Count the Number of Complex Roots

Wenda Li

June 24, 2019

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 *cindexE a b f* when *f* is a rational function. The idea is similar to and based on the evaluation of *cindex-poly* through $[\![?a < ?b; poly\ ?p\ ?a \neq 0; poly\ ?p\ ?b \neq 0]\!] \Longrightarrow cindex\text{-}poly\ ?a\ ?b\ ?q\ ?p = changes\text{-}itv\text{-}smods\ ?a\ ?b\ ?p\ ?q$.

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 $p::'a::field\ poly$
shows $is\text{-}unit\ p \longleftrightarrow (\exists\, a.\ a\neq 0 \wedge p=[:a:])$
⟨*proof*⟩

lemma *poly-gcd-iff*:
$poly\ (gcd\ p\ q)\ x=0 \longleftrightarrow poly\ p\ x=0 \wedge poly\ q\ x=0$
⟨*proof*⟩

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

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

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

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

1.2 Alternative definition of cross

definition *cross-alt* :: *real poly* ⇒*real poly* ⇒ *real* ⇒ *real* ⇒ *int* **where**
cross-alt p q a b=|*sign* (*poly p a*) − *sign* (*poly q a*)| − |*sign* (*poly p b*) − *sign*(*poly q b*)|

lemma *cross-alt-coprime-0*:
assumes *coprime p q p*=0∨*q*=0
shows *cross-alt p q a b*=0
⟨*proof*⟩

lemma *cross-alt-0*[*simp*]: *cross-alt 0 0 a b=0* ⟨*proof*⟩

lemma *cross-alt-poly-commute*:
cross-alt p q a b = cross-alt q p a b
⟨*proof*⟩

lemma *cross-alt-clear-n*:
assumes *coprime p q*
shows *cross-alt p q a b = cross-alt 1 (p∗q) a b*
⟨*proof*⟩

1.3 Alternative sign variation sequencse

fun *changes-alt*:: (*'a* ::*linordered-idom*) *list* ⇒ *int* **where**
changes-alt [] = *0*|
changes-alt [-] = *0* |
changes-alt (*x1*#*x2*#*xs*) = *abs*(*sign x1* − *sign x2*) + *changes-alt* (*x2*#*xs*)

definition *changes-alt-poly-at*::(*'a* ::*linordered-idom*) *poly list* ⇒ *'a* ⇒ *int* **where**
changes-alt-poly-at ps a= changes-alt (*map* (λ*p*. *poly p a*) *ps*)

definition *changes-alt-itv-smods*:: *real* ⇒ *real* ⇒*real poly* ⇒ *real poly* ⇒ *int* **where**
changes-alt-itv-smods a b p q= (*let ps*= *smods p q*
in changes-alt-poly-at ps a − *changes-alt-poly-at ps b*)

lemma *changes-alt-itv-smods-rec*:
assumes *a*<*b coprime p q*
shows *changes-alt-itv-smods a b p q* = *cross-alt p q a b* + *changes-alt-itv-smods a b q* (−(*p mod q*))
⟨*proof*⟩

1.4 jumpF on polynomials

definition *jumpF-polyR*:: *real poly* ⇒ *real poly* ⇒ *real* ⇒ *real* **where**
jumpF-polyR q p a = jumpF (λ*x*. *poly q x* / *poly p x*) (*at-right a*)

definition *jumpF-polyL*:: *real poly* ⇒ *real poly* ⇒ *real* ⇒ *real* **where**
jumpF-polyL q p a = jumpF (λ*x*. *poly q x* / *poly p x*) (*at-left a*)

definition *jumpF-poly-top*:: *real poly* ⇒ *real poly* ⇒ *real* **where**
jumpF-poly-top q p = jumpF (λ*x*. *poly q x* / *poly p x*) *at-top*

definition *jumpF-poly-bot*:: *real poly* ⇒ *real poly* ⇒ *real* **where**
jumpF-poly-bot q p = jumpF (λ*x*. *poly q x* / *poly p x*) *at-bot*

lemma *jumpF-polyR-0*[*simp*]: *jumpF-polyR 0 p a = 0 jumpF-polyR q 0 a = 0*
⟨*proof*⟩

lemma *jumpF-polyL-0*[*simp*]: *jumpF-polyL 0 p a = 0 jumpF-polyL q 0 a = 0*
⟨*proof*⟩

