

The Budan–Fourier Theorem and Counting Real Roots with Multiplicity

Wenda Li

June 15, 2026

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 Proof-Power¹). 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 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)  
by (induct xs) auto
```

```
lemma changes-filter-eq:  
  changes (filter (λx. x≠0) xs) = changes xs  
by (induct xs) (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 xs)
  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-butlast-last-id append-is-Nil-conv
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 (λx. x ≠ 0) ys = filter (λx. x ≠ 0) (tl ys)
        by (metis (mono-tags, lifting) filter.simps(2) list.exhaust-sel 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 changes (xs @ ys) = changes (xs @ tl ys)
  apply (rule Cons(1))
  using that Cons(3) by auto
  moreover have changes (a # xs @ ys) = (if ts = [] then 0 else if a * hd ts <
0

```

then $1 + \text{changes } (xs @ ys)$ else $\text{changes } (xs @ ys)$
using $\text{changes-Cons}[of a xs @ ys, folded ts-def]$.
moreover have $\text{changes } (a \# xs @ tl ys) = (\text{if } ts' = [] \text{ then } 0 \text{ else if } a * hd ts' < 0$
 then $1 + \text{changes } (xs @ tl ys)$ else $\text{changes } (xs @ tl ys)$
using $\text{changes-Cons}[of a xs @ tl ys, folded ts'-def]$.
ultimately show *?thesis* **by** *auto*
qed
ultimately show *?case* **by** *blast*
qed

lemma *Im-poly-of-real*:
 $Im (\text{poly } p (\text{of-real } x)) = \text{poly } (\text{map-poly } Im \ p) \ x$
apply (*induct p*)
by (*auto simp add: map-poly-pCons*)

lemma *Re-poly-of-real*:
 $Re (\text{poly } p (\text{of-real } x)) = \text{poly } (\text{map-poly } Re \ p) \ x$
apply (*induct p*)
by (*auto simp add: map-poly-pCons*)

1.2 More about *map-poly* and *of-real*

lemma *of-real-poly-map-pCons[simp]*: $\text{map-poly of-real } (pCons \ a \ p) = pCons (\text{of-real } a) (\text{map-poly of-real } p)$
by (*simp add: map-poly-pCons*)

lemma *of-real-poly-map-plus[simp]*: $\text{map-poly of-real } (p + q) = \text{map-poly of-real } p + \text{map-poly of-real } q$
apply (*rule poly-eqI*)
by (*auto simp add: coeff-map-poly*)

lemma *of-real-poly-map-smult[simp]*: $\text{map-poly of-real } (smult \ s \ p) = smult (\text{of-real } s) (\text{map-poly of-real } p)$
apply (*rule poly-eqI*)
by (*auto simp add: coeff-map-poly*)

lemma *of-real-poly-map-mult[simp]*: $\text{map-poly of-real } (p * q) = \text{map-poly of-real } p * \text{map-poly of-real } q$
by (*induct p, intro poly-eqI, auto*)

lemma *of-real-poly-map-poly*:
 $\text{of-real } (\text{poly } p \ x) = \text{poly } (\text{map-poly of-real } p) (\text{of-real } x)$
by (*induct p, auto*)

lemma *of-real-poly-map-power*: $\text{map-poly of-real } (p \hat{=} n) = (\text{map-poly of-real } p) \hat{=} n$
by (*induct n, auto*)

lemma *of-real-poly-eq-iff* [*simp*]: *map-poly of-real p = map-poly of-real q* \longleftrightarrow *p = q*
by (*auto simp: poly-eq-iff coeff-map-poly*)

lemma *of-real-poly-eq-0-iff* [*simp*]: *map-poly of-real p = 0* \longleftrightarrow *p = 0*
by (*auto simp: poly-eq-iff coeff-map-poly*)

1.3 More about *order*

lemma *order-multiplicity-eq*:
assumes *p* $\neq 0$
shows *order a p = multiplicity* $[-a, 1:]$ *p*
by (*metis assms multiplicity-eqI order-1 order-2*)

lemma *order-gcd*:
assumes *p* $\neq 0$ *q* $\neq 0$
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

lemma *map-poly-order-of-real*:
assumes *p* $\neq 0$
shows *order (of-real t) (map-poly of-real p) = order t p* **using** *assms*
proof (*induct p rule: poly-root-induct-alt [of - $\lambda x. True$]*)
case *0*
then show *?case* **by** *simp*

next

case (*no-roots p*)
then have *order t p = 0* **using** *order-root* **by** *blast*
moreover have *poly (map-poly of-real p) (of-real x) $\neq 0$* **for** *x*
apply (*subst of-real-poly-map-poly[symmetric]*)
using *no-roots order-root* **by** *simp*
then have *order (of-real t) (map-poly of-real p) = 0*
using *order-root* **by** *blast*
ultimately show *?case* **by** *auto*

next

case (*root a p*)
define *a1* **where** *a1* $[-a, 1:]$
have [*simp*]: *a1* $\neq 0$ *p* $\neq 0$ **unfolding** *a1-def* **using** *root* **by** *auto*
have *order (of-real t) (map-poly of-real a1) = order t a1*

```

    unfolding a1-def by simp
  then show ?case
    apply (fold a1-def)
    by (simp add:order-mult root)
qed

lemma order-pcompose:
  assumes pcompose p q ≠ 0
  shows order x (pcompose p q) = order x (q - [:poly q x:]) * order (poly q x) p
  using ⟨pcompose p q ≠ 0⟩
proof (induct p rule: poly-root-induct-alt [of - λx. True])
  case 0
  then show ?case by simp
next
  case (no-roots p)
  have order x (p ∘p q) = 0
  apply (rule order-0I)
  using no-roots by (auto simp:poly-pcompose)
  moreover have order (poly q x) p = 0
  apply (rule order-0I)
  using no-roots by (auto simp:poly-pcompose)
  ultimately show ?case by auto
next
  case (root a p)
  define a1 where a1 = [-a, 1:]
  obtain [simp]: a1 ≠ 0 p ≠ 0 a1 ∘p q ≠ 0 p ∘p q ≠ 0
  using root a1-def
  by (metis (no-types, opaque-lifting) mult-zero-left mult-zero-right pcompose-0
  pcompose-mult)
  have order x ((a1 * p) ∘p q) = order x (a1 ∘p q) + order x (p ∘p q)
  unfolding pcompose-mult by (auto simp: order-mult)
  also have ... = order x (q - [:poly q x:]) * (order (poly q x) a1 + order (poly q
  x) p)
  proof -
    have order x (a1 ∘p q) = order x (q - [:poly q x:]) * order (poly q x) a1
    unfolding a1-def
    apply (auto simp: pcompose-pCons algebra-simps diff-conv-add-uminus )
    by (simp add: order-0I)
    moreover have order x (p ∘p q) = order x (q - [:poly q x:]) * order (poly q
  x) p
    apply (rule root.hyps)
    by auto
    ultimately show ?thesis by (auto simp:algebra-simps)
  qed
  also have ... = order x (q - [:poly q x:]) * order (poly q x) (a1 * p)
  by (auto simp:order-mult)
  finally show ?case unfolding a1-def .
qed

```

1.4 Polynomial roots / zeros

definition *proots-within*::'a::comm-semiring-0 poly \Rightarrow 'a set \Rightarrow 'a set **where**
proots-within p s = {x \in s. poly p x=0}

abbreviation *proots*::'a::comm-semiring-0 poly \Rightarrow 'a set **where**
proots p \equiv *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-withinI[intro,simp]*:
poly p x=0 \implies x \in s \implies x \in *proots-within* p s
unfolding *proots-within-def* **by** auto

lemma *proots-within-iff[simp]*:
x \in *proots-within* p s \iff poly p x=0 \wedge x \in s
unfolding *proots-within-def* **by** auto

lemma *proots-within-union*:
proots-within p A \cup *proots-within* p B = *proots-within* p (A \cup 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 \cup *proots-within* q s
unfolding *proots-within-def* **by** auto

lemma *proots-within-gcd*:
fixes s::'a::{factorial-ring-gcd,semiring-gcd-mult-normalize} set
shows *proots-within* (gcd p q) s = *proots-within* p s \cap *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 \implies *proots-within* p s = *proots* p \cap s
unfolding *proots-within-def* **by** auto

lemma *proots-within-proots[simp]*:
proots-within p s \subseteq *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 $\text{proots-within } [-a, 1:] \ s = (\text{if } a \in s \text{ then } \{a\} \text{ else } \{\})$
 $\text{proots-within } [a, -1:] \ s = (\text{if } a \in s \text{ then } \{a\} \text{ else } \{\})$
by (*cases a ∈ s, auto*)

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

lemma *proots-within-smult*:
fixes $a :: 'a :: \{\text{semiring-no-zero-divisors, comm-semiring-0}\}$
assumes $a \neq 0$
shows $\text{proots-within } (\text{smult } a \ p) \ s = \text{proots-within } p \ s$
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**
 $\text{proots-count } p \ s = (\sum r \in \text{proots-within } p \ s. \text{order } r \ p)$

lemma *proots-count-empty[simp]*: $\text{proots-count } p \ \{\} = 0$
unfolding *proots-count-def* **by** *auto*

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

apply (*fold pts-def qts-def*)
by auto
qed

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

lemma *degree-proots-count*:
fixes $p :: \text{complex poly}$
shows $\text{degree } p = \text{proots-count } p \text{ UNIV}$
proof (*induct degree p arbitrary:p*)
case 0
then obtain c **where** $c\text{-def}: p = [:c:]$ **using** *degree-eq-zeroE* **by auto**
then show *?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 $\text{degree } p \neq 0$ **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[of p]*
by auto
define onez **where** $\text{onez} = [: - z, 1:]$
have [*simp*]: $\text{onez} \neq 0$ $\text{degree onez} = 1$ **unfolding** *onez-def* **by auto**
obtain q **where** $q\text{-def}: p = \text{onez} * q$
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 $\text{degree } q = \text{proots-count } q \text{ UNIV}$ **using** *Suc.hyps(1)* **by simp**
moreover have $\text{Suc } 0 = \text{proots-count onez UNIV}$
unfolding *onez-def* **using** *proots-count-power-n-n[of z 1 UNIV]*
by auto
ultimately show *?case*
unfolding $q\text{-def}$ **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 proots-count (smult a p) s = 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 proots-count  $[: - a, 1:]$  s = (if a ∈ s then 1 else 0)
  unfolding proots-count-def
  by (cases a ∈ s, auto simp add: proots-within-pCons-1-iff order-power-n-n[of - 1, simplified])

```

```

lemma proots-count-uminus[simp]:
  proots-count ( $- p$ ) s = proots-count p s
  unfolding proots-count-def by simp

```

```

lemma card-proots-within-leq:
  assumes  $p \neq 0$ 
  shows proots-count p s  $\geq$  card (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 proots-within p s = {} by auto
  then show ?case unfolding proots-count-def by auto
next
  case (root a p)
  have card (proots-within ( $[: - a, 1:] * p$ ) s)
     $\leq$  card (proots-within  $[: - a, 1:]$  s) + card (proots-within p s)
    unfolding proots-within-times by (auto simp add: card-Un-le)
  also have ...  $\leq 1 +$  proots-count p s
proof -
  have card (proots-within  $[: - a, 1:]$  s)  $\leq 1$ 
proof (cases a ∈ s)
  case True
  then have proots-within  $[: - a, 1:]$  s = {a} by auto
  then show ?thesis by auto
next
  case False
  then have proots-within  $[: - a, 1:]$  s = {} by auto
  then show ?thesis by auto
qed

```

```

moreover have card (proots-within p s) ≤ proots-count p s
apply (rule root.hyps)
using root by auto
ultimately show ?thesis by auto
qed
also have ... = proots-count ([: - a, 1:] * p) s
apply (subst proots-count-times)
subgoal by (metis mult-eq-0-iff pCons-eq-0-iff root.prem1 zero-neq-one)
using root by (auto simp add:proots-count-pCons-1-iff)
finally have card (proots-within ([: - a, 1:] * p) s) ≤ proots-count ([: - a, 1:] *
p) s .
then show ?case
by (metis (no-types, opaque-lifting) add.inverse-inverse add.inverse-neutral mi-
nus-pCons
mult-minus-left proots-count-uminus proots-within-uminus)
qed

```

```

lemma proots-count-0-imp-empty:
assumes proots-count p s = 0 p ≠ 0
shows proots-within p s = {}
proof -
have card (proots-within p s) = 0
using card-proots-within-leq[OF ⟨p ≠ 0⟩, of s] ⟨proots-count p s = 0⟩ by auto
moreover have finite (proots-within p s) using ⟨p ≠ 0⟩ by auto
ultimately show ?thesis by auto
qed

```

```

lemma proots-count-leq-degree:
assumes p ≠ 0
shows proots-count p s ≤ degree p using assms
proof (induct rule:poly-root-induct[of - λx. x ∈ s])
case 0
then show ?case by auto
next
case (no-roots p)
then have proots-within p s = {} by auto
then show ?case unfolding proots-count-def by auto
next
case (root a p)
have proots-count ([: a, - 1:] * p) s = proots-count [: a, - 1:] s + proots-count p
s
apply (subst proots-count-times)
using root by auto
also have ... = 1 + proots-count p s
proof -
have proots-count [: a, - 1:] s = 1
by (metis (no-types, lifting) add.inverse-inverse add.inverse-neutral mi-
nus-pCons)

```

```

      roots-count-pCons-1-iff roots-count-uminus root.hyps(1))
    then show ?thesis by auto
  qed
  also have ... ≤ degree ([:a,-1:] * p)
    apply (subst degree-mult-eq)
    subgoal by auto
    subgoal using root by auto
    subgoal using root by (simp add: ⟨p ≠ 0⟩)
  done
  finally show ?case .
qed

```

```

lemma roots-count-union-disjoint:
  assumes A ∩ B = {} p≠0
  shows roots-count p (A ∪ B) = roots-count p A + roots-count p B
  unfolding roots-count-def
  apply (subst roots-within-union[symmetric])
  apply (subst sum.union-disjoint)
  using assms by auto

```

```

lemma roots-count-cong:
  assumes order-eq:∀ x∈s. order x p = order x q and p≠0 and q≠0
  shows roots-count p s = roots-count q s unfolding roots-count-def
proof (rule sum.cong)
  have poly p x = 0 ⟷ poly q x = 0 when x∈s for x
    using order-eq that by (simp add: assms(2) assms(3) order-root)
  then show roots-within p s = roots-within q s by auto
  show ∧x. x ∈ roots-within q s ⟹ order x p = order x q
    using order-eq by auto
qed

```

```

lemma roots-count-of-real:
  assumes p≠0
  shows roots-count (map-poly of-real p) ((of-real::->'a::{real-algebra-1,idom}) '
s)
    = roots-count p s

```

```

proof -
  define k where k=(of-real::->'a)
  have roots-within (map-poly of-real p) (k ' s) =k ' (roots-within p s)
    unfolding roots-within-def k-def by (auto simp add: of-real-poly-map-poly[symmetric])
  then have roots-count (map-poly of-real p) (k ' s)
    = (∑ r∈k ' (roots-within p s). order r (map-poly of-real p))
    unfolding roots-count-def by simp
  also have ... = sum ((λr. order r (map-poly of-real p)) ∘ k) (roots-within p s)
    apply (subst sum.reindex)
    unfolding k-def by (auto simp add: inj-on-def)
  also have ... = roots-count p s unfolding roots-count-def
    apply (rule sum.cong)

