Cauchy's Mean Theorem and the Cauchy-Schwarz Inequality

Benjamin Porter

November 27, 2007

Contents

Abstract

This document presents the mechanised proofs of two popular theorems attributed to Augustin Louis Cauchy - Cauchy's Mean Theorem and the Cauchy-Schwarz Inequality.

Chapter 1

Cauchy's Mean Theorem

theory *CauchysMeanTheorem*
imports *Complex-Main*
begin

1.1 Abstract

The following document presents a proof of Cauchy's Mean theorem formalised in the Isabelle/Isar theorem proving system.

Theorem: For any collection of positive real numbers the geometric mean is always less than or equal to the arithmetic mean. In mathematical terms:

$$\sqrt[n]{x_1 x_2 \ldots x_n} \leq \frac{x_1 + \ldots + x_n}{n}$$

We will use the term *mean* to denote the arithmetic mean and *gmean* to denote the geometric mean.

Informal Proof:

This proof is based on the proof presented in [1]. First we need an auxillary lemma (the proof of which is presented formally below) that states:

Given two pairs of numbers of equal sum, the pair with the greater product is the pair with the least difference. Using this lemma we now present the proof -

Given any collection C of positive numbers with mean M and product P and with some element not equal to M we can choose two elements from the collection, a and b where $a > M$ and $b < M$. Remove these elements from the collection and replace them with two new elements, a' and b' such that $a' = M$ and $a' + b' = a + b$. This new collection C' now has a greater product P' but equal mean with respect to C. We can continue in this fashion until we have a collection C_n such that $P_n > P$ and $M_n = M$, but C_n has all its elements equal to M and thus $P_n = M^n$. Using the definition of geometric and arithmetic means above we can see that for any collection of positive

elements E it is always true that gmean E $\leq$ mean E. QED.

[1] Dorrie, H. "100 Great Problems of Elementary Mathematics." 1965, Dover.

1.2 Formal proof

1.2.1 Collection sum and product

The finite collections of numbers will be modelled as lists. We then define sum and product operations over these lists.

Sum and product definitions

definition
listsum :: (*real list*) ⇒ *real* ($\sum$:- [*999*] *1000*) **where**
listsum xs = *foldr op+ xs 0*

definition
listprod :: (*real list*) ⇒ *real* ($\prod$:- [*999*] *1000*) **where**
listprod xs = *foldr op∗ xs 1*

lemma *listsum-empty* [*simp*]: $\sum$:[] = *0*
unfolding *listsum-def* **by** *simp*

lemma *listsum-cons* [*simp*]: $\sum$:(*a*#*b*) = *a* + $\sum$:*b*
unfolding *listsum-def* **by** (*induct b*) *simp-all*

lemma *listprod-empty* [*simp*]: $\prod$:[] = *1*
unfolding *listprod-def* **by** *simp*

lemma *listprod-cons* [*simp*]: $\prod$:(*a*#*b*) = *a* ∗ $\prod$:*b*
unfolding *listprod-def* **by** (*induct b*) *simp-all*

Properties of sum and product

We now present some useful properties of sum and product over collections.

These lemmas just state that if all the elements in a collection C are less (greater than) than some value m, then the sum will less than (greater than) $m * length(C)$.

lemma *listsum-mono-lt* [*rule-format*]:
fixes *xs*::*real list*
shows *xs* ≠ [] ∧ (∀ *x*∈ *set xs*. *x* < *m*)
⟶ (($\sum$:*xs*) < (*m*∗(*real* (*length xs*))))
proof (*induct xs*)
case *Nil* **show** *?case* **by** *simp*
next

```
  case (Cons y ys)
  {
    assume ant: y#ys ≠ [] ∧ (∀ x∈set(y#ys). x < m)
    hence ylm: y < m by simp
    have ∑:(y#ys) < m * real (length (y#ys))
    proof cases
      assume ys ≠ []
      moreover with ant have ∀ x∈set ys. x < m by simp
      moreover with calculation Cons have ∑:ys < m*real (length ys) by simp
      hence ∑:ys + y < m*real(length ys) + y by simp
     with ylm have ∑:(y#ys) < m*(real(length ys) + 1) by(simp add:ring-simps)
      with real-of-nat-Suc have ∑:(y#ys) < m*(real(length ys + 1))
        apply -
        apply (drule meta-spec [of - length ys])
        apply (subst(asm) eq-sym-conv)
        by simp
      hence ∑:(y#ys) < m*(real (length(y#ys))) by simp
      thus ?thesis .
    next
      assume ¬ (ys ≠ [])
      hence ys = [] by simp
      with ylm show ?thesis by simp
    qed
  }
  thus ?case by simp
qed


lemma listsum-mono-gt [rule-format]:
  fixes xs::real list
  shows xs ≠ [] ∧ (∀ x∈set xs. x > m)
        ⟶ ((∑:xs) > (m*(real (length xs))))
```

proof omitted

qed

If a is in C then the sum of the collection D where D is C with a removed is the sum of C minus a.

```
lemma listsum-rmv1:
  a ∈ set xs ⟹ ∑:(remove1 a xs) = ∑:xs - a
by (induct xs) auto
```