lemma *jumpF-polyR-mult-cancel*:
assumes *p'≠0*
shows *jumpF-polyR (p' * q) (p' * p) a = jumpF-polyR q p a*
⟨*proof*⟩

lemma *jumpF-polyL-mult-cancel*:
assumes *p'≠0*
shows *jumpF-polyL (p' * q) (p' * p) a = jumpF-polyL q p a*
⟨*proof*⟩

lemma *jumpF-poly-noroot*:
assumes *poly p a≠0*
shows *jumpF-polyL q p a = 0 jumpF-polyR q p a = 0*
⟨*proof*⟩

lemma *jumpF-polyR-coprime*:
assumes *coprime p q*
shows *jumpF-polyR q p x = (if p ≠ 0 ∧ q ≠ 0 ∧ poly p x=0 then*
if sign-r-pos p x ⟷ poly q x>0 then 1/2 else − 1/2
else 0)
⟨*proof*⟩

lemma *jumpF-polyL-coprime*:
assumes *coprime p q*
shows *jumpF-polyL q p x = (if p ≠ 0 ∧ q ≠ 0 ∧ poly p x=0 then*
if even (order x p) ⟷ sign-r-pos p x ⟷ poly q x>0 then 1/2 else
− 1/2 else 0)
⟨*proof*⟩

lemma *jumpF-times*:
assumes *tendsto*:*(f ⟶ c) F* **and** *c≠0 F≠bot*
shows *jumpF (λx. f x * g x) F = sgn c * jumpF g F*
⟨*proof*⟩

lemma *jumpF-polyR-inverse-add*:
assumes *coprime p q*
shows *jumpF-polyR q p x + jumpF-polyR p q x = jumpF-polyR 1 (q*p) x*
⟨*proof*⟩

lemma *jumpF-polyL-inverse-add*:
assumes *coprime p q*
shows *jumpF-polyL q p x + jumpF-polyL p q x = 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* (λ*x*. *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 ∨ q=0*
shows *cindex-poly-ubd q p = 0*
⟨*proof*⟩

lemma *cindex-poly-ubd-code*:
shows *cindex-poly-ubd q p = changes-R-smods p q*
⟨*proof*⟩

lemma *cindexE-ubd-poly*: *cindexE-ubd* (*λx. poly q x/poly p x*) = *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≠0 q≠0*
shows *order x* (*gcd p q*) = *min* (*order x p*) (*order x q*)
⟨*proof*⟩

lemma *pderiv-power*: *pderiv* (*p ^ n*) = *smult* (*of-nat n*) (*p ^* (*n−1*)) ∗ *pderiv p*
⟨*proof*⟩

lemma *order-pderiv*:
fixes *p*::*'a*::{*idom*,*semiring-char-0*} *poly*
assumes *p≠0 poly p x=0*
shows *order x p = Suc* (*order x* (*pderiv p*)) ⟨*proof*⟩

2.2 More about *rsquarefree*

lemma *rsquarefree-0*[*simp*]: ¬ *rsquarefree 0*
⟨*proof*⟩

lemma *rsquarefree-times*:
assumes *rsquarefree* (*p*∗*q*)
shows *rsquarefree q* ⟨*proof*⟩

lemma *rsquarefree-smult-iff*:
assumes *s*≠*0*
shows *rsquarefree* (*smult s p*) ⟷ *rsquarefree p*
⟨*proof*⟩

lemma *card-proots-within-rsquarefree*:
assumes *rsquarefree p*
shows *proots-count p s* = *card* (*proots-within p s*) ⟨*proof*⟩

lemma *rsquarefree-gcd-pderiv*:
fixes *p*::*'a*::{*factorial-ring-gcd*,*semiring-char-0*} *poly*
assumes *p*≠*0*
shows *rsquarefree* (*p div* (*gcd p* (*pderiv p*)))
⟨*proof*⟩

lemma *poly-gcd-pderiv-iff*:
fixes *p*::*'a*::{*semiring-char-0*,*factorial-ring-gcd*} *poly*
shows *poly* (*p div* (*gcd p* (*pderiv p*))) *x* =*0* ⟷ *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*≡ *fcompose p* [:(*z0*+*r*)∗i,*z0*−*r*:] [:i,*1*:] **and** *q2* ≡ [:i,*1*:] ^ *degree p*
assumes *0*≤*t t*≤*1 t*≠*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 *path-image*(*part-circlepath z r st tt*) $\subseteq$ *sphere z r*
⟨*proof*⟩