```

```

unfolding k-def comp-def using ⟨p≠0⟩ by (auto simp add:map-poly-order-of-real)

finally show ?thesis unfolding k-def .
qed

lemma roots-pcompose:
  fixes p q::'a::field poly
  assumes p≠0 degree q=1
  shows roots-count (pcompose p q) s = roots-count p (poly q ' s)
proof –
  obtain a b where ab:q=[:a,b:] b≠0
    using ⟨degree q=1⟩ degree1-coeffs by metis
  define f where f=(λy. (y-a)/b)
  have f-eq:f (poly q x) = x poly q (f x) = x for x
    unfolding f-def using ab by auto
  have roots-count (p ∘p q) s = (∑ r∈ f ' roots-within p (poly q ' s). order r (p
    ∘p q))
    unfolding roots-count-def
    apply (rule arg-cong2[where f =sum])
    apply (auto simp:poly-pcompose roots-within-def f-eq)
    by (metis (mono-tags, lifting) f-eq(1) image-eqI mem-Collect-eq)
  also have ... = (∑ x∈roots-within p (poly q ' s). order (f x) (p ∘p q))
    apply (subst sum.reindex)
    subgoal unfolding f-def inj-on-def using ⟨b≠0⟩ by auto
    by simp
  also have ... = (∑ x∈roots-within p (poly q ' s). order x p)
proof –
  have p ∘p q ≠ 0 using assms(1) assms(2) pcompose-eq-0 by force
  moreover have order (f x) (q - [:x:]) = 1 for x
proof –
  have order (f x) (q - [:x:]) = order (f x) (smult b [:-((x - a) / b),1:])
    unfolding f-def using ab by auto
  also have ... = 1
    apply (subst order-smult)
    using ⟨b≠0⟩ unfolding f-def by auto
  finally show ?thesis .
qed
  ultimately have order (f x) (p ∘p q) = order x p for x
    apply (subst order-pcompose)
    using f-eq by auto
  then show ?thesis by auto
qed
  also have ... = roots-count p (poly q ' s)
    unfolding roots-count-def by auto
  finally show ?thesis .
qed

```

1.6 Composition of a polynomial and a rational function

definition $fcompose::'a ::field poly \Rightarrow 'a poly \Rightarrow 'a poly \Rightarrow 'a poly$ **where**
 $fcompose\ p\ q\ r = fst\ (fold-coeffs\ (\lambda a\ (c,d).\ (d*[:a:] + q * c,r*d))\ p\ (0,1))$

lemma $fcompose-0$ [*simp*]: $fcompose\ 0\ q\ r = 0$
by (*simp add: fcompose-def*)

lemma $fcompose-const$ [*simp*]: $fcompose\ [:a:]\ q\ r = [:a:]$
unfolding $fcompose-def$ **by** (*cases a=0*) *auto*

lemma $fcompose-pCons$:

$fcompose\ (pCons\ a\ p)\ q1\ q2 = smult\ a\ (q2^\wedge(degree\ (pCons\ a\ p))) + q1 * fcompose\ p\ q1\ q2$

proof (*cases p=0*)

case *False*

define ff **where** $ff = (\lambda a\ (c, d).\ (d * [:a:] + q1 * c, q2 * d))$

define fc **where** $fc = fold-coeffs\ ff\ p\ (0, 1)$

have $snd\ ff : snd\ fc = (if\ p=0\ then\ 1\ else\ q2^\wedge(degree\ p + 1))$ **unfolding** $fc-def$

apply (*induct p*)

subgoal **by** *simp*

subgoal **for** $a\ p$

by (*auto simp add: ff-def split:if-splits prod.splits*)

done

have $fcompose\ (pCons\ a\ p)\ q1\ q2 = fst\ (fold-coeffs\ ff\ (pCons\ a\ p)\ (0, 1))$

unfolding $fcompose-def\ ff-def$ **by** *simp*

also **have** $... = fst\ (ff\ a\ fc)$

using *False* **unfolding** $fc-def$ **by** *auto*

also **have** $... = snd\ fc * [:a:] + q1 * fst\ fc$

unfolding $ff-def$ **by** (*auto split:prod.splits*)

also **have** $... = smult\ a\ (q2^\wedge(degree\ (pCons\ a\ p))) + q1 * fst\ fc$

using $snd\ ff\ False$ **by** *auto*

also **have** $... = smult\ a\ (q2^\wedge(degree\ (pCons\ a\ p))) + q1 * fcompose\ p\ q1\ q2$

unfolding $fc-def\ ff-def\ fcompose-def$ **by** *simp*

finally **show** *?thesis* .

qed *simp*

lemma $fcompose-uminus$:

$fcompose\ (-p)\ q\ r = - fcompose\ p\ q\ r$

by (*induct p*) (*auto simp:fcompose-pCons*)

lemma $fcompose-add-less$:

assumes $degree\ p1 > degree\ p2$

shows $fcompose\ (p1+p2)\ q1\ q2$

$= fcompose\ p1\ q1\ q2 + q2^\wedge(degree\ p1 - degree\ p2) * fcompose\ p2\ q1\ q2$

using *assms*

proof (*induction p1 p2 rule: poly-induct2*)

case ($pCons\ a1\ p1\ a2\ p2$)

have *?case* **when** $p2=0$

using that by (*simp add:fcompose-pCons smult-add-left*)
moreover have ?case **when** $p2 \neq 0 \rightarrow \text{degree } p2 < \text{degree } p1$
using that *pCons(2)* **by** *auto*
moreover have ?case **when** $p2 \neq 0 \text{ degree } p2 < \text{degree } p1$
proof –
define *d1 d2* **where** $d1 = \text{degree } (pCons \ a1 \ p1)$ **and** $d2 = \text{degree } (pCons \ a2 \ p2)$
define *fp1 fp2* **where** $fp1 = fcompose \ p1 \ q1 \ q2$ **and** $fp2 = fcompose \ p2 \ q1 \ q2$

have $fcompose \ (pCons \ a1 \ p1 + pCons \ a2 \ p2) \ q1 \ q2$
 $= fcompose \ (pCons \ (a1 + a2) \ (p1 + p2)) \ q1 \ q2$
by *simp*
also have $\dots = smult \ (a1 + a2) \ (q2 \wedge d1) + q1 * fcompose \ (p1 + p2) \ q1 \ q2$
proof –
have $\text{degree } (pCons \ (a1 + a2) \ (p1 + p2)) = d1$
unfolding *d1-def* **using that** *degree-add-eq-left* **by** *fastforce*
then show ?thesis **unfolding** *fcompose-pCons* **by** *simp*
qed
also have $\dots = smult \ (a1 + a2) \ (q2 \wedge d1) + q1 * (fp1 + q2 \wedge (d1 - d2) * fp2)$
proof –
have $\text{degree } p1 - \text{degree } p2 = d1 - d2$
unfolding *d1-def d2-def* **using that** **by** *simp*
then show ?thesis
unfolding *pCons(1)[OF that(2),folded fp1-def fp2-def]* **by** *simp*
qed
also have $\dots = fcompose \ (pCons \ a1 \ p1) \ q1 \ q2 + q2 \wedge (d1 - d2) * fcompose \ (pCons \ a2 \ p2) \ q1 \ q2$
proof –
have $d1 > d2$ **unfolding** *d1-def d2-def* **using that** **by** *auto*
then show ?thesis
unfolding *fcompose-pCons*
apply (*fold d1-def d2-def fp1-def fp2-def*)
by (*simp add:algebra-simps smult-add-left power-add[symmetric]*)
qed
finally show ?thesis **unfolding** *d1-def d2-def* .
qed
ultimately show ?case **by** *blast*
qed *simp*

lemma *fcompose-add-eq*:
assumes $\text{degree } p1 = \text{degree } p2$
shows $q2 \wedge (\text{degree } p1 - \text{degree } (p1 + p2)) * fcompose \ (p1 + p2) \ q1 \ q2$
 $= fcompose \ p1 \ q1 \ q2 + fcompose \ p2 \ q1 \ q2$
using *assms*
proof (*induction p1 p2 rule: poly-induct2*)
case (*pCons a1 p1 a2 p2*)
have ?case **when** $p1 + p2 = 0$
proof –
have $p2 = -p1$ **using that** **by** *algebra*

```

then show ?thesis by (simp add:fcompose-pCons fcompose-uminus smult-add-left)
qed
moreover have ?case when p1=0
proof -
  have p2=0
  using pCons(2) that by (auto split:if-splits)
  then show ?thesis using that by simp
qed
moreover have ?case when p1≠0 p1+p2≠0
proof -
  define d1 d2 dp where d1=degree (pCons a1 p1) and d2=degree (pCons a2
p2)
  and dp = degree p1 - degree (p1+p2)
  define fp1 fp2 where fp1= fcompose p1 q1 q2 and fp2=fcompose p2 q1 q2

  have q2 ^ (degree (pCons a1 p1) - degree (pCons a1 p1 + pCons a2 p2)) *
    fcompose (pCons a1 p1 + pCons a2 p2) q1 q2
    = q2 ^ dp * fcompose (pCons (a1+a2) (p1 +p2)) q1 q2
  unfolding dp-def using that by auto
  also have ... = smult (a1 + a2) (q2 ^ d1) + q1 * (q2 ^ dp * fcompose (p1 +
p2) q1 q2)
  proof -
    have degree p1 ≥ degree (p1 + p2)
    by (metis degree-add-le degree-pCons-eq-if not-less-eq-eq order-refl pCons.prem
zero-le)
    then show ?thesis
    unfolding fcompose-pCons dp-def d1-def using that
    by (simp add:algebra-simps power-add[symmetric])
  qed
  also have ... = smult (a1 + a2) (q2 ^ d1) + q1 * (fp1 + fp2)
  apply (subst pCons(1)[folded dp-def fp1-def fp2-def])
  subgoal by (metis degree-pCons-eq-if diff-Suc-Suc diff-zero not-less-eq-eq
pCons.prem
zero-le)
  subgoal by simp
  done
  also have ... = fcompose (pCons a1 p1) q1 q2 + fcompose (pCons a2 p2) q1
q2
  proof -
    have *:d1 = degree (pCons a2 p2)
    unfolding d1-def using pCons(2) by simp
    show ?thesis
    unfolding fcompose-pCons
    apply (fold d1-def fp1-def fp2-def *)
    by (simp add:smult-add-left algebra-simps)
  qed
  finally show ?thesis .
qed
ultimately show ?case by blast
qed simp

```

lemma *fcompose-add-const*:
 $fcompose ([:a:] + p) q1 q2 = smult a (q2 \wedge degree p) + fcompose p q1 q2$
apply (*cases p*)
by (*auto simp add:fcompose-pCons smult-add-left*)

lemma *fcompose-smult*: $fcompose (smult a p) q1 q2 = smult a (fcompose p q1 q2)$
by (*induct p*) (*simp-all add:fcompose-pCons smult-add-right*)

lemma *fcompose-mult*: $fcompose (p1*p2) q1 q2 = fcompose p1 q1 q2 * fcompose p2 q1 q2$
proof (*induct p1*)
case 0
then show ?*case* **by** *simp*
next
case (*pCons a p1*)
have ?*case* **when** $p1=0 \vee p2=0$
using *that* **by** (*auto simp add:fcompose-smult*)
moreover have ?*case* **when** $p1 \neq 0 \wedge p2 \neq 0 \wedge a=0$
using *that* **by** (*simp add:fcompose-pCons pCons*)
moreover have ?*case* **when** $p1 \neq 0 \wedge p2 \neq 0 \wedge a \neq 0$
proof –
have $fcompose (pCons a p1 * p2) q1 q2$
 $= fcompose (pCons 0 (p1 * p2) + smult a p2) q1 q2$
by (*simp add:algebra-simps*)
also have ... = $fcompose (pCons 0 (p1 * p2)) q1 q2$
 $+ q2 \wedge (degree p1 + 1) * fcompose (smult a p2) q1 q2$
proof –
have $degree (pCons 0 (p1 * p2)) > degree (smult a p2)$
using *that* **by** (*simp add: degree-mult-eq*)
from *fcompose-add-less*[*OF this, of q1 q2*] *that*
show ?*thesis* **by** (*simp add:degree-mult-eq*)
qed
also have ... = $fcompose (pCons a p1) q1 q2 * fcompose p2 q1 q2$
using *that* **by** (*simp add:fcompose-pCons fcompose-smult pCons algebra-simps*)
finally show ?*thesis* .
qed
ultimately show ?*case* **by** *blast*
qed

lemma *fcompose-poly*:
assumes *poly q2 x* $x \neq 0$
shows $poly p (poly q1 x / poly q2 x) = poly (fcompose p q1 q2) x / poly (q2 \wedge (degree p)) x$
apply (*induct p*)
using *assms* **by** (*simp-all add:fcompose-pCons field-simps*)

lemma *poly-fcompose*:
assumes *poly q2 x* $x \neq 0$

```

    shows poly (fcompose p q1 q2) x = poly p (poly q1 x / poly q2 x) * (poly q2
x)^(degree p)
    using fcompose-poly[OF assms] assms by (auto simp add:field-simps)
lemma poly-fcompose-0-denominator:
  assumes poly q2 x=0
  shows poly (fcompose p q1 q2) x = poly q1 x ^ degree p * lead-coeff p
  apply (induct p)
  using assms by (auto simp add:fcompose-pCons)

lemma fcompose-0-denominator:fcompose p q1 0 = smult (lead-coeff p) (q1^degree
p)
  apply (induct p)
  by (auto simp:fcompose-pCons)

lemma fcompose-nzero:
  fixes p::'a::field poly
  assumes p≠0 and q2≠0 and nconst:∀ c. q1 ≠ smult c q2
    and infi:infinite (UNIV::'a set)
  shows fcompose p q1 q2 ≠ 0 using ⟨p≠0⟩
proof (induct p rule: poly-root-induct-alt [of - λx. True])
  case 0
  then show ?case by simp
next
  case (no-roots p)
  have False when fcompose p q1 q2 = 0
  proof -
    obtain x where poly q2 x≠0
    proof -
      have finite (proots q2) using ⟨q2≠0⟩ by auto
      then have ∃ x. poly q2 x≠0
        by (meson UNIV-I ex-new-if-finite infi proots-withinI)
      then show ?thesis using that by auto
    qed
    define y where y = poly q1 x / poly q2 x
    have poly p y = 0
      using ⟨fcompose p q1 q2 = 0⟩ fcompose-poly[OF ⟨poly q2 x≠0⟩,of p q1,folded
y-def]
      by simp
    then show False using no-roots(1) by auto
  qed
  then show ?case by auto
next
  case (root a p)
  have fcompose [:- a, 1:] q1 q2 ≠ 0
    unfolding fcompose-def using nconst[rule-format,of a]
    by simp
  moreover have fcompose p q1 q2 ≠ 0
    using root by fastforce
  ultimately show ?case unfolding fcompose-mult by auto