A handy addition and division distribution law over collection sums.

```
lemma list-sum-distrib-aux:
  shows (∑:xs/n + ∑:xs) = (1 + (1/n)) * ∑:xs
proof (induct xs)
  case Nil show ?case by simp
next
  case (Cons x xs)
```

```
  show ?case
  proof −
    have
      ∑:(x#xs)/n = x/n + ∑:xs/n
      by (simp add: add-divide-distrib)
    also with Cons have
      ... = x/n + (1+1/n)*∑:xs − ∑:xs
      by simp
    finally have
      ∑:(x#xs) / n + ∑:(x#xs) = x/n + (1+1/n)*∑:xs − ∑:xs + ∑:(x#xs)
      by simp
    also have
      ... = x/n + (1+(1/n)− 1)*∑:xs + ∑:(x#xs)
      by (subst real-mult-1 [symmetric, of ∑:xs], simp only: ring-simps)
    also have
      ... = x/n + (1/n)*∑:xs + ∑:(x#xs)
      by simp
    also have
      ... = (1/n)*∑:(x#xs) + 1*∑:(x#xs) by(simp add:ring-simps)
    finally show ?thesis by (simp only: ring-simps)
  qed
qed

lemma remove1-retains-prod:
  fixes a::real and xs::real list
  shows a : set xs ⟶ ∏:xs = ∏:(remove1 a xs) * a
  (is ?P xs)
proof (induct xs)
  case Nil
  show ?case by simp
next
  case (Cons aa list)
  assume plist: ?P list
  show ?P (aa#list)
  proof
    assume aml: a : set(aa#list)
    show ∏:(aa # list) = ∏:remove1 a (aa # list) * a
    proof (cases)
      assume aeq: a = aa
      hence
        remove1 a (aa#list) = list
        by simp
      hence
        ∏:(remove1 a (aa#list)) = ∏:list
        by simp
      moreover with aeq have
        ∏:(aa#list) = ∏:list * a
        by simp
      ultimately show
```

```
      ∏:(aa#list) = ∏:remove1 a (aa # list) * a
      by simp
    next
     assume naeq: a ≠ aa
     with aml have aml2: a : set list by simp
     from naeq have
      remove1 a (aa#list) = aa#(remove1 a list)
      by simp
     moreover hence
      ∏:(remove1 a (aa#list)) = aa * ∏:(remove1 a list)
      by simp
     moreover from aml2 plist have
      ∏:list = ∏:(remove1 a list) * a
      by simp
     ultimately show
      ∏:(aa#list) = ∏:remove1 a (aa # list) * a
      by simp
    qed
  qed
qed
```

The final lemma of this section states that if all elements are positive and non-zero then the product of these elements is also positive and non-zero.

```
lemma el-gt0-imp-prod-gt0 [rule-format]:
  fixes xs::real list
  shows ∀ y. y : set xs ⟶ y > 0 ⟹ ∏:xs > 0
proof (induct xs)
  case Nil show ?case by simp
next
  case (Cons a xs)
  have exp: ∏:(a#xs) = ∏:xs * a by simp
  with Cons have a > 0 by simp
  with exp Cons show ?case by (simp add: mult-pos-pos)
qed
```

1.2.2 Auxillary lemma

This section presents a proof of the auxillary lemma required for this theorem.

```
lemma prod-exp:
  fixes x::real
  shows 4*(x*y) = (x+y) ^2 - (x-y) ^2
apply (simp only: diff-minus)
apply (simp add: real-sum-squared-expand)
done

lemma abs-less-imp-sq-less [rule-format]:
  fixes x::real and y::real and z::real and w::real
```

assumes *diff*: *abs* $(x-y) < abs\ (z-w)$
shows $(x-y)\hat{}2 < (z-w)\hat{}2$
proof *cases*
assume $x=y$
hence *abs* $(x-y) = 0$ **by** *simp*
moreover with *diff* **have** $abs(z-w) > 0$ **by** *simp*
hence $(z-w)\hat{}2 > 0$ **by** *simp*
ultimately show *?thesis* **by** *auto*
next
assume $x \neq y$
hence *abs* $(x - y) > 0$ **by** *simp*
with *diff* **have** $(abs\ (x-y))\hat{}2 < (abs\ (z-w))\hat{}2$
by − (*drule power-strict-mono* [**where** *a*=*abs* $(x-y)$ **and** *n*=*2* **and** *b*=*abs* $(z-w)$], *auto*)
thus *?thesis* **by** *simp*
qed

The required lemma (phrased slightly differently than in the informal proof.) Here we show that for any two pairs of numbers with equal sums the pair with the least difference has the greater product.