proposition *in-path-image-part-circlepath*:
assumes $w \in$ *path-image*(*part-circlepath z r st tt*) $0 \leq r$
shows $norm(w - z) = r$
⟨*proof*⟩

lemma *infinite-ball*:
fixes $a :: 'a$::*euclidean-space*
assumes $r > 0$
shows *infinite* (*ball a r*)
⟨*proof*⟩

lemma *infinite-cball*:
fixes $a :: 'a$::*euclidean-space*
assumes $r > 0$
shows *infinite* (*cball a r*)
⟨*proof*⟩

lemma *infinite-sphere*:
fixes $a ::$ *complex*
assumes $r > 0$
shows *infinite* (*sphere a r*)
⟨*proof*⟩

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

lemma (**in** *ring-1*) *Ints-minus2*: $-\ a \in \mathbb{Z} \Longrightarrow a \in \mathbb{Z}$
⟨*proof*⟩

lemma *dvd-divide-Ints-iff*:
b dvd a $\vee$ *b=0* $\longleftrightarrow$ *of-int a / of-int b* $\in$ ($\mathbb{Z} :: 'a ::$ {*field*,*ring-char-0*} *set*)
⟨*proof*⟩

lemma *of-int-div-field*:
assumes *d dvd n*
shows (*of-int*::-⇒$'a$::*field-char-0*) (*n div d*) = *of-int n / of-int d*
⟨*proof*⟩

lemma *powr-eq-1-iff*:
assumes *a>0*
shows (*a*::*real*) *powr b* =*1* $\longleftrightarrow$ *a=1* $\vee$ *b=0*

⟨*proof*⟩

lemma *tan-inj-pi*:
  $-(pi/2) < x \Longrightarrow x < pi/2 \Longrightarrow -(pi/2) < y \Longrightarrow y < pi/2 \Longrightarrow tan\ x = tan\ y \Longrightarrow x = y$
  ⟨*proof*⟩

lemma *finite-ReZ-segments-poly-circlepath*:
    finite-ReZ-segments (*poly p* ∘ *circlepath z0 r*) *0*
⟨*proof*⟩

3.2 Some useful conformal/*bij-betw* properties

lemma *bij-betw-plane-ball*:*bij-betw* (λ*x*. (i−*x*)/(i+*x*)) {*x*. *Im x*>*0*} (*ball 0 1*)
⟨*proof*⟩

lemma *bij-betw-axis-sphere*:*bij-betw* (λ*x*. (i−*x*)/(i+*x*)) {*x*. *Im x*=*0*} (*sphere 0 1* − {−*1*})
⟨*proof*⟩

lemma *bij-betw-ball-uball*:
  assumes *r*>*0*
  shows *bij-betw* (λ*x*. *complex-of-real r*∗*x* + *z0*) (*ball 0 1*) (*ball z0 r*)
⟨*proof*⟩

lemma *bij-betw-sphere-usphere*:
  assumes *r*>*0*
  shows *bij-betw* (λ*x*. *complex-of-real r*∗*x* + *z0*) (*sphere 0 1*) (*sphere z0 r*)
⟨*proof*⟩

lemma *proots-ball-plane-eq*:
  defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
  assumes *p*≠*0*
  shows *proots-count p* (*ball 0 1*) = *proots-count* (*fcompose p q1 q2*) {*x*. *0* < *Im x*}
  ⟨*proof*⟩

lemma *proots-sphere-axis-eq*:
  defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
  assumes *p*≠*0*
  shows *proots-count p* (*sphere 0 1* − {− *1*}) = *proots-count* (*fcompose p q1 q2*) {*x*. *0* = *Im x*}
⟨*proof*⟩

lemma *proots-card-ball-plane-eq*:
  defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
  assumes *p*≠*0*
  shows *card* (*proots-within p* (*ball 0 1*)) = *card* (*proots-within* (*fcompose p q1 q2*)