```

qed

1.7 Bijection (*bij-betw*) and the number of polynomial roots

lemma *proots-fcompose-bij-eq*:

```
fixes p::'a::field poly
assumes bij:bij-betw ( $\lambda x. \text{poly } q1\ x / \text{poly } q2\ x$ ) A B and  $p \neq 0$ 
  and nzero: $\forall x \in A. \text{poly } q2\ x \neq 0$ 
  and max-deg:  $\max (\text{degree } q1) (\text{degree } q2) \leq 1$ 
  and nconst: $\forall c. q1 \neq \text{smult } c\ q2$ 
  and infi:infinite (UNIV::'a set)
shows proots-count p B = proots-count (fcompose p q1 q2) A
using  $\langle p \neq 0 \rangle$ 
proof (induct p rule: poly-root-induct-alt [of -  $\lambda x. \text{True}$ ])
  case 0
  then show ?case by simp
next
  case (no-proots p)
  have proots-count p B = 0
  proof -
    have proots-within p B = {}
    using no-proots by auto
    then show ?thesis unfolding proots-count-def by auto
  qed
  moreover have proots-count (fcompose p q1 q2) A = 0
  proof -
    have proots-within (fcompose p q1 q2) A = {}
    using no-proots unfolding proots-within-def
    by (smt (verit) div-0 empty-Collect-eq fcompose-poly nzero)
    then show ?thesis unfolding proots-count-def by auto
  qed
  ultimately show ?case by auto
next
  case (root b p)
  have proots-count ( $[: - b, 1:] * p$ ) B = proots-count  $[: - b, 1:] B$  + proots-count
p B
  using proots-count-times[OF  $\langle[: - b, 1:] * p \neq 0 \rangle$ ] by simp
  also have ... = proots-count (fcompose  $[: - b, 1:] q1 q2$ ) A
    + proots-count (fcompose p q1 q2) A
  proof -
    define g where  $g = (\lambda x. \text{poly } q1\ x / \text{poly } q2\ x)$ 

    have proots-count  $[: - b, 1:] B$  = proots-count (fcompose  $[: - b, 1:] q1 q2$ ) A
  proof (cases  $b \in B$ )
    case True
    then have proots-count  $[: - b, 1:] B$  = 1
      unfolding proots-count-pCons-1-iff by simp
    moreover have proots-count (fcompose  $[: - b, 1:] q1 q2$ ) A = 1
  proof -
```

```

obtain  $a$  where  $b=g$   $a \in A$ 
  using  $\text{bij}[\text{folded } g\text{-def}]$   $\text{True}$ 
  by ( $\text{metis } \text{bij-betwE } \text{bij-betw-the-inv-into } f\text{-the-inv-into-}f\text{-bij-betw}$ )
define  $qq$  where  $qq=q1 - \text{smult } b \ q2$ 
have  $qq-0$ :  $\text{poly } qq \ a=0$  and  $qq\text{-deg}$ :  $\text{degree } qq \leq 1$  and  $\langle qq \neq 0 \rangle$ 
  unfolding  $qq\text{-def}$ 
  subgoal using  $\langle b=g \ a \rangle$   $\text{nzero}[\text{rule-format}, OF \ \langle a \in A \rangle]$  unfolding  $g\text{-def}$  by
auto
  subgoal using  $\text{max-deg}$  by ( $\text{simp add: degree-diff-le}$ )
  subgoal using  $\text{nconst}[\text{rule-format}, of \ b]$  by auto
  done
have  $\text{proots-within } qq \ A = \{a\}$ 
proof  $-$ 
  have  $a \in \text{proots-within } qq \ A$ 
    using  $qq-0 \ \langle a \in A \rangle$  by auto
  moreover have  $\text{card } (\text{proots-within } qq \ A) = 1$ 
  proof  $-$ 
    have  $\text{finite } (\text{proots-within } qq \ A)$  using  $\langle qq \neq 0 \rangle$  by  $\text{simp}$ 
    moreover have  $\text{proots-within } qq \ A \neq \{\}$ 
      using  $\langle a \in \text{proots-within } qq \ A \rangle$  by auto
    ultimately have  $\text{card } (\text{proots-within } qq \ A) \neq 0$  by auto
    moreover have  $\text{card } (\text{proots-within } qq \ A) \leq 1$ 
      by ( $\text{meson } \langle qq \neq 0 \rangle \ \text{card-proots-within-leq } le\text{-trans } \text{proots-count-leq-degree}$ 
qq-deg)
    ultimately show  $?thesis$  by auto
  qed
  ultimately show  $?thesis$  by ( $\text{metis } \text{card-1-singletonE } \text{singletonD}$ )
qed
moreover have  $\text{order } a \ qq=1$ 
  by ( $\text{metis } \text{One-nat-def } \langle qq \neq 0 \rangle \ le\text{-antisym } le\text{-zero-eq } not\text{-less-eq-eq } or\text{-}$ 
der-degree
   $\text{order-root } qq-0 \ qq\text{-deg}$ )
  ultimately show  $?thesis$  unfolding  $f\text{compose-def } \text{proots-count-def } qq\text{-def}$ 
  by auto
qed
ultimately show  $?thesis$  by auto
next
case  $\text{False}$ 
then have  $\text{proots-count } [:- \ b, 1:] \ B = 0$ 
  unfolding  $\text{proots-count-pCons-1-iff}$  by  $\text{simp}$ 
moreover have  $\text{proots-count } (f\text{compose } [:- \ b, 1:] \ q1 \ q2) \ A = 0$ 
proof  $-$ 
  have  $\text{proots-within } (f\text{compose } [:- \ b, 1:] \ q1 \ q2) \ A = \{\}$ 
  proof ( $\text{rule ccontr}$ )
    assume  $\text{proots-within } (f\text{compose } [:- \ b, 1:] \ q1 \ q2) \ A \neq \{\}$ 
    then obtain  $a$  where  $a \in A$   $\text{poly } q1 \ a = b * \text{poly } q2 \ a$ 
      unfolding  $f\text{compose-def } \text{proots-within-def}$  by auto
    then have  $b = g \ a$ 
      unfolding  $g\text{-def}$  using  $\text{nzero}[\text{rule-format}, OF \ \langle a \in A \rangle]$  by auto

```

```

    then have  $b \in B$  using  $\langle a \in A \rangle$  bij[folded g-def] using bij-betwE by blast
    then show False using False by auto
  qed
  then show ?thesis unfolding roots-count-def by auto
  qed
  ultimately show ?thesis by simp
  qed
  moreover have  $\text{roots-count } p \ B = \text{roots-count } (fcompose \ p \ q1 \ q2) \ A$ 
  apply (rule root.hyps)
  using mult-eq-0-iff root.prems by blast
  ultimately show ?thesis by auto
  qed
  also have  $\dots = \text{roots-count } (fcompose \ ([:- \ b, \ 1:] \ * \ p) \ q1 \ q2) \ A$ 
  proof (cases A = {})
  case False
  have  $fcompose \ ([:- \ b, \ 1:] \ q1 \ q2) \neq 0$ 
  using nconst[rule-format, of b] unfolding fcompose-def by auto
  moreover have  $fcompose \ p \ q1 \ q2 \neq 0$ 
  apply (rule fcompose-nzero[OF - - nconst infi])
  subgoal using  $\langle[:- \ b, \ 1:] \ * \ p \neq 0 \rangle$  by auto
  subgoal using nzero False by auto
  done
  ultimately show ?thesis unfolding fcompose-mult
  apply (subst roots-count-times)
  by auto
  qed auto
  finally show ?case .
  qed

lemma roots-card-fcompose-bij-eq:
  fixes  $p::'a::field \ poly$ 
  assumes bij:bij-betw  $(\lambda x. \ poly \ q1 \ x / \ poly \ q2 \ x) \ A \ B$  and  $p \neq 0$ 
    and nzero:  $\forall x \in A. \ poly \ q2 \ x \neq 0$ 
    and max-deg:  $\max(\text{degree } q1) (\text{degree } q2) \leq 1$ 
    and nconst:  $\forall c. \ q1 \neq \text{smult } c \ q2$ 
    and infi: infinite  $(UNIV::'a \ set)$ 
  shows  $\text{card } (\text{roots-within } p \ B) = \text{card } (\text{roots-within } (fcompose \ p \ q1 \ q2) \ A)$ 
  using  $\langle p \neq 0 \rangle$ 
  proof (induct p rule: poly-root-induct-alt [of -  $\lambda x. \ True$ ])
  case 0
  then show ?case by simp
  next
  case (no-roots p)
  have  $\text{roots-within } p \ B = \{\}$  using no-roots by auto
  moreover have  $\text{roots-within } (fcompose \ p \ q1 \ q2) \ A = \{\}$ 
  using no-roots fcompose-poly
  by (smt (verit) Collect-empty-eq divide-eq-0-iff nzero roots-within-def)
  ultimately show ?case by auto
  next

```

```

case (root b p)
then have [simp]:p≠0 by auto

have ?case when b∉B ∨ poly p b=0
proof -
  have roots-within ([:- b, 1:] * p) B = roots-within p B
  using that by auto
  moreover have roots-within (fcompose ([:- b, 1:] * p) q1 q2) A
    = roots-within (fcompose p q1 q2) A
  using that nzero unfolding fcompose-mult roots-within-times
  apply (auto simp add: poly-fcompose)
  using bij bij-betwE by blast
  ultimately show ?thesis using root by auto
qed
moreover have ?case when b∈B poly p b≠0
proof -
  define bb where bb=[:- b, 1:]
  have card (roots-within (bb * p) B) = card {b} + card (roots-within p B)
  proof -
    have roots-within bb B = {b}
    using that unfolding bb-def by auto
    then show ?thesis unfolding roots-within-times
    apply (subst card-Un-disjoint)
    by (use that in auto)
  qed
  also have ... = 1 + card (roots-within (fcompose p q1 q2) A)
  using root.hyps by simp
  also have ... = card (roots-within (fcompose (bb * p) q1 q2) A)
  unfolding roots-within-times fcompose-mult
  proof (subst card-Un-disjoint)
    obtain a where b-poly:b=poly q1 a / poly q2 a and a∈A
    by (metis (no-types, lifting) ⟨b ∈ B⟩ bij bij-betwE bij-betw-the-inv-into
      f-the-inv-into-f-bij-betw)
    define bbq pq where bbq=fcompose bb q1 q2 and pq=fcompose p q1 q2
    have bbq-0:poly bbq a=0 and bbq-deg: degree bbq≤1 and bbq≠0
    unfolding bbq-def bb-def
    subgoal using ⟨a ∈ A⟩ b-poly nzero poly-fcompose by fastforce
    subgoal by (metis (no-types, lifting) degree-add-le degree-pCons-eq-if de-
      gree-smult-le
        dual-order.trans fcompose-const fcompose-pCons max.boundedE max-deg
        mult-cancel-left2
        one-neq-zero one-poly-eq-simps(1) power.simps)
    subgoal by (metis ⟨a ∈ A⟩ ⟨poly (fcompose ([:- b, 1:] q1 q2) a = 0⟩
      fcompose-nzero infi
        nconst nzero one-neq-zero pCons-eq-0-iff))
  done
  show finite (roots-within bbq A) using ⟨bbq≠0⟩ by simp
  show finite (roots-within pq A) unfolding pq-def
  by (metis ⟨a ∈ A⟩ ⟨p ≠ 0⟩ fcompose-nzero finite-roots infi nconst nzero

```

```

poly-0 pq-def)
  have bbq-a:proots-within bbq A = {a}
  proof -
    have a∈proots-within bbq A
      by (simp add: ⟨a ∈ A⟩ bbq-0)
    moreover have card (proots-within bbq A) = 1
    proof -
      have card (proots-within bbq A) ≠ 0
        using ⟨a∈proots-within bbq A⟩ ⟨finite (proots-within bbq A)⟩
        by auto
      moreover have card (proots-within bbq A) ≤ 1
      by (meson ⟨bbq ≠ 0⟩ card-proots-within-leq le-trans proots-count-leq-degree
bbq-deg)
      ultimately show ?thesis by auto
    qed
    ultimately show ?thesis by (metis card-1-singletonE singletonD)
  qed
  show proots-within (bbq) A ∩ proots-within (pq) A = {}
    using b-poly bbq-a fcompose-poly nzero pq-def that(2) by fastforce
  show 1 + card (proots-within pq A) = card (proots-within bbq A) + card
(proots-within pq A)
    using bbq-a by simp
  qed
  finally show ?thesis unfolding bb-def .
  qed
  ultimately show ?case by auto
qed

lemma proots-pcompose-bij-eq:
  fixes p::'a::idom poly
  assumes bij:bij-betw (λx. poly q x) A B and p≠0
    and q-deg: degree q = 1
  shows proots-count p B = proots-count (p ∘p q) A using ⟨p≠0⟩
proof (induct p rule: poly-root-induct-alt [of - λx. True])
  case 0
  then show ?case by simp
next
  case (no-proots p)
  have proots-count p B = 0
  proof -
    have proots-within p B = {}
    using no-proots by auto
    then show ?thesis unfolding proots-count-def by auto
  qed
  moreover have proots-count (p ∘p q) A = 0
  proof -
    have proots-within (p ∘p q) A = {}
    using no-proots unfolding proots-within-def
    by (auto simp:poly-pcompose)

```

```

    then show ?thesis unfolding roots-count-def by auto
  qed
  ultimately show ?case by auto
next
case (root b p)
have roots-count ([: - b, 1:] * p) B = roots-count [: - b, 1:] B + roots-count
p B
  using roots-count-times[OF <[: - b, 1:] * p ≠ 0>] by simp
also have ... = roots-count ([: - b, 1:] ∘p q) A + roots-count (p ∘p q) A
proof -
  have roots-count [: - b, 1:] B = roots-count ([: - b, 1:] ∘p q) A
  proof (cases b∈B)
    case True
    then have roots-count [: - b, 1:] B = 1
      unfolding roots-count-pCons-1-iff by simp
    moreover have roots-count ([: - b, 1:] ∘p q) A = 1
    proof -
      obtain a where b=poly q a a∈A
      using True bij by (metis bij-betwE bij-betw-the-inv-into-f-the-inv-into-f-bij-betw)
      define qq where qq=[: - b:] + q
      have qq-0:poly qq a=0 and qq-deg: degree qq≤1 and <qq≠0>
        unfolding qq-def
        subgoal using <b=poly q a> by auto
        subgoal using q-deg by (simp add: degree-add-le)
        subgoal using q-deg add.inverse-unique by force
      done
      have roots-within qq A = {a}
      proof -
        have a∈roots-within qq A
          using qq-0 <a∈A> by auto
        moreover have card (roots-within qq A) = 1
        proof -
          have finite (roots-within qq A) using <qq≠0> by simp
          moreover have roots-within qq A ≠ {}
            using <a∈roots-within qq A> by auto
          ultimately have card (roots-within qq A) ≠ 0 by auto
          moreover have card (roots-within qq A) ≤ 1
            by (meson <qq ≠ 0> card-roots-within-leq le-trans roots-count-leq-degree
qq-deg)
          ultimately show ?thesis by auto
        qed
      qed
    ultimately show ?thesis by (metis card-1-singletonE singletonD)
  qed
  moreover have order a qq=1
    by (metis One-nat-def <qq ≠ 0> le-antisym le-zero-eq not-less-eq-eq or-
der-degree
      order-root qq-0 qq-deg)
  ultimately show ?thesis unfolding pcompose-def roots-count-def qq-def
    by auto