lemma *le-diff-imp-gt-prod* [*rule-format*]:
fixes *x*::*real* **and** *y*::*real* **and** *z*::*real* **and** *w*::*real*
assumes *diff*: *abs* $(x-y) < abs\ (z-w)$ **and** *sum*: $x+y = z+w$
shows $x*y > z*w$
proof −
from *sum* **have** $(x+y)\hat{}2 = (z+w)\hat{}2$ **by** *simp*
moreover from *diff* **have** $(x-y)\hat{}2 < (z-w)\hat{}2$ **by** (*rule abs-less-imp-sq-less*)
ultimately have $(x+y)\hat{}2 - (x-y)\hat{}2 > (z+w)\hat{}2 - (z-w)\hat{}2$ **by** *auto*
thus $x*y > z*w$ **by** (*simp only*: *prod-exp* [*symmetric*])
qed

1.2.3 Mean and GMean

Now we introduce definitions and properties of arithmetic and geometric means over collections of real numbers.

Definitions

Arithmetic mean

definition
mean :: (*real list*)⇒*real* **where**
mean s = $(\sum$:*s* / *real* (*length s*))

Geometric mean

definition
gmean :: (*real list*)⇒*real* **where**
gmean s = *root* (*length s*) $(\prod$:*s*)

Properties

Here we present some trival properties of *mean* and *gmean*.

lemma *list-sum-mean*:
fixes *xs*::*real list*
shows $\sum$:*xs* = ((*mean xs*) * (*real* (*length xs*)))
apply (*induct-tac xs*)
apply *simp*
apply *clarsimp*
apply (*unfold mean-def*)
apply *clarsimp*
done

lemma *list-mean-eq-iff*:
fixes *one*::*real list* **and** *two*::*real list*
assumes
se: ($\sum$:*one* = $\sum$:*two*) **and**
le: (*length one* = *length two*)
shows (*mean one* = *mean two*)
proof −
from *se le* **have**
($\sum$:*one* / *real* (*length one*)) = ($\sum$:*two* / *real* (*length two*))
by *auto*
thus *?thesis* **unfolding** *mean-def* **.**
qed

lemma *list-gmean-gt-iff*:
fixes *one*::*real list* **and** *two*::*real list*
assumes
gz1: $\prod$:*one* > *0* **and** *gz2*: $\prod$:*two* > *0* **and**
ne1: *one* ≠ [] **and** *ne2*: *two* ≠ [] **and**
pe: ($\prod$:*one* > $\prod$:*two*) **and**
le: (*length one* = *length two*)
shows (*gmean one* > *gmean two*)
unfolding *gmean-def*
using *le ne2 pe* **by** *simp*

This slightly more complicated lemma shows that for every non-empty collection with mean M, adding another element a where $a = M$ results in a new list with the same mean M.

lemma *list-mean-cons* [*rule-format*]:
fixes *xs*::*real list*
shows *xs* ≠ [] ⟶ *mean* ((*mean xs*)#*xs*) = *mean xs*
proof
assume *lne*: *xs* ≠ []
obtain *len* **where** *ld*: *len* = *real* (*length xs*) **by** *simp*
with *lne* **have** *lgt0*: *len* > *0* **by** *simp*
hence *lnez*: *len* ≠ *0* **by** *simp*
from *lgt0* **have** *l1nez*: *len* + *1* ≠ *0* **by** *simp*

```
  from ld have mean: mean xs = ∑:xs / len unfolding mean-def by simp
  with ld real-of-nat-add real-of-one mean-def
  have mean ((mean xs)#xs) = (∑:xs/len + ∑:xs) / (1+len)
    by simp
  also from list-sum-distrib-aux have
    ... = (1 + (1/len))*∑:xs / (1+len) by simp
  also with lnez have
    ... = (len + 1)*∑:xs / (len * (1+len))
    apply −
    apply (drule mult-divide-mult-cancel-left
      [symmetric, where c=len and a=(1 + 1 / len) * ∑:xs and b=1+len])
    apply (clarsimp simp:ring-simps)
    done
  also from l1nez have ... = ∑:xs / len
    apply (subst real-mult-commute [where z=len])
    apply (drule mult-divide-mult-cancel-left
      [where c=len+1 and a=∑:xs and b=len])
    by (simp add: mult-ac add-ac)
  finally show mean ((mean xs)#xs) = mean xs by (simp add: mean)
qed
```

For a non-empty collection with positive mean, if we add a positive number to the collection then the mean remains positive.

```
lemma mean-gt-0 [rule-format]:
  xs≠[] ∧ 0 < x ∧ 0 < (mean xs) ⟶ 0 < (mean (x#xs))
proof
  assume a: xs ≠ [] ∧ 0 < x ∧ 0 < mean xs
  hence xgt0: 0 < x and mgt0: 0 < mean xs by auto
  from a have lxsgt0: length xs ≠ 0 by simp
  from mgt0 have xsgt0: 0 < ∑:xs
  proof −
    have mean xs = ∑:xs / real (length xs) unfolding mean-def by simp
    hence ∑:xs = mean xs * real (length xs) by simp
    moreover from lxsgt0 have real (length xs) > 0 by simp
    moreover with calculation lxsgt0 mgt0 real-mult-order show ?thesis by auto
  qed
  with xgt0 have ∑:(x#xs) > 0 by simp
  thus 0 < (mean (x#xs))
  proof −
    assume 0 < ∑:(x#xs)
    moreover have real (length (x#xs)) > 0 by simp
    ultimately show ?thesis unfolding mean-def by (rule divide-pos-pos)
  qed
qed
```

1.2.4 *list-neq*, *list-eq*

This section presents a useful formalisation of the act of removing all the elements from a collection that are equal (not equal) to a particular value.

