

Count the Number of Complex Roots

Wenda Li

February 23, 2021

Abstract

Based on evaluating Cauchy indices through remainder sequences [1] [2, Chapter 11], this entry provides an effective procedure to count the number of complex roots (with multiplicity) of a polynomial within a rectangle box or a half-plane. Potential applications of this entry include certified complex root isolation (of a polynomial) and testing the Routh-Hurwitz stability criterion (i.e., to check whether all the roots of some characteristic polynomial have negative real parts).

1 An alternative Sturm sequences

theory *Extended-Sturm imports*

Sturm-Tarski.Sturm-Tarski

Winding-Number-Eval.Cauchy-Index-Theorem

begin

The main purpose of this theory is to provide an effective way to compute $\text{cindex} E a b f$ when f is a rational function. The idea is similar to and based on the evaluation of cindex-poly through $\llbracket ?a < ?b; \text{poly } ?p \text{ ?}a \neq 0; \text{poly } ?p \text{ ?}b \neq 0 \rrbracket \implies \text{cindex-poly } ?a \text{ ?}b \text{ ?}q \text{ ?}p = \text{changes-itv-smods } ?a \text{ ?}b \text{ ?}p \text{ ?}q$.

This alternative version of remainder sequences is inspired by the paper "The Fundamental Theorem of Algebra made effective: an elementary real-algebraic proof via Sturm chains" by Michael Eisermann.

hide-const *Permutations.sign*

1.1 Misc

lemma *is-unit-pCons-ex-iff:*

fixes $p::'a::\text{field } \text{poly}$

shows $\text{is-unit } p \iff (\exists a. a \neq 0 \wedge p = [a])$

<proof>

lemma *poly-gcd-iff:*

$\text{poly } (\text{gcd } p \ q) \ x=0 \iff \text{poly } p \ x=0 \wedge \text{poly } q \ x=0$

<proof>

lemma *eventually-poly-nz-at-within*:
fixes $x :: 'a::\{idom,euclidean-space\}$
assumes $p \neq 0$
shows *eventually* $(\lambda x. \text{poly } p \ x \neq 0)$ (at x within S)
 \langle proof \rangle

lemma *sgn-power*:
fixes $x::'a::linordered-idom$
shows $\text{sgn } (x^n) = (\text{if } n=0 \text{ then } 1 \text{ else if even } n \text{ then } |\text{sgn } x| \text{ else } \text{sgn } x)$
 \langle proof \rangle

lemma *poly-divide-filterlim-at-top*:
fixes $p \ q::\text{real poly}$
defines $ll \equiv (\text{if degree } q < \text{degree } p \text{ then}$
 $\text{at } 0$
 $\text{else if degree } q = \text{degree } p \text{ then}$
 $\text{nhds } (\text{lead-coeff } q / \text{lead-coeff } p)$
 $\text{else if sgn-pos-inf } q * \text{sgn-pos-inf } p > 0 \text{ then}$
 at-top
 else
 $\text{at-bot})$
assumes $p \neq 0 \ q \neq 0$
shows *filterlim* $(\lambda x. \text{poly } q \ x / \text{poly } p \ x)$ ll at-top
 \langle proof \rangle

lemma *poly-divide-filterlim-at-bot*:
fixes $p \ q::\text{real poly}$
defines $ll \equiv (\text{if degree } q < \text{degree } p \text{ then}$
 $\text{at } 0$
 $\text{else if degree } q = \text{degree } p \text{ then}$
 $\text{nhds } (\text{lead-coeff } q / \text{lead-coeff } p)$
 $\text{else if sgn-neg-inf } q * \text{sgn-neg-inf } p > 0 \text{ then}$
 at-top
 else
 $\text{at-bot})$
assumes $p \neq 0 \ q \neq 0$
shows *filterlim* $(\lambda x. \text{poly } q \ x / \text{poly } p \ x)$ ll at-bot
 \langle proof \rangle

1.2 Alternative definition of cross

definition *cross-alt* :: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{int}$ **where**
 $\text{cross-alt } p \ q \ a \ b = |\text{sign } (\text{poly } p \ a) - \text{sign } (\text{poly } q \ a)| - |\text{sign } (\text{poly } p \ b) - \text{sign } (\text{poly } q \ b)|$

lemma *cross-alt-coprime-0*:
assumes $\text{coprime } p \ q \ p = 0 \vee q = 0$
shows $\text{cross-alt } p \ q \ a \ b = 0$
 \langle proof \rangle

lemma *cross-alt-0[simp]*: $\text{cross-alt } 0 \ 0 \ a \ b = 0$ $\langle \text{proof} \rangle$

lemma *cross-alt-poly-commute*:
 $\text{cross-alt } p \ q \ a \ b = \text{cross-alt } q \ p \ a \ b$
 $\langle \text{proof} \rangle$

lemma *cross-alt-clear-n*:
assumes *coprime* $p \ q$
shows $\text{cross-alt } p \ q \ a \ b = \text{cross-alt } 1 \ (p * q) \ a \ b$
 $\langle \text{proof} \rangle$

1.3 Alternative sign variation sequence

fun *changes-alt*:: $('a :: \text{linordered-idom}) \ \text{list} \Rightarrow \text{int}$ **where**
 $\text{changes-alt } [] = 0$
 $\text{changes-alt } [-] = 0$ |
 $\text{changes-alt } (x1 \# x2 \# xs) = \text{abs}(\text{sign } x1 - \text{sign } x2) + \text{changes-alt } (x2 \# xs)$

definition *changes-alt-poly-at*:: $('a :: \text{linordered-idom}) \ \text{poly list} \Rightarrow 'a \Rightarrow \text{int}$ **where**
 $\text{changes-alt-poly-at } ps \ a = \text{changes-alt } (\text{map } (\lambda p. \text{poly } p \ a) \ ps)$

definition *changes-alt-itv-smods*:: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{int}$ **where**
 $\text{changes-alt-itv-smods } a \ b \ p \ q = (\text{let } ps = \text{smods } p \ q$
 $\text{in } \text{changes-alt-poly-at } ps \ a - \text{changes-alt-poly-at } ps \ b)$

lemma *changes-alt-itv-smods-rec*:
assumes $a < b$ *coprime* $p \ q$
shows $\text{changes-alt-itv-smods } a \ b \ p \ q = \text{cross-alt } p \ q \ a \ b + \text{changes-alt-itv-smods } a \ b \ q \ (- (p \ \text{mod } q))$
 $\langle \text{proof} \rangle$