```

```

qed
ultimately show ?thesis by auto
next
case False
then have proots-count  $[- b, 1:] B = 0$ 
  unfolding proots-count-pCons-1-iff by simp
moreover have proots-count  $([- b, 1:] \circ_p q) A = 0$ 
proof -
  have proots-within  $([- b, 1:] \circ_p q) A = \{\}$ 
  unfolding pcompose-def
  apply auto
  using False bij bij-betwE by blast
  then show ?thesis unfolding proots-count-def by auto
qed
ultimately show ?thesis by simp
qed
moreover have proots-count  $p B = \text{proots-count } (p \circ_p q) A$ 
  apply (rule root.hyps)
  using  $\langle [- b, 1:] * p \neq 0 \rangle$  by auto
ultimately show ?thesis by auto
qed
also have ... = proots-count  $(([- b, 1:] * p) \circ_p q) A$ 
  unfolding pcompose-mult
  apply (subst proots-count-times)
  subgoal by (metis (no-types, lifting) One-nat-def add.right-neutral degree-0
degree-mult-eq
degree-pCons-eq-if degree-pcompose mult-eq-0-iff one-neq-zero one-pCons pcompose-mult
q-deg root.prem)
  by simp
  finally show ?case .
qed

lemma proots-card-pcompose-bij-eq:
  fixes  $p::'a::idom$  poly
  assumes bij:bij-betw  $(\lambda x. poly q x) A B$  and  $p \neq 0$ 
  and q-deg: degree  $q = 1$ 
  shows card (proots-within  $p B) = \text{card } (\text{proots-within } (p \circ_p q) A)$  using  $\langle p \neq 0 \rangle$ 
proof (induct p rule: poly-root-induct-alt [of -  $\lambda x. True$ ])
  case 0
  then show ?case by auto
next
  case (no-proots p)
  have proots-within  $p B = \{\}$  using no-proots by auto
  moreover have proots-within  $(p \circ_p q) A = \{\}$  using no-proots
  by (simp add: poly-pcompose proots-within-def)
  ultimately show ?case by auto
next
  case (root b p)

```

```

then have [simp]:p≠0 by auto
have ?case when b∉B ∨ poly p b=0
proof -
  have roots-within ([:- b, 1:] * p) B = roots-within p B
  using that by auto
  moreover have roots-within (([:- b, 1:] * p) ∘p q) A = roots-within (p ∘p
q) A
  using that unfolding pcompose-mult roots-within-times
  apply (auto simp add: poly-pcompose)
  using bij bij-betwE by blast
  ultimately show ?thesis using ⟨p ≠ 0⟩ root.hyps by auto
qed
moreover have ?case when b∈B poly p b≠0
proof -
  define bb where bb=[:- b, 1:]
  have card (roots-within (bb * p) B) = card {b} + card (roots-within p B)
  proof -
    have roots-within bb B = {b}
    using that unfolding bb-def by auto
    then show ?thesis unfolding roots-within-times
    apply (subst card-Un-disjoint)
    by (use that in auto)
  qed
  also have ... = 1 + card (roots-within (p ∘p q) A)
  using root.hyps by simp
  also have ... = card (roots-within ((bb * p) ∘p q) A)
  unfolding roots-within-times pcompose-mult
  proof (subst card-Un-disjoint)
    obtain a where b=poly q a a∈A
    by (metis ⟨b ∈ B⟩ bij bij-betwE bij-betw-the-inv-into f-the-inv-into-f-bij-betw)
    define bbq pq where bbq=bb ∘p q and pq=p ∘p q
    have bbq-0:poly bbq a=0 and bbq-deg: degree bbq≤1 and bbq≠0
    unfolding bbq-def bb-def poly-pcompose
    subgoal using ⟨b=poly q a⟩ by auto
    subgoal using q-deg by (simp add: degree-add-le degree-pcompose)
    subgoal using pcompose-eq-0 q-deg by fastforce
    done
  show finite (roots-within bbq A) using ⟨bbq≠0⟩ by simp
  show finite (roots-within pq A) unfolding pq-def
  by (metis ⟨p ≠ 0⟩ finite-roots pcompose-eq-0 q-deg zero-less-one)
  have bbq-a:roots-within bbq A = {a}
  proof -
    have a∈roots-within bbq A
    unfolding bb-def roots-within-def poly-pcompose bbq-def
    using ⟨b=poly q a⟩ ⟨a∈A⟩ by simp
    moreover have card (roots-within bbq A) = 1
  proof -
    have card (roots-within bbq A) ≠0
    using ⟨a∈roots-within bbq A⟩ ⟨finite (roots-within bbq A)⟩

```

```

    by auto
    moreover have card (proots-within bbq A) ≤ 1
    by (meson ⟨bbq ≠ 0⟩ card-proots-within-leq le-trans proots-count-leq-degree
bbq-deg)
    ultimately show ?thesis by auto
  qed
  ultimately show ?thesis by (metis card-1-singletonE singletonD)
  qed
  show proots-within (bbq) A ∩ proots-within (pq) A = {}
  using bbq-a ⟨b = poly q a⟩ that(2) unfolding pq-def by (simp add:poly-pcompose)
  show 1 + card (proots-within pq A) = card (proots-within bbq A) + card
(proots-within pq A)
  using bbq-a by simp
  qed
  finally show ?thesis unfolding bb-def .
  qed
  ultimately show ?case by auto
  qed
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 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.

The proof mainly follows Theorem 2.35 in Basu, S., Pollack, R., Roy, M.-F.: Algorithms in Real Algebraic Geometry. Springer Berlin Heidelberg, Berlin, Heidelberg (2006).

2.1 More results related to *sign-r-pos*

lemma *sign-r-pos-nzero-right*:

```

  assumes nzero:∀ x. c < x ∧ x ≤ d → poly p x ≠ 0 and c < d
  shows if sign-r-pos p c then poly p d > 0 else poly p d < 0

```

proof (cases *sign-r-pos p c*)

```

  case True

```

```

  then obtain d' where d' > c and d'-pos:∀ y > c. y < d' → 0 < poly p y

```

```

    unfolding sign-r-pos-def eventually-at-right by auto

```

```

  have False when ¬ poly p d > 0

```

```

  proof –

```

```

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

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

```

```

proof –
  have  $\forall_F x$  in at c. poly p x < 0
    using that
    by (metis (no-types, lifting) less-linear no-roots.hyps not-eventuallyD
        poly-IVT-neg poly-IVT-pos)
  then have  $\forall_F x$  in at-left c. poly p x < 0
    using eventually-at-split by blast
  moreover have  $\neg$  sign-r-pos p c using sign-r-pos-rec[OF ‹p≠0›] that by auto
  ultimately show ?thesis by simp
qed
ultimately show ?case using no-roots(1)[of c] by argo
next
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 p x
  else  $\forall_F x$  in at-left c. poly p x < 0)
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)  $\longleftrightarrow$  ( $\forall_F x$  in at-left c. 0
  < poly p x)
    ( $\forall_F x$  in at-left c. 0 > poly (aa * p) x)  $\longleftrightarrow$  ( $\forall_F x$  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)
by (subst order-mult) auto
moreover have \neg 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 = (\neg 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 p x
else $\forall_F x$ in at-left c. poly p x < 0)
proof –
have $\forall_F x$ in at-left c. poly aa x < 0
apply (simp add:aa-def)
using that eventually-at-filter **by** fastforce
then have ($\forall_F x$ in at-left c. 0 < poly (aa * p) x) \longleftrightarrow ($\forall_F x$ in at-left c.
poly p x < 0)
($\forall_F x$ in at-left c. 0 > poly (aa * p) x) \longleftrightarrow ($\forall_F x$ 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. 0 > poly (aa * p) x
else $\forall_F x$ in at-left c. poly (aa * p) x > 0)
proof –
have order c aa=1 **unfolding** aa-def **using** that
by (metis order-power-n-n power-one-right)
then have even (order c (aa * p)) = odd (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 pderiv-pCons)
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.  $0 < poly\ p\ x$ 
  else  $\forall_F x$  in at-left c.  $poly\ p\ x < 0$ )
proof -
  have  $\forall_F x$  in at-left c.  $0 > poly\ aa\ x$ 
  apply (simp add:aa-def)
  using that by (simp add: eventually-at-filter)
  then have  $(\forall_F x$  in at-left c.  $0 < poly\ (aa * p)\ x$ )  $\longleftrightarrow$   $(\forall_F x$  in at-left c.  $0 > poly\ p\ x$ )
   $(\forall_F x$  in at-left c.  $0 > poly\ (aa * p)\ x$ )  $\longleftrightarrow$   $(\forall_F x$  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
ultimately show ?case by argo
qed

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

```

```

have False when  $\neg$  poly p d < 0
proof -
  have  $\exists x > d. x < (c + \max d d') / 2 \wedge \text{poly } p x = 0$ 
  apply (rule poly-IVT-neg)
  using  $\langle d' < c \rangle \langle c > d \rangle$  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 ?thesis using False by auto
qed

```

2.2 Fourier sequences

```

function pders::real poly  $\Rightarrow$  real poly list where
  pders p = (if p = 0 then [] else Cons p (pders (pderiv p)))
  by auto
termination
  apply (relation measure ( $\lambda p. \text{if } p = 0 \text{ then } 0 \text{ else degree } p + 1$ ))
  by (auto simp: degree-pderiv pderiv-eq-0-iff)

```

```

declare pders.simps[simp del]

```

```

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

```

2.3 Sign variations for Fourier sequences

```

definition changes-itv-der:: real  $\Rightarrow$  real  $\Rightarrow$  real poly  $\Rightarrow$  int where
  changes-itv-der a b p = (let ps = pders p in changes-poly-at ps a - changes-poly-at
ps b)

```

```

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

```

```

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

```

```

lemma changes-poly-pos-inf-pders[simp]: 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≠0 p≠0 using pderiv-eq-0-iff by force+
define ps where ps=pders (pderiv (pderiv p))
have ps:pders p = p# pderiv p #ps pders (pderiv p) = pderiv p#ps
  unfolding ps-def by (simp-all add: ⟨p ≠ 0⟩ ⟨pderiv p ≠ 0⟩ pders.simps)
have hyps: changes-poly-pos-inf (pders (pderiv p)) = 0
  apply (rule Suc(1))
  using ⟨Suc x = degree p⟩ by (metis degree-pderiv diff-Suc-1)
moreover have sgn-pos-inf p * sgn-pos-inf (pderiv p) > 0
  unfolding sgn-pos-inf-def lead-coeff-pderiv
  apply (simp add: algebra-simps sgn-mult)
  using Suc.hyps(2) ⟨p ≠ 0⟩ by linarith
ultimately show ?case unfolding changes-poly-pos-inf-def ps by auto
qed

lemma changes-poly-neg-inf-pders[simp]: changes-poly-neg-inf (pders p) = degree
p
proof (induct degree p arbitrary:p)
  case 0
  then obtain a where p=[:a:] using degree-eq-zeroE by auto
  then show ?case unfolding changes-poly-neg-inf-def by (auto simp: pders.simps)
next
case (Suc x)
then have pderiv p≠0 p≠0 using pderiv-eq-0-iff by force+
then have changes-poly-neg-inf (pders p)
  = changes-poly-neg-inf (p # pderiv p#pders (pderiv (pderiv p)))
  by (simp add: pders.simps)
also have ... = 1 + changes-poly-neg-inf (pderiv p#pders (pderiv (pderiv p)))
proof -
  have sgn-neg-inf p * sgn-neg-inf (pderiv p) < 0
    unfolding sgn-neg-inf-def using ⟨p≠0⟩ ⟨pderiv p≠0⟩
  by (auto simp add: lead-coeff-pderiv degree-pderiv coeff-pderiv sgn-mult pderiv-eq-0-iff)
  then show ?thesis unfolding changes-poly-neg-inf-def by auto
qed
also have ... = 1 + changes-poly-neg-inf (pders (pderiv p))
  using ⟨pderiv p≠0⟩ by (simp add: pders.simps)
also have ... = 1 + degree (pderiv p)
  apply (subst Suc(1))
  using Suc(2) by (auto simp add: degree-pderiv)
also have ... = degree p
  by (metis Suc.hyps(2) degree-pderiv diff-Suc-1 plus-1-eq-Suc)

```

```

finally show ?case .
qed

lemma pders-coeffs-sgn-eq:map (λp. sgn(poly p 0)) (pders p) = map sgn (coeffs p)
proof (induct degree p arbitrary:p)
  case 0
  then obtain a where p=[:a:] using degree-eq-zeroE by auto
  then show ?case by (auto simp: pders.simps)
next
  case (Suc x)
  then have pderiv p≠0 p≠0 using pderiv-eq-0-iff by force+

  have map (λp. sgn (poly p 0)) (pders p)
    = sgn (poly p 0) # map (λp. sgn (poly p 0)) (pders (pderiv p))
  apply (subst pders.simps)
  using ⟨p≠0⟩ by simp
  also have ... = sgn (coeff p 0) # map sgn (coeffs (pderiv p))
  proof -
  have sgn (poly p 0) = sgn (coeff p 0) by (simp add: poly-0-coeff-0)
  then show ?thesis
  apply (subst Suc(1))
  subgoal by (metis Suc.hyps(2) degree-pderiv diff-Suc-1)
  subgoal by auto
  done
qed
  also have ... = map sgn (coeffs p)
  proof (rule nth-equalityI)
  show p-length:length (sgn (coeff p 0) # map sgn (coeffs (pderiv p)))
    = length (map sgn (coeffs p))
  by (metis Suc.hyps(2) ⟨p ≠ 0⟩ ⟨pderiv p ≠ 0⟩ degree-pderiv diff-Suc-1
length-Cons
length-coeffs-degree length-map)
  show (sgn (coeff p 0) # map sgn (coeffs (pderiv p))) ! i = map sgn (coeffs p)
! i
  if i < length (sgn (coeff p 0) # map sgn (coeffs (pderiv p))) for i
  proof -
  show (sgn (coeff p 0) # map sgn (coeffs (pderiv p))) ! i = map sgn (coeffs p)
! i
  proof (cases i)
  case 0
  then show ?thesis
  by (simp add: ⟨p ≠ 0⟩ coeffs-nth)
  next
  case (Suc i')
  then show ?thesis
  using that p-length
  apply simp
  apply (subst (1 2) coeffs-nth)
  by (auto simp add: ⟨p ≠ 0⟩ ⟨pderiv p ≠ 0⟩ length-coeffs-degree coeff-pderiv

```