We use this to extract all the non-mean elements from a collection as is required by the proof.

Definitions

list-neq and *list-eq* just extract elements from a collection that are not equal (or equal) to some value.

```
abbreviation
  list-neq :: ('a list) ⇒ 'a ⇒ ('a list) where
  list-neq xs el == filter (λx. x≠el) xs
```

```
abbreviation
  list-eq :: ('a list) ⇒ 'a ⇒ ('a list) where
  list-eq xs el == filter (λx. x=el) xs
```

Properties

This lemma just proves a required fact about *list-neq*, *remove1* and *length*.

```
lemma list-neq-remove1 [rule-format]:
  shows a≠m ∧ a : set xs
  ⟶ length (list-neq (remove1 a xs) m) < length (list-neq xs m)
  (is ?A xs ⟶ ?B xs is ?P xs)
proof (induct xs)
  case Nil show ?case by simp
next
  case (Cons x xs)
  note ‹?P xs›
  {
    assume a: ?A (x#xs)
    hence
      a-ne-m: a≠m and
      a-mem-x-xs: a : set(x#xs)
      by auto
    have b: ?B (x#xs)
    proof cases
      assume xs = []
      with a-ne-m a-mem-x-xs show ?thesis
        apply (cases x=a)
        by auto
    next
      assume xs-ne: xs ≠ []
      with a-ne-m a-mem-x-xs show ?thesis
      proof cases
        assume a=x with a-ne-m show ?thesis by simp
      next
        assume a-ne-x: a≠x
        with a-mem-x-xs have a-mem-xs: a : set xs by simp
```

```
      with xs-ne a-ne-m Cons have
        rel: length (list-neq (remove1 a xs) m) < length (list-neq xs m)
        by simp
      show ?thesis
      proof cases
        assume x-e-m: x=m
        with Cons xs-ne a-ne-m a-mem-xs show ?thesis by simp
      next
        assume x-ne-m: x≠m
        from a-ne-x have
          remove1 a (x#xs) = x#(remove1 a xs)
          by simp
        hence
          length (list-neq (remove1 a (x#xs)) m) =
           length (list-neq (x#(remove1 a xs)) m)
          by simp
        also with x-ne-m have
          ... = 1 + length (list-neq (remove1 a xs) m)
          by simp
        finally have
          length (list-neq (remove1 a (x#xs)) m) =
           1 + length (list-neq (remove1 a xs) m)
          by simp
        moreover with x-ne-m a-ne-x have
          length (list-neq (x#xs) m) =
           1 + length (list-neq xs m)
          by simp
        moreover with rel show ?thesis by simp
      qed
    qed
  qed
 }
 thus ?P (x#xs) by simp
qed
```

We now prove some facts about *list-eq*, *list-neq*, length, sum and product.

```
lemma list-eq-sum [simp]:
 fixes xs::real list
 shows ∑:(list-eq xs m) = (m * (real (length (list-eq xs m))))
apply (induct-tac xs)
apply simp
apply clarsimp
apply (subst real-of-nat-Suc)
apply (simp add:ring-simps)
done

lemma list-eq-prod [simp]:
 fixes xs::real list
 shows ∏:(list-eq xs m) = (m ^ (length (list-eq xs m)))
```

apply (*induct-tac xs*)
apply *simp*
apply *clarsimp*
done

lemma *listsum-split*:
fixes *xs*::*real list*
shows $\sum$:*xs* = ($\sum$:(*list-neq xs m*) + $\sum$:(*list-eq xs m*))
apply (*induct xs*)
apply *simp*
apply *clarsimp*
done

lemma *listprod-split*:
fixes *xs*::*real list*
shows $\prod$:*xs* = ($\prod$:(*list-neq xs m*) * $\prod$:(*list-eq xs m*))
apply (*induct xs*)
apply *simp*
apply *clarsimp*
done

lemma *listsum-length-split*:
fixes *xs*::*real list*
shows *length xs* = *length* (*list-neq xs m*) + *length* (*list-eq xs m*)
apply (*induct xs*)
apply *simp+*
done

1.2.5 Element selection

We now show that given after extracting all the elements not equal to the mean there exists one that is greater then (or less than) the mean.