$\{x.\ 0 < Im\ x\}$)
⟨*proof*⟩

lemma *proots-card-sphere-axis-eq*:
defines *q1*≡[:i,−*1*:] **and** *q2*≡[:i,*1*:]
assumes *p*≠*0*
shows *card* (*proots-within p* (*sphere 0 1* − {− *1*}))
= *card* (*proots-within* (*fcompose p q1 q2*) {*x. 0* = *Im x*})
⟨*proof*⟩

lemma *proots-uball-eq*:
fixes *z0*::*complex* **and** *r*::*real*
defines *q*≡[:*z0*, *of-real r*:]
assumes *p*≠*0* **and** *r*>*0*
shows *proots-count p* (*ball z0 r*) = *proots-count* (*p* ∘$_p$ *q*) (*ball 0 1*)
⟨*proof*⟩

lemma *proots-card-uball-eq*:
fixes *z0*::*complex* **and** *r*::*real*
defines *q*≡[:*z0*, *of-real r*:]
assumes *r*>*0*
shows *card* (*proots-within p* (*ball z0 r*)) = *card* (*proots-within* (*p* ∘$_p$ *q*) (*ball 0 1*))
⟨*proof*⟩

lemma *proots-card-usphere-eq*:
fixes *z0*::*complex* **and** *r*::*real*
defines *q*≡[:*z0*, *of-real r*:]
assumes *r*>*0*
shows *card* (*proots-within p* (*sphere z0 r*)) = *card* (*proots-within* (*p* ∘$_p$ *q*) (*sphere 0 1*))
⟨*proof*⟩

3.3 Combining two real polynomials into a complex one

definition *cpoly-of*:: *real poly* ⇒ *real poly* ⇒ *complex poly* **where**
cpoly-of pR pI = *map-poly of-real pR* + *smult* i (*map-poly of-real pI*)

lemma *cpoly-of-eq-0-iff* [*iff*]:
cpoly-of pR pI = *0* ⟷ *pR* = *0* ∧ *pI* = *0*
⟨*proof*⟩

lemma *cpoly-of-decompose*:
p = *cpoly-of* (*map-poly Re p*) (*map-poly Im p*)
⟨*proof*⟩

lemma *cpoly-of-dist-right*:
cpoly-of (*pR*∗*g*) (*pI*∗*g*) = *cpoly-of pR pI* ∗ (*map-poly of-real g*)
⟨*proof*⟩

lemma *poly-cpoly-of-real*:
  poly (*cpoly-of pR pI*) (*of-real x*) = *Complex* (*poly pR x*) (*poly pI x*)
⟨*proof*⟩

lemma *poly-cpoly-of-real-iff*:
shows *poly* (*cpoly-of pR pI*) (*of-real t*) =*0* ⟷ *poly pR t* = *0* ∧ *poly pI t*=*0*
⟨*proof*⟩

lemma *order-cpoly-gcd-eq*:
assumes *pR*≠*0* ∨ *pI*≠*0*
shows *order t* (*cpoly-of pR pI*) = *order t* (*gcd pR pI*)
⟨*proof*⟩

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

— 1 dimensional hyperplane
definition *unbounded-line*::′*a*::*real-vector* ⇒ ′*a* ⇒ ′*a set* **where**
  unbounded-line a b = ({*x*. ∃ *u*::*real*. *x*= (*1* − *u*) ∗$_R$ *a* + *u* ∗$_R$ *b*})

definition *proots-line-card*:: *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
proots-line-card p st tt = *card* (*proots-within p* (*open-segment st tt*))

definition *proots-unbounded-line-card*:: *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
proots-unbounded-line-card p st tt = *card* (*proots-within p* (*unbounded-line st tt*))

definition *proots-unbounded-line* :: *complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
proots-unbounded-line p st tt = *proots-count p* (*unbounded-line st tt*)

lemma *card-proots-open-segments*:
assumes *poly p st* ≠*0 poly p tt* ≠ *0*
shows *card* (*proots-within p* (*open-segment st tt*)) =
  (*let pc* = *pcompose p* [:*st*, *tt* − *st*:];
    pR = *map-poly Re pc*;
    pI = *map-poly Im pc*;
    g = *gcd pR pI*
  in changes-itv-smods 0 1 g (*pderiv g*)) (**is** *?L* = *?R*)
⟨*proof*⟩

lemma *unbounded-line-closed-segment*: *closed-segment a b* ⊆ *unbounded-line a b*
⟨*proof*⟩

lemma *card-proots-unbounded-line*:
assumes *st*≠*tt*
shows *card* (*proots-within 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 g (pderiv g))) (is ?L = ?R)
⟨proof⟩

