The Budan-Fourier Theorem and Counting Real Roots with Multiplicity

Wenda Li

September 4, 2018

Abstract

This entry is mainly about counting and approximating real roots (of a polynomial) with multiplicity. We have first formalised the Budan-Fourier theorem: given a polynomial with real coefficients, we can calculate sign variations on Fourier sequences to over-approximate the number of real roots (counting multiplicity) within an interval. When all roots are known to be real, the over-approximation becomes tight: we can utilise this theorem to count real roots exactly. It is also worth noting that Descartes’ rule of sign is a direct consequence of the Budan-Fourier theorem, and has been included in this entry. In addition, we have extended previous formalised Sturm’s theorem to count real roots with multiplicity, while the original Sturm’s theorem only counts distinct real roots. Compared to the Budan-Fourier theorem, our extended Sturm’s theorem always counts roots exactly but may suffer from greater computational cost.

Many problems in real algebraic geometry is about counting or approximating roots of a polynomial. Previous formalised results are mainly about counting distinct real roots (i.e. Sturm’s theorem in Isabelle/HOL [5, 2], HOL Light [4], PVS [9] and Coq [8]) and limited support for multiple real roots (i.e. Descartes’ rule of signs in Isabelle/HOL [3], HOL Light and ProofPower1). In comparison, this entry provides more comprehensive support for reasoning about multiple real roots.

The main motivation of this entry is to cope with the roots-on-the-border issue when counting complex roots [7, 6], but the results here should be beneficial to other developments.

Our proof of the Budan-Fourier theorem mainly follows Theorem 2.35 in the book by Basu et al. [1] and that of the extended Sturm’s theorem is inspired by Theorem 10.5.6 in Rahman and Schmeisser’s book [10].

1According to Freek Wiedijk’s "Formalising 100 Theorems" (http://www.cs.ru.nl/~freek/100/index.html)
1 Misc results for polynomials and sign variations

theory BF-Misc imports 
  HOL Computational-Algebra.Polynomial-Factorial 
  HOL Computational-Algebra.Fundamental-Theorem-Algebra 
  Sturm-Tarski.Sturm-Tarski
begin

1.1 Misc

lemma lead-coeff-pderiv:
  fixes p :: 'a::{comm-semiring-1,semiring-no-zero-divisors,semiring-char-0} poly 
  shows lead-coeff (pderiv p) = of-nat (degree p) * lead-coeff p 
  apply (auto simp:degree-pderiv coeff-pderiv) 
  apply (cases degree p) 
  by (auto simp add:coeff-eq-0)

lemma gcd-degree-le-min:
  assumes p≠0 q≠0 
  shows degree (gcd p q) ≤ min (degree p) (degree q) 
  by (simp add:assms(1) assms(2) dvd-imp-degree-le)

lemma lead-coeff-normalize-field:
  fixes p :: 'a::{field,semidom-divide-unit-factor} poly 
  assumes p≠0 
  shows lead-coeff (normalize p) = 1 
  using assms by (metis gcd.normalize-idem gcd-eq-0-iff lead-coeff-normalize-field)

lemma smult-normalize-field-eq:
  fixes p :: 'a::{field,semidom-divide-unit-factor} poly 
  shows p = smult (lead-coeff p) (normalize p) 
  proof (rule poly-eqI)
    fix n 
    have unit-factor (lead-coeff p) = lead-coeff p 
      by (metis dvd-field-iff is-unit-unit-factor unit-factor-0)
    then show coeff p n = coeff (smult (lead-coeff p) (normalize p)) n 
      by simp
  qed

lemma lead-coeff-gcd-field:
  fixes p q :: 'a::{field,semidom-divide-unit-factor,factorial-ring-gcd} poly 
  assumes p≠0 ∨ q≠0 
  shows lead-coeff (gcd p q) = 1 
  using assms by (metis gcd.normalize-idem gcd-eq-0-iff lead-coeff-normalize-field)

lemma poly-gcd-0-iff:
  poly (gcd p q) x = 0 ⟷ poly p x=0 ∧ poly q x=0 
  by (simp add:poly-eq-0-iff-dvd)
lemma order-multiplicity-eq:
assumes p≠0
shows order a p = multiplicity [:-a.|] p
by (metis assms multiplicity-eqI order-1 order-2)

lemma order-gcd:
assumes p≠0 q≠0
shows order x (gcd p q) = min (order x p) (order x q)
proof
have prime [:- x, 1:]
apply (auto simp add: prime-elem-linear-poly normalize-poly-def intro!:primeI)
by (simp add: pCons-one)
then show ?thesis
using assms
by (auto simp add: order-multiplicity-eq intro: multiplicity-gcd)
qed

1.2 More results about sign variations (i.e. changes

lemma changes-0[simp]: changes (0#xs) = changes xs
by (cases xs) auto

lemma changes-Cons: changes (x#xs) = (if filter (λx. x≠0) xs = [] then
0
else if x* hd (filter (λx. x≠0) xs) < 0 then
1 + changes xs
else changes xs)
apply (induct xs)
by auto

lemma changes-filter-eq:
changes (filter (λx. x≠0) xs) = changes xs
apply (induct xs)
by (auto simp add: changes-Cons)

lemma changes-filter-empty:
assumes filter (λx. x≠0) xs = []
shows changes xs = 0 changes (a#xs) = 0 using assms
apply (induct xs)
apply auto
by (metis changes-0 neq-Nil-conv)

lemma changes-append:
assumes xs≠ [] ∧ ys≠ [] → (last xs = hd ys ∧ last xs≠0)
shows changes (xs@ys) = changes xs + changes ys
using assms
proof (induct xs)
case Nil
then show ?case by simp
next
case (Cons a xs)
  have ?case when xs=[]
    using that Cons
    apply (cases ys)
    by auto
moreover have ?case when ys=[]
  using that Cons by auto
moreover have ?case when xs≠[] ys≠[]
proof −
  have filter (λx. x ≠ 0) xs ≠[]
    using that Cons
    apply auto
    by (metis (mono-tags, lifting) filter.simps(1) filter.simps(2) filter-append snoc-eq-iff-butlast)
  then have changes (a ≠ xs @ ys) = changes (a ≠ xs) + changes ys
    apply (subst (1 2) changes-Cons)
    using that Cons by auto
  then show ?thesis by auto
qed
ultimately show ?case by blast
qed

lemma changes-drop-dup:
  assumes xs≠ [] ys≠ [] → last xs=hd ys
  shows changes (xs@ys) = changes (xs@tl ys)
  using assms
proof (induct zs)
case Nil
  then show ?case by simp
next
case (Cons a xs)
  have ?case when ys=[]
    using that by simp
moreover have ?case when ys≠[] xs=[]
  using that Cons
  apply auto
  by (metis changes.simps(3) list.exhaust-sel not-square-less-zero)
moreover have ?case when ys≠[] xs≠[]
proof −
  define ts ts' where ts = filter (λx. x ≠ 0) (xs @ ys)
    and ts' = filter (λx. x ≠ 0) (xs @ tl ys)
  have (ts = [] ←→ ts' = []) ∧ hd ts = hd ts'
    proof (cases filter (λx. x ≠ 0) xs = [])
      case True
      then have last xs = 0 using (xs≠[])
        by (metis (mono-tags, lifting) append-butfast-last-id append-is-Nil-cone
          filter.simps(2) filter-append list.simps(3))
    then have hd ys=0 using Cons(3)[rule-format, OF (ys≠[]) (xs≠[])] by auto
then have filter (\(\lambda x. \; x \neq 0\)) \(ys = \) filter (\(\lambda x. \; x \neq 0\)) (\(tl \; ys\))

by (metis (mono-tags, lifting) filter.simps(2) list.exhaust-set that(1))

then show \(?thesis unfolding ts-def ts'-def by auto\)

next

case False

then show \(?thesis unfolding ts-def ts'-def by auto\)

qed

moreover have \(\text{changes} \; (xs @ ys) = \text{changes} \; (xs @ tl \; ys)\)

apply (rule Cons(1))

using that Cons(3) by auto

moreover have \(\text{changes} \; (a # xs @ ys) = (\text{if} \; ts = [] \; \text{then} \; 0 \; \text{else} \; \text{if} \; a * \text{hd} \; ts < 0 \; \text{then} \; 1 + \text{changes} \; (xs @ ys) \; \text{else} \; \text{changes} \; (xs @ ys))\)

using \(\text{changes-Cons[of} \; a \; xs \; @ \; ys \; \text{folded} \; ts-def\] .

moreover have \(\text{changes} \; (a # xs \; @ \; tl \; ys) = (\text{if} \; ts' = [] \; \text{then} \; 0 \; \text{else} \; \text{if} \; a * \text{hd} \; ts < 0 \; \text{then} \; 1 + \text{changes} \; (xs \; @ \; tl \; ys) \; \text{else} \; \text{changes} \; (xs \; @ \; tl \; ys))\)

using \(\text{changes-Cons[of} \; a \; xs \; @ \; tl \; ys \; \text{folded} \; ts'-def\] .

ultimately show \(?thesis by auto\)

qed

ultimately show \(?case by blast\)

qed

1.3 Induction on polynomial roots

lemma poly-root-induct-alt [case-names 0 no-proots root]:

fixes \(p :: \alpha :: \text{idom poly}\)

assumes \(Q 0\)

assumes \(\forall p. \; (\forall a. \; \text{poly} \; p \; a \neq 0) \implies Q \; p\)

assumes \(\forall a. \; Q \; p \implies Q \; ([: -a, 1:] * \; p)\)

shows \(Q \; p\)

proof (induction degree \(p\) arbitrary; \(p\) rule: less-induct)

case (less \(p\))

have \(?case when \; p=0 using (Q 0): that by auto\)

moreover have \(?case when \; \exists a. \; \text{poly} \; p \; a = 0\)

using assms(2) that by blast

moreover have \(?case when \; \exists a. \; \text{poly} \; p \; a = 0 \; p\neq 0\)

proof –

obtain \(a\) where \(\text{poly} \; p \; a = 0\) using \(\exists a. \; \text{poly} \; p \; a = 0\) by auto

then obtain \(q\) where \(pq:p=[: -a, 1:] * \; q\) by (meson dvdE poly-eq-0-iff-dvd)

then have \(q\neq 0\) using \((p\neq 0)\) by auto

then have \(\text{degree} \; q < \text{degree} \; p\) unfolding \(pq\) by (subst degree-mult-eq,auto)

then have \(Q \; q\) using less by auto

then show \(?case using assms(3) unfolding \(pq\) by auto\)

qed

ultimately show \(?case by auto\)

qed
1.4 Polynomial roots / zeros

definition proots-within::'a::comm-semiring-0 poly ⇒ 'a set ⇒ 'a set where
    proots-within p s = {x∈s. poly p x=0}

abbreviation proots::'a::comm-semiring-0 poly ⇒ 'a set where
    proots p ≡ proots-within p UNIV

lemma proots-def: proots p = {x. poly p x=0}
  unfolding proots-within-def by auto

lemma proots-within-empty[simp]:
    proots-within p {} = {} unfolding proots-within-def by auto

lemma proots-within-0[simp]:
    proots-within 0 s = s unfolding proots-within-def by auto

lemma proots-within-intro[simp]:
    poly p x=0 ⇒ x∈s ⇒ x∈proots-within p s
  unfolding proots-within-def by auto

lemma proots-within-iff[simp]:
    x∈proots-within p s ⇔ poly p x=0 ∧ x∈s
  unfolding proots-within-def by auto

lemma proots-within-union:
    proots-within p A ∪ proots-within p B = proots-within p (A ∪ B)
  unfolding proots-within-def by auto

lemma proots-within-times:
    fixes s::'a::{semiring-no-zero-divisors,comm-semiring-0} set
    shows proots-within (p*q) s = proots-within p s ∪ proots-within q s
  unfolding proots-within-def by auto

lemma proots-within-gcd:
    fixes s::'a::factorial-ring-gcd set
    shows proots-within (gcd p q) s = proots-within p s ∩ proots-within q s
  unfolding proots-within-def by (auto simp add: poly-eq-0-iff-dvd)

lemma proots-within-inter:
    NO-MATCH UNIV s ⇒ proots-within p s = proots p ∩ s
  unfolding proots-within-def by auto

lemma proots-within-proots[simp]:
    proots-within p s ⊆ proots p
  unfolding proots-within-def by auto

lemma finite-proots[simp]:
    fixes p :: 'a::idom poly
shows \( p \neq 0 \implies \text{finite } (\text{proots-within } p \ s) \)

unfolding proots-within-def using poly-roots-finite by fast

lemma proots-within-pCons-1-iff:
fixes \( a :: 'a :: \text{idom} \)
shows proots-within \([-a, 1] s = (\text{if } a \in s \text{ then } \{a\} \text{ else } \{\})\)
proots-within \([a, -1] s = (\text{if } a \in s \text{ then } \{a\} \text{ else } \{\})\)
by (cases \( a \in s \), auto)

lemma proots-within-uminus[simp]:
fixes \( p :: 'a :: \text{comm-ring poly} \)
shows proots-within \((-p) s = \text{proots-within } p s\)
by auto

lemma proots-within-smult:
fixes \( a :: 'a :: \{\text{semiring-no-zero-divisors, comm-semiring-0}\} \)
assumes \( a \neq 0 \)
shows proots-within \((\text{smult } a \ p) s = \text{proots-within } p s\)
unfolding proots-within-def using assms by auto