lemma *pick-one-gt*:
fixes *xs*::*real list* **and** *m*::*real*
defines *m*: *m* ≡ (*mean xs*) **and** *neq*: *noteq* ≡ *list-neq xs m*
assumes *asum*: *noteq*≠[]
shows ∃ *e*. *e* : *set noteq* ∧ *e* > *m*
proof (*rule ccontr*)
let *?m* = (*mean xs*)
let *?neq* = *list-neq xs ?m*
let *?eq* = *list-eq xs ?m*
from *list-eq-sum* **have** ($\sum$:*?eq*) = *?m* * (*real* (*length ?eq*)) **by** *simp*
from *asum* **have** *neq-ne*: *?neq* ≠ [] **unfolding** *m neq* **.**
assume *not-el*: ¬(∃ *e*. *e* : *set noteq* ∧ *m* < *e*)
hence *not-el-exp*: ¬(∃ *e*. *e* : *set ?neq* ∧ *?m* < *e*) **unfolding** *m neq* **.**
hence ∀ *e*. ¬(*e* : *set ?neq*) ∨ ¬(*e* > *?m*) **by** *simp*
hence ∀ *e*. *e* : *set ?neq* ⟶ ¬(*e* > *?m*) **by** *blast*
hence ∀ *e*. *e* : *set ?neq* ⟶ *e* ≤ *?m* **by** (*simp add*: *linorder-not-less*)
hence ∀ *e*. *e* : *set ?neq* ⟶ *e* < *?m* **by** (*simp add*:*order-le-less*)

with *prems listsum-mono-lt* **have** $(\sum :?neq) < ?m * (real\ (length\ ?neq))$ **by** *blast*
hence
$(\sum :?neq) + (\sum :?eq) < ?m * (real\ (length\ ?neq)) + (\sum :?eq)$ **by** *simp*
also have
$\ldots = (?m * ((real\ (length\ ?neq) + (real\ (length\ ?eq)))))$
by (*simp add*:*ring-simps*)
also have
$\ldots = (?m * (real\ (length\ xs)))$
apply (*subst real-of-nat-add* [*symmetric*])
by (*simp add*: *listsum-length-split* [*symmetric*])
also have
$\ldots = \sum :xs$
by (*simp add*: *list-sum-mean* [*symmetric*])
also from *not-el calculation* **show** *False* **by** (*simp only*: *listsum-split* [*symmetric*])
qed

lemma *pick-one-lt*:
fixes *xs*::*real list* **and** *m*::*real*
defines *m*: $m \equiv (mean\ xs)$ **and** *neq*: $noteq \equiv list\text{-}neq\ xs\ m$
assumes *asum*: $noteq \neq []$
shows $\exists\, e.\ e : set\ noteq \wedge e < m$
proof (*rule ccontr*) — reductio ad absurdum
let $?m = (mean\ xs)$
let $?neq = list\text{-}neq\ xs\ ?m$
let $?eq = list\text{-}eq\ xs\ ?m$
from *list-eq-sum* **have** $(\sum :?eq) = ?m * (real\ (length\ ?eq))$ **by** *simp*
from *asum* **have** *neq-ne*: $?neq \neq []$ **unfolding** *m neq* **.**
assume *not-el*: $\neg(\exists\, e.\ e : set\ noteq \wedge m > e)$
hence *not-el-exp*: $\neg(\exists\, e.\ e : set\ ?neq \wedge ?m > e)$ **unfolding** *m neq* **.**
hence $\forall\, e.\ \neg(e : set\ ?neq) \vee \neg(e < ?m)$ **by** *simp*
hence $\forall\, e.\ e : set\ ?neq \longrightarrow \neg(e < ?m)$ **by** *blast*
hence $\forall\, e.\ e : set\ ?neq \longrightarrow e \geq ?m$ **by** (*simp add*: *linorder-not-less*)
hence $\forall\, e.\ e : set\ ?neq \longrightarrow e > ?m$ **by** (*auto simp*: *order-le-less*)
with *prems listsum-mono-gt* **have** $(\sum :?neq) > ?m * (real\ (length\ ?neq))$ **by** *blast*
hence
$(\sum :?neq) + (\sum :?eq) > ?m * (real\ (length\ ?neq)) + (\sum :?eq)$ **by** *simp*
also have
$(?m * (real\ (length\ ?neq)) + (\sum :?eq)) =$
$(?m * (real\ (length\ ?neq)) + (?m * (real\ (length\ ?eq))))$
by *simp*
also have
$\ldots = (?m * ((real\ (length\ ?neq) + (real\ (length\ ?eq)))))$
by (*simp add*:*ring-simps*)
also have
$\ldots = (?m * (real\ (length\ xs)))$
apply (*subst real-of-nat-add* [*symmetric*])
by (*simp add*: *listsum-length-split* [*symmetric*])
also have

$\ldots = \sum$:*xs*
by (*simp add*: *list-sum-mean* [*symmetric*])
also from *not-el calculation* **show** *False* **by** (*simp only*: *listsum-split* [*symmetric*])
qed

1.2.6 Abstract properties

In order to maintain some comprehension of the following proofs we now introduce some properties of collections.