1.4 jumpF on polynomials

definition *jumpF-polyR*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 $\text{jumpF-polyR } q \ p \ a = \text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ (\text{at-right } a)$

definition *jumpF-polyL*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real} \Rightarrow \text{real}$ **where**
 $\text{jumpF-polyL } q \ p \ a = \text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ (\text{at-left } a)$

definition *jumpF-poly-top*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real}$ **where**
 $\text{jumpF-poly-top } q \ p = \text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ \text{at-top}$

definition *jumpF-poly-bot*:: $\text{real poly} \Rightarrow \text{real poly} \Rightarrow \text{real}$ **where**
 $\text{jumpF-poly-bot } q \ p = \text{jumpF } (\lambda x. \text{poly } q \ x / \text{poly } p \ x) \ \text{at-bot}$

lemma *jumpF-polyR-0[simp]*: $\text{jumpF-polyR } 0 \ p \ a = 0$ $\text{jumpF-polyR } q \ 0 \ a = 0$
 $\langle \text{proof} \rangle$

lemma *jumpF-polyL-0[simp]*: $\text{jumpF-polyL } 0 \ p \ a = 0 \ \text{jumpF-polyL } q \ 0 \ a = 0$
(proof)

lemma *jumpF-polyR-mult-cancel*:
assumes $p' \neq 0$
shows $\text{jumpF-polyR } (p' * q) \ (p' * p) \ a = \text{jumpF-polyR } q \ p \ a$
(proof)

lemma *jumpF-polyL-mult-cancel*:
assumes $p' \neq 0$
shows $\text{jumpF-polyL } (p' * q) \ (p' * p) \ a = \text{jumpF-polyL } q \ p \ a$
(proof)

lemma *jumpF-poly-noroot*:
assumes $\text{poly } p \ a \neq 0$
shows $\text{jumpF-polyL } q \ p \ a = 0 \ \text{jumpF-polyR } q \ p \ a = 0$
(proof)

lemma *jumpF-polyR-coprime*:
assumes $\text{coprime } p \ q$
shows $\text{jumpF-polyR } q \ p \ x = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{poly } p \ x = 0 \text{ then}$
 $\text{if sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \text{ then } 1/2 \text{ else } - 1/2 \text{ else}$
 $0)$
(proof)

lemma *jumpF-polyL-coprime*:
assumes $\text{coprime } p \ q$
shows $\text{jumpF-polyL } q \ p \ x = (\text{if } p \neq 0 \wedge q \neq 0 \wedge \text{poly } p \ x = 0 \text{ then}$
 $\text{if even } (\text{order } x \ p) \longleftrightarrow \text{sign-r-pos } p \ x \longleftrightarrow \text{poly } q \ x > 0 \text{ then } 1/2 \text{ else}$
 $- 1/2 \text{ else } 0)$
(proof)

lemma *jumpF-times*:
assumes $\text{tendsto}(f \longrightarrow c) \ F$ and $c \neq 0 \ F \neq \text{bot}$
shows $\text{jumpF } (\lambda x. f \ x * g \ x) \ F = \text{sgn } c * \text{jumpF } g \ F$
(proof)

lemma *jumpF-polyR-inverse-add*:
assumes $\text{coprime } p \ q$
shows $\text{jumpF-polyR } q \ p \ x + \text{jumpF-polyR } p \ q \ x = \text{jumpF-polyR } 1 \ (q * p) \ x$
(proof)

lemma *jumpF-polyL-inverse-add*:
assumes $\text{coprime } p \ q$
shows $\text{jumpF-polyL } q \ p \ x + \text{jumpF-polyL } p \ q \ x = \text{jumpF-polyL } 1 \ (q * p) \ x$
(proof)

lemma *jumpF-polyL-smult-1*:

*jumpF-polyL (smult c q) p x = sgn c * jumpF-polyL q p x*
(proof)

lemma *jumpF-polyR-smult-1*:

*jumpF-polyR (smult c q) p x = sgn c * jumpF-polyR q p x*
(proof)

lemma

shows *jumpF-polyR-mod:jumpF-polyR q p x = jumpF-polyR (q mod p) p x* **and**
jumpF-polyL-mod:jumpF-polyL q p x = jumpF-polyL (q mod p) p x
(proof)

lemma *jumpF-poly-top-0[simp]*: *jumpF-poly-top 0 p = 0 jumpF-poly-top q 0 = 0*
(proof)

lemma *jumpF-poly-bot-0[simp]*: *jumpF-poly-bot 0 p = 0 jumpF-poly-bot q 0 = 0*
(proof)

lemma *jumpF-poly-top-code*:

jumpF-poly-top q p = (if p≠0 ∧ q≠0 ∧ degree q > degree p then
*if sgn-pos-inf q * sgn-pos-inf p > 0 then 1/2 else -1/2 else 0)*
(proof)

lemma *jumpF-poly-bot-code*:

jumpF-poly-bot q p = (if p≠0 ∧ q≠0 ∧ degree q > degree p then
*if sgn-neg-inf q * sgn-neg-inf p > 0 then 1/2 else -1/2 else 0)*
(proof)