1.5 Polynomial roots counting multiplicities.

definition proots-count::\('a :: \text{idom poly } \Rightarrow 'a \text{ set } \Rightarrow \text{nat}\) where
proots-count \( p \ s = (\sum r \in \text{proots-within } p \ s. \text{order } r p)\)

lemma proots-count-emtpy[simp]:
proots-count \( p \ \{\} = 0\)
unfolding proots-count-def by auto

lemma proots-count-times:
fixes \( s :: 'a :: \text{idom set} \)
assumes \( p \cdot q \neq 0 \)
shows proots-count \((p \cdot q) s = \text{proots-count } p s + \text{proots-count } q s\)
proof
  define \( pts \) where \( pts = \text{proots-within } p s\)
  define \( qts \) where \( qts = \text{proots-within } q s\)
  have [simp]: finite \( pts \) finite \( qts \)
    using \((p \cdot q \neq 0)\)
    unfolding pts-def qts-def by auto
  have \((\sum r \in pts \cup qts. \text{order } r p) = (\sum r \in pts. \text{order } r p)\)
  proof
    (rule \text{comm-monoid-add-class.sum.mono-neutral-cong-right,simp-all})
    show \( \forall i \in pts \cup qts - pts. \text{order } i p = 0 \)
      unfolding pts-def qts-def proots-within-def using order-root by fastforce
  qed
  moreover have \((\sum r \in pts \cup qts. \text{order } r q) = (\sum r \in qts. \text{order } r q)\)
  proof
    (rule \text{comm-monoid-add-class.sum.mono-neutral-cong-right,simp-all})
    show \( \forall i \in pts \cup qts - qts. \text{order } i q = 0 \)
      unfolding pts-def qts-def proots-within-def using order-root by fastforce
  qed
  ultimately show \(?thesis\)
    unfolding proots-count-def
    apply (simp add:proots-within-times order-mult[OF \( p \cdot q \neq 0 \)] sum.distrib)
apply (fold pts-def qts-def) by auto

qed

lemma proots-count-power-n-n:
shows \( \text{proots-count} \left( \left[ -a, 1 \right] \right)^n \ s = (\text{if } a \in s \land n > 0 \text{ then } n \text{ else } 0) \)
proof –
  have \( \text{proots-within} \left( \left[ -a, 1 \right] \right)^n s = (\text{if } a \in s \land n > 0 \text{ then } \{a\} \text{ else } \{\}) \)
  unfolding proots-within-def by auto
  thus \( ?\text{thesis} \) unfolding proots-count-def using order-power-n-n by auto
qed

lemma degree-proots-count:
fixes \( p :: \text{complex poly} \)
shows \( \text{degree } p = \text{proots-count } p \ UNIV \)
proof (induct \( \text{degree } p \) arbitrary: \( p \))
  case 0
  then obtain \( c \) where \( \text{c-def: } p = \left[ :c: \right] \) using degree-eq-zeroE by auto
  then show \( ?\text{case} \) unfolding proots-count-def apply (cases \( c = 0 \))
    by (auto intro!: sum.infinite simp add: infinite-UNIV-char-0 order-0I)
  next
  case (Suc \( n \))
  then have \( \text{degree } p \neq 0 \text{ and } p \neq 0 \) by auto
  obtain \( z \) where \( \text{poly } p z = 0 \)
    using Fundamental-Theorem-Algebra.fundamental-theorem-of-algebra \( \langle \text{degree } p \neq 0 \rangle \) constant-degree \( \text{of } p \)
    by auto
  define \( \text{onez where onez=}\left[ :\mathbf{z},1 : \right] \)
  have \( \text{[simp]: onez}\neq0 \text{ degree } \text{onez} = 1 \) unfolding onez-def by auto
  obtain \( q \) where \( \text{q-def: } p = \text{onez } \ast \ q \)
    using poly-eq-0-iff-dvd \( \langle \text{poly } p z = 0 \rangle \) dvdE unfolding onez-def by blast
  hence \( q \neq 0 \) using \( \langle p \neq 0 \rangle \) by auto
  hence \( n=\text{degree } q \) using degree-mult-eq[of onez q] \( \langle \text{Suc } n = \text{degree } p \rangle \)
    apply (fold q-def)
    by auto
  hence \( \text{degree } q = \text{proots-count } q \ UNIV \) using Suc.hyps(1) by simp
  moreover have \( \text{Suc } 0 = \text{proots-count } \text{onez } UNIV \)
    unfolding onez-def using proots-count-power-n-n[of z 1 UNIV]
    by auto
  ultimately show \( ?\text{case} \)
    unfolding q-def using degree-mult-eq[of onez q] proots-count-times[of onez q UNIV] \( \langle q \neq 0 \rangle \)
    by auto
qed

lemma proots-count-smult:
fixes \( a ::'a::\{\text{semiring-no-zero-divisors, idom}\} \)
assumes \( a \neq 0 \)
shows \(\text{proots-count} (\text{smult} a p) s = \text{proots-count} p s\)

proof (cases \(p=0\))
  case True
  then show ?thesis by auto
next
case False
then show ?thesis unfolding proots-count-def
  using order-smult[OF assms] proots-within-smult[OF assms] by auto
qed

lemma proots-count-pCons-1-iff:
  fixes \(a::'a::idom\)
  shows \(\text{proots-count} [:-a, 1:] s = (\text{if } a \in s \text{ then } 1 \text{ else } 0)\)
  unfolding proots-count-def by (cases \(a \in s\), auto simp add: proots-within-pCons-1-iff order-power-n-n[of - 1, simplified])

lemma proots-count-uminus[simp]:
  \(\text{proots-count} (- p) s = \text{proots-count} p s\)
  unfolding proots-count-def by simp

lemma card-proots-within-leq:
  assumes \(p \neq 0\)
  shows \(\text{proots-count} p s \geq \text{card} (\text{proots-within} p s)\) using assms
  proof (induct rule: poly-root-induct[of -\(\lambda x. x \in s\)])
  case 0
  then show ?case unfolding proots-within-def proots-count-def by auto
next
case (no-roots p)
  then have \(\text{proots-within} p s = {}\) by auto
  then show ?case unfolding proots-count-def by auto
next
case (root a p)
  have \(\text{card} (\text{proots-within} [:- a, 1:] * p) s \leq \text{card} (\text{proots-within} [:- a, 1:] s) + \text{card} (\text{proots-within} p s)\)
    unfolding proots-within-times by (auto simp add: card-Un-le)
  also have ... \(\leq 1 + \text{proots-count} p s\)
  proof
    have \(\text{card} (\text{proots-within} [:- a, 1:] s) \leq 1\)
      proof (cases \(a \in s\))
        case True
        then have \(\text{proots-within} [:- a, 1:] s = \{a\}\) by auto
        then show ?thesis by auto
      next
case False
      then have \(\text{proots-within} [:- a, 1:] s = {}\) by auto
      then show ?thesis by auto
  qed
qed
moreover have \( \text{card} (\text{proots-within } p \ s) \leq \text{proots-count } p \ s \)
apply (rule root.hyps)
using root by auto
ultimately show \(?\text{thesis}\) by auto
qed
also have ... \( \text{proots-count } ([:- a,1:] \ast p) \ s \)
apply (subst proots-count-times)
subgoal by (metis mult-eq-0-iff pCons-eq-0-iff root.prems zero-neq-one)
using root by (auto simp add:proots-count-pCons-1-iff)
finally have \( \text{card} (\text{proots-within } ([:- a,1:] \ast p) \ s) \leq \text{proots-count } ([:- a,1:] \ast p) \ s \).
then show \(?\text{case}\)
by (metis (no-types, hide-lams) add.inverse-inverse add.inverse-neutral minus-pCons

\multminusleft \text{proots-count-uminus} \text{proots-within-uminus})
qed

lemma proots-count-leq-degree:
assumes \( p \neq 0 \)
shows \( \text{proots-count } p \ s \leq \text{degree } p \) using assms
proof (induct rule:poly-root-induct[of \(-\lambda x. x\in s\)])
case 0
then show \(?\text{case}\) by auto
next
case (no-roots \( p \))
then have \( \text{proots-within } p \ s = \{\} \) by auto
then show \(?\text{case}\) unfolding \text{proots-count-def} by auto
next
case (root \( a \) \( p \))
have \( \text{proots-count } ([:a, - 1:] \ast p) \ s = \text{proots-count } [:a, - 1:] \ s + \text{proots-count } p \ s \)
apply (subst proots-count-times)
using root by auto
also have ... \( = 1 + \text{proots-count } p \ s \)
proof –
have \( \text{proots-count } [:a, - 1:] \ s = 1 \)
by (metis (no-types, lifting) add.inverse-inverse add.inverse-neutral minus-pCons

\text{proots-count-pCons-1-iff} \text{proots-count-uminus} \text{root.hyps}(1))
then show \(?\text{thesis}\) by auto
qed
also have ... \( \leq \text{degree } ([:a, - 1:] \ast p) \)
apply (subst degree-mult-eq)
subgoal by auto
subgoal using root by auto
subgoal using root by (simp add: \( \langle p \neq 0 \rangle \))
done
finally show \(?\text{case}\).
qed
lemma proots-count-union-disjoint:
assumes \( A \cap B = \{\} \) \( p \neq 0 \)
shows \( \text{proots-count}_p (A \cup B) = \text{proots-count}_p A + \text{proots-count}_p B \)
unfolding proots-count-def
apply (subst proots-within-union[symmetric])
apply (subst sum.union-disjoint)
using assms by auto
end

2 Budan-Fourier theorem

theory Budan-Fourier imports 
  BF-Misc
begin

The Budan-Fourier theorem is a classic result in real algebraic geometry to over-approximate real roots of a polynomial (counting multiplicity) within an interval. When all roots of the polynomial are known to be real, the over-approximation becomes tight – the number of roots are counted exactly. Also note that Descartes’ rule of sign is a direct consequence of the Budan-Fourier theorem.


2.1 More results related to \text{sign-r-pos}

lemma sign-r-pos-nzero-right:
assumes \( \forall x. c < x \land x \leq d \rightarrow \text{poly } p x \neq 0 \) and \( c < d \)
shows if \( \text{sign-r-pos } p c \) then \( \text{poly } p d > 0 \) else \( \text{poly } p d < 0 \)
proof (cases sign-r-pos p c)
case True
then obtain \( d' \) where \( d' > c \) and \( d'-\text{pos} \forall y. y > c \land y < d' 
\rightarrow 0 < \text{poly } p y \)
unfolding sign-r-pos-def eventually-at-right by auto
have False when \( \neg \text{poly } p d > 0 \)
proof
have \( \exists x > (c + \min d d') / 2. x < d \land \text{poly } p x = 0 \)
apply (rule poly-IVT-neg)
using \( d' > c \) that nzero[rule-format,of d,simplified]
by (auto intro:d'-pos[rule-format])
then show False using nzero \( c < d' \) by auto
qed
then show \( ?\text{thesis using True by auto} \)
next
case False
then have \( \text{sign-r-pos} \ (\neg p) \ c \)
  using \( \text{sign-r-pos-minus} \ [\text{of } p \ c] \) \( \text{nzero} \ [\text{rule-format, of } d, simplified] \) \( \langle c < d \rangle \)
by fastforce
then obtain \( d' \) where \( d' > c \) and \( d' \neg:\forall y. y < d' \rightarrow 0 > poly \ p \ y \)
unfolding \( \text{sign-r-pos-def} \) eventually-at-right by auto
have \( \text{False} \) when \( \neg poly \ p \ d' < 0 \)
proof
  have \( \exists x > (c + \min d \ d') / 2. \ x < d \land poly \ p \ x = 0 \)
  apply (rule poly-IVT-pos)
  using \( d' > c \) \( \langle c < d \rangle \) that \( \text{nzero} \ [\text{rule-format, of } d, simplified] \)
  by (auto intro: \( d' \neg \) [rule-format])
  then show \( \text{False} \) using \( \text{nzero} \ [\langle c < d' \rangle \) by auto
qed
then show \( \text{thesis} \) using \( \text{False} \) by auto
qed

lemma \( \text{sign-r-pos-at-left} \):
assumes \( p \neq 0 \)
says if even \( (\text{order } c \ p) \leftrightarrow \text{sign-r-pos} \ p \ c \) then eventually \( \lambda x. \ poly \ p \ x > 0 \) (at-left \( c \))
else eventually \( \lambda x. \ poly \ p \ x < 0 \) (at-left \( c \))
  using \( \text{assms} \)
proof (induct \( p \) rule: \( \text{poly-root-induct-alt} \))
case \( 0 \)
  then show \( ? \) case by simp
next
case \( \text{no-proots } p \)
then have \( [\text{simp}]: \text{order } c \ p = 0 \) using \( \text{order-root} \) by blast
have \( ? \) case when \( \text{poly } p \ c > 0 \)
proof
  have \( \forall F x \in \text{at } c. \ 0 < \text{poly } p \ x \)
    using that
    by (metis \( \text{no-types, lifting} \) \( \text{less-linear no-proots, hyps not-eventuallyD} \)
      \( \text{poly-IVT-neg poly-IVT-pos} \))
  then have \( \forall F x \in \text{at-left } c. \ 0 < \text{poly } p \ x \)
    using eventually-at-split by blast
  moreover have \( \text{sign-r-pos } p \ c \) using \( \text{sign-r-pos-rec} \ [OF \ \langle p \neq 0 \rangle] \) that by auto
  ultimately show \( \text{thesis} \) by simp
qed
moreover have \( ? \) case when \( \text{poly } p \ c < 0 \)
proof
  have \( \forall F x \in \text{at } c. \ \text{poly } p \ x < 0 \)
    using that
    by (metis \( \text{no-types, lifting} \) \( \text{less-linear no-proots, hyps not-eventuallyD} \)
      \( \text{poly-IVT-neg poly-IVT-pos} \))
  then have \( \forall F x \in \text{at-left } c. \ \text{poly } p \ x < 0 \)
    using eventually-at-split by blast
  moreover have \( \neg \text{sign-r-pos } p \ c \) using \( \text{sign-r-pos-rec} \ [OF \ \langle p \neq 0 \rangle] \) that by auto
  ultimately show \( \text{thesis} \) by simp
qed
ultimately show \( \text{?case using no-roots(1)[of c]} \) by \text{argo}