Definitions

het: The heterogeneity of a collection is the number of elements not equal to its mean. A heterogeneity of zero implies the all the elements in the collection are the same (i.e. homogeneous).

definition
het :: *real list* ⇒ *nat* **where**
het l = *length* (*list-neq l* (*mean l*))

lemma *het-gt-0-imp-noteq-ne*: *het l* > *0* ⟹ *list-neq l* (*mean l*) ≠ []
unfolding *het-def* **by** *simp*

$\gamma-eq$: Two lists are γ-equivalent if and only if they both have the same number of elements and the same arithmetic means.

definition
γ-*eq* :: ((*real list*)∗(*real list*)) ⇒ *bool* **where**
γ-*eq a* ⟷ *mean* (*fst a*) = *mean* (*snd a*) ∧ *length* (*fst a*) = *length* (*snd a*)

γ-*eq* is transitive and symmetric.

lemma γ-*eq-sym*: γ-*eq* (*a*,*b*) = γ-*eq* (*b*,*a*)
unfolding γ-*eq-def* **by** *auto*

lemma γ-*eq-trans*:
γ-*eq* (*x*,*y*) ⟹ γ-*eq* (*y*,*z*) ⟹ γ-*eq* (*x*,*z*)
unfolding γ-*eq-def* **by** *simp*

pos: A list is positive if all its elements are greater than 0.

definition
pos :: *real list* ⇒ *bool* **where**
pos l ⟷ (*if l*=[] *then False else* ∀ *e*. *e* : *set l* ⟶ *e* > *0*)

lemma *pos-empty* [*simp*]: *pos* [] = *False* **unfolding** *pos-def* **by** *simp*
lemma *pos-single* [*simp*]: *pos* [*x*] = (*x* > *0*) **unfolding** *pos-def* **by** *simp*
lemma *pos-imp-ne*: *pos xs* ⟹ *xs*≠[] **unfolding** *pos-def* **by** *auto*

lemma *pos-cons* [*simp*]:
xs ≠ [] ⟶ *pos* (*x*#*xs*) =

```
   (if (x>0) then pos xs else False)
  (is ?P x xs is ?A xs ⟶ ?S x xs)
proof (simp add: split-if, rule impI)
  assume xsne: xs ≠ []
  hence pxs-simp:
    pos xs = (∀ e. e : set xs ⟶ e > 0)
    unfolding pos-def by simp
  show
    (0 < x ⟶ pos (x # xs) = pos xs) ∧
     (¬ 0 < x ⟶ ¬ pos (x # xs))
  proof
    {
      assume xgt0: 0 < x
      {
        assume pxs: pos xs
        with pxs-simp have ∀ e. e : set xs ⟶ e > 0 by simp
        with xgt0 have ∀ e. e : set (x#xs) ⟶ e > 0 by simp
        hence pos (x#xs) unfolding pos-def by simp
      }
      moreover
      {
        assume pxxs: pos (x#xs)
        hence ∀ e. e : set (x#xs) ⟶ e > 0 unfolding pos-def by simp
        hence ∀ e. e : set xs ⟶ e > 0 by simp
        with xsne have pos xs unfolding pos-def by simp
      }
      ultimately have pos (x # xs) = pos xs
        apply -
        apply (rule iffI)
        apply auto
        done
    }
    thus 0 < x ⟶ pos (x # xs) = pos xs by simp
  next
    {
      assume xngt0: ¬ (0<x)
      {
        assume pxs: pos xs
        with pxs-simp have ∀ e. e : set xs ⟶ e > 0 by simp
        with xngt0 have ¬ (∀ e. e : set (x#xs) ⟶ e > 0) by auto
        hence ¬ (pos (x#xs)) unfolding pos-def by simp
      }
      moreover
      {
        assume pxxs: ¬pos xs
        with xsne have ¬ (∀ e. e : set xs ⟶ e > 0) unfolding pos-def by simp
        hence ¬ (∀ e. e : set (x#xs) ⟶ e > 0) by auto
        hence ¬ (pos (x#xs)) unfolding pos-def by simp
      }
```

```
    ultimately have ¬ pos (x#xs) by auto
   }
   thus ¬ 0 < x ⟶ ¬ pos (x # xs) by simp
  qed
qed
```

Properties

Here we prove some non-trivial properties of the abstract properties.