1.5 The extended Cauchy index on polynomials

definition *cindex-polyE*:: *real ⇒ real ⇒ real poly ⇒ real poly ⇒ real* **where**

cindex-polyE a b q p = jumpF-polyR q p a + cindex-poly a b q p - jumpF-polyL
q p b

definition *cindex-poly-ubd*:: *real poly ⇒ real poly ⇒ int* **where**

cindex-poly-ubd q p = (THE l. (∀_F r in at-top. cindexE (-r) r (λx. poly q x / poly
p x) = of-int l))

lemma *cindex-polyE-0[simp]*: *cindex-polyE a b 0 p = 0 cindex-polyE a b q 0 = 0*
(proof)

lemma *cindex-polyE-mult-cancel*:

fixes *p q p'::real poly*

assumes *p'≠ 0*

shows *cindex-polyE a b (p' * q) (p' * p) = cindex-polyE a b q p*

<proof>

lemma *cindexE-eq-cindex-polyE*:

assumes $a < b$

shows $cindexE\ a\ b\ (\lambda x. poly\ q\ x / poly\ p\ x) = cindex-polyE\ a\ b\ q\ p$

<proof>

lemma *cindex-polyE-cross*:

fixes $p::real\ poly$ **and** $a\ b::real$

assumes $a < b$

shows $cindex-polyE\ a\ b\ 1\ p = cross-alt\ 1\ p\ a\ b / 2$

<proof>

lemma *cindex-polyE-inverse-add*:

fixes $p\ q::real\ poly$

assumes $cp::coprime\ p\ q$

shows $cindex-polyE\ a\ b\ q\ p + cindex-polyE\ a\ b\ p\ q = cindex-polyE\ a\ b\ 1\ (q*p)$

<proof>

lemma *cindex-polyE-inverse-add-cross*:

fixes $p\ q::real\ poly$

assumes $a < b\ coprime\ p\ q$

shows $cindex-polyE\ a\ b\ q\ p + cindex-polyE\ a\ b\ p\ q = cross-alt\ p\ q\ a\ b / 2$

<proof>

lemma *cindex-polyE-smult-1*:

fixes $p\ q::real\ poly$ **and** $c::real$

shows $cindex-polyE\ a\ b\ (smult\ c\ q)\ p = (sgn\ c) * cindex-polyE\ a\ b\ q\ p$

<proof>

lemma *cindex-polyE-mod*:

fixes $p\ q::real\ poly$

shows $cindex-polyE\ a\ b\ q\ p = cindex-polyE\ a\ b\ (q\ mod\ p)\ p$

<proof>

lemma *cindex-polyE-rec*:

fixes $p\ q::real\ poly$

assumes $a < b\ coprime\ p\ q$

shows $cindex-polyE\ a\ b\ q\ p = cross-alt\ q\ p\ a\ b / 2 + cindex-polyE\ a\ b\ (- (p\ mod\ q))\ q$

<proof>

lemma *cindex-polyE-changes-alt-itv-mods*:

assumes $a < b\ coprime\ p\ q$

shows $cindex-polyE\ a\ b\ q\ p = changes-alt-itv-smods\ a\ b\ p\ q / 2$ *<proof>*

lemma *cindex-poly-ubd-eventually*:

shows $\forall_F\ r\ in\ at-top.\ cindexE\ (-r)\ r\ (\lambda x. poly\ q\ x / poly\ p\ x) = of-int\ (cindex-poly-ubd\ q\ p)$

<proof>

lemma *cindex-poly-ubd-0*:

assumes $p=0 \vee q=0$

shows *cindex-poly-ubd* $q\ p = 0$

<proof>

lemma *cindex-poly-ubd-code*:

shows *cindex-poly-ubd* $q\ p = \text{changes-R-smods } p\ q$

<proof>

lemma *cindexE-ubd-poly*: *cindexE-ubd* $(\lambda x. \text{poly } q\ x / \text{poly } p\ x) = \text{cindex-poly-ubd } q$

p

<proof>

end

2 More useful lemmas related polynomials

theory *More-Polynomials* **imports**

Winding-Number-Eval.Missing-Algebraic

Winding-Number-Eval.Missing-Transcendental

Sturm-Tarski.PolyMisc

Budan-Fourier.BF-Misc

begin

2.1 More about *order*

lemma *order-normalize[simp]*: *order* x (*normalize* p) = *order* $x\ p$

<proof>

lemma *order-gcd*:

assumes $p \neq 0\ q \neq 0$

shows *order* x (*gcd* $p\ q$) = *min* (*order* $x\ p$) (*order* $x\ q$)

<proof>

lemma *pderiv-power*: *pderiv* $(p \wedge n)$ = *smult* (*of-nat* n) $(p \wedge (n-1)) * \text{pderiv } p$

<proof>

lemma *order-pderiv*:

fixes $p::'a::\{\text{idom, semiring-char-0}\}$ *poly*

assumes $p \neq 0\ \text{poly } p\ x=0$

shows *order* $x\ p = \text{Suc } (\text{order } x\ (\text{pderiv } p))$ *<proof>*

2.2 More about *rsquarefree*

lemma *rsquarefree-0[simp]*: $\neg \text{rsquarefree } 0$

<proof>

lemma *rsquarefree-times*:
 assumes *rsquarefree* (p*q)
 shows *rsquarefree* q *<proof>*

lemma *rsquarefree-smult-iff*:
 assumes $s \neq 0$
 shows *rsquarefree* (smult s p) \longleftrightarrow *rsquarefree* p
 <proof>

lemma *card-roots-within-rsquarefree*:
 assumes *rsquarefree* p
 shows *roots-count* p s = *card* (*roots-within* p s) *<proof>*

lemma *rsquarefree-gcd-pderiv*:
 fixes p::'a::{*factorial-ring-gcd, semiring-gcd-mult-normalize, semiring-char-0*} *poly*
 assumes $p \neq 0$
 shows *rsquarefree* (p div (gcd p (pderiv p)))
 <proof>

lemma *poly-gcd-pderiv-iff*:
 fixes p::'a::{*semiring-char-0, factorial-ring-gcd, semiring-gcd-mult-normalize*} *poly*
 shows *poly* (p div (gcd p (pderiv p))) $x=0 \longleftrightarrow$ *poly* p $x=0$
 <proof>