```

sgn-mult)
  qed
  qed
  qed
  finally show ?case .
qed

```

```

lemma changes-poly-at-pders-0:changes-poly-at (pders p) 0 = changes (coeffs p)
  unfolding changes-poly-at-def
  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  $p \neq 0$ 
  assumes  $\forall x. c < x \wedge x \leq d2 \longrightarrow (\forall q \in \text{set } (pders\ p). \text{poly } q\ x \neq 0)$ 
  shows changes-itv-der  $c\ d2\ p = 0$ 
  using assms(2–3)
proof (induct degree p arbitrary:p)
  case 0
  then obtain a where  $p = [:a:]$   $a \neq 0$  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]:  $pderiv\ p \neq 0$  by (metis nat.distinct(1) pderiv-eq-0-iff)
  note  $nzero = \langle \forall x. c < x \wedge x \leq d2 \longrightarrow (\forall q \in \text{set } (pders\ p). \text{poly } q\ x \neq 0) \rangle$ 

  have  $hyps: \text{changes-itv-der } c\ d2\ (pderiv\ p) = 0$ 
  apply (rule Suc(1))
  subgoal by (metis Suc.hyps(2) degree-pderiv diff-Suc-1)
  subgoal by (simp add: Suc.prem(1) Suc.prem(2) pders.simps)
  subgoal by (simp add: Suc.prem(1) nonzero pders.simps)
  done

  have  $pders\ \text{changes-c:changes-poly-at } (r\ \# \text{pders } q)\ c = (\text{if sign-r-pos } q\ c \longleftrightarrow \text{poly } r\ c > 0$ 
    then  $\text{changes-poly-at } (pders\ q)\ c$  else  $1 + \text{changes-poly-at } (pders\ q)\ c)$ 
  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  $pderiv\ q = 0$ 
  proof –
    have 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

```

```

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
  then show ?thesis unfolding qs
  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-d2:changes-poly-at (r# pders q) d2 = (if sign-r-pos q c ⟷
poly r c>0
  then changes-poly-at (pders q) d2 else 1+changes-poly-at (pders q) d2)
  when poly r c≠0 q≠0 and qr-nzero:∀x. c < x ∧ x ≤ d2 ⟶ 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 sign-r-pos r c then 0 < poly r d2 else poly r d2 < 0
  if sign-r-pos q c then 0 < poly q d2 else poly q d2 < 0
  subgoal by (rule sign-r-pos-nzero-right[of c d2 r]) (use qr-nzero ⟨c<d2⟩ in
auto)
  subgoal by (rule sign-r-pos-nzero-right[of c d2 q]) (use qr-nzero ⟨c<d2⟩ in
auto)
  done
  then 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 d2c-nzero:∀x. c<x ∧ x≤d2 ⟶ poly p x≠0 ∧ poly (pderiv p) x ≠ 0
  and p-cons:pders p = p#pders(pderiv p)
  subgoal by (simp add: nzero Suc.prem(1) pders.simps)
  subgoal by (simp add: Suc.prem(1) pders.simps)
  done

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 changes-poly-at (p # pderiv p # ps) c = changes-poly-at (pderiv p # ps)
c
  unfolding changes-poly-at-def using that by auto
  moreover have changes-poly-at (p # pderiv p # ps) d2 = changes-poly-at
(pderiv p # ps) d2
  proof -
    have if sign-r-pos p c then 0 < poly p d2 else poly p d2 < 0
      apply (rule sign-r-pos-nzero-right[OF - ⟨c<d2⟩])
      using nzero[folded ps-cons] assms(1-2) by auto
    moreover have if sign-r-pos (pderiv p) c then 0 < poly (pderiv p) d2
      else poly (pderiv p) d2 < 0
      apply (rule sign-r-pos-nzero-right[OF - ⟨c<d2⟩])
      using nzero[folded ps-cons] assms(1-2) by auto
    ultimately have poly p d2 * poly (pderiv p) d2 > 0
      unfolding zero-less-mult-iff sign-r-pos-rec[OF ⟨p≠0⟩] 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≠0 sign-r-pos (pderiv p) c ⟷ 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 ⟨poly p c≠0⟩])
      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 ⟨poly p c≠0⟩ - 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≠0 ¬ sign-r-pos (pderiv p) c ⟷ 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 ⟨poly p c≠0⟩])
      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 ⟨poly p c≠0⟩ - 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≠0
  assumes ∀ x. d1 ≤ x ∧ x < c ⟶ (∀ q ∈ set (pders p). poly q x ≠ 0)
  shows changes-itv-der d1 c p ≥ order c p ∧ even (changes-itv-der d1 c p - order
  c p)
  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
    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 [simp]: pderiv p ≠ 0 by (metis nat.distinct(1) pderiv-eq-0-iff)
  note nzero = ⟨∀ x. d1 ≤ x ∧ x < c ⟶ (∀ q ∈ set (pders p). poly q x ≠ 0)⟩
  define v where v = order c (pderiv p)

  have hyps: v ≤ changes-itv-der d1 c (pderiv p) ∧ even (changes-itv-der d1 c (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.prem(1) Suc.prem(2) pders.simps)
    subgoal by (simp add: Suc.prem(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 q ≠ 0 for q r
    using ⟨q ≠ 0⟩
  proof (induct q rule: pders.induct)
    case (1 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
  qed

```

moreover have *?case when pderiv q ≠ 0*
proof –
obtain *qs* **where** *qs:pders q = q#qs pders (pderiv q) = qs*
using $\langle q \neq 0 \rangle$ **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 $1 \langle pderiv q \neq 0 \rangle$ **unfolding** *qs* **by** *simp*
then show *?thesis unfolding qs*
apply (*cases poly q c = 0*)
subgoal unfolding *changes-poly-at-def* **by** (*auto simp:sign-r-pos-rec[OF*
 $\langle q \neq 0 \rangle, of c]$)
subgoal unfolding *changes-poly-at-def* **using** $\langle poly r c \neq 0 \rangle$
by (*auto simp:sign-r-pos-rec[OF \langle q \neq 0 \rangle, 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)*
 $\longleftrightarrow sign-r-pos q c \longleftrightarrow poly r c > 0$
then changes-poly-at (pders q) d1 else 1+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 $\langle q \neq 0 \rangle$ **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 \langle d1 < c \rangle in*
auto)
subgoal by (*rule sign-r-pos-nzero-left[of d1 c q]*) (*use qr-nzero \langle d1 < c \rangle in*
auto)
done
moreover have *order c r = 0* **by** (*simp add: order-0I that(1)*)
ultimately show *?thesis unfolding qs changes-poly-at-def*
using $\langle poly r c \neq 0 \rangle$ **by** (*auto split:if-splits simp:mult-less-0-iff sign-r-pos-rec[OF*
 $\langle r \neq 0 \rangle]$)
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.prem(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** $\langle p \neq 0 \rangle$ **by** (*auto simp:pders.simps*)

have $p\text{-order:order } c \ p = \text{Suc } v$
apply (*subst order-pderiv*)
using *Suc.prem1* *order-root* **that** **unfolding** *v-def* **by** *auto*
moreover have $\text{changes-poly-at } (p \# \text{pderiv } p \ \# \ \text{ps}) \ d1 = \text{changes-poly-at } (\text{pderiv } p \ \# \ \text{ps}) \ d1 \ +1$
proof –
have *if even* $(\text{order } c \ p) = \text{sign-r-pos } p \ c$ **then** $0 < \text{poly } p \ d1$ **else** $\text{poly } p \ d1 < 0$
apply (*rule sign-r-pos-nzero-left*[*OF - <d1<c>*])
using *nzero*[*folded ps-cons*] *assms(1-2)* **by** *auto*
moreover have *if even* $v = \text{sign-r-pos } (\text{pderiv } p) \ c$
then $0 < \text{poly } (\text{pderiv } p) \ d1$ **else** $\text{poly } (\text{pderiv } p) \ d1 < 0$
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 $\text{poly } p \ d1 * \text{poly } (\text{pderiv } p) \ d1 < 0$
unfolding *mult-less-0-iff sign-r-pos-rec*[*OF <p≠0>*] **using** $\langle \text{poly } p \ c = 0 \rangle$
p-order
by (*auto split:if-splits*)
then show *?thesis*
unfolding *changes-poly-at-def* **by** *auto*
qed
moreover have $\text{changes-poly-at } (p \ \# \ \text{pderiv } p \ \# \ \text{ps}) \ c = \text{changes-poly-at } (\text{pderiv } p \ \# \ \text{ps}) \ c$
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** $\text{poly } p \ c \neq 0$ *odd* $v = \text{sign-r-pos } (\text{pderiv } p) \ c \longleftrightarrow \text{poly } p \ c > 0$
proof –
have $\text{order } c \ p = 0$ **by** (*simp add: order-0I that(1)*)
moreover have $\text{changes-poly-at } (\text{pders } p) \ d1 = \text{changes-poly-at } (\text{pders } (\text{pderiv } p)) \ d1 \ +1$
unfolding *p-cons*
apply (*subst pders-changes-d1*[*OF <poly p c≠0> - d1c-nzero*])
using *that* **unfolding** *v-def* **by** *auto*
moreover have $\text{changes-poly-at } (\text{pders } p) \ c = \text{changes-poly-at } (\text{pders } (\text{pderiv } p)) \ c$
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* $\langle \text{odd } v \rangle$ **unfolding** *changes-itv-der-def*
Let-def
by *auto*
qed
moreover have *?case* **when** $\text{poly } p \ c \neq 0$ *odd* $v \neg \text{sign-r-pos } (\text{pderiv } p) \ c \longleftrightarrow$

poly p c > 0
proof –
 have $v \geq 1$ **using** $\langle \text{odd } v \rangle$ **using** *not-less-eq-eq* **by** *auto*
 moreover 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 + 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* $\langle \text{even } v \rangle$ **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 $nzero: \forall x. d1 < x \wedge x < c \longrightarrow (\forall q \in \text{set } (pders\ p). \text{poly } q\ x \neq 0)$
shows $\text{changes-itv-der } d1\ c\ p \geq \text{order } c\ p \text{ even } (\text{changes-itv-der } d1\ c\ p - \text{order } c\ p)$
proof –
define d **where** $d = (d1 + c) / 2$
have $d1 < d < c$ **unfolding** *d-def* **using** $\langle d1 < c \rangle$ **by** *auto*

have $\text{changes-itv-der } d1\ d\ p = 0$
apply (*rule budan-fourier-aux-right*[*OF* $\langle d1 < d \rangle \langle p \neq 0 \rangle$])
using $nzero \langle d1 < d \rangle \langle d < c \rangle$ **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* $\langle d < c \rangle \langle p \neq 0 \rangle$])
using $nzero \langle d1 < d \rangle \langle d < c \rangle$ **by** *auto*
ultimately show $\text{changes-itv-der } d1\ c\ p \geq \text{order } c\ p \text{ even } (\text{changes-itv-der } d1\ c\ p - \text{order } c\ p)$
unfolding *changes-itv-der-def* *Let-def* **by** *auto*
qed

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 $\langle a < b \rangle$
proof (*induct card* $\{x. \exists p \in \text{set } (pders\ p). \text{poly } p\ x = 0 \wedge a < x \wedge x < b\}$ *arbitrary:b*)
case 0
have $nzero: \forall x. a < x \wedge x < b \longrightarrow (\forall q \in \text{set } (pders\ p). \text{poly } q\ x \neq 0)$
proof –
define S **where** $S = \{x. \exists p \in \text{set } (pders\ p). \text{poly } p\ x = 0 \wedge a < x \wedge x < b\}$
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 *set-pders-nzero*[*OF* $\langle p \neq 0 \rangle$] **by** *auto*
ultimately show *?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 ?thesis unfolding S-def using ⟨a < b⟩ assms(2) pders.simps by
fastforce
qed
from budan-fourier-aux-left[OF ⟨a < b⟩ ⟨p ≠ 0⟩ this]
have order b p ≤ changes-itv-der a b p even (changes-itv-der a b p - order b p)
by simp-all
moreover have roots-count p {x. a < x ∧ x ≤ b} = order b p
proof -
  have p-cons: pders p = p#pders (pderiv p) by (simp add: assms(2) pders.simps)
  have roots-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 roots-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). roots p)
  unfolding S-def P-def by auto
  moreover have finite (⋃ p ∈ set (pders p). roots 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 Max-in[OF f-S])
  using 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 ccontr)
  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 bb ∈ S b unfolding S-def using ⟨a < b'⟩ ⟨b' < b⟩ by auto
  from Max-ge[OF f-S this, folded b'-def] have bb ≤ b'.
  then show False using ⟨b' < bb⟩ by auto
qed

have hyps: roots-count p {x. a < x ∧ x ≤ b'} ≤ changes-itv-der a b' p ∧
  even (changes-itv-der a b' p - roots-count p {x. a < x ∧ x ≤ b'})
proof (rule Suc(1)[OF - ⟨a < b'⟩])
  have S b = {b'} ∪ S b'

```

proof –
have $\{x. P x \wedge b' < x \wedge x < b\} = \{\}$
using $b'\text{-nzero}$ **unfolding** $P\text{-def}$ **by** *auto*
then have $\{x. P x \wedge b' \leq x \wedge x < b\} = \{b'\}$
using $\langle b' \in S b \rangle$ **unfolding** $S\text{-def}$ **by** *force*
moreover have $S b = S b' \cup \{x. P x \wedge b' \leq x \wedge x < b\}$
unfolding $S\text{-def}$ **using** $\langle a < b' \rangle \langle b' < b \rangle$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed
moreover have $Suc n = card (S b)$ **using** $Suc(2)$ **unfolding** $S\text{-def}$ $P\text{-def}$ **by** *simp*
moreover have $b' \notin S b'$ **unfolding** $S\text{-def}$ **by** *auto*
ultimately have $n = card (S b')$ **using** $f\text{-S}$ **by** *auto*
then show $n = card \{x. \exists p \in set (pders p). poly p x = 0 \wedge a < x \wedge x < b'\}$
unfolding $S\text{-def}$ $P\text{-def}$ **by** *simp*
qed
moreover have $proots\text{-count } p \{x. a < x \wedge x \leq b\}$
 $= proots\text{-count } p \{x. a < x \wedge x \leq b'\} + order b p$
proof –
have $p\text{-cons}: pders p = p \# pders (pderiv p)$ **by** (*simp add: assms(2) pders.simps*)
have $proots\text{-within } p \{x. b' < x \wedge x \leq b\} = (if poly p b = 0 then \{b\} else \{\})$
using $b'\text{-nzero}$ $\langle b' < b \rangle$ **unfolding** $p\text{-cons}$
apply *auto*
using *not-le* **by** *fastforce*
then have $proots\text{-count } p \{x. b' < x \wedge x \leq b\} = order b p$
unfolding $proots\text{-count-def}$ **using** $order\text{-root}$ **by** *auto*
moreover have $proots\text{-count } p \{x. a < x \wedge x \leq b\} = proots\text{-count } p \{x. a <$
 $x \wedge x \leq b'\} +$
 $proots\text{-count } p \{x. b' < x \wedge x \leq b\}$
apply (*subst proots-count-union-disjoint[symmetric]*)
using $\langle a < b' \rangle \langle b' < b \rangle \langle p \neq 0 \rangle$ **by** (*auto intro: arg-cong2[where f=proots-count]*)
ultimately show *?thesis* **by** *auto*
qed
moreover note $budan\text{-fourier-aux-left}[OF \langle b' < b \rangle \langle p \neq 0 \rangle b'\text{-nzero}]$
ultimately show *?case* **unfolding** $changes\text{-itv-der-def}$ $Let\text{-def}$ **by** *auto*
qed