Two lemmas regarding *pos*. The first states the removing an element from a positive collection (of more than 1 element) results in a positive collection. The second asserts that the mean of a positive collection is positive.

```
lemma pos-imp-rmv-pos:
  assumes (remove1 a xs)≠[] pos xs shows pos (remove1 a xs)
proof −
  from assms have pl: pos xs and rmvne: (remove1 a xs)≠[] by auto
  from pl have xs ≠ [] by (rule pos-imp-ne)
  with pl pos-def have ∀ x. x : set xs ⟶ x > 0 by simp
  hence ∀ x. x : set (remove1 a xs) ⟶ x > 0
    using set-remove1-subset[of - xs] by(blast)
  with rmvne show pos (remove1 a xs) unfolding pos-def by simp
qed

lemma pos-mean: pos xs ⟹ mean xs > 0
proof (induct xs)
  case Nil thus ?case by(simp add: pos-def)
next
  case (Cons x xs)
  show ?case
  proof cases
    assume xse: xs = []
    hence pos (x#xs) = (x > 0) by simp
    with Cons(2) have x>0 by(simp)
    with xse have 0 < mean (x#xs) by(auto simp:mean-def)
    thus ?thesis by simp
  next
    assume xsne: xs ≠ []
    show ?thesis
    proof cases
      assume pxs: pos xs
      with Cons(1) have z-le-mxs: 0 < mean xs by(simp)
      {
        assume ass: x > 0
        with ass z-le-mxs xsne have 0 < mean (x#xs)
          apply −
          apply (rule mean-gt-0)
          by simp
      }
```

```
    moreover
    {
      from xsne pxs have 0 < x
      proof cases
        assume 0 < x thus ?thesis by simp
      next
        assume ¬(0 < x)
        with xsne pos-cons have pos (x#xs) = False by simp
        with Cons(2) show ?thesis by simp
      qed
    }
    ultimately have 0 < mean (x#xs) by simp
    thus ?thesis by simp
  next
    assume npxs: ¬pos xs
    with xsne pos-cons have pos (x#xs) = False  by simp
    thus ?thesis using Cons(2) by simp
  qed
 qed
qed
```

We now show that homogeneity of a non-empty collection x implies that its product is equal to $(mean\ x)\ \hat{}\ (length\ x)$.

```
lemma listprod-het0:
 shows x≠[] ∧ het x = 0 ⟹ ∏:x = (mean x) ^ (length x)
proof −
 assume x≠[] ∧ het x = 0
 hence xne: x≠[] and hetx: het x = 0 by auto
 from hetx have lz: length (list-neq x (mean x)) = 0 unfolding het-def .
 hence ∏:(list-neq x (mean x)) = 1 by simp
 with listprod-split have ∏:x = ∏:(list-eq x (mean x))
   apply −
   apply (drule meta-spec [of - x])
   apply (drule meta-spec [of - mean x])
   by simp
 also with list-eq-prod have
   ... = (mean x) ^ (length (list-eq x (mean x))) by simp
 also with calculation lz listsum-length-split have
   ∏:x = (mean x) ^ (length x)
   apply −
   apply (drule meta-spec [of - x])
   apply (drule meta-spec [of - mean x])
   by simp
 thus ?thesis by simp
qed
```

Furthermore we present an important result - that a homogeneous collection has equal geometric and arithmetic means.

```
lemma het-base:
```

shows *pos x* $\wedge$ *x*$\neq$[] $\wedge$ *het x* = *0* $\Longrightarrow$ *gmean x* = *mean x*
proof −
assume *ass*: *pos x* $\wedge$ *x*$\neq$[] $\wedge$ *het x* = *0*
hence
xne: *x*$\neq$[] **and**
hetx: *het x* = *0* **and**
posx: *pos x*
by *auto*
from *posx pos-mean* **have** *mxgt0*: *mean x* > *0* **by** *simp*
from *xne* **have** *lxgt0*: *length x* > *0* **by** *simp*
with *ass listprod-het0* **have**
root (*length x*) ($\prod$:*x*) = *root* (*length x*) ((*mean x*) ^(*length x*))
by *simp*
also from *lxgt0 mxgt0 real-root-power-cancel* **have** ... = *mean x* **by** *auto*
finally show *gmean x* = *mean x* **unfolding** *gmean-def* **.**
qed

1.2.7 Existence of a new collection

We now present the largest and most important proof in this document. Given any positive and non-homogeneous collection of real numbers there exists a new collection that is γ-equivalent, positive, has a strictly lower heterogeneity and a greater geometric mean.

lemma *new-list-gt-gmean*:
fixes *xs*::*real list* **and** *m*::*real*
defines
m: *m* $\equiv$ (*mean xs*) **and**
neq: *noteq* $\equiv$ *list-neq xs m* **and**
eq: *eq* $\equiv$ *list-eq xs m*
assumes *pos-xs*: *pos xs* **and** *het-gt-0*: *het xs* > *0*
shows
$\exists$ *xs'*. *gmean xs'* > *gmean xs* $\wedge$ γ-*eq* (*xs'*,*xs*) $\wedge$
het xs' < *het xs* $\wedge$ *pos xs'*
proof −
from *pos-xs pos-imp-ne* **have**
pos-els: $\forall$ *y*. *y* : *set xs* $\longrightarrow$ *y* > *0* **by** (*unfold pos-def*, *simp*)
with *el-gt0-imp-prod-gt0* **have** *pos-asm*: $\prod$:*xs* > *0* **by** *simp*
from *neq het-gt-0 het-gt-0-imp-noteq-ne m* **have**
neqne: *noteq* $\neq$ [] **by** *simp*

