ *lj* = *nn* − *1*}

lemma *C-nonempty*: *C* ≠ {}
⟨*proof*⟩

lemma *C-n*: *C* ⊆ {..<*n*} ⟨*proof*⟩

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*⟩

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

d la ≠ *0* ∧ *la* ^(*d la*) = *1*)

lemma *d*: **assumes** (*k*,*la*) ∈ *set n-as cmod la* = *1*
shows *la* ^(*d la*) = *1* ∧ *d la* ≠ *0*
⟨*proof*⟩

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

lemma *D0*: *D* ≠ *0* ⟨*proof*⟩

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

lemma *mono-f*: *strict-mono* (*f off*) ⟨*proof*⟩

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

lemma *limit-jordan-block*: **assumes** *kla*: (*k*, *la*) ∈ *set n-as*

and *ij*: $i < k$ $j < k$
shows $(\lambda N.\ (\textit{jordan-block}\ k\ la\ \widehat{\ }_m\ (f\ \textit{off}\ N))\ \$\$\ (i,\ j) * \textit{inv-op off}\ N)$
$\longrightarrow (\textit{if}\ i = 0 \wedge j = k - 1 \wedge \textit{cmod}\ la = 1 \wedge k = m\ \textit{then}\ la\widehat{\ }\textit{off else}\ 0)$
⟨*proof*⟩

definition *lambda* **where** $\textit{lambda}\ i = \textit{snd}\ (\textit{n-as}\ !\ \textit{fst}\ (\textit{j-to-jb-index}\ \textit{n-as}\ i))$

lemma *cmod-lambda*: $i \in C \Longrightarrow \textit{cmod}\ (\textit{lambda}\ i) = 1$
⟨*proof*⟩

lemma *R-lambda*: **assumes** *i*: $i \in C$
shows $(m,\ \textit{lambda}\ i) \in \textit{set n-as}$
⟨*proof*⟩

lemma *limit-jordan-matrix*: **assumes** *ij*: $i < n$ $j < n$
shows $(\lambda N.\ (J\ \widehat{\ }_m\ (f\ \textit{off}\ N))\ \$\$\ (i,\ j) * \textit{inv-op off}\ N)$
$\longrightarrow (\textit{if}\ j \in C \wedge i = j - (m - 1)\ \textit{then}\ (\textit{lambda}\ j)\widehat{\ }\textit{off else}\ 0)$
⟨*proof*⟩

declare *sumlist-nf*[*simp*]

lemma *A-power-P*: $cA\ \widehat{\ }_m\ k * P = P * J\ \widehat{\ }_m\ k$
⟨*proof*⟩

lemma *inv-op-nonneg*: $\textit{inv-op off}\ k \geq 0$ ⟨*proof*⟩

lemma *P-nonzero-entry*: **assumes** *j*: $j < n$
shows $\exists\ i < n.\ P\ \$\$\ (i,j) \neq 0$
⟨*proof*⟩

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

lemma *j*: $j \in C$ ⟨*proof*⟩

lemma *j-n*: $j < n$ ⟨*proof*⟩

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

lemma *i*: $i < n$ **and** *P-ij0*: $P\ \$\$\ (i,\ j - (m - 1)) \neq 0$
⟨*proof*⟩

definition $w = P *_v \textit{unit-vec}\ n\ j$

lemma *w*: $w \in \textit{carrier-vec}\ n$ ⟨*proof*⟩

definition $v = \textit{map-vec cmod}\ w$

lemma *v*: $v \in \textit{carrier-vec}\ n$ ⟨*proof*⟩

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

lemma *u*: $u \in$ *carrier-vec n* ⟨*proof*⟩

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

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

lemma *main-theorem*: $(m, 1) \in$ *set n-as*
⟨*proof*⟩
end

lemma *nonneg-sr-1-largest-jb*:
  assumes *nonneg*: *nonneg-mat A*
  and *jnf*: *jordan-nf* (*map-mat complex-of-real A*) *n-as*
  and *mem*: (*m*, *lam*) $\in$ *set n-as*
  and *lam1*: *cmod lam* = *1*
  and *sr1*: $\bigwedge x.$ *poly* (*char-poly A*) $x = 0 \Longrightarrow x \leq 1$
  shows $\exists\ M.\ M \geq m \wedge (M,1) \in$ *set n-as*
⟨*proof*⟩
hide-const(**open**) *spectral-radius*

lemma (**in** *ring-hom*) *hom-smult-mat*: $mat_h\ (a \cdot_m A) = hom\ a \cdot_m mat_h\ A$
  ⟨*proof*⟩

lemma *root-char-poly-smult*: **fixes** *A* :: *complex mat*
  assumes *A*: $A \in$ *carrier-mat n n*
  and *k*: $k \neq 0$
shows (*poly* (*char-poly* $(k \cdot_m A)$) $x = 0$) = (*poly* (*char-poly A*) $(x / k) = 0$)
  ⟨*proof*⟩

theorem *real-nonneg-mat-spectral-radius-largest-jordan-block*:
  assumes *real-nonneg-mat A*
  and *jordan-nf A n-as*
  and (*m*, *lam*) $\in$ *set n-as*
  and *cmod lam* = *spectral-radius A*
shows $\exists\ M \geq m.$ (*M*, *of-real* (*spectral-radius A*)) $\in$ *set n-as*
⟨*proof*⟩