theorem $budan\text{-fourier-gt}$:
assumes $p \neq 0$
shows $changes\text{-gt-der } a p \geq proots\text{-count } p \{x. a < x\} \wedge$
 $even (changes\text{-gt-der } a p - proots\text{-count } p \{x. a < x\})$
proof –
define ps **where** $ps = pders p$
obtain ub **where** $ub\text{-root}: \forall p \in set ps. \forall x. poly p x = 0 \longrightarrow x < ub$
and $ub\text{-sgn}: \forall x \geq ub. \forall p \in set ps. sgn (poly p x) = sgn\text{-pos-inf } p$
and $a < ub$
using $root\text{-list-ub}[of ps a]$ $set\text{-pders-nzero}[OF \langle p \neq 0 \rangle, folded ps\text{-def}]$ **by** *blast*
have $proots\text{-count } p \{x. a < x\} = proots\text{-count } p \{x. a < x \wedge x \leq ub\}$
proof –

have $p \in \text{set } ps$ **unfolding** $ps\text{-def}$ **by** (*simp add: assms pders.simps*)
then have $\text{roots-within } p \{x. a < x\} = \text{roots-within } p \{x. a < x \wedge x \leq ub\}$
using $ub\text{-root}$ **by** *fastforce*
then show *?thesis* **unfolding** roots-count-def **by** *auto*
qed
moreover have $\text{changes-gt-der } a \ p = \text{changes-itv-der } a \ ub \ p$
proof –
have $\text{map } (sgn \circ (\lambda p. \text{poly } p \ ub)) \ ps = \text{map } sgn\text{-pos-inf } ps$
using $ub\text{-sgn}$ [*THEN spec, of ub, simplified*]
by (*metis (mono-tags, lifting) comp-def list.map-cong0*)
hence $\text{changes-poly-at } ps \ ub = \text{changes-poly-pos-inf } ps$
unfolding $\text{changes-poly-pos-inf-def}$ $\text{changes-poly-at-def}$
by (*subst changes-map-sgn-eq, metis map-map*)
then have $\text{changes-poly-at } ps \ ub = 0$ **unfolding** $ps\text{-def}$ **by** *simp*
thus *?thesis* **unfolding** $\text{changes-gt-der-def}$ $\text{changes-itv-der-def}$ $ps\text{-def}$
by (*simp add: Let-def*)
qed
moreover have $\text{roots-count } p \{x. a < x \wedge x \leq ub\} \leq \text{changes-itv-der } a \ ub \ p \wedge$
even ($\text{changes-itv-der } a \ ub \ p - \text{roots-count } p \{x. a < x \wedge x \leq ub\}$)
using $\text{budan-fourier-interval}$ [*OF ‹a < ub› ‹p ≠ 0›*].
ultimately show *?thesis* **by** *auto*
qed

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

theorem *descartes-sign:*

fixes $p :: \text{real poly}$

assumes $p \neq 0$

shows $\text{changes } (\text{coeffs } p) \geq \text{roots-count } p \{x. 0 < x\} \wedge$
even ($\text{changes } (\text{coeffs } p) - \text{roots-count } p \{x. 0 < x\}$)

using budan-fourier-gt [*OF ‹p ≠ 0›, of 0*] **unfolding** $\text{changes-gt-der-def}$
by (*simp add: changes-poly-at-pders-0*)

theorem *budan-fourier-le:*

assumes $p \neq 0$

shows $\text{changes-le-der } b \ p \geq \text{roots-count } p \{x. x \leq b\} \wedge$
even ($\text{changes-le-der } b \ p - \text{roots-count } p \{x. x \leq b\}$)

proof –

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

obtain lb **where** $lb\text{-root} : \forall p \in \text{set } ps. \forall x. \text{poly } p \ x = 0 \longrightarrow x > lb$

and $lb\text{-sgn} : \forall x \leq lb. \forall p \in \text{set } ps. \text{sgn } (\text{poly } p \ x) = \text{sgn-neg-inf } p$

and $lb < b$

using root-list-lb [*of ps b*] set-pders-nzero [*OF ‹p ≠ 0›, folded ps-def*] **by** *blast*

have $\text{roots-count } p \{x. x \leq b\} = \text{roots-count } p \{x. lb < x \wedge x \leq b\}$

proof –

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

then have $\text{roots-within } p \{x. x \leq b\} = \text{roots-within } p \{x. lb < x \wedge x \leq b\}$

using $lb\text{-root}$ **by** *fastforce*

then show *?thesis* **unfolding** roots-count-def **by** *auto*

qed
moreover have $\text{changes-le-der } b \ p = \text{changes-itv-der } lb \ b \ p$
proof –
have $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \ lb)) \ ps = \text{map } \text{sgn-neg-inf } ps$
using $\text{lb-sgn}[\text{THEN } \text{spec, of } lb, \text{simplified}]$
by $(\text{metis } (\text{mono-tags, lifting}) \text{comp-def list.map-cong0})$
hence $\text{changes-poly-at } ps \ lb = \text{changes-poly-neg-inf } ps$
unfolding $\text{changes-poly-neg-inf-def changes-poly-at-def}$
by $(\text{subst changes-map-sgn-eq,metis map-map})$
then have $\text{changes-poly-at } ps \ lb = \text{degree } p$ **unfolding** ps-def **by** simp
thus $?thesis$ **unfolding** $\text{changes-le-der-def changes-itv-der-def ps-def}$
by $(\text{simp add:Let-def})$
qed
moreover have $\text{roots-count } p \ \{x. \ lb < x \wedge x \leq b\} \leq \text{changes-itv-der } lb \ b \ p \wedge$
even $(\text{changes-itv-der } lb \ b \ p - \text{roots-count } p \ \{x. \ lb < x \wedge x \leq b\})$
using $\text{budan-fourier-interval}[\text{OF } \langle lb < b \rangle \langle p \neq 0 \rangle]$.
ultimately show $?thesis$ **by** auto
qed

2.5 Count exactly when all roots are real

definition $\text{all-roots-real}:: \text{real poly} \Rightarrow \text{bool}$ **where**
 $\text{all-roots-real } p = (\forall r \in \text{proots } (\text{map-poly of-real } p). \ \text{Im } r = 0)$

lemma $\text{all-roots-real-mult}[\text{simp}]$:
 $\text{all-roots-real } (p * q) \longleftrightarrow \text{all-roots-real } p \wedge \text{all-roots-real } q$
unfolding $\text{all-roots-real-def}$ **by** auto

lemma $\text{all-roots-real-const-iff}$:
assumes $\text{all-real:all-roots-real } p$
shows $\text{degree } p \neq 0 \longleftrightarrow (\exists x. \ \text{poly } p \ x = 0)$

proof
assume $\text{degree } p \neq 0$
moreover have $\text{degree } p = 0$ **when** $\forall x. \ \text{poly } p \ x \neq 0$
proof –
define pp **where** $pp = \text{map-poly complex-of-real } p$
have $\forall x. \ \text{poly } pp \ x \neq 0$
proof (rule ccontr)
assume $\neg (\forall x. \ \text{poly } pp \ x \neq 0)$
then obtain x **where** $\text{poly } pp \ x = 0$ **by** auto
moreover have $\text{Im } x = 0$
using $\text{all-real}[\text{unfolded all-roots-real-def, rule-format, of } x, \text{folded } pp\text{-def}] \ \langle \text{poly } pp \ x = 0 \rangle$
by auto
ultimately have $\text{poly } pp \ (\text{of-real } (\text{Re } x)) = 0$
by $(\text{simp add: complex-is-Real-iff})$
then have $\text{poly } p \ (\text{Re } x) = 0$
unfolding $pp\text{-def}$
by $(\text{metis Re-complex-of-real of-real-poly-map-poly zero-complex.simps}(1))$

```

    then show False using that by simp
  qed
  then obtain a where pp = [:of-real a:] a ≠ 0
    by (metis ‹degree p ≠ 0› constant-degree degree-map-poly
        fundamental-theorem-of-algebra of-real-eq-0-iff pp-def)
  then have p = [:a:] unfolding pp-def
    by (metis map-poly-0 map-poly-pCons of-real-0 of-real-poly-eq-iff)
  then show ?thesis by auto
  qed
  ultimately show  $\exists x. \text{poly } p \ x = 0$  by auto
next
  assume  $\exists x. \text{poly } p \ x = 0$ 
  then show degree p ≠ 0
    by (metis UNIV-I all-roots-real-def assms degree-pCons-eq-if
        imaginary-unit.sel(2) map-poly-0 nat.simps(3) order-root pCons-eq-0-iff
        proots-within-iff synthetic-div-eq-0-iff synthetic-div-pCons zero-neq-one)
  qed

lemma all-roots-real-degree:
  assumes all-roots-real p
  shows proots-count p UNIV = degree p using assms
proof (induct p rule: poly-root-induct-alt [of -  $\lambda x. \text{True}$ ])
  case 0
    then have False using imaginary-unit.sel(2) unfolding all-roots-real-def by
  auto
    then show ?case by simp
  next
  case (no-proots p)
  from all-roots-real-const-iff[OF this(2)] this(1)
  have degree p = 0 by auto
  then obtain a where p = [:a:] a ≠ 0
    by (metis degree-eq-zeroE no-proots.hyps poly-const-conv)
  then have proots p = {} by auto
  then show ?case using ‹p = [:a:]› by (simp add: proots-count-def)
  next
  case (root a p)
  define a1 where a1 = [:- a, 1:]
  have p ≠ 0 using root.prems
    apply auto
    using imaginary-unit.sel(2) unfolding all-roots-real-def by auto
  have a1 ≠ 0 unfolding a1-def by auto

  have proots-count (a1 * p) UNIV = proots-count a1 UNIV + proots-count p
  UNIV
    using ‹p ≠ 0› ‹a1 ≠ 0› by (subst proots-count-times, auto)
  also have ... = 1 + degree p
  proof -
  have proots-count a1 UNIV = 1 unfolding a1-def by (simp add: proots-count-pCons-1-iff)
  moreover have hyps: proots-count p UNIV = degree p

```

```

    apply (rule root.hyps)
    using root.premis[folded a1-def] unfolding all-roots-real-def by auto
    ultimately show ?thesis by auto
qed
also have ... = degree (a1*p)
  apply (subst degree-mult-eq)
  using ‹a1≠0› ‹p≠0› unfolding a1-def by auto
  finally show ?case unfolding a1-def .
qed

lemma all-real-roots-mobius:
  fixes a b::real
  assumes all-roots-real p and a<b
  shows all-roots-real (fcompose p [:a,b:] [:1,1:]) using assms(1)
proof (induct p rule: poly-root-induct-alt [of - λx. True])
  case 0
  then show ?case by simp
next
  case (no-roots p)
  from all-roots-real-const-iff[OF this(2)] this(1)
  have degree p=0 by auto
  then obtain a where p=[:a:] a≠0
    by (metis degree-eq-zeroE no-roots.hyps poly-const-conv)
  then show ?case by (auto simp add:all-roots-real-def)
next
  case (root x p)
  define x1 where x1=[:- x, 1:]
  define fx where fx=fcompose x1 [:a, b:] [:1, 1:]

  have all-roots-real fx
  proof (cases x=b)
    case True
    then have fx = [:a-x:] a≠x
      subgoal unfolding fx-def by (simp add:fcompose-def smult-add-right x1-def)
      subgoal using ‹a<b› True by auto
    done
    then have roots (map-poly complex-of-real fx) = {}
      by auto
    then show ?thesis unfolding all-roots-real-def by auto
  next
    case False
    then have fx = [:a-x,b-x:]
      unfolding fx-def by (simp add:fcompose-def smult-add-right x1-def)
    then have roots (map-poly complex-of-real fx) = {of-real ((x-a)/(b-x))}
      using False by (auto simp add:field-simps)
    then show ?thesis unfolding all-roots-real-def by auto
  qed
  moreover have all-roots-real (fcompose p [:a, b:] [:1, 1:])
    using root[folded x1-def] all-roots-real-mult by auto

```

ultimately show ?case
 apply (fold x1-def)
 by (auto simp add:fcompose-mult fx-def)
 qed

If all roots are real, we can use the Budan–Fourier theorem to EXACTLY count the number of real roots.

corollary budan-fourier-real:

assumes $p \neq 0$
 assumes all-roots-real p
 shows $\text{roots-count } p \{x. x \leq a\} = \text{changes-le-der } a \ p$
 $a < b \implies \text{roots-count } p \{x. a < x \wedge x \leq b\} = \text{changes-itv-der } a \ b \ p$
 $\text{roots-count } p \{x. b < x\} = \text{changes-gt-der } b \ p$

proof –

have *: $\text{roots-count } p \{x. x \leq a\} = \text{changes-le-der } a \ p$
 $\wedge \text{roots-count } p \{x. a < x \wedge x \leq b\} = \text{changes-itv-der } a \ b \ p$
 $\wedge \text{roots-count } p \{x. b < x\} = \text{changes-gt-der } b \ p$
 when $a < b$ for $a \ b$

proof –

define $c1 \ c2 \ c3$ where
 $c1 = \text{changes-le-der } a \ p - \text{roots-count } p \{x. x \leq a\}$ and
 $c2 = \text{changes-itv-der } a \ b \ p - \text{roots-count } p \{x. a < x \wedge x \leq b\}$ and
 $c3 = \text{changes-gt-der } b \ p - \text{roots-count } p \{x. b < x\}$

have $c1 \geq 0 \ c2 \geq 0 \ c3 \geq 0$

using budan-fourier-interval[OF $\langle a < b \rangle \langle p \neq 0 \rangle$] budan-fourier-gt[OF $\langle p \neq 0 \rangle$, of
 b]

budan-fourier-le[OF $\langle p \neq 0 \rangle$, of a]

unfolding c1-def c2-def c3-def by auto

moreover have $c1 + c2 + c3 = 0$

proof –

have $\text{roots-deg: roots-count } p \ UNIV = \text{degree } p$
 using all-roots-real-degree[OF $\langle \text{all-roots-real } p \rangle$].

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

unfolding changes-le-der-def changes-itv-der-def changes-gt-der-def

by (auto simp add:Let-def)

moreover have $\text{roots-count } p \{x. x \leq a\} + \text{roots-count } p \{x. a < x \wedge x \leq b\}$

$+ \text{roots-count } p \{x. b < x\} = \text{degree } p$

using $\langle p \neq 0 \rangle \langle a < b \rangle$

apply (subst roots-count-union-disjoint[symmetric], auto)+

apply (subst roots-deg[symmetric])

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

ultimately show ?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 ?thesis unfolding c1-def c2-def c3-def by auto

```

qed
show roots-count  $p \{x. x \leq a\} = \text{changes-le-der } a \ p$  using *[of a a+1] by auto
show roots-count  $p \{x. a < x \wedge x \leq b\} = \text{changes-itv-der } a \ b \ p$  when  $a < b$ 
using *[OF that] by auto
show roots-count  $p \{x. b < x\} = \text{changes-gt-der } b \ p$ 
using *[of b-1 b] by auto
qed

```

Similarly, Descartes' rule of sign counts exactly when all roots are real.

```

corollary descartes-sign-real:
  fixes  $p::\text{real poly}$  and  $a \ b::\text{real}$ 
  assumes  $p \neq 0$ 
  assumes all-roots-real  $p$ 
  shows roots-count  $p \{x. 0 < x\} = \text{changes (coeffs } p)$ 
  using budan-fourier-real(3)[OF <p≠0> <all-roots-real p>]
  unfolding changes-gt-der-def by (simp add:changes-poly-at-pders-0)

end

```

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.