next

\begin{verbatim}
case (root a p)
define aa where aa=[:-a,1:]
have [simp]:aa\#0 p\#0 using [:- a, 1:] * p \# 0 unfolding aa-def by auto
have ?case when c>a
  proof
    have ?thesis = (if even (order c p) = sign-r-pos p c
                   then \( \forall F \) x in at-left c. 0 < poly (aa * p) x
                   else \( \forall F \) x in at-left c. poly (aa * p) x < 0)
    proof
      have order c aa=0 unfolding aa-def using order-0I that by force
      then have even (order c (aa * p)) = even (order c p)
        by (subst order-mult) auto
      moreover have sign-r-pos aa c unfolding aa-def using that
        by (auto simp: sign-r-pos-rec)
      then have sign-r-pos (aa * p) c = sign-r-pos p c
        by (subst sign-r-pos-mult) auto
      ultimately show ?thesis
        by (fold aa-def) auto
    qed
  also have ... = (if even (order c p) = sign-r-pos p c
                   then \( \forall F \) x in at-left c. 0 < poly x
                   else \( \forall F \) x in at-left c. 0 > poly x)
  proof
    have \( \forall F \) x in at-left c. 0 < poly aa x
      apply (simp add:aa-def)
      using that eventually-at-left-field by blast
    then have (\( \forall F \) x in at-left c. 0 < poly (aa * p) x) \iff (\forall F \) x in at-left c. 0 < poly p x)
      (\( \forall F \) x in at-left c. 0 > poly (aa * p) x) \iff (\forall F \) x in at-left c. 0 > poly p x)
      (apply auto)
      by (erule (1) eventually-elim2,simp add: zero-less-mult-iff mult-less-0-iff)+
    then show ?thesis by simp
  qed
also have ... using root.hyps by simp
finally show ?thesis .
qed
moreover have ?case when c<a
proof
  have ?thesis = (if even (order c p) = sign-r-pos p c
                   then \( \forall F \) x in at-left c. poly (aa * p) x < 0
                   else \( \forall F \) x in at-left c. 0 < poly (aa * p) x)
  proof
    have order c aa=0 unfolding aa-def using order-0I that by force
    then have even (order c (aa * p)) = even (order c p)
\end{verbatim}
by (subst order-mult) auto
moreover have \(\neg\ \text{sign-r-pos} \ a \ a \ c\)
  unfolding aa-def using that
  by (auto simp: sign-r-pos-rec)
then have \(\text{sign-r-pos} \ (a * p) \ c = (\neg\ \text{sign-r-pos} \ p \ c)\)
  by (subst sign-r-pos-mult) auto
ultimately show \(?thesis\)
  by (fold aa-def) auto
qed
also have \(\ldots = (\text{if even} \ (\text{order} \ c \ p) = \text{sign-r-pos} \ p \ c \)
  then \(\forall F \ x \in \text{at-left} \ c. \ 0 < \text{poly} \ p \ x\)
  else \(\forall F \ x \in \text{at-left} \ c. \ \text{poly} \ p \ x < 0\)\)
proof –
have \(\forall F \ x \in \text{at-left} \ c. \ \text{poly} \ a \ a \ x < 0\)
  apply (simp add: aa-def)
  using that \(\text{Eventually-at-filter} \ 
  \text{by fastforce}\)
then have \(\forall F \ x \in \text{at-left} \ c. \ 0 < \text{poly} \ (a * p) \ x \leftrightarrow \forall F \ x \in \text{at-left} \ c. \ \text{poly} \ p \ x < 0\)
  apply auto
  by (erule (1) eventually-elim2, simp add: zero-less-mult-iff mult-less-0-iff)+
then show \(?thesis\) by simp
qed
also have \(\ldots \text{using root.hyps by simp}\)
finally show \(?thesis\).
qed
moreover have \(?case\ when \ c=a\)
proof –
have \(?thesis\) = \(\text{if even} \ (\text{order} \ c \ p) = \text{sign-r-pos} \ p \ c \)
  then \(\forall F \ x \in \text{at-left} \ c. \ 0 > \text{poly} \ (a * p) \ x\)
  else \(\forall F \ x \in \text{at-left} \ c. \ \text{poly} \ (a * p) \ x > 0\)
proof –
have \(\text{order} \ c \ a a = 1\) unfolding aa-def using that
  by (metis order-power-n-n power-one-right)
then have \(\text{even} \ (\text{order} \ c \ (a * p)) = \text{odd} \ (\text{order} \ c \ p)\)
  by (subst order-mult) auto
moreover have \(\text{sign-r-pos} \ a a \ c\)
  unfolding aa-def using that
  by (auto simp: sign-r-pos-rec pderiv-pCons)
then have \(\text{sign-r-pos} \ (a * p) \ c = \text{sign-r-pos} \ p \ c\)
  by (subst sign-r-pos-mult) auto
ultimately show \(?thesis\)
  by (fold aa-def) auto
qed
also have \(\ldots = (\text{if even} \ (\text{order} \ c \ p) = \text{sign-r-pos} \ p \ c \)
  then \(\forall F \ x \in \text{at-left} \ c. \ 0 < \text{poly} \ p \ x\)
  else \(\forall F \ x \in \text{at-left} \ c. \ \text{poly} \ p \ x < 0\)\)
proof –
have $\forall F \ x \ in \ \text{at-left} \ c. \ 0 > \text{poly} \ aa \ x$
apply (simp add:aa-def)
using that by (simp add: eventually-at-filter)
then have $(\forall F \ x \ in \ \text{at-left} \ c. \ 0 < \text{poly} \ (aa * p) \ x) \iff (\forall F \ x \ in \ \text{at-left} \ c. \ 0 > \text{poly} \ p \ x)$
apply auto
by (erule (1) eventually elimin2, simp add: zero-less-mult-iff mult-less-0-iff)+
then show ?thesis by simp
qed
also have ... using root.hyps by simp
finally show ?thesis.
qed
ultimately show ?case by argo
qed

lemma sign-r-pos-nzero-left:
assumes nzero: $\forall x. \ d \leq x \land x < c \longrightarrow \text{poly} \ p \ x \neq 0$ \text{ and } d < c
shows if even \ (order c p) $\longleftrightarrow$ sign-r-pos p c then \text{poly} p d > 0 else \text{poly} p d < 0
proof (cases even \ (order c p) $\longleftrightarrow$ sign-r-pos p c)
case True
then have eventually $(\lambda x. \ \text{poly} \ p \ x > 0) \ (\text{at-left} \ c)$
using nzero[rule-format,of d,simplified] \ (d < c) sign-r-pos-at-left
by (simp add: order-root)
then obtain $d'$ where $d' < c$ \text{ and } d'-pos: $\forall y > d'. \ y < c \longrightarrow 0 < \text{poly} \ p \ y$
unfolding eventually-at-left by auto
have False when $\neg \ \text{poly} \ p \ d > 0$
proof
have $\exists x > d. \ x < (\ c + \ \text{max} \ d \ d') / 2 \land \ \text{poly} \ p \ x = 0$
apply (rule poly-IVT-pos)
using (d' < c) \ (c > d) that nzero[rule-format,of d,simplified]
by (auto intro:d'-pos[rule-format])
then show False using nzero \ (c > d') by auto
qed
then show ?thesis using True by auto
next
case False
then have eventually $(\lambda x. \ \text{poly} \ p \ x < 0) \ (\text{at-left} \ c)$
using nzero[rule-format,of d,simplified] \ (d < c) sign-r-pos-at-left
by (simp add: order-root)
then obtain $d'$ where $d' < c$ \text{ and } d'-neg: $\forall y > d'. \ y < c \longrightarrow 0 > \text{poly} \ p \ y$
unfolding eventually-at-left by auto
have False when $\neg \ \text{poly} \ p \ d < 0$
proof
have $\exists x > d. \ x < (\ c + \ \text{max} \ d \ d') / 2 \land \ \text{poly} \ p \ x = 0$
apply (rule poly-IVT-neg)
using (d' < c) \ (c > d) that nzero[rule-format,of d,simplified]
by (auto intro:d'-neg[rule-format])
then show False using \( nzero \langle c > d \rangle \) by auto 
qed 
then show \( \theta \) using False by auto 
qed 

2.2 Fourier sequences 

function \( pders :: \text{real poly} \Rightarrow \text{real poly list} \) where 
\( pders p = (\text{if } p = 0 \text{ then } [] \text{ else } \text{Cons } p (pders (pderiv p))) \) 
by auto 

termination 
apply (relation measure \( \lambda p. \text{if } p = 0 \text{ then } 0 \text{ else } \text{degree } p + 1 \)) 
by (auto simp:degree-pderiv pderiv-eq-0-iff)

declare pders.simps[simp del]

lemma set-pders-nzero: 
assumes \( p \neq 0 \) \( q \in \text{set} (pders p) \) 
shows \( q \neq 0 \) 
using assms 
proof (induct p rule:pders.induct) 
case (1 p) 
then have \( q \in \text{set} (p \neq \text{pders (pderiv p)}) \) 
by (simp add: pders.simps) 
then have \( q = p \lor q \in \text{set} (pders (pderiv p)) \) by auto 
moreover have \( ? \) case when \( q = p \) 
using that \( p \neq 0 \) by auto 
moreover have \( ? \) case when \( q \in \text{set} (pders (pderiv p)) \) 
using 1 pders.simps by fastforce 
ultimately show \( ? \) case by auto 
qed 

2.3 Sign variations for Fourier sequences 

definition \( \text{changes-it}\_\text{v-der} :: \text{real} \Rightarrow \text{real} \Rightarrow \text{real poly} \Rightarrow \text{int} \) where 
\( \text{changes-it}\_\text{v-der} a b p = (\text{let } ps = \text{pders p in changes-poly-at ps a} - \text{changes-poly-at ps b}) \)

definition \( \text{changes-gt-der} :: \text{real} \Rightarrow \text{real poly} \Rightarrow \text{int} \) where 
\( \text{changes-gt-der} a p = \text{changes-poly-at (pders p) a} \)

definition \( \text{changes-le-der} :: \text{real} \Rightarrow \text{real poly} \Rightarrow \text{int} \) where 
\( \text{changes-le-der} b p = (\text{degree p} - \text{changes-poly-at (pders p) b}) \)

lemma \( \text{changes-poly-pos-inf-pders}[simp]:\text{changes-poly-pos-inf (pders p)} = 0 \) 
proof (induct degree p arbitrary:p) 
case 0 
then obtain a where \( p = [a:] \) using degree-eq-zeroE by auto 
then show \( ? \) case 
apply (cases a=0)
by (auto simp: changes-poly-pos-inf-def pders.simps)

next

  case (Suc x)

  then have pderiv \( p \neq 0 \) \( p \neq 0 \) \textbf{using} pderiv-eq-0-iff \textbf{by} force+

  define \( \text{ps} \) where \( \text{ps} = \text{pders} (\text{pderiv} \ (pderiv \ p)) \)

  have \( \text{ps} = p \# \) \( \text{pderiv} \) \( p \# \) \( \text{ps} \) \( \text{pders} \ (\text{pderiv} \ p) = p \) \( \text{pderiv} \ p \# \) \( \text{ps} \) \( \text{pders} \ (\text{pderiv} \ (pderiv \ p)) \)

  unfolding ps-def \textbf{by} (simp-all add: \( p \neq 0 \), \( \text{pderiv} \ p \neq 0 \), pders.simps)

  have \( \text{hyps} = \text{changes-poly-pos-inf} \ (\text{pders} \ (\text{pderiv} \ p)) \) \( = 0 \)

  apply (rule Suc(1))

  using \( \text{Suc} x = \) \( \text{degree} \ p \) \textbf{by} (metis degree-pderiv diff-Suc-1)

  moreover have sgn-pos-inf \( p \) \( * \) \( \text{sgn-pos-inf} \) \( \text{pderiv} \ p \) \( > 0 \)

  unfolding sgn-pos-inf-def lead-coeff-pderiv

  apply (simp add: algebra-simps sgn-mult)

  using Suc.hyps(2) \( p \neq 0 \) \textbf{by} linarith

  ultimately show \( ?\text{case unfolding changes-poly-pos-inf-def ps by auto} \)

  qed

lemma \( \text{changes-poly-neg-inf-pders[simp]} : \text{changes-poly-neg-inf} \ (\text{pders} \ p) = \) \( \text{degree} \ p \)

proof (induct \( \text{degree} \ p \) arbitrary: \( p \))

  case 0

  then obtain \( a \) where \( p = [: a :] \) \textbf{using} degree-eq-zeroE \textbf{by} auto

  then show \( ?\text{case unfolding changes-poly-neg-inf-def by (auto simp: pders.simps)} \)

next

  case (Suc x)

  then have pderiv \( p \neq 0 \) \( p \neq 0 \) \textbf{using} pderiv-eq-0-iff \textbf{by} force+

  then have changes-poly-neg-inf \( \text{pders} \ (\text{pderiv} \ p) \)