2.3 Composition of a polynomial and a circular path

lemma *poly-circlepath-tan-eq*:
 fixes z0::*complex* **and** r::*real* **and** p::*complex poly*
 defines q1 \equiv *fcompose* p [:(z0+r)*i, z0-r:] [i, I:] **and** q2 \equiv [i, I:] \wedge *degree* p
 assumes $0 \leq t \leq 1$ $t \neq 1/2$
 shows *poly* p (*circlepath* z0 r t) = *poly* q1 (*tan* (pi*t)) / *poly* q2 (*tan* (pi*t))
 (is ?L = ?R)
 <proof>

end

3 Procedures to count the number of complex roots

theory *Count-Complex-Roots* **imports**
 Winding-Number-Eval. Winding-Number-Eval
 Extended-Sturm
 More-Polynomials
 Budan-Fourier. Sturm-Multiple-Roots
begin

3.1 Misc

corollary *path-image-part-circlepath-subset*:

assumes $r \geq 0$
shows $\text{path-image}(\text{part-circlepath } z \ r \ st \ tt) \subseteq \text{sphere } z \ r$
<proof>

proposition *in-path-image-part-circlepath*:

assumes $w \in \text{path-image}(\text{part-circlepath } z \ r \ st \ tt)$ $0 \leq r$
shows $\text{norm}(w - z) = r$
<proof>

lemma *infinite-ball*:

fixes $a :: 'a::\text{euclidean-space}$
assumes $r > 0$
shows *infinite* ($\text{ball } a \ r$)
<proof>

lemma *infinite-cball*:

fixes $a :: 'a::\text{euclidean-space}$
assumes $r > 0$
shows *infinite* ($\text{cball } a \ r$)
<proof>

lemma *infinite-sphere*:

fixes $a :: \text{complex}$
assumes $r > 0$
shows *infinite* ($\text{sphere } a \ r$)
<proof>

lemma *infinite-halfspace-Im-gt*: *infinite* $\{x. \text{Im } x > b\}$

<proof>

lemma (*in ring-1*) *Ints-minus2*: $- a \in \mathbb{Z} \implies a \in \mathbb{Z}$

<proof>

lemma *dvd-divide-Ints-iff*:

$b \text{ dvd } a \vee b=0 \iff \text{of-int } a / \text{of-int } b \in (\mathbb{Z} :: 'a :: \{\text{field}, \text{ring-char-0}\} \text{ set})$
<proof>

lemma *of-int-div-field*:

assumes $d \text{ dvd } n$
shows $(\text{of-int}::\implies 'a::\text{field-char-0}) (n \text{ div } d) = \text{of-int } n / \text{of-int } d$
<proof>

lemma *powr-eq-1-iff*:

assumes $a > 0$
shows $(a::\text{real}) \text{ powr } b = 1 \iff a=1 \vee b=0$

<proof>

lemma *tan-inj-pi*:

$-(\pi/2) < x \implies x < \pi/2 \implies -(\pi/2) < y \implies y < \pi/2 \implies \tan x = \tan y$
 $\implies x = y$

<proof>

lemma *finite-ReZ-segments-poly-circlepath*:

finite-ReZ-segments (poly p o circlepath z0 r) 0

<proof>

3.2 Some useful conformal/*bij-betw* properties

lemma *bij-betw-plane-ball*:*bij-betw* $(\lambda x. (i-x)/(i+x)) \{x. \text{Im } x > 0\}$ (ball 0 1)

<proof>

lemma *bij-betw-axis-sphere*:*bij-betw* $(\lambda x. (i-x)/(i+x)) \{x. \text{Im } x = 0\}$ (sphere 0 1 -
 $\{-1\}$)

<proof>

lemma *bij-betw-ball-uball*:

assumes $r > 0$

shows *bij-betw* $(\lambda x. \text{complex-of-real } r*x + z0)$ (ball 0 1) (ball z0 r)

<proof>

lemma *bij-betw-sphere-usphere*:

assumes $r > 0$

shows *bij-betw* $(\lambda x. \text{complex-of-real } r*x + z0)$ (sphere 0 1) (sphere z0 r)

<proof>

lemma *proots-ball-plane-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$

assumes $p \neq 0$

shows *proots-count* p (ball 0 1) = *proots-count* (fcompose p $q1$ $q2$) $\{x. 0 < \text{Im } x\}$

<proof>

lemma *proots-sphere-axis-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$

assumes $p \neq 0$

shows *proots-count* p (sphere 0 1 - $\{-1\}$) = *proots-count* (fcompose p $q1$ $q2$)
 $\{x. 0 = \text{Im } x\}$

<proof>

lemma *proots-card-ball-plane-eq*:

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$

assumes $p \neq 0$

shows *card* (*proots-within* p (ball 0 1)) = *card* (*proots-within* (fcompose p $q1$ $q2$)
 $\{x. 0 < \text{Im } x\}$)

<proof>

lemma *roots-card-sphere-axis-eq:*

defines $q1 \equiv [i, -1:]$ **and** $q2 \equiv [i, 1:]$

assumes $p \neq 0$

shows $\text{card} (\text{roots-within } p (\text{sphere } 0 \ 1 - \{-1\}))$
 $= \text{card} (\text{roots-within} (\text{fcompose } p \ q1 \ q2) \{x. \ 0 = \text{Im } x\})$

<proof>

lemma *roots-uball-eq:*

fixes $z0::\text{complex}$ **and** $r::\text{real}$

defines $q \equiv [z0, \text{of-real } r:]$

assumes $p \neq 0$ **and** $r > 0$

shows $\text{roots-count } p (\text{ball } z0 \ r) = \text{roots-count} (p \circ_p \ q) (\text{ball } 0 \ 1)$

<proof>

lemma *roots-card-uball-eq:*

fixes $z0::\text{complex}$ **and** $r::\text{real}$

defines $q \equiv [z0, \text{of-real } r:]$

assumes $r > 0$

shows $\text{card} (\text{roots-within } p (\text{ball } z0 \ r)) = \text{card} (\text{roots-within} (p \circ_p \ q) (\text{ball } 0 \ 1))$