Our formal proof is inspired by Theorem 10.5.6 in Rahman, Q.I., Schmeisser, G.: Analytic Theory of Polynomials. Oxford University Press (2002).

3.1 More results for *smods*

```

lemma last-smods-gcd:
  fixes  $p \ q ::\text{real poly}$ 
  defines  $pp \equiv \text{last (smods } p \ q)$ 
  assumes  $p \neq 0$ 
  shows  $pp = \text{smult (lead-coeff } pp) (\text{gcd } p \ q)$ 
  using  $\langle p \neq 0 \rangle$  unfolding pp-def
proof (induct smods p q arbitrary:p q rule:length-induct)
  case 1
  have ?case when  $q = 0$ 
    using that <p≠0> by (simp add:smult-normalize-field-eq)
  moreover have ?case when  $q \neq 0$ 
  proof –

```

```

define r where  $r = -(p \bmod q)$ 
have smods-cons:smods  $p\ q = p \# \text{smods } q\ r$ 
  unfolding r-def using  $\langle p \neq 0 \rangle$  by simp
have  $\text{last } (\text{smods } q\ r) = \text{smult } (\text{lead-coeff } (\text{last } (\text{smods } q\ r))) (\text{gcd } q\ r)$ 
  by (metis 1.hyps length-Cons lessI smods-cons that)
moreover have  $\text{gcd } p\ q = \text{gcd } q\ r$ 
  unfolding r-def by (simp add: gcd.commute that)
ultimately show ?thesis unfolding smods-cons using  $\langle q \neq 0 \rangle$ 
  by simp
qed
ultimately show ?case by argo
qed

```

```

lemma last-smods-nzero:
  assumes  $p \neq 0$ 
  shows  $\text{last } (\text{smods } p\ q) \neq 0$ 
  by (metis assms last-in-set no-0-in-smods smods-nil-eq)

```

3.2 Alternative signed remainder sequences

```

function smods-ext::real poly  $\Rightarrow$  real poly  $\Rightarrow$  real poly list where
  smods-ext  $p\ q = (\text{if } p=0 \text{ then } [] \text{ else}$ 
    ( $\text{if } p \bmod q \neq 0$ 
       $\text{then Cons } p (\text{smods-ext } q\ (-(p \bmod q)))$ 
       $\text{else Cons } p (\text{smods-ext } q\ (\text{pderiv } q))$ 
    )
  )
  by auto
termination
  apply (relation measure  $(\lambda(p,q).\text{if } p=0 \text{ then } 0 \text{ else if } q=0 \text{ then } 1 \text{ else } 2+\text{degree } q)$ )
  using degree-mod-less by (auto simp add:degree-pderiv pderiv-eq-0-iff)

```

```

lemma smods-ext-prefix:
  fixes  $p\ q::\text{real poly}$ 
  defines  $pp \equiv \text{last } (\text{smods } p\ q)$ 
  assumes  $p \neq 0\ q \neq 0$ 
  shows  $\text{smods-ext } p\ q = \text{smods } p\ q @ \text{tl } (\text{smods-ext } pp\ (\text{pderiv } pp))$ 
  unfolding pp-def using assms(2,3)
proof (induct smods-ext p q arbitrary:p q rule:length-induct)
  case 1
  have ?case when  $p \bmod q \neq 0$ 
  proof –
    define pp where  $pp = \text{last } (\text{smods } q\ (-(p \bmod q)))$ 
    have smods-cons:smods  $p\ q = p \# \text{smods } q\ (-(p \bmod q))$ 
      using  $\langle p \neq 0 \rangle$  by auto
    then have  $pp\text{-last}:pp = \text{last } (\text{smods } p\ q)$  unfolding pp-def
      by (simp add: 1.prem(2) pp-def)
    have smods-ext-cons:smods-ext  $p\ q = p \# \text{smods-ext } q\ (-(p \bmod q))$ 
      using that  $\langle p \neq 0 \rangle$  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≠0⟩ ⟨q≠0⟩ that by auto
moreover have smods-ext p q = [p,q]
using that ⟨p≠0⟩ by auto
ultimately show ?case using ⟨p≠0⟩ ⟨q≠0⟩ that(1) by auto
qed
moreover have ?case when p mod q = 0 pderiv q ≠ 0
proof -
have smods-cons:smods p q = [p,q]
using ⟨p≠0⟩ ⟨q≠0⟩ that by auto
have smods-ext-cons:smods-ext p q = p#smods-ext q (pderiv q)
using that ⟨p≠0⟩ by auto
show ?case unfolding smods-cons smods-ext-cons
apply (simp del:smods-ext.simps)
by (simp add: 1.prem(2))
qed
ultimately show ?case by argo
qed

lemma no-0-in-smods-ext: 0 ∉ 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*:: $real \Rightarrow real \Rightarrow real \Rightarrow real \Rightarrow int$ **where**

changes-itv-smods-ext a b p q = (let ps = smods-ext p q in *changes-poly-at ps a* - *changes-poly-at ps b*)

definition *changes-gt-smods-ext*:: $real \Rightarrow real \Rightarrow real \Rightarrow int$ **where**
changes-gt-smods-ext a p q = (let ps = smods-ext p q in *changes-poly-at ps a* - *changes-poly-pos-inf ps*)

definition *changes-le-smods-ext*:: $real \Rightarrow real \Rightarrow real \Rightarrow int$ **where**

changes-le-smods-ext b p q = (let ps = *smods-ext* p q in *changes-poly-neg-inf* ps
– *changes-poly-at* ps b)

definition *changes-R-smods-ext*:: *real poly* \Rightarrow *real poly* \Rightarrow *int* **where**
changes-R-smods-ext p q = (let ps = *smods-ext* p q in *changes-poly-neg-inf* ps
– *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 *proots-count* p $\{x. a < x \wedge x < b\}$ = *changes-itu-smods-ext* a b p (*pderiv* p)

using *assms*(2,3)

proof (*induct smods-ext* p (*pderiv* p) *arbitrary:p* *rule:length-induct*)

case 1

have $p \neq 0$ **using** \langle *poly* p $a \neq 0$ \rangle **by** *auto*

have $?case$ **when** *pderiv* p = 0

proof –

obtain c **where** $p = [:c:]$ $c \neq 0$

using \langle $p \neq 0$ \rangle \langle *pderiv* p = 0 \rangle *pderiv-iszero* **by** *force*

then have *proots-count* p $\{x. a < x \wedge x < b\}$ = 0

unfolding *proots-count-def* **by** *auto*

moreover have *changes-itu-smods-ext* a b p (*pderiv* p) = 0

unfolding *changes-itu-smods-ext-def* **using** \langle $p = [:c:]$ \rangle \langle $c \neq 0$ \rangle **by** *auto*

ultimately show $?thesis$ **by** *auto*

qed

moreover have $?case$ **when** *pderiv* $p \neq 0$

proof –

define pp **where** pp = *last* (*smods* p (*pderiv* p))

define lp **where** lp = *lead-coeff* pp

define S **where** $S = \{x. a < x \wedge x < b\}$

have *prefix:smods-ext* p (*pderiv* p) = *smods* p (*pderiv* p) @ *tl* (*smods-ext* pp
(*pderiv* pp))

using *smods-ext-prefix*[*OF* \langle $p \neq 0$ \rangle \langle *pderiv* $p \neq 0$ \rangle ,*folded pp-def*] .

have pp -*gcd*: pp = *smult* lp (*gcd* p (*pderiv* p))

using *last-smods-gcd*[*OF* \langle $p \neq 0$ \rangle ,*of pderiv p*,*folded pp-def lp-def*] .

have $pp \neq 0$ $lp \neq 0$ **unfolding** *pp-def lp-def*

subgoal by (*rule last-smods-nzero*[*OF* \langle $p \neq 0$ \rangle])

subgoal using \langle *last* (*smods* p (*pderiv* p)) $\neq 0$ \rangle **by** *auto*

done

have *poly* pp $a \neq 0$ *poly* pp $b \neq 0$

unfolding pp -*gcd* **using** \langle *poly* p $a \neq 0$ \rangle \langle *poly* p $b \neq 0$ \rangle \langle $lp \neq 0$ \rangle

by (*simp-all add:poly-gcd-0-iff*)

have *proots-count* pp S = *changes-itu-smods-ext* a b pp (*pderiv* pp) **unfolding**
S-def

proof (*rule 1(1)*[*rule-format,of smods-ext pp (pderiv pp) pp*])

show *length* (*smods-ext* pp (*pderiv* pp)) < *length* (*smods-ext* p (*pderiv* p))

unfolding prefix by (*simp add: $\langle p \neq 0 \rangle$ that*)
qed (*use $\langle \text{poly } pp \ a \neq 0 \rangle \ \langle \text{poly } pp \ b \neq 0 \rangle$ in *simp-all**)
moreover have $\text{proots-count } p \ S = \text{card } (\text{proots-within } p \ S) + \text{proots-count } pp \ S$
proof –
have $(\sum_{r \in \text{proots-within } p \ S} \text{order } r \ p) = (\sum_{r \in \text{proots-within } p \ S} \text{order } r \ pp + 1)$
proof (*rule sum.cong*)
fix x **assume** $x \in \text{proots-within } p \ S$
have $\text{order } x \ pp = \text{order } x \ (\text{gcd } p \ (\text{pderiv } p))$
unfolding *pp-gcd using $\langle lp \neq 0 \rangle$ by (simp add:order-smult)*
also have $\dots = \min (\text{order } x \ p) (\text{order } x \ (\text{pderiv } p))$
apply (*subst order-gcd*)
using $\langle p \neq 0 \rangle \ \langle \text{pderiv } p \neq 0 \rangle$ **by** *simp-all*
also have $\dots = \text{order } x \ (\text{pderiv } p)$
apply (*subst order-pderiv*)
using $\langle \text{pderiv } p \neq 0 \rangle \ \langle p \neq 0 \rangle \ \langle x \in \text{proots-within } p \ S \rangle$ **order-root by auto**
finally have $\text{order } x \ pp = \text{order } x \ (\text{pderiv } p)$.
moreover have $\text{order } x \ p = \text{order } x \ (\text{pderiv } p) + 1$
apply (*subst order-pderiv*)
using $\langle \text{pderiv } p \neq 0 \rangle \ \langle p \neq 0 \rangle \ \langle x \in \text{proots-within } p \ S \rangle$ **order-root by auto**
ultimately show $\text{order } x \ p = \text{order } x \ pp + 1$ **by auto**
qed *simp*
also have $\dots = \text{card } (\text{proots-within } p \ S) + (\sum_{r \in \text{proots-within } p \ S} \text{order } r \ pp)$
apply (*subst sum.distrib*)
by auto
also have $\dots = \text{card } (\text{proots-within } p \ S) + (\sum_{r \in \text{proots-within } pp \ S} \text{order } r \ pp)$
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 $\langle p \neq 0 \rangle$ **by auto**
subgoal unfolding *pp-gcd using $\langle lp \neq 0 \rangle$ by (auto simp:poly-gcd-0-iff)*
subgoal unfolding *pp-gcd using $\langle lp \neq 0 \rangle$*
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 $\text{card } (\text{proots-within } p \ S) = \text{changes-itv-smods } a \ b \ p \ (\text{pderiv } p)$
using *sturm-interval[OF $\langle a < b \rangle \ \langle \text{poly } p \ a \neq 0 \rangle \ \langle \text{poly } p \ b \neq 0 \rangle$,symmetric]*
unfolding *S-def proots-within-def*
by (*auto intro!:arg-cong[where f=card]*)
moreover have $\text{changes-itv-smods-ext } a \ b \ p \ (\text{pderiv } p)$

$= \text{changes-itv-smods } a \ b \ p \ (\text{pderiv } p) + \text{changes-itv-smods-ext } a \ b \ pp$
 $(\text{pderiv } pp)$

proof –

define $xs \ ys$ **where** $xs = \text{smods } p \ (\text{pderiv } p)$ **and** $ys = \text{smods-ext } pp \ (\text{pderiv } pp)$

have $xys: xs \neq [] \ ys \neq [] \ \text{last } xs = \text{hd } ys \ \text{poly } (\text{last } xs) \ a \neq 0 \ \text{poly } (\text{last } xs) \ b \neq 0$

subgoal unfolding $xs\text{-def}$ **using** $\langle p \neq 0 \rangle$ **by** $auto$

subgoal unfolding $ys\text{-def}$ **using** $\langle pp \neq 0 \rangle$ **by** $auto$

subgoal using $\langle pp \neq 0 \rangle$ **unfolding** $xs\text{-def}$ $ys\text{-def}$

apply $(\text{fold } pp\text{-def})$

by $auto$

subgoal using $\langle \text{poly } pp \ a \neq 0 \rangle$ **unfolding** $pp\text{-def}$ $xs\text{-def}$.

subgoal using $\langle \text{poly } pp \ b \neq 0 \rangle$ **unfolding** $pp\text{-def}$ $xs\text{-def}$.

done

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

$ys \ a$

proof –

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

unfolding $\text{changes-poly-at-def}$

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

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

using $\text{that } xys$ **by** $(\text{auto } \text{simp } \text{add: hd-map last-map})$

also have $\dots = \text{changes-poly-at } xs \ a + \text{changes-poly-at } ys \ a$

unfolding $\text{changes-poly-at-def}$

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

using xys **by** $(\text{auto } \text{simp } \text{add: hd-map last-map})$

finally show $?thesis$.

qed

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

$\text{changes-poly-at } ys \ b$

proof –

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

unfolding $\text{changes-poly-at-def}$

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

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

using $\text{that } xys$ **by** $(\text{auto } \text{simp } \text{add: hd-map last-map})$

also have $\dots = \text{changes-poly-at } xs \ b + \text{changes-poly-at } ys \ b$

unfolding $\text{changes-poly-at-def}$

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

using xys **by** $(\text{auto } \text{simp } \text{add: hd-map last-map})$

finally show $?thesis$.

qed

ultimately show $?thesis$ **unfolding** $\text{changes-itv-smods-ext-def}$ $\text{changes-itv-smods-def}$

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

by $auto$

qed

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

by $auto$

qed

ultimately show $?case$ **by** $argo$

qed

theorem *sturm-ext-above*:

assumes *poly p a ≠ 0*

shows *roots-count p {x. a < x} = changes-gt-smods-ext a p (pderiv p)*

proof –

define *ps* where *ps ≡ smods-ext p (pderiv p)*

have *p ≠ 0* and *p ∈ set ps* using *⟨poly p a ≠ 0⟩ 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 > a*

using *root-list-ub[OF no-0-in-smods-ext, of p pderiv p, folded ps-def]*

by *auto*

have *roots-count p {x. a < x} = roots-count p {x. a < x ∧ x < ub}*

unfolding *roots-count-def*

apply (*rule sum.cong*)

by (*use ub ⟨p ∈ set ps⟩ in auto*)

moreover have *changes-gt-smods-ext a p (pderiv p) = changes-itv-smods-ext a ub p (pderiv p)*

proof –

have *map (sgn ∘ (λp. poly p ub)) ps = map sgn-pos-inf ps*

using *ub-sgn[THEN spec, of ub, simplified]*

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*)

thus *?thesis unfolding changes-gt-smods-ext-def changes-itv-smods-ext-def ps-def*

by *metis*

qed

moreover have *poly p ub ≠ 0* using *ub ⟨p ∈ set ps⟩* by *auto*

ultimately show *?thesis using sturm-ext-interval[OF ⟨ub > a⟩ assms]* by *auto*

qed

theorem *sturm-ext-below*:

assumes *poly p b ≠ 0*

shows *roots-count p {x. x < b} = changes-le-smods-ext b p (pderiv p)*

proof –

define *ps* where *ps ≡ smods-ext p (pderiv p)*

have *p ≠ 0* and *p ∈ set ps* using *⟨poly p b ≠ 0⟩ ps-def* 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 < b*

using *root-list-lb[OF no-0-in-smods-ext, of p pderiv p, folded ps-def]*

by *auto*

have *roots-count p {x. x < b} = roots-count p {x. lb < x ∧ x < b}*

unfolding *roots-count-def* by (*rule sum.cong, insert lb ⟨p ∈ set ps⟩, auto*)

moreover have *changes-le-smods-ext b p (pderiv p) = changes-itv-smods-ext lb b p (pderiv p)*

proof –
have $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \text{ lb})) \text{ ps} = \text{map } \text{sgn-neg-inf } \text{ps}$
using $\text{lb-sgn}[\text{THEN spec, of lb, simplified}]$
by $(\text{metis } (\text{mono-tags, lifting}) \text{comp-def list.map-cong0})$
hence $\text{changes-poly-at } \text{ps } \text{lb} = \text{changes-poly-neg-inf } \text{ps}$
unfolding $\text{changes-poly-neg-inf-def changes-poly-at-def}$
by $(\text{subst changes-map-sgn-eq,metis map-map})$
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** $\text{lb } \langle p \in \text{set } \text{ps} \rangle$ **by** auto
ultimately show $?thesis$ **using** $\text{sturm-ext-interval}[\text{OF } \langle \text{lb} < b \rangle - \text{assms}]$ **by** auto
qed

theorem sturm-ext-R :

assumes $p \neq 0$
shows $\text{roots-count } p \text{ UNIV} = \text{changes-R-smods-ext } p \text{ (pderiv } p)$
proof –
define ps **where** $\text{ps} \equiv \text{smods-ext } p \text{ (pderiv } p)$
have $p \in \text{set } \text{ps}$ **using** $\text{ps-def } \langle p \neq 0 \rangle$ **by** auto
obtain lb **where** $\text{lb} : \forall p \in \text{set } \text{ps}. \forall x. \text{poly } p \text{ } x = 0 \longrightarrow x > \text{lb}$
and $\text{lb-sgn} : \forall x \leq \text{lb}. \forall p \in \text{set } \text{ps}. \text{sgn } (\text{poly } p \text{ } x) = \text{sgn-neg-inf } p$
and $\text{lb} < 0$
using $\text{root-list-lb}[\text{OF no-0-in-smods-ext, of } p \text{ pderiv } p, \text{folded ps-def}]$
by auto
obtain ub **where** $\text{ub} : \forall p \in \text{set } \text{ps}. \forall x. \text{poly } p \text{ } x = 0 \longrightarrow x < \text{ub}$
and $\text{ub-sgn} : \forall x \geq \text{ub}. \forall p \in \text{set } \text{ps}. \text{sgn } (\text{poly } p \text{ } x) = \text{sgn-pos-inf } p$
and $\text{ub} > 0$
using $\text{root-list-ub}[\text{OF no-0-in-smods-ext, of } p \text{ pderiv } p, \text{folded ps-def}]$
by auto
have $\text{roots-count } p \text{ UNIV} = \text{roots-count } p \{x. \text{lb} < x \wedge x < \text{ub}\}$
unfolding roots-count-def **by** $(\text{rule sum.cong, insert lb ub } \langle p \in \text{set } \text{ps} \rangle, \text{auto})$
moreover **have** $\text{changes-R-smods-ext } p \text{ (pderiv } p) = \text{changes-itv-smods-ext lb ub}$
 $p \text{ (pderiv } p)$
proof –
have $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \text{ lb})) \text{ ps} = \text{map } \text{sgn-neg-inf } \text{ps}$
and $\text{map } (\text{sgn} \circ (\lambda p. \text{poly } p \text{ ub})) \text{ ps} = \text{map } \text{sgn-pos-inf } \text{ps}$
using $\text{lb-sgn}[\text{THEN spec, of lb, simplified}] \text{ub-sgn}[\text{THEN spec, of ub, simplified}]$
by $(\text{metis } (\text{mono-tags, lifting}) \text{comp-def list.map-cong0})$
hence $\text{changes-poly-at } \text{ps } \text{lb} = \text{changes-poly-neg-inf } \text{ps}$
 $\wedge \text{changes-poly-at } \text{ps } \text{ub} = \text{changes-poly-pos-inf } \text{ps}$
unfolding $\text{changes-poly-neg-inf-def changes-poly-at-def changes-poly-pos-inf-def}$
by $(\text{subst } (1 \ 3) \ \text{changes-map-sgn-eq,metis map-map})$
thus $?thesis$ **unfolding** $\text{changes-R-smods-ext-def changes-itv-smods-ext-def ps-def}$
by metis
qed
moreover **have** $\text{poly } p \text{ lb} \neq 0$ **and** $\text{poly } p \text{ ub} \neq 0$ **using** $\text{lb ub } \langle p \in \text{set } \text{ps} \rangle$ **by** auto
moreover **have** $\text{lb} < \text{ub}$ **using** $\langle \text{lb} < 0 \rangle \langle 0 < \text{ub} \rangle$ **by** auto
ultimately show $?thesis$ **using** $\text{sturm-ext-interval}$ **by** auto

qed

end

4 Descartes Roots Test

theory *Descartes-Roots-Test* **imports** *Budan-Fourier*
begin

The Descartes roots test is a consequence of Descartes' rule of signs: through counting sign variations on coefficients of a base-transformed (i.e. Taylor shifted) polynomial, it can over-approximate the number of real roots (counting multiplicity) within an interval. Its ability is similar to the Budan–Fourier theorem, but is far more efficient in practice. Therefore, this test is widely used in modern root isolation procedures.

More information can be found in the wiki page about Vincent's theorem: https://en.wikipedia.org/wiki/Vincent%27s_theorem and Collins and Akritas's classic paper of root isolation: Collins, G.E., Akritas, A.G.: Polynomial real root isolation using Descarte's rule of signs. SYMSACC. 272–275 (1976). A more modern treatment is available from a recent implementation of isolating real roots: Kobel, A., Rouillier, F., Sagraloff, M.: Computing Real Roots of Real Polynomials ... and now For Real! Proceedings of ISSAC '16, New York, New York, USA (2016).

lemma *bij-betw-pos-interval*:

fixes $a\ b::\text{real}$

assumes $a < b$

shows *bij-betw* $(\lambda x. (a+b * x) / (1+x)) \{x. x > 0\} \{x. a < x \wedge x < b\}$

proof (*rule* *bij-betw-imageI*)

show *inj-on* $(\lambda x. (a + b * x) / (1 + x)) \{x. 0 < x\}$

unfolding *inj-on-def*

apply (*auto simp add:field-simps*)

using *assms crossproduct-noteq* **by** *fastforce*

have $x \in (\lambda x. (a + b * x) / (1 + x)) \{x. 0 < x\}$ **when** $a < x < b$ **for** x

proof (*rule* *rev-image-eqI*[*of* $(x-a)/(b-x)$])

define bx **where** $bx = b - x$

have $x = b - bx$ **unfolding** *bx-def* **by** *auto*

have $bx \neq 0$ $b > a$ **unfolding** *bx-def* **using** *that* **by** *auto*

then show $x = (a + b * ((x - a) / (b - x))) / (1 + (x - a) / (b - x))$

apply (*fold* *bx-def,unfold* x)

by (*auto simp add:field-simps*)

show $(x - a) / (b - x) \in \{x. 0 < x\}$ **using** *that* **by** *auto*

qed

then show $(\lambda x. (a + b * x) / (1 + x)) \{x. 0 < x\} = \{x. a < x \wedge x < b\}$

using *assms* **by** (*auto simp add:divide-simps algebra-simps*)

qed

lemma *proots-sphere-pos-interval*:

```

fixes  $a\ b::\text{real}$ 
defines  $q1\equiv[:a,b:]$  and  $q2\equiv[:1,1:]$ 
assumes  $p\neq 0\ a<b$ 
shows  $\text{roots-count } p \{x. a < x \wedge x < b\} = \text{roots-count } (fcompose\ p\ q1\ q2) \{x.$ 
 $0 < x\}$ 
apply (rule roots-fcompose-bij-eq[OF - <p≠0>])
unfolding  $q1\text{-def } q2\text{-def}$  using bij-betw-pos-interval[OF <a<b> <a<b>]
by (auto simp add: algebra-simps infinite-UNIV-char-0)

```

definition *descartes-roots-test*:: $\text{real} \Rightarrow \text{real} \Rightarrow \text{real poly} \Rightarrow \text{nat}$ **where**
descartes-roots-test $a\ b\ p = \text{nat } (\text{changes } (\text{coeffs } (fcompose\ p\ [:a,b:]\ [:1,1:])))$

theorem *descartes-roots-test*:

```

fixes  $p::\text{real poly}$ 
assumes  $p\neq 0\ a<b$ 
shows  $\text{roots-count } p \{x. a < x \wedge x < b\} \leq \text{descartes-roots-test } a\ b\ p \wedge$ 
 $\text{even } (\text{descartes-roots-test } a\ b\ p - \text{roots-count } p \{x. a < x \wedge x < b\})$ 

```

proof –

```

define  $q$  where  $q=fcompose\ p\ [:a,b:]\ [:1,1:]$ 
have  $q\neq 0$ 
unfolding  $q\text{-def}$ 
apply (rule fcompose-nzero[OF <p≠0>])
using  $<a<b>$  infinite-UNIV-char-0 by auto
have  $\text{roots-count } p \{x. a < x \wedge x < b\} = \text{roots-count } q \{x. 0 < x\}$ 
using roots-sphere-pos-interval[OF <p≠0> <a<b>,folded q-def].
moreover have  $\text{int } (\text{roots-count } q \{x. 0 < x\}) \leq \text{changes } (\text{coeffs } q) \wedge$ 
 $\text{even } (\text{changes } (\text{coeffs } q) - \text{int } (\text{roots-count } q \{x. 0 < x\}))$ 
by (rule descartes-sign[OF <q≠0>])
then have  $\text{roots-count } q \{x. 0 < x\} \leq \text{nat } (\text{changes } (\text{coeffs } q)) \wedge$ 
 $\text{even } (\text{nat } (\text{changes } (\text{coeffs } q)) - \text{roots-count } q \{x. 0 < x\})$ 
using even-nat-iff by auto
ultimately show ?thesis
unfolding descartes-roots-test-def
apply (fold q-def)
by auto

```

qed

The roots test *descartes-roots-test* is exact if its result is 0 or 1.

corollary *descartes-roots-test-zero*:

```

fixes  $p::\text{real poly}$ 
assumes  $p\neq 0\ a<b\ \text{descartes-roots-test } a\ b\ p = 0$ 
shows  $\forall x. a < x \wedge x < b \longrightarrow \text{poly } p\ x\neq 0$ 

```

proof –

```

have  $\text{roots-count } p \{x. a < x \wedge x < b\} = 0$ 
using descartes-roots-test[OF assms(1,2)] assms(3) by auto
from roots-count-0-imp-empty[OF this <p≠0>]
show ?thesis by auto

```

qed

corollary *descartes-roots-test-one*:

fixes $p::\text{real poly}$
assumes $p \neq 0 \ a < b \ \text{descartes-roots-test } a \ b \ p = 1$
shows $\text{roots-count } p \ \{x. \ a < x \wedge x < b\} = 1$
using $\text{descartes-roots-test}[OF \ \langle p \neq 0 \rangle \ \langle a < b \rangle] \ \langle \text{descartes-roots-test } a \ b \ p = 1 \rangle$
by (*metis dvd-diffD even-zero le-neq-implies-less less-one odd-one*)

Similar to the Budan–Fourier theorem, the Descartes roots test result is exact when all roots are real.

corollary *descartes-roots-test-real*:

fixes $p::\text{real poly}$
assumes $p \neq 0 \ a < b$
assumes *all-roots-real* p
shows $\text{roots-count } p \ \{x. \ a < x \wedge x < b\} = \text{descartes-roots-test } a \ b \ p$
proof –
define q **where** $q = fcompose \ p \ [:a,b:] \ [:1,1:]$
have $q \neq 0$
unfolding $q\text{-def}$ **using** *assms*
by (*metis all-real-roots-mobius all-roots-real-const-iff degree-0 poly-0*)
have $\text{roots-count } p \ \{x. \ a < x \wedge x < b\} = \text{roots-count } q \ \{x. \ 0 < x\}$
using $\text{roots-sphere-pos-interval}[OF \ \langle p \neq 0 \rangle \ \langle a < b \rangle, \text{folded } q\text{-def}]$.
moreover have $\text{int} (\text{roots-count } q \ \{x. \ 0 < x\}) = \text{changes} (\text{coeffs } q)$
apply (*rule descartes-sign-real[OF \ \langle q \neq 0 \rangle]*)
unfolding $q\text{-def}$ **by** (*rule all-real-roots-mobius[OF \ \langle all-roots-real \ p \rangle \ \langle a < b \rangle]*)
then have $\text{roots-count } q \ \{x. \ 0 < x\} = \text{nat} (\text{changes} (\text{coeffs } q))$
by *simp*
ultimately show *?thesis* **unfolding** $\text{descartes-roots-test-def } q\text{-def}$
by *argo*
qed
end

5 Acknowledgements

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

References

- [1] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
- [2] M. Eberl. Sturm’s theorem. *Archive of Formal Proofs*, Jan. 2014. http://isa-afp.org/entries/Sturm_Sequences.html, Formal proof development.

- [3] M. Eberl. Descartes' rule of signs. *Archive of Formal Proofs*, Dec. 2015. http://isa-afp.org/entries/Descartes_Sign_Rule.html, Formal proof development.
- [4] J. Harrison. Verifying the accuracy of polynomial approximations in HOL. In E. L. Gunter and A. Felty, editors, *Theorem Proving in Higher Order Logics: 10th International Conference, TPHOLs'97*, volume 1275 of *Lecture Notes in Computer Science*, pages 137–152, Murray Hill, NJ, 1997. Springer-Verlag.
- [5] W. Li. The Sturm–Tarski Theorem. *Archive of Formal Proofs*, Sept. 2014.
- [6] W. Li. Count the Number of Complex Roots. *Archive of Formal Proofs*, Oct. 2017.
- [7] W. Li and L. C. Paulson. Evaluating Winding Numbers and Counting Complex Roots through Cauchy Indices in Isabelle/HOL. *CoRR*, abs/1804.03922, 2018.
- [8] A. Mahboubi and C. Cohen. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Logical Methods in Computer Science*, 8(1), 2012.
- [9] A. Narkawicz, C. A. Muñoz, and A. Dutle. Formally-Verified Decision Procedures for Univariate Polynomial Computation Based on Sturm's and Tarski's Theorems. *Journal of Automated Reasoning*, 54(4):285–326, 2015.
- [10] Q. I. Rahman and G. Schmeisser. *Analytic Theory of Polynomials*. Oxford University Press, 2002.