  = changes-poly-neg-inf \( p \# \text{pders} \ (\text{pderiv} \ p) \)

  unfolding pders-def \textbf{by} (simp: changes-poly-neg-inf-def)

  also have \( \ldots = 1 + \) \text{changes-poly-neg-inf} \( \text{pders} \ (\text{pderiv} \ p) \)

  proof

    have sgn-pos-inf \( p \) \( * \) \( \text{sgn-pos-inf} \) \( \text{pderiv} \ p \) \( < 0 \)

    unfolding sgn-pos-inf-def \textbf{using} \( p \neq 0 \), \( \text{pderiv} \ p \neq 0 \)

    by (auto simp add: lead-coeff-pderiv degree-pderiv coeff-pderiv sgn-mult pderiv-eq-0-iff)

    then show \( ?\text{thesis unfolding changes-poly-neg-inf-def by auto} \)

    qed

  also have \( \ldots = 1 + \) \text{changes-poly-neg-inf} \( \text{pders} \ (\text{pderiv} \ p) \)

  using \( \text{pderiv} \ p \neq 0 \) \textbf{by} (simp add: pders.simps)

  also have \( \ldots = 1 + \) \text{degree} \( \text{pderiv} \ p \)

  apply (subst Suc(1))

  using Suc(2) \textbf{by} (auto simp add: degree-pderiv)

  also have \( \ldots = \) \text{degree} \( p \)

  by (metis Suc.hyps(2) degree-pderiv diff-Suc-1 plus-1-eq-Suc)

  finally show \( ?\text{case .} \)

  qed

lemma \( \text{pders-coeffs-sgn-eq} : \) \( \lambda p. \) \text{sgn} \( \text{(poly p 0)} \) \( = \) \text{map} \( \text{sgn} \) \( \text{(coeffs p)} \)

proof (induct \( \text{degree} \ p \) \text{arbitrary:}p)


case 0
then obtain \( a \) where \( p = [a:] \) using degree-eq-zeroE by auto
then show \( \text{?case by (auto simp: pders.simps)} \)
next
case (Suc \( x \))
then have \( \text{pderiv } p \neq 0 \) \( \text{p} \neq 0 \) using pderie-eq-0-iff by force
have \( \text{map } (\lambda p. \text{sgn } (\text{poly } p \ 0)) (\text{pders } p) = \text{sgn } (\text{poly } p \ 0) \neq \text{map } (\lambda p. \text{sgn } (\text{poly } p \ 0)) (\text{pders } (\text{pderiv } p)) \)
apply (subst pders.simps)
using \( (p \neq 0) \) by simp
also have \( \ldots = \text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p)) \)
proof
have \( \text{sgn } (\text{poly } p \ 0) = \text{sgn } (\text{coeff } p \ 0) \) by (simp add: poly-0-coeff-0)
then show \( \text{?thesis} \)
apply (subst Suc(1))
subgoal by (metis Suc.hyps(2) degree-pderiv diff-Suc-1)
subgoal by auto
done
qed
also have \( \ldots = \text{map } \text{sgn } (\text{coeffs } p) \)
proof (rule nth-equalityI)
show \( \text{p-length:}\text{length } (\text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p)))) = \text{length } (\text{map } \text{sgn } (\text{coeffs } p)) \)
by (metis Suc.hyps(2) \( p \neq 0 \) \( \text{pderiv } p \neq 0 \) \( \text{degree-pderiv diff-Suc-1} \))
proof
fix \( i \) assume \( \text{asm: } i < \text{length } (\text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p)))) \)
show \( (\text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p))) ! i = \text{map } \text{sgn } (\text{coeffs } p) ! i \)
proof (rule+)
fix \( i \) assume \( \text{asm: } i < \text{length } (\text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p)))) \)
show \( (\text{sgn } (\text{coeff } p \ 0) \neq \text{map } \text{sgn } (\text{coeffs } (\text{pderiv } p))) ! i = \text{map } \text{sgn } (\text{coeffs } p) ! i \)
apply (cases \( i \))
subgoal by (simp add: \( p \neq 0 \) \( \text{coeffs-nth} \))
subgoal for \( n \) using \( \text{asm } \text{p-length} \)
apply simp
apply (subst (1 2) \( \text{coeffs-nth} \))
by (metis Suc.hyps(2) \( p \neq 0 \) \( \text{pderiv } p \neq 0 \) \( \text{length-coeffs-degree coeff-pderiv sgn-mult} \))
done
qed
qed
finally show \( \text{?case} . \)
qed

lemma changes-poly-at-pders-0\:changes-poly-at \( (\text{pders } p) \ 0 = \text{changes } (\text{coeffs } p) \)
unfolding changes-poly-at-def

18
apply (subst (1 2) changes-map-sgn-eq)
by (auto simp add:pders-coeffs-sgn-eq comp-def)

2.4 Budan-Fourier theorem

lemma budan-fourier-aux-right:
assumes c < d2 and p ≠ 0
assumes ∀ x. c < x ∧ x ≤ d2 → (∀ q ∈ set (pders p). poly q x ≠ 0)
shows changes-itv-der c d2 p = 0
using assms(2-3)
proof (induct degree p arbitrary:p)
case 0
then obtain a where p = [a:] a ≠ 0 by (metis degree-eq-zeroE pCons-0-0)
then show ?case by (auto simp add: changes-itv-der-def pders.
simps intro:order-0I)
next
  case (Suc n)
  then have [simp]: pderiv p ≠ 0 by (metis nat.distinct(1) pderiv-eq-0-iff)
  note nzero = ⟨∀ x. c < x ∧ x ≤ d2 −→ (∀ q ∈ set (pders p). poly q x ≠ 0)⟩
  have hyps: changes-itv-der c d2 (pderiv p) = 0
    apply (rule Suc(1))
    subgoal by (metis Suc.hyps(2) degree-pderiv diff-Suc-1)
    subgoal by (simp add: Suc.prems(1) Suc.prems(2) pders.simps)
    subgoal by (simp add: Suc.prems(1) nzero pders.simps)
    done
  have pders-changes-c: changes-poly-at (r # pders q) c = (if sign-r-pos q c −→
    poly r c > 0
    then changes-poly-at (pders q) c else 1 + changes-poly-at (pders q) c)
    when poly r c ≠ 0 for q r
    using (q ≠ 0)
    proof (induct q rule:pders.induct)
      case (Suc q)
      have ?case when pderiv q = 0
        proof –
          have degree q = 0 using that pderiv-eq-0-iff by blast
          then obtain a where q = [a:] a ≠ 0 using (q ≠ 0) by (metis degree-eq-zeroE
            pCons-0-0)
          then show ?thesis using (poly r c ≠ 0)
            by (auto simp add:sign-r-pos-rec changes-poly-at-def mult-less-0-iff pders.simps)
        qed
        moreover have ?case when pderiv q ≠ 0
        proof –
          obtain qs where qs:pders q = q # qs pders (pderiv q) = qs
            using (q ≠ 0) by (simp add:pders.simps)
          have changes-poly-at (r # qs) c = (if sign-r-pos (pderiv q) c = (0 < poly r
c)
            then changes-poly-at qs c else 1 + changes-poly-at qs c)
            using 1 (pderiv q ≠ 0) unfolding qs by simp

19
then show \( \text{thesis} \) unfolding \( qs \)
apply (cases \( \text{poly} \) \( q \) \( c = 0 \))
subgoal unfolding \( \text{changes-poly-at-def} \) by (auto simp:sign-r-pos-rec[OF \( q \neq 0 \), of \( c \)]
subgoal unfolding \( \text{changes-poly-at-def} \) using (\( \text{poly} \) \( r \) \( c 
eq 0 \))
  by (auto simp:sign-r-pos-rec[OF \( q \neq 0 \), of \( c \)] mult-less-0-iff)
done
qed
ultimately show ?case by blast
qed
have \( \text{pders-changes-d2:changes-poly-at} \) \( (r \# \text{pders} \ q) \) \( d2 = (\text{sign-r-pos} \ q \ c \mapsto \text{poly} \ r \ c > 0) \)
  then \( \text{changes-poly-at} \) \( (\text{pders} \ q) \) \( d2 \) else \( 1 + \text{changes-poly-at} \) \( (\text{pders} \ q) \) \( d2 \)
when \( \text{poly} \) \( r \) \( c \neq 0 \) \( q \neq 0 \) and \( \text{qr-nzero}:\forall x \cdot c < x \land x \leq d2 \mapsto \text{poly} \ r \ x \neq 0 \land \text{poly} \ q \ x \neq 0 \)
for \( q \) \( r \)
proof –
  have \( r \neq 0 \) using that(1) using \( \text{poly-0} \) by blast
obtain \( qs \) where \( qs:pders \ q = \#qs \) \( \text{pders} \ (\text{pderiv} \ q) = \ qs \)
    using \( q \neq 0 \) by (simp add:pders.simps)
  have if \( \text{sign-r-pos} \ r \ c \) then \( \text{poly} \ r \ d2 \) else \( \text{poly} \ r \ d2 < 0 \)
    if \( \text{sign-r-pos} \ q \ c \) then \( \text{poly} \ q \ d2 \) else \( \text{poly} \ q \ d2 < 0 \)
subgoal by (rule sign-r-pos-nzero-right[OF \( c \) \( d2 \) \( r \)]) (use \( \text{qr-nzero} \) \( \langle c < d2 \rangle \) in auto)
subgoal by (rule sign-r-pos-nzero-right[OF \( c \) \( d2 \) \( q \)]) (use \( \text{qr-nzero} \) \( \langle c < d2 \rangle \) in auto)
done
then show \( \text{thesis} \) unfolding \( qs \) changes-poly-at-def
  using \( \text{poly} \ r \ c \neq 0 \) by (auto split:if-splits simp:mult-less-0-iff sign-r-pos-rec[OF \( r \neq 0 \)])
qed
have \( \text{d2c-nzero}:\forall x \cdot c < x \land x \leq d2 \mapsto \text{poly} \ p \ x \neq 0 \land \text{poly} \ (\text{pderiv} \ p) \ x \neq 0 \)
  and \( \text{p-cons:pders} \ p = \#\text{pders}(\text{pderiv} \ p) \)
subgoal by (simp add: nzero Suc.prems(1) pders.simps)
subgoal by (simp add: Suc.prems(1) pders.simps)
done
have \( ?\text{case when} \) \( \text{poly} \ p \ c = 0 \)
proof –
  define \( ps \) where \( ps:pders \ (\text{pderiv} \ (\text{pderiv} \ p)) \)
  have \( \text{ps-cons:p\#pderiv \ p\#ps = pders \ p deriv \ p\#ps = pders \ (pderiv p)} \)
    unfolding \( ps-def \) using \( p \neq 0 \) by (auto simp:pders.simps)
  have \( \text{changes-poly-at} \) \( (p \# \text{pderiv} \ p \# \text{ps}) \) \( c = \text{changes-poly-at} \) \( (\text{pderiv} \ p \# \text{ps}) \) \( c \)
    unfolding \( \text{changes-poly-at-def} \) using that by auto
  moreover have \( \text{changes-poly-at} \) \( (p \# \text{pderiv} \ p \# \text{ps}) \) \( d2 = \text{changes-poly-at} \)
    \( (\text{pderiv} \ p \# \text{ps}) \) \( d2 \)
    proof –
have if sign-r-pos p c then \( 0 < \text{poly} p \, d2 \) else \( \text{poly} p \, d2 < 0 \)
apply (rule sign-r-pos-nzero-right[OF \( \langle \cdot < d2 \rangle \)])
using nzero[folded ps-cons] assms(1-2) by auto
moreover have if sign-r-pos (pderiv p) c then \( 0 < \text{poly} (pderiv p) \, d2 \)
else \( \text{poly} (pderiv p) \, d2 < 0 \)
apply (rule sign-r-pos-nzero-right[OF \( \langle \cdot < d2 \rangle \)])
using nzero[folded ps-cons] assms(1-2) by auto
ultimately have \( \text{poly} p \, d2 \cdot \text{poly} (pderiv p) \, d2 > 0 \)
unfolding zero-less-mult-iff sign-r-pos-rec[OF \( \langle p \neq 0 \rangle \)] using (poly p c = 0)
by (auto split:if-splits)
then show \(?thesis\) unfolding changes-poly-at-def by auto
qed
ultimately show \(?thesis\) using hyps unfolding changes-itv-der-def
apply (fold ps-cons)
by (auto simp:Let-def)
qed
moreover have \(?case when\) poly p c \(\neq 0\) sign-r-pos (pderiv p) c \(\leftrightarrow\) poly p c > 0
proof –
have changes-poly-at (pders p) c = changes-poly-at (pders (pderiv p)) c
unfolding p-cons
apply (subst pders-changes-c[OF \( \langle \cdot \neq 0 \rangle \)])
using that by auto
moreover have changes-poly-at (pders p) d2 = changes-poly-at (pders (pderiv p)) d2
unfolding p-cons
apply (subst pders-changes-d2[OF \( \langle \cdot \neq 0 \rangle \cdot -d2c-nzero \)])
using that by auto
ultimately show \(?thesis\) using hyps unfolding changes-itv-der-def Let-def
by auto
qed
moreover have \(?case when\) poly p c \(\neq 0\) \(\neg\) sign-r-pos (pderiv p) c \(\leftrightarrow\) poly p c > 0
proof –
have changes-poly-at (pders p) c = changes-poly-at (pders (pderiv p)) c + 1
unfolding p-cons
apply (subst pders-changes-c[OF \( \langle \cdot \neq 0 \rangle \)])
using that by auto
moreover have changes-poly-at (pders p) d2 = changes-poly-at (pders (pderiv p)) d2 + 1
unfolding p-cons
apply (subst pders-changes-d2[OF \( \langle \cdot \neq 0 \rangle \cdot -d2c-nzero \)])
using that by auto
ultimately show \(?thesis\) using hyps unfolding changes-itv-der-def Let-def
by auto
qed
ultimately show \(?case by blast\)
qed