end

8 Homomorphisms of Gauss-Jordan Elimination, Kernel and More

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

Jordan-Normal-Form.Jordan-Normal-Form-Uniqueness
begin

lemma (**in** *comm-ring-hom*) *similar-mat-wit-hom*: **assumes**
similar-mat-wit A B C D
shows *similar-mat-wit* (*mat$_h$ A*) (*mat$_h$ B*) (*mat$_h$ C*) (*mat$_h$ D*)
⟨*proof*⟩

lemma (**in** *comm-ring-hom*) *similar-mat-hom*:
similar-mat A B ⟹ *similar-mat* (*mat$_h$ A*) (*mat$_h$ B*)
⟨*proof*⟩

context *field-hom*
begin
lemma *hom-swaprows*: *i* < *dim-row A* ⟹ *j* < *dim-row A* ⟹
swaprows i j (*mat$_h$ A*) = *mat$_h$* (*swaprows i j A*)
⟨*proof*⟩

lemma *hom-gauss-jordan-main*: *A* ∈ *carrier-mat nr nc* ⟹ *B* ∈ *carrier-mat nr nc2* ⟹
gauss-jordan-main (*mat$_h$ A*) (*mat$_h$ B*) *i j* =
map-prod mat$_h$ mat$_h$ (*gauss-jordan-main A B i j*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *hom-gauss-jordan-single*[*simp*]: *gauss-jordan-single* (*mat$_h$ A*) = *mat$_h$* (*gauss-jordan-single A*)
⟨*proof*⟩

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*⟩

lemma *hom-pivot-positions*[*simp*]: *pivot-positions* (*mat$_h$ A*) = *pivot-positions A*
⟨*proof*⟩

lemma *hom-kernel-dim*[*simp*]: *kernel-dim* (*mat$_h$ A*) = *kernel-dim A*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩
end
end

9 Combining Spectral Radius Theory with Perron Frobenius theorem

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

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

lemma *jnf-complexity-generic*: **fixes** *A* :: *complex mat*
assumes *A*: *A* ∈ *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* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *of-nat k* ^ *d*)
⟨*proof*⟩

lemma *norm-bound-complex-to-real*: **fixes** *A* :: *real mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *bnd*: ∃ *c1 c2*. ∀ *k*. *norm-bound* ((*map-mat complex-of-real A*) $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *of-nat k* ^ *d*)
shows ∃ *c1 c2*. ∀ *k a*. *a* ∈ *elements-mat* (*A* $\widehat{\ }_m$ *k*) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *of-nat k* ^ *d*)
⟨*proof*⟩

lemma *dim-gen-eigenspace-max-jordan-block*: **assumes** *jnf*: *jordan-nf A n-as*
shows *dim-gen-eigenspace A l d* = *order l* (*char-poly A*) ⟷
(∀ *n*. (*n*,*l*) ∈ *set n-as* ⟶ *n* ≤ *d*)
⟨*proof*⟩

lemma *jnf-complexity-1-complex*: **fixes** *A* :: *complex mat*
assumes *A*: *A* ∈ *carrier-mat n n*
and *nonneg*: *real-nonneg-mat A*
and *sr*: ⋀ *x*. *poly* (*char-poly A*) *x* = *0* ⟹ *cmod x* ≤ *1*
and *1*: *poly* (*char-poly A*) *1* = *0* ⟹
order 1 (*char-poly A*) > *d* + *1* ⟹
dim-gen-eigenspace A 1 (*d*+*1*) = *order 1* (*char-poly A*)
shows ∃ *c1 c2*. ∀ *k*. *norm-bound* (*A* $\widehat{\ }_m$ *k*) (*c1* + *c2* ∗ *of-nat k* ^ *d*)
⟨*proof*⟩

lemma *jnf-complexity-1-real*: **fixes** *A* :: *real mat*

assumes *A*: $A \in$ *carrier-mat n n*
and *nonneg*: *nonneg-mat A*
and *sr*: $\bigwedge$ *x. poly* (*char-poly A*) $x = 0 \Longrightarrow x \leq 1$
and *jb*: *poly* (*char-poly A*) $1 = 0 \Longrightarrow$
order 1 (*char-poly A*) $> d + 1 \Longrightarrow$
dim-gen-eigenspace A 1 (d+1) = *order 1* (*char-poly A*)
shows $\exists$ *c1 c2*. $\forall$ *k a*. $a \in$ *elements-mat* (A $\hat{}_m$ k) $\longrightarrow |a| \leq c1 + c2 *$ *real k* ^ *d*
⟨*proof*⟩
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*)) $\hat{}_m$ (*d* + *1*)) = *ord*)))

lemma *check-matrix-complexity*: **assumes** *A*: $A \in$ *carrier-mat n n* **and** *nn*: *nonneg-mat A*
and *check*: *check-matrix-complexity A* (*char-poly A*) *d*
shows $\exists$ *c1 c2*. $\forall$ *k a*. $a \in$ *elements-mat* (A $\hat{}_m$ k) ⟶ *abs a* ≤ (*c1* + *c2* ∗ *of-nat k* ^ *d*)
⟨*proof*⟩
end

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

References

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

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

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

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