lemma proots-unbounded-line:
  assumes st≠tt p≠0
  shows (proots-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 proots-unbounded-line-card-code[code]:
  proots-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 ''proots-unbounded-line-card fails due to invalid
hyperplanes.'')
                  (λ-. proots-unbounded-line-card p st tt))
  ⟨proof⟩

lemma proots-unbounded-line-code[code]:
  proots-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 ''proots-unbounded-line fails due to p=0'')
                    (λ-. proots-unbounded-line p st tt)
              else
                    Code.abort (STR ''proots-unbounded-line fails due to invalid
hyperplanes.'')
                    (λ-. proots-unbounded-line p st tt) )
  ⟨proof⟩
```

3.5 Checking if there a polynomial root on a closed segment

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

lemma *no-proots-line-code*[*code*]: *no-proots-line p st tt* = (*if poly p st* ≠*0* ∧ *poly p tt* ≠ *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 *proots-rectangle* ::*complex poly* ⇒ *complex* ⇒ *complex* ⇒ *nat* **where**
proots-rectangle p lb ub = *proots-count p* (*box lb ub*)

lemma *closed-segment-imp-Re-Im*:
fixes *x*::*complex*
assumes *x*∈*closed-segment lb ub*
shows *Re lb* ≤ *Re ub* ⟹ *Re lb* ≤ *Re x* ∧ *Re x* ≤ *Re ub*
Im lb ≤ *Im ub* ⟹ *Im lb* ≤ *Im x* ∧ *Im x* ≤ *Im ub*
⟨*proof*⟩

lemma *closed-segment-degen-complex*:
⟦*Re lb* = *Re ub*; *Im lb* ≤ *Im ub* ⟧
⟹ *x* ∈ *closed-segment lb ub* ⟷ *Re x* = *Re lb* ∧ *Im lb* ≤ *Im x* ∧ *Im x* ≤ *Im ub*
⟦*Im lb* = *Im ub*; *Re lb* ≤ *Re ub* ⟧
⟹ *x* ∈ *closed-segment lb ub* ⟷ *Im x* = *Im lb* ∧ *Re lb* ≤ *Re x* ∧ *Re x* ≤ *Re ub*
⟨*proof*⟩

lemma *complex-box-ne-empty*:
fixes *a b*::*complex*
shows
cbox a b ≠ {} ⟷ (*Re a* ≤ *Re b* ∧ *Im a* ≤ *Im b*)
box a b ≠ {} ⟷ (*Re a* < *Re b* ∧ *Im a* < *Im b*)
⟨*proof*⟩

lemma *proots-rectangle-code1*:
proots-rectangle p lb ub = (*if Re lb* < *Re ub* ∧ *Im lb* < *Im ub then*
if p≠*0 then*
if no-proots-line p lb (*Complex* (*Re ub*) (*Im lb*))
∧ *no-proots-line p* (*Complex* (*Re ub*) (*Im lb*)) *ub*

```
        ∧ no-proots-line p ub (Complex (Re lb) (Im ub))
        ∧ no-proots-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*: ((λ*x*. *Im* (*Ln* (*Complex a x*))) ⟶ *pi/2*) *at-top*
⟨*proof*⟩

lemma *Im-Ln-tendsto-at-bot*: ((λ*x*. *Im* (*Ln* (*Complex a x*))) ⟶ − *pi/2*) *at-bot*

⟨*proof*⟩

lemma *Re-winding-number-tendsto-part-circlepath*:
shows ((λ*r*. *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*::*complex poly*
defines *g* ≡ (λr. *poly pp o linepath* (−*r*) (*of-real r*))
assumes *lead-coeff pp=1* **and** *no-real-zero*:∀ *x*∈*proots pp*. *Im x*≠*0*
shows ((λr. *2*∗*Re* (*winding-number* (*g r*) *0*) + *cindex-pathE* (*g r*) *0*) ⟶ *0*) *at-top*
⟨*proof*⟩

lemma *proots-upper-cindex-eq*:
assumes *lead-coeff p=1* **and** *no-real-roots*:∀ *x*∈*proots p*. *Im x*≠*0*
shows *proots-upper p* =
(*degree p* − *cindex-poly-ubd* (*map-poly Im p*) (*map-poly Re p*)) */2*
⟨*proof*⟩