lemma budan-fourier-aux-left:\
assumes $d1 < c$ and $p \neq 0$
assumes $\forall x.\; d1 \leq x \wedge x < c \rightarrow (\forall q \in \text{set} (\text{pders } p).\; \text{poly } q \ x \neq 0)$
shows $\text{changes-itv-der } d1 \ c \ p \geq \text{order } c \ p \wedge \text{even} (\text{changes-itv-der } d1 \ c \ p - \text{order } c \ p)$
using assms $(2-3)$
proof (induct degree $p$ arbitrary:$p$)
case 0
then obtain $a$ where $p = [a]$ $a \neq 0$ by (metis degree-eq-zeroE pCons-0-0)
then show ?case
apply (auto simp add: changes-itv-der-def pders.simps intro:order-0I)
by (metis add.right-neutral dvd-0-right mult-zero-right order-root poly-pCons)
next
case (Suc $n$)
then have $[\text{simp}]:\text{pderiv } p \neq 0$ by (metis nat.distinct(1) pderiv-eq-0-iff)
note $nzero = \langle \forall x.\; d1 \leq x \wedge x < c \rightarrow (\forall q \in \text{set} (\text{pders } p).\; \text{poly } q \ x \neq 0)\rangle$
define $v$ where $v = \text{order } c (\text{pderiv } p)$

have $\text{hyps} : v \leq \text{changes-itv-der } d1 \ c \ (\text{pderiv } p) \wedge \text{even} (\text{changes-itv-der } d1 \ c \ (\text{pderiv } p) - v)$
unfolding $v$-def
apply (rule Suc(1))
subgoal by (metis Suc.hyps(2) degree-pderiv diff-Suc-1)
subgoal by (simp add: Suc.prems(1) Suc.prems(2) pders.simps)
subgoal by (simp add: Suc.prems(1) nzero pders.simps)
done
have $\text{pders-changes-c:changes-poly-at } (r \# \text{pders } q) \ c = (\text{if sign-r-pos } q \ c \leftarrow \rightarrow \text{poly } r \ c > 0)$
when $\text{poly } r \ c \neq 0 \ q \neq 0$ for $q \ r$
using $\langle q \neq 0 \rangle$
proof (induct $q$ rule:pders.induct)
case (1 $q$)
have $?case$ when $\text{pderv } q = 0$
proof
have $\text{degree } q = 0$ using that pderiv-eq-0-iff by blast
then obtain $a$ where $q = [a]$ $a \neq 0$ using $\langle q \neq 0 \rangle$ by (metis degree-eq-zeroE pCons-0-0)
then show $?thesis$ using $\langle \text{poly } r \ c \neq 0 \rangle$.
by (auto simp add:sign-r-pos-rec changes-poly-at-def mult-less-0-iff pders.simps)
qed
moreover have $?case$ when $\text{pderiv } q \neq 0$
proof
obtain $qs$ where $qs:pders \ q = q \# qs \ \text{pders} (\text{pderiv } q) = qs$
using $\langle q \neq 0 \rangle$ by (simp add:pders.simps)
have $\text{changes-poly-at } (r \# qs) \ c = (\text{if sign-r-pos } (\text{pderiv } q) \ c = (0 < \text{poly } r \ c))$
then $\text{changes-poly-at } qs \ c \ else \ 1 + \text{changes-poly-at } qs \ c$
using 1 $\langle \text{pderiv } q \neq 0 \rangle$ unfolding $qs$ by simp
then show $?thesis$ unfolding $qs$

22
apply (cases poly q c=0)

subgoal unfolding changes-poly-at-def by (auto simp:sign-r-pos-rec[OF ⟨q≠0; of c⟩])

subgoal unfolding changes-poly-at-def using ⟨poly r c≠0⟩ by (auto simp:sign-r-pos-rec[OF ⟨q≠0; of c⟩] mult-less-0-iff)

done

qed

ultimately show ?case by blast

qed

have pders-changes-d1:changes-poly-at (r# pders q) d1 = (if even (order c q)
                        then changes-poly-at (pders q) d1 else changes-poly-at (pders q) d1)
                        when poly r c≠0 q≠0 and qr-nzero:
                                ∀ x. d1 ≤ x ∧ x < c → poly r x ≠ 0 ∧ poly q x≠0
                                for q r

proof –

  have r≠0 using that(1) using poly-0 by blast

  obtain qs where qs:pders q=q#qs pders (pderiv q) = qs

  using ⟨q≠0⟩ by (simp add:pders.simps)

  have if even (order c r) = sign-r-pos r c then 0 < poly r d1 else poly r d1 < 0

  if even (order c q) = sign-r-pos q c then 0 < poly q d1 else poly q d1 < 0

  subgoal by (rule sign-r-pos-nzero-left[of d1 c r]) (use qr-nzero ⟨d1<c⟩ in auto)

  subgoal by (rule sign-r-pos-nzero-left[of d1 c q]) (use qr-nzero ⟨d1<c⟩ in auto)

  done

moreover have order c r=0 by (simp add: order-0I that)

ultimately show ?thesis unfolding qs changes-poly-at-def

using ⟨poly r c≠0⟩ by (auto split:if-splits simp:mult-less-0-iff sign-r-pos-rec[OF ⟨r≠0⟩])

qed

have d1c-nzero:∀ x. d1 ≤ x ∧ x < c → poly p x ≠ 0 ∧ poly (pderiv p) x ≠ 0

and p-cons: pders p = p#pders(pderiv p)

by (simp-all add: nzero Suc.prems(1) pders.simps)

have ?case when poly p c=0

proof –

  define ps where ps=pders (pderiv (pderiv p))

  have ps-cons:p#pderiv p#ps = pders p pderiv p#ps=pders (pderiv p)

  unfolding ps-def using ⟨p≠0⟩ by (auto simp:pders.simps)

  have p-order: order c p = Suc v

  apply (subst order-pderiv)

  using Suc.prems(1) order-root that unfolding v-def by auto

moreover have changes-poly-at (p#pderiv p # ps) d1 = changes-poly-at (pderiv p#ps) d1 + 1

proof –

  have if even (order c p) = sign-r-pos p c then 0 < poly p d1 else poly p d1 < 0
apply (rule sign-r-pos-nzero-left[OF \((d1 < c)\)])
using nzero[folded ps-cons] assms(1-2) by auto
moreover have if even v = sign-r-pos (pderiv p) c
then \(0 < \text{poly} (pderiv p) \cdot d1\) else \(\text{poly} (pderiv p) \cdot d1 < 0\)
unfolding v-def
apply (rule sign-r-pos-nzero-left[OF \((d1 < c)\)])
using nzero[folded ps-cons] assms(1-2) by auto
ultimately have \(\text{poly} p \cdot d1 < \text{poly} (pderiv p) \cdot d1 < 0\)
unfolding mult-less-0-iff sign-r-pos-rec[OF \(p\neq 0\)] using \(\text{poly} p \cdot c=0\)
p-order
by (auto split:if-splits)
then show ?thesis
unfolding changes-poly-at-def by auto
qed
moreover have \(\text{changes-poly-at} (p \# pderiv p \# ps) c = \text{changes-poly-at} (pderiv p \# ps) c\)
unfolding changes-poly-at-def using that by auto
ultimately show ?thesis using hyps unfolding changes-itv-der-def
apply (fold ps-cons)
by (auto simp:Let-def)
qed
moreover have \(?case when \(\text{poly} p \neq 0\) odd v sign-r-pos (pderiv p) c \longleftrightarrow \text{poly} p \cdot c>0\)
proof
have order c p=0 by (simp add: order-0I that(1))
moreover have \(\text{changes-poly-at} (pders p) \cdot d1 = \text{changes-poly-at} (pders (pderiv p)) \cdot d1 +1\)
unfolding p-cons
apply (subst pders-changes-d1[OF \(\text{poly} p \neq 0\) - d1c-nzero])
using that unfolding v-def by auto
moreover have \(\text{changes-poly-at} (pders p) c = \text{changes-poly-at} (pders (pderiv p)) c\)
unfolding p-cons
apply (subst pders-changes-c[OF \(\text{poly} p \neq 0\)])
using that unfolding v-def by auto
ultimately show ?thesis using hyps \(\text{odd v}\) unfolding changes-itv-der-def
Let-def
by auto
qed
moreover have \(?case when \(\text{poly} p \neq 0\) odd v \neg \text{sign-r-pos} (pderiv p) c \longleftrightarrow \text{poly} p \cdot c>0\)
proof
have v\geq1 using \(\text{odd v}\) using not-less-eq-eq by auto
moreover have order c p=0 by (simp add: order-0I that(1))
moreover have \(\text{changes-poly-at} (pders p) \cdot d1 = \text{changes-poly-at} (pders (pderiv p)) \cdot d1\)
unfolding p-cons
apply (subst pders-changes-d1[OF \(\text{poly} p \neq 0\) - d1c-nzero])
using that unfolding v-def by auto
moreover have changes-poly-at (pders p) c = changes-poly-at (pders (pderiv p)) c + 1
  unfolding p-cons
  apply (subst pders-changes-c[OF (poly p c≠0)])
  using that unfolding v-def by auto
  ultimately show ?thesis using hyps ⟨odd v⟩ unfolding changes-itv-der-def
Let-def
  by auto
qed
moreover have ?case when poly p c≠0 even v sign-r-pos (pderiv p) c ←→ poly p c>0
proof –
  have order c p=0 by (simp add: order-0I that(1))
  moreover have changes-poly-at (pders p) d1 = changes-poly-at (pders (pderiv p)) d1
    unfolding p-cons
    apply (subst pders-changes-d1[OF (poly p c≠0) - d1c-nzero])
    using that unfolding v-def by auto
  moreover have changes-poly-at (pders p) c = changes-poly-at (pders (pderiv p)) c
    unfolding p-cons
    apply (subst pders-changes-c[OF (poly p c≠0)])
    using that unfolding v-def by auto
  ultimately show ?thesis using hyps ⟨even v⟩ unfolding changes-itv-der-def
Let-def
  by auto
qed
moreover have ?case when poly p c≠0 even v ¬ sign-r-pos (pderiv p) c ←→ poly p c>0
proof –
  have order c p=0 by (simp add: order-0I that(1))
  moreover have changes-poly-at (pders p) d1 = changes-poly-at (pders (pderiv p)) d1 + 1
    unfolding p-cons
    apply (subst pders-changes-d1[OF (poly p c≠0) - d1c-nzero])
    using that unfolding v-def by auto
  moreover have changes-poly-at (pders p) c = changes-poly-at (pders (pderiv p)) c + 1
    unfolding p-cons
    apply (subst pders-changes-c[OF (poly p c≠0)])
    using that unfolding v-def by auto
  ultimately show ?thesis using hyps ⟨even v⟩ unfolding changes-itv-der-def
Let-def
  by auto
qed
ultimately show ?case by blast
qed

lemma budan-fourier-aux-left:
assumes \( d_1 < c \) and \( p \neq 0 \)
assumes \( \text{nzero} \forall x. \ d_1 < x \land x < c \rightarrow (\forall q \in \text{set (pders p)}. \ poly \ q x \neq 0)\)
shows \( \text{changes-itv-der} \ d_1 \ c \ p \geq \text{order} \ c \ p \ \text{even} \ (\text{changes-itv-der} \ d_1 \ c \ p - \text{order} \ c \ p)\)
proof –
  define \( d \) where \( d = (d_1 + c) / 2 \)
have \( d_1 < d \ < c \) unfolding \( d \)-def using \( (d_1 < c) \) by auto

have \( \text{changes-itv-der} \ d_1 \ d \ p = 0 \)
  apply (rule budan-fourier-aux-right[\( OF (d_1 < d) (p \neq 0) \)])
  using \( \text{nzero} \ (d_1 < d) (d < c) \) by auto
moreover have \( \text{order} \ c \ p \leq \text{changes-itv-der} \ d \ c \ p \ \wedge \ \text{even} \ (\text{changes-itv-der} \ d \ c \ p - \text{order} \ c \ p)\)
  apply (rule budan-fourier-aux-left[\( OF (d < c) (p \neq 0) \)])
  using \( \text{nzero} \ (d_1 < d) (d < c) \) by auto
ultimately show \( \text{changes-itv-der} \ d_1 \ c \ p \geq \text{order} \ c \ p \ \text{even} \ (\text{changes-itv-der} \ d_1 \ c \ p - \text{order} \ c \ p)\)