<proof>

lemma *roots-card-usphere-eq:*

fixes $z0::\text{complex}$ **and** $r::\text{real}$

defines $q \equiv [z0, \text{of-real } r:]$

assumes $r > 0$

shows $\text{card} (\text{roots-within } p (\text{sphere } z0 \ r)) = \text{card} (\text{roots-within} (p \circ_p \ q) (\text{sphere } 0 \ 1))$

<proof>

3.3 Combining two real polynomials into a complex one

definition *cpoly-of:: real poly \Rightarrow real poly \Rightarrow complex poly where*

cpoly-of $pR \ pI = \text{map-poly of-real } pR + \text{smult } i (\text{map-poly of-real } pI)$

lemma *cpoly-of-eq-0-iff[iff]:*

cpoly-of $pR \ pI = 0 \iff pR = 0 \wedge pI = 0$

<proof>

lemma *cpoly-of-decompose:*

$p = \text{cpoly-of} (\text{map-poly } \text{Re } p) (\text{map-poly } \text{Im } p)$

<proof>

lemma *cpoly-of-dist-right:*

cpoly-of $(pR * g) (pI * g) = \text{cpoly-of } pR \ pI * (\text{map-poly of-real } g)$

<proof>

lemma *poly-cpoly-of-real:*

$\text{poly } (\text{cpoly-of } pR \ pI) \ (\text{of-real } x) = \text{Complex } (\text{poly } pR \ x) \ (\text{poly } pI \ x)$
 ⟨proof⟩

lemma *poly-cpoly-of-real-iff*:

shows $\text{poly } (\text{cpoly-of } pR \ pI) \ (\text{of-real } t) = 0 \iff \text{poly } pR \ t = 0 \wedge \text{poly } pI \ t = 0$
 ⟨proof⟩

lemma *order-cpoly-gcd-eq*:

assumes $pR \neq 0 \vee pI \neq 0$

shows $\text{order } t \ (\text{cpoly-of } pR \ pI) = \text{order } t \ (\text{gcd } pR \ pI)$

⟨proof⟩

3.4 Number of roots on a (bounded or unbounded) segment

— 1 dimensional hyperplane

definition *unbounded-line::'a::real-vector \Rightarrow 'a \Rightarrow 'a set* **where**

$\text{unbounded-line } a \ b = \{\{x. \exists u::\text{real. } x = (1 - u) *R \ a + u *R \ b\}\}$

definition *roots-line-card:: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat* **where**

$\text{roots-line-card } p \ st \ tt = \text{card } (\text{roots-within } p \ (\text{open-segment } st \ tt))$

definition *roots-unbounded-line-card:: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat* **where**

$\text{roots-unbounded-line-card } p \ st \ tt = \text{card } (\text{roots-within } p \ (\text{unbounded-line } st \ tt))$

definition *roots-unbounded-line :: complex poly \Rightarrow complex \Rightarrow complex \Rightarrow nat* **where**

$\text{roots-unbounded-line } p \ st \ tt = \text{roots-count } p \ (\text{unbounded-line } st \ tt)$

lemma *card-proots-open-segments*:

assumes $\text{poly } p \ st \neq 0 \ \text{poly } p \ tt \neq 0$

shows $\text{card } (\text{roots-within } p \ (\text{open-segment } st \ tt)) =$

$(\text{let } pc = \text{pcompose } p \ [:\text{st}, \text{tt} - \text{st};$

$\text{pR} = \text{map-poly } \text{Re } pc;$

$\text{pI} = \text{map-poly } \text{Im } pc;$

$g = \text{gcd } pR \ pI$

$\text{in changes-itv-smods } 0 \ 1 \ g \ (\text{pderiv } g)) \ (\text{is } ?L = ?R)$

⟨proof⟩

lemma *unbounded-line-closed-segment*: $\text{closed-segment } a \ b \subseteq \text{unbounded-line } a \ b$

⟨proof⟩

lemma *card-proots-unbounded-line*:

assumes $st \neq tt$

shows $\text{card } (\text{roots-within } p \ (\text{unbounded-line } st \ tt)) =$

$(\text{let } pc = \text{pcompose } p \ [:\text{st}, \text{tt} - \text{st};$

$\text{pR} = \text{map-poly } \text{Re } pc;$

$\text{pI} = \text{map-poly } \text{Im } pc;$

$g = \text{gcd } pR \ pI$

in nat (changes-R-smods g (pderiv g)) (is ?L = ?R)

⟨proof⟩

lemma *roots-unbounded-line*:
assumes *st≠tt p≠0*
shows (*roots-count p (unbounded-line st tt)*) =
 (*let pc = pcompose p [:st, tt - st];*
 pR = map-poly Re pc;
 pI = map-poly Im pc;
 g = gcd pR pI
in nat (changes-R-smods-ext g (pderiv g)) (is ?L = ?R))

⟨proof⟩

lemma *roots-unbounded-line-card-code*[code]:
roots-unbounded-line-card p st tt =
 (*if st≠tt then*
 (*let pc = pcompose p [:st, tt - st];*
 pR = map-poly Re pc;
 pI = map-poly Im pc;
 g = gcd pR pI
 in nat (changes-R-smods g (pderiv g)))
else
Code.abort (STR "roots-unbounded-line-card fails due to invalid
hyperplanes.")
 (*λ-. roots-unbounded-line-card p st tt*)

⟨proof⟩

lemma *roots-unbounded-line-code*[code]:
roots-unbounded-line p st tt =
 (*if st≠tt then*
 if p≠0 then
 (*let pc = pcompose p [:st, tt - st];*
 pR = map-poly Re pc;
 pI = map-poly Im pc;
 g = gcd pR pI
 in nat (changes-R-smods-ext g (pderiv g)))
 else
 Code.abort (STR "roots-unbounded-line fails due to p=0")
 (*λ-. roots-unbounded-line p st tt*)
 else
 Code.abort (STR "roots-unbounded-line fails due to invalid
 hyperplanes.")
 (*λ-. roots-unbounded-line p st tt*))