lemma *cindexE-roots-on-horizontal-border*:
fixes *a*::*complex* **and** *s*::*real*
defines *g*≡*linepath a* (*a* + *of-real s*)
assumes *pqr*:*p* = *q* ∗ *r* **and** *r-monic*:*lead-coeff r=1* **and** *r-proots*:∀ *x*∈*proots r*. *Im x=Im a*
shows *cindexE lb ub* (λt. *Im* ((*poly p* ∘ *g*) *t*) / *Re* ((*poly p* ∘ *g*) *t*)) =
cindexE lb ub (λt. *Im* ((*poly q* ∘ *g*) *t*) / *Re* ((*poly q* ∘ *g*) *t*))
⟨*proof*⟩

lemma *poly-decompose-by-proots*:
fixes *p* ::′*a*::*idom poly*
assumes *p*≠*0*
shows ∃ *q r*. *p* = *q* ∗ *r* ∧ *lead-coeff q=1* ∧ (∀ *x*∈*proots q*. *P x*) ∧ (∀ *x*∈*proots r*. ¬*P x*) ⟨*proof*⟩

lemma *proots-upper-cindex-eq*′:
assumes *lead-coeff p=1*
shows *proots-upper p* = (*degree p* − *proots-count p* {*x*. *Im x=0*}
− *cindex-poly-ubd* (*map-poly Im p*) (*map-poly Re p*)) */2*
⟨*proof*⟩

lemma *proots-within-upper-squarefree*:
assumes *rsquarefree p*
shows *card* (*proots-within p* {*x*. *Im x* >*0*}) = (*let*
pp = *smult* (*inverse* (*lead-coeff p*)) *p*;
pI = *map-poly Im pp*;
pR = *map-poly Re pp*;

```
        g = gcd pR pI
      in
        nat ((degree p − changes-R-smods g (pderiv g) − changes-R-smods pR
pI) div 2)
    )
⟨proof⟩

lemma proots-upper-code1[code]:
  proots-upper p =
    (if p ≠ 0 then
      (let pp=smult (inverse (lead-coeff p)) p;
        pI=map-poly Im pp;
        pR=map-poly Re pp;
        g = gcd pI pR
      in
       nat ((degree p − nat (changes-R-smods-ext g (pderiv g)) − changes-R-smods
pR pI) div 2)
      )
    else
      Code.abort (STR ''proots-upper fails when p=0.'') (λ-. proots-upper p))
⟨proof⟩

lemma proots-upper-card-code[code]:
  proots-upper-card p = (if p=0 then 0 else
    (let
        pf = p div (gcd p (pderiv p));
        pp = smult (inverse (lead-coeff pf)) pf;
        pI = map-poly Im pp;
        pR = map-poly Re pp;
        g = gcd pR pI
      in
        nat ((degree pf − changes-R-smods g (pderiv g) − changes-R-smods pR
pI) 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 proots-half ::complex poly ⇒ complex ⇒ complex ⇒ nat where
  proots-half p a b = proots-count p {w. Im ((w−a) / (b−a)) > 0}

lemma proots-half-empty:
  assumes a=b
  shows proots-half p a b = 0
⟨proof⟩
```

lemma *proots-half-proots-upper*:
  assumes *a≠b p≠0*
  shows *proots-half p a b= proots-upper (pcompose p [:a, (b−a):])*
⟨*proof*⟩

lemma *proots-half-code1*[*code*]:
  proots-half p a b = (if a≠b then
                    if p≠0 then proots-upper (p ∘$_p$ [: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 ⇒ complex ⇒ real ⇒ nat* **where**
  proots-ball p z0 r = proots-count p (ball z0 r)

— Roots counted WITHOUT multiplicity
definition *proots-ball-card* ::*complex poly ⇒ complex ⇒ real ⇒ 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 ≤ 0 then
                          0
                        else if p≠0 then
                         proots-upper (fcompose (p ∘$_p$ [: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 ≤ 0 ∨ p=0 then
                   0
                else
                   proots-upper-card (fcompose (p ∘$_p$ [: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 ⇒ complex ⇒ real ⇒ nat* **where**
  proots-sphere p z0 r = proots-count p (sphere z0 r)

— Roots counted WITHOUT multiplicity

definition *proots-sphere-card* ::*complex poly* ⇒ *complex* ⇒ *real* ⇒ *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*:])
0 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

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.