ultimately have \( \text{order} \ b \ p \leq \text{changes-itv-der} \ a \ b \ p \ \text{even} \ (\text{changes-itv-der} \ a \ b \ p - \text{order} \ b \ p)\)

ultimately have \( \text{proots-count} \ p \{x. \ a < x \wedge x \leq b\} = \text{order} \ b \ p \)

ultimately show \( \text{changes-itv-der-def Let-def} \) by auto

theorem budan-fourier-interval:
  assumes \( a < b \ p \neq 0 \)
  shows \( \text{changes-itv-der} \ a \ b \ p \geq \text{proots-count} \ p \{x. \ a < x \wedge x \leq b\} \wedge \text{even} \ (\text{changes-itv-der} \ a \ b \ p - \text{proots-count} \ p \{x. \ a < x \wedge x \leq b\})\)
  using \( (a < b) \)
proof (induct \( \text{card} \\{x. \ \exists p \in \text{set (pders p)}. \ poly p x = 0 \wedge a < x \wedge x < b\}\) arbitrary:b)
  case 0
  have \( \text{nzero} \forall x. \ a < x \wedge x < b \rightarrow (\forall q \in \text{set (pders p)}. \ poly q x \neq 0)\)
  proof –
  define \( S \) where \( S = \{x. \ \exists p \in \text{set (pders p)}. \ poly p x = 0 \wedge a < x \wedge x < b\}\)
  have finite \( S \)
  proof –
  have \( S \subseteq (\bigcup p \in \text{set (pders p)}. \ \text{proots} p)\)
  unfolding \( S \)-def by auto
  moreover have finite \( (\bigcup p \in \text{set (pders p)}. \ \text{proots} p)\)
  apply (subst finite-UN)
  using \( \text{set-pders-nzero[OF (p \neq 0)]} \) by auto
  ultimately show \( \text{?thesis} \) by (simp add: finite-subset)
qed
moreover have \( \text{card} \ S = 0 \) unfolding \( S \)-def using \( 0 \) by auto
ultimately have \( S = \{\} \) by auto
then show \( \text{?thesis} \) unfolding \( S \)-def using \( (a < b) \) \( \text{assms(2)} \) \( \text{pders.simps by fastforce} \)
  qed
from \( \text{budan-fourier-aux-left[OF (a < b) (p \neq 0)] this} \)
have \( \text{order} \ b \ p \leq \text{changes-itv-der} \ a \ b \ p \ \text{even} \ (\text{changes-itv-der} \ a \ b \ p - \text{order} \ b \ p)\)
by simp-all
moreover have \( \text{proots-count} \ p \{x. \ a < x \wedge x \leq b\} = \text{order} \ b \ p \)
proof –

have p-cons: pders p = p#pders (pderiv p) by (simp add: assms(2) pders.simps)

have proots-within p \{ x. a < x ∧ x ≤ b \} = (if poly p b=0 then \{ b \} else {}) using nzero (a < b) unfolding p-cons

apply auto using not-le by fastforce

then show ?thesis unfolding proots-count_def using order-root by auto

qed ultimately show ?case by auto

next
case (Suc n)
define P where P = (λx. ∃ p ∈ set (pders p). poly p x = 0)
define S where S = (λb. \{ x. P x ∧ a < x ∧ x < b \})
define b' where b' = Max (S b)
have f-S: finite (S x) for x

proof -
  have S x ⊆ (⋃ p ∈ set (pders p). proots p) unfolding S-def P-def by auto
  moreover have finite (⋃ p ∈ set (pders p). proots p) apply (subst finite-UN)
  using set-pders-nzero[OF \{ p ≠ 0 \}] by auto
  ultimately show ?thesis by (simp add: finite-subset)

qed

have b' ∈ S b unfolding b'-def
  apply (rule Suc(2) unfolding S-def P-def by force)
then have a < b' b' ≤ b unfolding S-def by auto

have b'-nzero: ∀ x. b' < x ∧ x < b → (∀ q ∈ set (pders p). poly q x ≠ 0)

proof (rule contr)
  assume ¬ (∀ x. b' < x ∧ x < b → (∀ q ∈ set (pders p). poly q x ≠ 0))
then obtain bb where P bb b' < bb bb < b unfolding P-def by auto

then have bbc < S b unfolding S-def using (a < b') (b' < b) by auto
from Max_neq[OF f-S this, folded b'-def] have bb ≤ b'.
then show False using (b' < bb) by auto

qed

have hyps: proots-count p \{ x. a < x ∧ x ≤ b' \} ≤ changes-itv-der a b' p ∧
even (changes-itv-der a b' p − proots-count p \{ x. a < x ∧ x ≤ b' \})

proof (rule Suc(1))[OF \{ a < b' \}]

have S b = \{ b' \} ∪ S b'

proof -
  have \{ x. P x ∧ b' < x ∧ x < b \} = {}
    using b'-nzero unfolding P-def by auto
  then have \{ x. P x ∧ b' ≤ x ∧ x < b \} = \{ b' \}
    using \{ b' ∈ S b \} unfolding S-def by force
  moreover have S b = S b' ∪ \{ x. P x ∧ b' ≤ x ∧ x < b \}
    unfolding S-def using (a < b') (b' < b) by auto
  ultimately show ?thesis by auto

qed
moreover have $Suc \ n = card \ (S \ b)$ using $Suc(2)$ unfolding $S$-def $P$-def by simp

moreover have $b' \sqsubseteq S \ b'$ unfolding $S$-def by auto

ultimately have $n = card \ \{x. \ \exists \ p \in \text{set} \ (pder p). \ \text{poly} \ p \ x = 0 \ \land \ a < x \ \land \ x < b'\}$

unfolding $S$-def $P$-def by simp

qed

moreover have $proots-count \ p \ \{x. \ a < x \ \land \ x \leq b\}$

proof -

have $p-consp:pders \ p=p\#pders \ (pder p)$ by (simp add: assms(2) pders.simps)

have $proots-within \ p \ \{x. \ b' < x \ \land \ x \leq b\} = (\text{if poly} \ p \ b=0 \ \text{then} \ \{b\} \ \text{else} \ \{\})$

using $b'$-nzero $\langle b' < b\rangle$ unfolding $p$-cons

apply auto

using not-le by fastforce

then have $proots-count \ p \ \{x. \ b' < x \ \land \ x \leq b\} = order \ b \ p$

unfolding $proots-count-def$ using $order-root$ by auto

moreover have $proots-count \ p \ \{x. \ a < x \ \land \ x \leq b\} = proots-count \ p \ \{x. \ a < x \ \land \ x \leq b\}$ + $proots-count \ p \ \{x. \ b' < x \ \land \ x \leq b\}$

apply (subst $proots-count$-union-disjoint[symmetric])

using $(a\ < \ b') \ \langle b' < b \rangle \ (p\neq0)$ by (auto intro:arg-cong2\where $f=proots-count$)

ultimately show $\theta$thesis by auto

qed

moreover note $budan-fourier-aux-left[OF \ \langle b' < b \rangle \ \langle p\neq0\rangle \ b'$-nzero]

ultimately show $\exists$case unfolding $changes$-ite$-der-def \ Let-def$ by auto

qed

theorem $budan-fourier$-gt:

assumes $p\neq0$

shows $changes$-gt$-der \ a \ p \geq \ proots-count \ p \ \{x. \ a < x\} \ \land$

even $(changes$-gt$-der \ a \ p - proots-count \ p \ \{x. \ a < x\})$

proof -

define $ps \ where \ ps=pders \ p$

obtain $ub$ where $ub-root\forall p\in \text{set} \ ps. \ \forall x. \ \text{poly} \ p \ x = 0 \ \longrightarrow \ x < ub$

and $ub-sgn:\forall x \geq ub. \ \forall p \in \text{set} \ ps. \ \text{sgn} \ (\text{poly} \ p \ x) = \text{sgn}$-pos-inf $p$

and $a < ub$

using root-list-$ub[\text{of} \ ps \ a]$ set-$pders$-nzero[OF $\langle p\neq0\rangle$\folded $ps$-def] by blast

have $proots-count \ p \ \{x. \ a < x\} = proots-count \ p \ \{x. \ a < x \ \land \ x \leq ub\}$

proof -

have $p \in \text{set} \ ps$ unfolding $ps$-def by (simp add: assms pders.simps)

then have $proots-within \ p \ \{x. \ a < x\} = proots-within \ p \ \{x. \ a < x \ \land \ x \leq ub\}$

using $ub-root$ by fastforce

then show $\theta$thesis unfolding $proots-count-def$ by auto

qed

moreover have $changes$-gt$-der \ a \ p = changes$-ite$-der \ a \ ub \ p$

proof -

have $map \ (\text{sgn} \circ (\lambda p. \ \text{poly} \ p \ \text{ub})) \ ps = map \ \text{sgn}$-pos-inf $ps$

using $ub-sgn[THEN \ spec,of \ ub,\simplified]$

28
by (metis (mono-tags, lifting) comp-def list_map_cong0)

hence changes_poly_at ps ub = changes_poly_pos_inf ps

unfolding changes_poly_pos_inf_def changes_poly_at_def

by (subst changes_map_sgn_eq, metis map_map)

then have changes_poly_at ps ub = 0

unfolding ps_def by simp

thus ?thesis unfolding changes_gt_der_def changes_itv_der_def

by (simp add: Let_def)

qed

moreover have proots_count p {x. a < x ∧ x ≤ ub} ≤ changes_itv_der a ub p ∧

even (changes_itv_der a ub p ∨ proots_count p {x. a < x ∧ x ≤ ub})

using budan_fourier_gt[OF ⟨a < ub⟩ ⟨p̸=0⟩] unfolding changes_gt_der_def

by (simp add: changes_poly_at_pders_0)

ultimately show ?thesis by auto

qed

Descartes' rule of signs is a direct consequence of the Budan-Fourier theorem

theorem descartes_sign:

fixes p :: real poly

assumes p̸=0

shows changes (coeffs p) ≥ proots_count p {x. 0 < x} ∧

even (changes (coeffs p) − proots_count p {x. 0 < x})

using budan_fourier_gt[OF ⟨p̸=0⟩, of 0] unfolding changes_gt_der_def

by (simp add: changes_poly_at_pders_0)

theorem budan_fourier_le:

assumes p̸=0

shows changes_le_der b p ≥ proots_count p {x. x ≤ b} ∧

even (changes_le_der b p ∨ proots_count p {x. x ≤ b})

proof —

define ps where ps = pders p

obtain lb where lb_root: ∀ p ∈ set ps. ∀ x. poly p x = 0 → x > lb

and lb_sgn: ∀ x. lb ≤ x, ∀ p ∈ set ps. sgn (poly p x) = sgn_neg_inf p

and lb < b

using root_list_lb[OF ps b] set_pders_nzero[OF ⟨p̸=0⟩, of 0, folded ps_def] by blast

have proots_count p {x. x ≤ b} = proots_count p {x. lb < x ∧ x ≤ b}

proof —

have p ∈ set ps unfolding ps_def by (simp add: assms pders_simps)

then have proots_within p {x. x ≤ b} = proots_within p {x. lb < x ∧ x ≤ b}

using lb_root by fastforce

then show ?thesis unfolding proots_count_def by auto

qed

moreover have changes_le_der b p = changes_itv_der lb b p

proof —

have map (sgn ∘ λ x. poly p lb)) ps = map sgn_neg_inf ps

using lb_sgn[THEN spec, of lb, simplified]

by (metis (mono_tags, lifting) comp_def list_map_cong0)

hence changes_poly_at ps lb = changes_poly_neg_inf ps

unfolding changes_poly_neg_inf_def changes_poly_at_def

by (subst changes_map_sgn_eq, metis map_map)

29
then have \( \text{changes-poly-at } ps \text{ lb}=\text{degree } p \) unfolding \( ps \text{-def} \) by simp

thus \( \text{thesis unfolding changes-le-der-def changes-itv-der-def ps-def} \)

by (simp add: Let-def)

qed

moreover have \( \text{proots-count } p \{x. \text{ lb} < x \wedge x \leq b\} \leq \text{changes-itv-der lb } b \text{ p } \wedge \)

\begin{itemize}
\item even \( (\text{changes-itv-der lb } b \text{ p} - \text{proots-count } p \{x. \text{ lb} < x \wedge x \leq b\}) \)
\end{itemize}

using \( \text{budan-fourier-interval}[OF \langle lb \text{<} b \rangle \langle p\neq 0 \rangle] \).

ultimately show \( \text{thesis by } auto \)

qed

If we knowing that all roots of a polynomial are real, we can use the Budan-Fourier theorem to EXACTLY count the number of real roots.

corollary \( \text{budan-fourier-real}: \)

assumes \( p\neq 0 \text{ a} \text{<} b \)

assumes \( \text{proots-deg;proots-count } p \text{ UNIV } =\text{degree } p \) — All of \( p \)'s roots are real.

shows \( \text{proots-count } p \{x. \text{x} \leq a\} = \text{changes-le-der a } p \)

\( \text{proots-count } p \{x. \text{ a} <x \wedge x \leq b\} = \text{changes-itv-der a } b \text{ p} \)

\( \text{proots-count } p \{x. \text{ b} <x\} = \text{changes-gt-der b } p \)

proof —

define \( c1 \text{ \ c2 \ c3 where} \)

\( c1=\text{changes-le-der a } p - \text{proots-count } p \{x. \text{x} \leq a\} \) and

\( c2=\text{changes-itv-der a } b \text{ p} - \text{proots-count } p \{x. \text{ a} <x \wedge x \leq b\} \) and