⟨proof⟩

3.5 Checking if there a polynomial root on a closed segment

definition *no-roots-line::complex poly ⇒ complex ⇒ complex ⇒ bool* **where**
no-roots-line p st tt = (*roots-within p (closed-segment st tt)* = {})

lemma *no-roots-line-code*[code]: *no-roots-line* p st $tt = (if$ *poly* p $st \neq 0 \wedge$ *poly* p $tt \neq 0$ *then*

(let $pc = pcompose$ p $[:st, tt - st];$
 $pR = map-poly$ Re $pc;$
 $pI = map-poly$ Im $pc;$
 $g = gcd$ pR pI
in if *changes-itv-smods* 0 1 g (*pderiv* g) = 0 *then* *True* *else* *False*) *else*

False)

(is ?L = ?R)

<proof>

3.6 Counting roots in a rectangle

definition *roots-rectangle* :: *complex poly* \Rightarrow *complex* \Rightarrow *complex* \Rightarrow *nat* **where**
roots-rectangle p lb $ub = roots-count$ p (*box* lb ub)

lemma *closed-segment-imp-Re-Im*:

fixes $x :: complex$

assumes $x \in closed-segment$ lb ub

shows Re $lb \leq Re$ $ub \implies Re$ $lb \leq Re$ $x \wedge Re$ $x \leq Re$ ub

Im $lb \leq Im$ $ub \implies Im$ $lb \leq Im$ $x \wedge Im$ $x \leq Im$ ub

<proof>

lemma *closed-segment-degen-complex*:

[[Re $lb = Re$ $ub; Im$ $lb \leq Im$ ub]]

$\implies x \in closed-segment$ lb $ub \iff Re$ $x = Re$ $lb \wedge Im$ $lb \leq Im$ $x \wedge Im$ $x \leq Im$ ub

[[Im $lb = Im$ $ub; Re$ $lb \leq Re$ ub]]

$\implies x \in closed-segment$ lb $ub \iff Im$ $x = Im$ $lb \wedge Re$ $lb \leq Re$ $x \wedge Re$ $x \leq Re$ ub

<proof>

lemma *complex-box-ne-empty*:

fixes a $b :: complex$

shows

cbox a $b \neq \{\}$ $\iff (Re$ $a \leq Re$ $b \wedge Im$ $a \leq Im$ $b)$

box a $b \neq \{\}$ $\iff (Re$ $a < Re$ $b \wedge Im$ $a < Im$ $b)$

<proof>

lemma *roots-rectangle-code1*:

roots-rectangle p lb $ub = (if$ Re $lb < Re$ $ub \wedge Im$ $lb < Im$ ub *then*

if $p \neq 0$ *then*

no-roots-line p lb (*Complex* (Re ub) (Im lb))

\wedge *no-roots-line* p (*Complex* (Re ub) (Im lb)) ub

\wedge *no-roots-line* p ub (*Complex* (Re lb) (Im ub))

\wedge *no-roots-line* p (*Complex* (Re lb) (Im ub)) lb *then*

(

```

let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0];
  pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
  p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb)];
  pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
  p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0];
  pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
  p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub)];
  pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
  nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)
+ changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
+ changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4)) div 4)
)
else Code.abort (STR "proots-rectangle fails when there is a root on the
border.")
(λ-. proots-rectangle p lb ub)
else Code.abort (STR "proots-rectangle fails when p=0.")
(λ-. proots-rectangle p lb ub)
else 0)
⟨proof⟩

```

lemma *proots-rectangle-code2*[code]:

```

proots-rectangle p lb ub = (if Re lb < Re ub ∧ Im lb < Im ub then
if p≠0 then
if poly p lb ≠ 0 ∧ poly p (Complex (Re ub) (Im lb)) ≠ 0
∧ poly p ub ≠ 0 ∧ poly p (Complex (Re lb) (Im ub)) ≠ 0
then
  (let p1 = pcompose p [:lb, Complex (Re ub - Re lb) 0];
    pR1 = map-poly Re p1; pI1 = map-poly Im p1; gc1 = gcd pR1 pI1;
    p2 = pcompose p [:Complex (Re ub) (Im lb), Complex 0 (Im ub - Im
lb)];
    pR2 = map-poly Re p2; pI2 = map-poly Im p2; gc2 = gcd pR2 pI2;
    p3 = pcompose p [:ub, Complex (Re lb - Re ub) 0];
    pR3 = map-poly Re p3; pI3 = map-poly Im p3; gc3 = gcd pR3 pI3;
    p4 = pcompose p [:Complex (Re lb) (Im ub), Complex 0 (Im lb - Im
ub)];
    pR4 = map-poly Re p4; pI4 = map-poly Im p4; gc4 = gcd pR4 pI4
in
  if changes-itv-smods 0 1 gc1 (pderiv gc1) = 0
  ∧ changes-itv-smods 0 1 gc2 (pderiv gc2) = 0
  ∧ changes-itv-smods 0 1 gc3 (pderiv gc3) = 0
  ∧ changes-itv-smods 0 1 gc4 (pderiv gc4) = 0
  then
    nat ( - (changes-alt-itv-smods 0 1 (pR1 div gc1) (pI1 div gc1)
+ changes-alt-itv-smods 0 1 (pR2 div gc2) (pI2 div gc2)

```

```

      + changes-alt-itv-smods 0 1 (pR3 div gc3) (pI3 div gc3)
      + changes-alt-itv-smods 0 1 (pR4 div gc4) (pI4 div gc4) div 4)
    else Code.abort (STR "proots-rectangle fails when there is a root on
the border.")
      (λ-. proots-rectangle p lb ub)
    else Code.abort (STR "proots-rectangle fails when there is a root on the
border.")
      (λ-. proots-rectangle p lb ub)
    else Code.abort (STR "proots-rectangle fails when p=0.")
      (λ-. proots-rectangle p lb ub)
    else 0)
⟨proof⟩

```

3.7 Polynomial roots on the upper half-plane

— Roots counted WITH multiplicity

definition *proots-upper* :: *complex poly* ⇒ *nat* **where**
proots-upper p = *proots-count* p {z. *Im* z > 0}

— Roots counted WITHOUT multiplicity

definition *proots-upper-card* :: *complex poly* ⇒ *nat* **where**
proots-upper-card p = *card* (*proots-within* p {x. *Im* x > 0})

lemma *Im-Ln-tendsto-at-top*: ((λx. *Im* (Ln (Complex a x))) → pi/2) *at-top*
⟨proof⟩

lemma *Im-Ln-tendsto-at-bot*: ((λx. *Im* (Ln (Complex a x))) → - pi/2) *at-bot*
⟨proof⟩

lemma *Re-winding-number-tendsto-part-circlepath*:

shows ((λr. *Re* (*winding-number* (*part-circlepath* z0 r 0 pi) a)) → 1/2)
at-top
⟨proof⟩

lemma *not-image-at-top-poly-part-circlepath*:

assumes *degree* p > 0
shows ∀ *F* r *in* *at-top*. b ∉ *path-image* (*poly* p o *part-circlepath* z0 r st tt)
⟨proof⟩

lemma *not-image-poly-part-circlepath*:

assumes *degree* p > 0
shows ∃ r > 0. b ∉ *path-image* (*poly* p o *part-circlepath* z0 r st tt)
⟨proof⟩

lemma *Re-winding-number-poly-part-circlepath*:

assumes *degree* p > 0
shows ((λr. *Re* (*winding-number* (*poly* p o *part-circlepath* z0 r 0 pi) 0)) →
degree p / 2) *at-top*

<proof>

lemma *Re-winding-number-poly-linepth:*

fixes $pp::\text{complex poly}$

defines $g \equiv (\lambda r. \text{poly } pp \circ \text{linepath } (-r) \text{ (of-real } r))$

assumes $\text{lead-coeff } pp=1$ **and** $\text{no-real-zero}:\forall x \in \text{roots } pp. \text{Im } x \neq 0$

shows $((\lambda r. 2 * \text{Re } (\text{winding-number } (g \ r) \ 0) + \text{cindex-pathE } (g \ r) \ 0) \longrightarrow 0)$

at-top

<proof>

lemma *roots-upper-cindex-eq:*

assumes $\text{lead-coeff } p=1$ **and** $\text{no-real-roots}:\forall x \in \text{roots } p. \text{Im } x \neq 0$

shows $\text{roots-upper } p =$

$(\text{degree } p - \text{cindex-poly-ubd } (\text{map-poly } \text{Im } p) (\text{map-poly } \text{Re } p)) / 2$

<proof>

lemma *cindexE-roots-on-horizontal-border:*

fixes $a::\text{complex}$ **and** $s::\text{real}$

defines $g \equiv \text{linepath } a \ (a + \text{of-real } s)$

assumes $pqr:p = q * r$ **and** $r\text{-monic}:\text{lead-coeff } r=1$ **and** $r\text{-roots}:\forall x \in \text{roots } r. \text{Im } x = \text{Im } a$

shows $\text{cindexE } lb \ ub \ (\lambda t. \text{Im } ((\text{poly } p \circ g) \ t) / \text{Re } ((\text{poly } p \circ g) \ t)) =$
 $\text{cindexE } lb \ ub \ (\lambda t. \text{Im } ((\text{poly } q \circ g) \ t) / \text{Re } ((\text{poly } q \circ g) \ t))$

<proof>

lemma *poly-decompose-by-roots:*

fixes $p :: 'a::\text{idom poly}$

assumes $p \neq 0$

shows $\exists q \ r. p = q * r \wedge \text{lead-coeff } q=1 \wedge (\forall x \in \text{roots } q. P \ x) \wedge (\forall x \in \text{roots } r. \neg P \ x)$ *<proof>*

lemma *roots-upper-cindex-eq':*

assumes $\text{lead-coeff } p=1$

shows $\text{roots-upper } p = (\text{degree } p - \text{roots-count } p \ \{x. \text{Im } x=0\} - \text{cindex-poly-ubd } (\text{map-poly } \text{Im } p) (\text{map-poly } \text{Re } p)) / 2$

<proof>

lemma *roots-within-upper-squarefree:*

assumes $\text{rsquarefree } p$

shows $\text{card } (\text{roots-within } p \ \{x. \text{Im } x > 0\}) = (\text{let}$

$pp = \text{smult } (\text{inverse } (\text{lead-coeff } p)) \ p;$

$pI = \text{map-poly } \text{Im } pp;$

$pR = \text{map-poly } \text{Re } pp;$

$g = \text{gcd } pR \ pI$

in

$\text{nat } ((\text{degree } p - \text{changes-R-smods } g \ (\text{pderiv } g) - \text{changes-R-smods } pR$

pI) $\text{div } 2$)
)
 ⟨*proof*⟩

lemma *roots-upper-code1*[*code*]:

roots-upper $p =$
 (if $p \neq 0$ then
 (let $pp = \text{smult } (\text{inverse } (\text{lead-coeff } p)) p;$
 $pI = \text{map-poly } \text{Im } pp;$
 $pR = \text{map-poly } \text{Re } pp;$
 $g = \text{gcd } pI pR$
 in
 $\text{nat } ((\text{degree } p - \text{nat } (\text{changes-R-smods-ext } g (pderiv g)) - \text{changes-R-smods } pR pI) \text{div } 2)$
)
 else
 $\text{Code.abort } (\text{STR } "roots-upper \text{ fails when } p=0.") (\lambda-. \text{roots-upper } p)$)
 ⟨*proof*⟩

lemma *roots-upper-card-code*[*code*]:

roots-upper-card $p = (\text{if } p=0 \text{ then } 0 \text{ else}$
 (let
 $pf = p \text{div } (\text{gcd } p (pderiv p));$
 $pp = \text{smult } (\text{inverse } (\text{lead-coeff } pf)) pf;$
 $pI = \text{map-poly } \text{Im } pp;$
 $pR = \text{map-poly } \text{Re } pp;$
 $g = \text{gcd } pR pI$
 in
 $\text{nat } ((\text{degree } pf - \text{changes-R-smods } g (pderiv g)) - \text{changes-R-smods } pR pI) \text{div } 2)$
))
 ⟨*proof*⟩