\( c3=\text{changes-gt-der b } p - \text{proots-count } p \{x. \text{ b} <x\} \)

have \( c1\geq 0 \text{ c2}\geq 0 \text{ c3}\geq 0 \)

using \( \text{budan-fourier-interval}[OF \langle a \text{<} b \rangle \langle p\neq 0 \rangle,of b] \)

\( \text{budan-fourier-le[OF } p\neq 0,\text{of a]} \)

unfolding \( \text{c1-def c2-def c3-def by } auto \)

moreover have \( c1+c2+c3=0 \)

proof —

have \( \text{changes-le-der a } p + \text{changes-itv-der a } b \text{ p} + \text{changes-gt-der b } p = \text{degree } p \)

unfolding \( \text{changes-le-der-def changes-itv-der-def changes-gt-der-def} \)

by (auto simp add: Let-def)

moreover have \( \text{proots-count } p \{x. \text{x} \leq a\} + \text{proots-count } p \{x. \text{ a} <x \wedge x \leq b\} \)

+ \( \text{proots-count } p \{x. \text{ b} <x\} = \text{degree } p \)

using \( p\neq 0, \text{a} \text{<} b \)

apply (subst \( \text{proots-count-union-disjoint}[\text{symmetric},auto]+ \)

apply (subst \( \text{proots-deg}[\text{symmetric}]\))

by (auto intro!:arg-cong2[where \( f=\text{proots-count}]) \)

ultimately show \( \text{thesis unfolding c1-def c2-def c3-def} \)

by (auto simp add: algebra-simps)

qed

ultimately have \( c1 =0 \wedge c2=0 \wedge c3=0 \) by auto

then show \( \text{proots-count } p \{x. \text{x} \leq a\} = \text{changes-le-der a } p \)

\( \text{proots-count } p \{x. \text{ a} <x \wedge x \leq b\} = \text{changes-itv-der a } b \text{ p} \)

\( \text{proots-count } p \{x. \text{ b} <x\} = \text{changes-gt-der b } p \)
3 Extension of Sturm’s theorem for multiple roots

theory Sturm-Multiple-Roots imports BF-Misc begin

The classic Sturm’s theorem is used to count real roots WITHOUT multiplicity of a polynomial within an interval. Surprisingly, we can also extend Sturm’s theorem to count real roots WITH multiplicity by modifying the signed remainder sequence, which seems to be overlooked by many textbooks.


3.1 More results for smods

lemma last-smods-gcd:
  fixes p q :: real poly
  defines pp ≡ last (smods p q)
  assumes p ≠ 0
  shows pp = smult (lead-coeff pp) (gcd p q)
  using ⟨p ≠ 0⟩ unfolding pp-def
proof (induct smods p q arbitrary: p q rule: length-induct)
  case 1
  have ?case when q = 0
    using that smult-normalize-field-eq ⟨p ≠ 0⟩ by auto
  moreover have ?case when q ≠ 0
    proof
      define r where r = − (p mod q)
      have smods-cons:smods p q = p # smods q r
        unfolding r-def using ⟨p ≠ 0⟩ by simp
      have last (smods q r) = smult (lead-coeff (last (smods q r))) (gcd q r)
        apply (rule 1 (1)[rule-format, af smods q r q r])
        using smods-cons ⟨q ≠ 0⟩ by simp
      moreover have gcd p q = gcd q r
        unfolding r-def by (simp add: gcd.commute that)
      ultimately show ?thesis unfolding smods-cons using ⟨q ≠ 0⟩
        by simp
    qed
  ultimately show ?thesis by argo
qed

qed
lemma last-smods-nzero:
  assumes p≠0
  shows last (smods p q) ≠ 0
  by (metis assms last-in-set no-0-in-smods smods-nil-eq)

3.2 Alternative signed remainder sequences

function smods-ext::real poly ⇒ real poly ⇒ real poly list where
  smods-ext p q = (if p=0 then [] else
    (if p mod q ≠ 0
     then Cons p (smods-ext q (−(p mod q)))
      else Cons p (smods-ext q (pderiv q)))
    )
  
by auto
termination
  apply (relation measure (λ(p,q).if p=0 then 0 else if q=0 then 1 else 2+degree q))
  using degree-mod-less by (auto simp add: degree-pderiv pderiv-eq-0-iff)

lemma smods-ext-prefix:
  fixes p q::real poly
defines pp ≡ last (smods p q)
  assumes p≠0 q≠0
  shows smods-ext p q = smods p q @ tl (smods-ext pp (pderiv pp))
  unfolding pp-def using assms(2,3)
  proof (induct smods-ext p q arbitrary: p q rule: length-induct)
    case 1
    have ?case when p mod q ≠ 0
      proof
        define pp where pp=last (smods q (−(p mod q)))
        have smods-cons:smods p q = p# smods q (−(p mod q))
          using (p≠0) by auto
        then have pp-last: pp=last (smods p q) unfolding pp-def
          by (simp add: 1.prems(2) pp-def)
        have smods-ext-cons: smods-ext p q = p# smods-ext q (−(p mod q))
          using that (p≠0) by auto
        have smods-ext q (−(p mod q)) = smods q (−(p mod q)) @ tl (smods-ext pp (pderiv pp))
          apply (rule 1(1)[rule-format,of smods-ext q (−(p mod q)) q −(p mod q),folded pp-def])
          using smods-ext-cons ⟨q≠0⟩ that by auto
        then show ?thesis unfolding pp-last
          apply (subst smods-cons)
          apply (subst smods-ext-cons)
          by auto
      qed
    moreover have ?case when p mod q =0 pderiv q = 0
      proof
        have smods p q = [p,q]
using \( p \neq 0 \) \( q \neq 0 \) that by auto

moreover have smods-ext \( p \ q = [p,q] \)

using that \( p \neq 0 \) by auto

ultimately show \( \text{case using} \ p \neq 0 \ q \neq 0 \ (1) \) by auto

qed

moreover have \( \text{case when} \ p \mod q = 0 p \text{deriv} q \neq 0 \)

proof –

have smods-cons:smods \( p \ q = [p,q] \)

using \( p \neq 0 \) \( q \neq 0 \) that by auto

have smods-ext-cons:smods-ext \( p \ q = p \# \text{smods-ext} q \ (p \text{deriv} q) \)

using that \( p \neq 0 \) by auto

show \( \text{case unfolding} \ \text{smods-cons} \ \text{smods-ext-cons} \)

apply \( \text{simp del:smods-ext.simps} \)

by (simp add: 1.prems(2))

qed

ultimately show \( \text{case by argo} \)

qed

lemma no-0-in-smods-ext: \( 0 \notin \text{set (smods-ext} p \ q) \)

apply (induct smods-ext \( p \ q \) arbitrary: \( p \ q \))

apply simp

by (metis list.distinct(1) list.inject set-ConsD smods-ext.simps)

3.3 Sign variations on the alternative signed remainder sequences

definition changes-itv-smods-ext:: real \( \Rightarrow \) real \( \Rightarrow \) real poly \( \Rightarrow \) real poly \( \Rightarrow \) int where

changes-itv-smods-ext \( a \ b \ p \ q = (\text{let ps}= \text{smods-ext} p \ q \text{ in changes-poly-at ps} a \)

− changes-poly-at ps b)\)

definition changes-gt-smods-ext:: real \( \Rightarrow \) real poly \( \Rightarrow \) real poly \( \Rightarrow \) int where

changes-gt-smods-ext \( a \ p \ q = (\text{let ps}= \text{smods-ext} p \ q \text{ in changes-poly-at ps} a \)

− changes-poly-pos-inf ps)\)

definition changes-le-smods-ext:: real \( \Rightarrow \) real poly \( \Rightarrow \) real poly \( \Rightarrow \) int where

changes-le-smods-ext \( b \ p \ q = (\text{let ps}= \text{smods-ext} p \ q \text{ in changes-poly-neg-inf ps} a \)

− changes-poly-at ps b)\)

definition changes-R-smods-ext:: real \( \Rightarrow \) real poly \( \Rightarrow \) real poly \( \Rightarrow \) int where

changes-R-smods-ext \( p \ q = (\text{let ps}= \text{smods-ext} p \ q \text{ in changes-poly-neg-inf ps} a \)

− changes-poly-pos-inf ps)\)

3.4 Extension of Sturm’s theorem for multiple roots

theorem sturm-ext-interval:

assumes \( a < b \) poly \( p \ a \neq 0 \) poly \( p \ b \neq 0 \)

shows \( \text{proots-count} \ p \ (x. \ a < x \wedge x < b) = \text{changes-itv-smods-ext} a \ b \ (\text{pderiv} p) \)

using assms(2,3)
proof (induct \textit{smods-ext} \(p\) (pderiv \(p\)) arbitrary; \(p\) rule: length-induct)

\textbf{case 1}

have \(p\neq 0\) using (poly \(p\) \(a\neq 0\)) by auto

have \(?case\ when\ \textit{pderiv}\ \(p\)=0\)

proof –

obtain \(c\) where \(p=:[c:]\ c\neq 0\)

using \(p\neq 0\), (pderiv \(p\)=0), pderiv-iszero by force

then have \(\textit{proots-count} \(p\) \{x. \(a < x \land x < b\}\} = 0\)

un folding \(\textit{proots-count-def}\) by auto

moreover have \(\textit{changes-itv-smods-ext} \(a\ \(b\ \(p\)\))) (pderiv \(p\)) = 0

un folding \(\textit{changes-itv-smods-ext-def}\) using \(\langle p=:[c:]\ \langle c\neq 0\rangle\) by auto

ultimately show \(?thesis\ by\ auto\)

qed

moreover have \(?case\ when\ \textit{pderiv}\ \(p\neq 0\)\)

proof –

define \(pp\) where \(pp = \textit{last} (\textit{smods} \(p\) (pderiv \(p\)))\)

define \(lp\) where \(lp = \textit{lead-coef f} \(pp\)\)

define \(S\) where \(S = \{x. \(a < x \land x < b\}\)\)

have \(\textit{prefix-smods-ext} \(p\) (pderiv \(p\)) = \textit{smods} \(p\) (pderiv \(p\)) @ tl (\textit{smods-ext} \(pp\) (pderiv \(pp\)))\)

using \(\textit{smods-ext-prefix}[\textit{OF} \langle p\neq 0\rangle, \textit{pderiv} \(p\neq 0\), \textit{folded} \(pp\)-def]\) .

have \(\textit{pp-gcd} \(pp\) = \textit{smult} \(lp\) (gcd \(p\) (pderiv \(p\)))\)

using \(\textit{last-smods-gcd}[\textit{OF} \langle p\neq 0\rangle, \textit{of} \(p\) \textit{deriv} \(p\), \textit{folded} \(pp\)-def \(lp\)-def]\) .

have \(p\neq 0\) \(lp\neq 0\) unfolding \(pp\)-def \(lp\)-def

subgoal by (rule \(\textit{last-smods-nzero}[\textit{OF} \langle p\neq 0\rangle])

subgoal using \(\textit{last} (\textit{smods} \(p\) (pderiv \(p\))) \neq 0\); by auto

done

have \(\textit{poly} \(pp\) a\neq 0\) \(\textit{poly} \(pp\) b \neq 0\)

un folding \(\textit{pp-gcd}\) using \(\langle \textit{poly} \(p\) a\neq 0\rangle, \langle \textit{poly} \(p\) b\neq 0\rangle, \langle lp\neq 0\rangle\)

by (simp-all add: \(\textit{poly-gcd-0-iff}\))

have \(\textit{proots-count} \(pp\) \(S\) = \textit{changes-itv-smods-ext} \(a\ \(b\ \(pp\))\) unfolding \(S\)-def

proof (rule \(1(1)[\textit{rule-format.of} \textit{smods-ext} \(pp\) (pderiv \(pp\)) \(pp\)]\)

show \(\textit{length} (\textit{smods-ext} \(pp\) (pderiv \(pp\))) < \textit{length} (\textit{smods-ext} \(p\) (pderiv \(p\)))\)

un folding \(\textit{prefix}\) by (simp add: \(\langle p \neq 0\rangle\) that)

qed (use \(\langle \textit{poly} \(pp\) a\neq 0\rangle, \langle \textit{poly} \(pp\) b\neq 0\rangle, \textit{in} \textit{simp-all})

moreover have \(\textit{proots-count} \(p\) \(S\) = \textit{card} (\textit{proots-within} \(p\) \(S\)) + \textit{proroots-count} \(pp\) \(S\)

proof –

have \((\sum r\in \textit{proots-within} \(p\) \(S\). \textit{order} \(r\) \(p\)) = (\sum r\in \textit{proots-within} \(p\) \(S\). \textit{order} \(r\) \(pp\) + 1)\)

proof (rule \textit{sum.cong})

fix \(x\) assume \(x \in \textit{proots-within} \(p\) \(S\)

have \(\textit{order} \(x\) \(pp\) = \textit{order} \(x\) (gcd \(p\) (pderiv \(p\)))\)

un folding \(\textit{pp-gcd}\) using \(\langle lp\neq 0\rangle\) by (simp add: \textit{order-smult})

also have ... = \textit{min} (\textit{order} \(x\) \(p\)) (\textit{order} \(x\) (pderiv \(p\)))