3.8 Polynomial roots on a general half-plane

the number of roots of polynomial p , counted with multiplicity, on the left half plane of the vector $b - a$.

definition *roots-half* :: *complex poly* \Rightarrow *complex* \Rightarrow *complex* \Rightarrow *nat* **where**
 $\text{roots-half } p a b = \text{roots-count } p \{w. \text{Im } ((w-a) / (b-a)) > 0\}$

lemma *roots-half-empty*:

assumes $a=b$
shows $\text{roots-half } p a b = 0$
 ⟨*proof*⟩

lemma *roots-half-roots-upper*:

assumes $a \neq b \ p \neq 0$
shows $\text{roots-half } p a b = \text{roots-upper } (p \text{compose } p [:a, (b-a):])$

<proof>

lemma *proots-half-code1*[code]:

proots-half p a b = (if $a \neq b$ then
if $p \neq 0$ then *proots-upper* ($p \circ_p$ [: a , $b - a$:])
else *Code.abort* (STR "proots-half fails when $p=0$.")
(λ -. *proots-half* p a b)
else 0)

<proof>

3.9 Polynomial roots within a circle (open ball)

— Roots counted WITH multiplicity

definition *proots-ball*::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-ball p $z0$ r = *proots-count* p (*ball* $z0$ r)

— Roots counted WITHOUT multiplicity

definition *proots-ball-card* ::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-ball-card p $z0$ r = *card* (*proots-within* p (*ball* $z0$ r))

lemma *proots-ball-code1*[code]:

proots-ball p $z0$ r = (if $r \leq 0$ then
0
else if $p \neq 0$ then
proots-upper (*fcompose* ($p \circ_p$ [: $z0$, of-real r :]) [:i,-1:] [:i,1:])
else
Code.abort (STR "proots-ball fails when $p=0$.")
(λ -. *proots-ball* p $z0$ r)
)

<proof>

lemma *proots-ball-card-code1*[code]:

proots-ball-card p $z0$ r =
(if $r \leq 0 \vee p=0$ then
0
else
proots-upper-card (*fcompose* ($p \circ_p$ [: $z0$, of-real r :]) [:i,-1:] [:i,1:])
)

<proof>

3.10 Polynomial roots on a circle (sphere)

— Roots counted WITH multiplicity

definition *proots-sphere*::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-sphere p $z0$ r = *proots-count* p (*sphere* $z0$ r)

— Roots counted WITHOUT multiplicity

definition *proots-sphere-card* ::*complex poly* \Rightarrow *complex* \Rightarrow *real* \Rightarrow *nat* **where**
proots-sphere-card p $z0$ r = *card* (*proots-within* p (*sphere* $z0$ r))

```

lemma proots-sphere-card-code1[code]:
  proots-sphere-card p z0 r =
    ( if r=0 then
      (if poly p z0=0 then 1 else 0)
    else if r < 0 ∨ p=0 then
      0
    else
      (if poly p (z0-r) =0 then 1 else 0) +
      proots-unbounded-line-card (fcompose (p ∘p [:z0, of-real r:])) [:i,-1:]
    )
  [:i,1:]
  )
  ⟨proof⟩

```

3.11 Polynomial roots on a closed ball

— Roots counted WITH multiplicity

definition *proots-cball::complex poly ⇒ complex ⇒ real ⇒ nat* **where**
proots-cball p z0 r = *proots-count* p (*cball* z0 r)

— Roots counted WITHOUT multiplicity

definition *proots-cball-card ::complex poly ⇒ complex ⇒ real ⇒ nat* **where**
proots-cball-card p z0 r = *card* (*proots-within* p (*cball* z0 r))

```

lemma proots-cball-card-code1[code]:
  proots-cball-card p z0 r =
    ( if r=0 then
      (if poly p z0=0 then 1 else 0)
    else if r < 0 ∨ p=0 then
      0
    else
      ( let pp=fcompose (p ∘p [:z0, of-real r:])) [:i,-1:] [:i,1:]
        in
        (if poly p (z0-r) =0 then 1 else 0)
        + proots-unbounded-line-card pp 0 1
        + proots-upper-card pp
      )
    )
  ⟨proof⟩

```

end

4 Some examples for complex root counting

```

theory Count-Complex-Roots-Examples
  imports Count-Complex-Roots
begin

```

value *proots-rectangle* [:2*i,0,i:] (*Complex* (-1) 0) (*Complex* 2 2)

value *proots-rectangle* [-1,-2*i,1:]
(*Complex* (-1) 0) (*Complex* 2 2)

value *proots-half* [:1-i,2-i,1:]
0 (*Complex* 0 1)

value *proots-half* [:1-i,2-i,1:] (*Complex* 0 1) 0

value [*code*] *proots-ball* ([: -2,1:]*[: -2,1:]*[: -3,1:]) 0 4

value [*code*] *proots-ball-card* ([: -2,1:]*[: -2,1:]*[: -3,1:]) 0 3

end

5 Acknowledgements

The work was supported by the ERC Advanced Grant ALEXANDRIA (Project 742178), funded by the European Research Council and led by Professor Lawrence Paulson at the University of Cambridge, UK.

References

- [1] M. Eisermann. The fundamental theorem of algebra made effective: An elementary real-algebraic proof via Sturm chains. *American Mathematical Monthly*, 119(9):715–752, 2012.
- [2] Q. I. Rahman and G. Schmeisser. *Analytic theory of polynomials*. Number 26. Oxford University Press, 2002.