apply (subst \textit{order-gcd})
using \( p \neq 0 \) \( p \text{deriv} \neq 0 \) by simp-all
also have \( \ldots = \text{order} x \ (p \text{deriv} p) \)
  apply (subst order-pderiv)
  using \( p \text{deriv} p \neq 0 \) \( p \neq 0 \) \( x \in \text{proots-within} \ p \) order-root by auto
  finally have \( \text{order} x \ p p = \text{order} x \ (p \text{deriv} p) \).
  moreover have \( \text{order} x \ p = \text{order} x \ (p \text{deriv} p) + 1 \) by auto
  ultimately show \( \text{order} x \ p = \text{order} x \ p p + 1 \) by auto
  qed simp
also have \( \ldots = \text{card} \ (\text{proots-within} \ p \) S) + (\( \sum r \in \text{proots-within} \ p \) S. \( \text{order} r \) pp)
  apply (subst sum.distrib)
  by auto
also have \( \ldots = \text{card} \ (\text{proots-within} \ p \) S) + (\( \sum r \in \text{proots-within} \ pp \) S. \( \text{order} r \) pp)
proof 
  have \( (\sum r \in \text{proots-within} \ p \) S. \( \text{order} r \) pp) = (\( \sum r \in \text{proots-within} \ pp \) S. \( \text{order} r \) pp)
  apply (rule sum.mono-neutral-right)
  subgoal using \( p \neq 0 \) by auto
  subgoal unfolding pp-gcd using \( lp \neq 0 \) by (auto simp;poly-gcd-0-iff)
  subgoal unfolding pp-gcd using \( lp \neq 0 \) by auto
  apply (auto simp;poly-gcd-0-iff order-smult)
  apply (subst order-gcd)
  by (auto simp add: order-root)
  done
  then show \?thesis by simp
  qed
finally show \?thesis unfolding proots-count-def .
qed
moreover have \( \text{card} \ (\text{proots-within} \ p \) S) = \text{changes-itv-smods} a b p (pderiv p)
  using sturm-interval\[OF \ (a < b) \ (\text{poly} \ p \ a \neq 0) \ (\text{poly} \ p \ b \neq 0) ; \text{symmetric}]\]
  unfolding S-def proots-within-def
  by (auto intro!:arg-cong[where \( f = \text{card} \)]
moreover have \( \text{changes-itv-smods-ext} a b p (pderiv p) = \text{changes-itv-smods} a b p (pderiv p) + \text{changes-itv-smods-ext} a b pp \) (pderiv pp)
proof 
  define \( xx \ ys \ where \ xx = \text{smods} p \ (pderiv p) \) and \( ys = \text{smods-ext} pp \ (pderiv pp) \)
  have \( xxys : xx \neq [] \ ys \neq [] \) last xx=hd yy poly (last xx) \( a \neq 0 \ poly \ (last xx) \) b \( \neq 0 \)
    subgoal unfolding xx-def using \( p \neq 0 \) by auto
    subgoal unfolding ys-def using \( pp \neq 0 \) by auto
    subgoal using \( pp \neq 0 \) unfolding xx-def ys-def
      apply (fold pp-def)
    by auto
    subgoal using \( \text{poly} \ pp \ a \neq 0 \) unfolding pp-def xx-def .
    subgoal using \( \text{poly} \ pp \ b \neq 0 \) unfolding pp-def xx-def .
    done

35
have \( \text{changes-poly-at} (xs @ tl ys) a = \text{changes-poly-at} xs a + \text{changes-poly-at} ys a \)

proof –

have \( \text{changes-poly-at} (xs @ tl ys) a = \text{changes-poly-at} (xs @ ys) a \)

unfolding \( \text{changes-poly-at-def} \)

apply \( \text{simp add:map-tl} \)

apply \( \text{subst \emph{changes-drop-dup}[symmetric]} \)

using \( \text{that \emph{ys} by \emph{auto simp add: hd-map last-map}} \)

also have \( \ldots = \text{changes-poly-at} xs a + \text{changes-poly-at} ys a \)

unfolding \( \text{changes-poly-at-def} \)

apply \( \text{subst \emph{changes-append}[symmetric]} \)

using \( \text{\emph{ys} by \emph{auto simp add: hd-map last-map}} \)

finally show \( \text{?thesis} \).
qed

moreover have \( \text{changes-poly-at} (xs @ tl ys) b = \text{changes-poly-at} xs b + \text{changes-poly-at} ys b \)

proof –

have \( \text{changes-poly-at} (xs @ tl ys) b = \text{changes-poly-at} (xs @ ys) b \)

unfolding \( \text{changes-poly-at-def} \)

apply \( \text{simp add:map-tl} \)

apply \( \text{subst \emph{changes-drop-dup}[symmetric]} \)

using \( \text{that \emph{ys} by \emph{auto simp add: hd-map last-map}} \)

also have \( \ldots = \text{changes-poly-at} xs b + \text{changes-poly-at} ys b \)

unfolding \( \text{changes-poly-at-def} \)

apply \( \text{subst \emph{changes-append}[symmetric]} \)

using \( \text{\emph{ys} by \emph{auto simp add: hd-map last-map}} \)

finally show \( \text{?thesis} \).
qed

ultimately show \( \text{?thesis} \)
unfolding \( \text{changes-itv-smods-ext-def changes-itv-smods-def} \)

apply \( \text{fold \emph{xs-def \emph{ys-def}}, unfold \emph{prefix[folded \emph{xs-def \emph{ys-def}] Let-def}}} \)

by \emph{auto}

qed

ultimately show \( \text{proots-count} p S = \text{changes-itv-smods-ext} a b \text{ (pderiv \emph{p})} \)

by \emph{auto}

qed

ultimately show \( \text{?case by \emph{argo}} \)
qed

theorem \emph{sturm-extend-above}:

assumes \( \text{poly} \emph{p} \emph{a} \neq 0 \)

shows \( \text{proots-count} \emph{p} \{x. \emph{a} < x\} = \text{changes-gt-smods-ext} \emph{a} \emph{p} \) (pderiv \emph{p})

proof –

define \emph{ps} where \( \emph{ps} \equiv \text{smods-ext} \emph{p} \) (pderiv \emph{p})

have \( \emph{p} \neq 0 \) and \( \emph{p} \in \text{set} \emph{ps} \) using \( \text{poly} \emph{p} \emph{a} \neq 0 \) \emph{ps-def} by \emph{auto}

obtain \emph{ub} where \( \emph{ub} : \forall \emph{p} \in \text{set} \emph{ps} \\forall x. \text{poly} \emph{p} x = 0 \rightarrow x < \emph{ub} \)

and \( \emph{ub-sgn} : \forall x \geq \emph{ub} . \forall \emph{p} \in \text{set} \emph{ps} . \text{sgn} \ (\text{poly} \emph{p} x) = \text{sgn-pos-inf} \emph{p} \)

and \( \emph{ub} > \emph{a} \)

using \( \text{root-list-ub[\emph{OF \text{no-0-in-smods-extend},of \emph{p} pderiv \emph{p},folded \emph{ps-def}]} \) \)

by \emph{auto}

36
have \( \text{proots-count} \ p \ \{ x. \ a < x \} = \text{proots-count} \ p \ \{ x. \ a < x \land x < \text{ub} \} \)

unfolding \( \text{proots-count-def} \)
apply (rule sum.cong)
by (use \( \text{ub} \ \{ p \in \text{set} \ ps \} \) in auto)
moreover have \( \text{changes-gl-smods-ext} a \ p \ (pderiv \ p) = \text{changes-itv-smods-ext} a \ \text{ub} \ p \ (pderiv \ p) \)

proof –

\([\text{have map (sgn o (\lambda p. poly p \text{ub})) ps = map sgn-pos-inf ps using ub-sgn[THEN spec,of ub,simplified]}]
\)

hence unfolding \( \text{changes-poly-pos-inf-def changes-poly-at-def} \)
by (subst changes-map-sgn-eq, metis map-map)
thus \(?thesis \) unfolding \( \text{changes-gl-smods-ext-def changes-itv-smods-ext-def} \) \( \text{ps-def} \)
by metis
qed

moreover have \( \text{poly} \ p \ \text{ub} \neq 0 \) using \( \text{ub} \ \{ p \in \text{set} \ ps \} \) by auto
ultimately show \(?thesis \) using \( \text{sturm-ext-interval[OF} \ \langle \text{ub} > \text{a}\rangle \ \text{assms}] \) by auto
qed

theorem \( \text{sturm-ext-below} : \)
assumes \( \text{poly} \ p \ \text{b} \neq 0 \)
shows \( \text{proots-count} \ p \ \{ x. \ x < \text{b} \} = \text{changes-le-smods-ext} \ b \ p \ (pderiv \ p) \)

proof –

define \( \text{ps} \ where \ \text{ps}\equiv\text{smods-ext} \ p \ (pderiv \ p) \)

have \( p \neq 0 \) and \( p \in \text{set} \ \text{ps} \) using \( \langle \text{poly} \ p \ \text{b} \neq 0 \rangle ; \ \text{ps-def} \) by auto
obtain \( \text{lb} \ where \ \text{lb}\equiv\forall p \in \text{set} \ \text{ps}. \ \forall x. \ \text{poly} \ p \ x = 0 \rightarrow x > \text{lb} \)
\( \text{and} \ \text{lb-sgn}\equiv\forall x \leq \text{lb}. \ \forall p \in \text{set} \ \text{ps}. \ \text{sgn} \ (\text{poly} \ p \ x) = \text{sgn-neg-inf} \ p \)
\( \text{and} \ \text{lb}<\text{b} \)
using root-list-lb[\( \text{OF no-0-in-smods-ext,of} \ \langle \text{pderiv} \ p,\text{folded ps-def} \rangle \)]
by auto

have \( \text{proots-count} \ p \ \{ x. \ x < \text{b} \} = \text{proots-count} \ p \ \{ x. \ \text{lb} < x \land x < \text{b} \} \)

unfolding \( \text{proots-count-def by} \) (rule sum.cong,insert \( \langle \text{p} \in \text{set} \ \text{ps},\text{auto} \rangle \))
moreover have \( \text{changes-le-smods-ext} \ b \ p \ (pderiv \ p) = \text{changes-itv-smods-ext} \ \text{lb} \ b \ p \ (pderiv \ p) \)

proof –

\([\text{have map (sgn o (\lambda p. poly p \text{lb})) ps = map sgn-neg-inf ps using lb-sgn[THEN spec,of lb,simplified]}]
\)

hence unfolding \( \text{changes-poly-neg-inf-def changes-poly-at-def} \)
by (subst changes-map-sgn-eq, metis map-map)
thus \(?thesis \) unfolding \( \text{changes-le-smods-ext-def changes-itv-smods-ext-def ps-def} \)
by metis
qed

moreover have \( \text{poly} \ p \ \text{lb} \neq 0 \) using \( \text{lb} \ \{ p \in \text{set} \ \text{ps} \} \) by auto
ultimately show \(?thesis \) using \( \text{sturm-ext-interval[OF} \ \langle \text{lb} < \text{b}\rangle - \text{assms}] \) by auto
qed

37
theorem sturm-ext-R:
  assumes p ≠ 0
  shows proots-count p UNIV = changes-R-smods-ext p (pderiv p)
proof -
  define ps where ps ≡ smods-ext p (pderiv p)
  have p ∈ set ps using ps-def ⟨ p ≠ 0 ⟩ by auto
  obtain lb where lb: ∀ p ∈ set ps. ∀ x. poly p x = 0 → x > lb
    and lb-sgn: ∀ x ≤ lb. ∀ p ∈ set ps. sgn (poly p x) = sgn-neg-inf p
    and lb < 0
      using root-list-lb[OF no-0-in-smods-ext, of p pderiv p, folded ps-def]
      by auto
  obtain ub where ub: ∀ p ∈ set ps. ∀ x. poly p x = 0 → x < ub
    and ub-sgn: ∀ x ≥ ub. ∀ p ∈ set ps. sgn (poly p x) = sgn-pos-inf p
    and ub > 0
      using root-list-ub[OF no-0-in-smods-ext, of p pderiv p, folded ps-def]
      by auto
  have proots-count p UNIV = proots-count p { x. lb < x ∧ x < ub }
    unfolding proots-count-def by (rule sum.cong, insert lb ub ⟨ p ∈ set ps ⟩, auto)
  moreover have changes-R-smods-ext p (pderiv p) = changes-itv-smods-ext lb ub p (pderiv p)
    proof -
      have map (sgn o (λ p. poly p lb)) ps = map sgn-neg-inf ps
        and map (sgn o (λ p. poly p ub)) ps = map sgn-pos-inf ps
        using lb-sgn[THEN spec, of lb, simplified] ub-sgn[THEN spec, of ub, simplified]
        by (metis mono-tags, lifting, comp-def list.map-cong)+
      hence changes-poly-at ps lb = changes-poly-neg-inf ps
        ∧ changes-poly-at ps ub = changes-poly-pos-inf ps
        unfolding changes-poly-neg-inf-def changes-poly-at-def changes-poly-pos-inf-def
        by (subst (1 3) changes-map-sgn-eq, metis map-map)
      thus ?thesis unfolding changes-R-smods-ext-def changes-itv-smods-ext-def ps-def
        by metis
      qed
  moreover have poly p lb ≠ 0 and poly p ub ≠ 0 using lb ub ⟨ p ∈ set ps ⟩ by auto
  moreover have lb < ub using ⟨ lb < 0 ⟩ ⟨ 0 < ub ⟩ by auto
  ultimately show ?thesis using sturm-ext-interval by auto
qed

end

References


opment.