Pick two elements from xs, one greater than m, one less than m.

from *prems pick-one-gt neqne* **obtain** α **where**
α-*def*: α : *set noteq* $\wedge$ α > *m* **unfolding** *neq m* **by** *auto*
from *prems pick-one-lt neqne* **obtain** β **where**
β-*def*: β : *set noteq* $\wedge$ β < *m* **unfolding** *neq m* **by** *auto*
from α-*def* β-*def* **have** α-*gt*: α > *m* **and** β-*lt*: β < *m* **by** *auto*
from *prems* **have** *el-neq*: $\beta \neq \alpha$ **by** *simp*
from *neqne neq* **have** *xsne*: *xs* $\neq$ [] **by** *auto*

```
  from prems have βmem: β : set xs by (auto simp: neq)
  from prems have αmem: α : set xs by (auto simp: neq)

  from pos-xs pos-def xsne αmem βmem α-def β-def have
    α-pos: α > 0 and β-pos: β > 0 by auto

  — remove these elements from xs, and insert two new elements
  obtain left-over where lo: left-over = (remove1 β (remove1 α xs)) by simp
  obtain b where bdef: m + b = α + β
    by (drule meta-spec [of - α + β − m], simp)

  from m pos-xs pos-def pos-mean have m-pos: m > 0 by simp
  with bdef α-pos β-pos α-gt β-lt have b-pos: b > 0 by simp

  obtain new-list where nl: new-list = m#b#(left-over) by auto

  from el-neq βmem αmem have β : set xs ∧ α : set xs ∧ β ≠ α by simp
   hence α : set (remove1 β xs) ∧ β : set(remove1 α xs) by (auto simp add:
in-set-remove1)
  moreover hence (remove1 α xs) ≠ [] ∧ (remove1 β xs) ≠ [] by (auto)
  ultimately have
    mem : α : set(remove1 β xs) ∧ β : set(remove1 α xs) ∧
          (remove1 α xs) ≠ [] ∧ (remove1 β xs) ≠ [] by simp
  — prove that new list is positive
  from nl have nl-pos: pos new-list
  proof cases
    assume left-over = []
    with nl b-pos m-pos show ?thesis by simp
  next
    assume lone: left-over ≠ []
    from mem pos-imp-rmv-pos pos-xs have pos (remove1 α xs) by simp
    with lo lone pos-imp-rmv-pos have pos left-over by simp
    with lone mem nl m-pos b-pos show ?thesis by simp
  qed

  — now show that the new list has the same mean as the old list
  with mem prems lo bdef αmem βmem
    have ∑:new-list = ∑:xs
      apply clarsimp
      apply (subst listsum-rmv1)
        apply simp
      apply (subst listsum-rmv1)
        apply simp
      apply clarsimp
    done
  moreover from lo nl βmem αmem mem have
    leq: length new-list = length xs
    apply −
```

```
  apply (erule conjE)+
  apply (clarsimp)
  apply (subst length-remove1, simp)
  apply (simp add: length-remove1)
  apply (auto dest!:length-pos-if-in-set)
  done
ultimately have eq-mean: mean new-list = mean xs by (rule list-mean-eq-iff)

— finally show that the new list has a greater gmean than the old list
have gt-gmean: gmean new-list > gmean xs
proof −
  from bdef α-gt β-lt have abs (m − b) < abs (α − β) by arith
  moreover from bdef have m+b = α+β .
  ultimately have mb-gt-gt: m∗b > α∗β by (rule le-diff-imp-gt-prod)
  moreover from nl have
    ∏:new-list = ∏:left-over ∗ (m∗b) by auto
  moreover
  from lo αmem βmem mem remove1-retains-prod have
    xsprod: ∏:xs = ∏:left-over ∗ (α∗β) by auto
  moreover from xsne have
    xs ≠ [] .
  moreover from nl have
    nlne: new-list ≠ [] by simp
  moreover from pos-asm lo have
    ∏:left-over > 0
    proof −
      from pos-asm have ∏:xs > 0 .
      moreover
      from xsprod have ∏:xs = ∏:left-over ∗ (α∗β) .
      ultimately have ∏:left-over ∗ (α∗β) > 0 by simp
      moreover
      from pos-els αmem βmem have α > 0 and β > 0 by auto
      hence α∗β > 0 by (rule real-mult-order)
      ultimately show ∏:left-over > 0
        apply −
        apply (rule zero-less-mult-pos2 [where a=(α ∗ β)])
        by auto
    qed
  ultimately have ∏:new-list > ∏:xs
    apply clarsimp
    apply (rule real-mult-less-mono2)
    apply assumption
    apply assumption
    done
  moreover with pos-asm nl have ∏:new-list > 0 by auto
  moreover from calculation pos-asm xsne nlne leq list-gmean-gt-iff
  show gmean new-list > gmean xs by simp
qed
```

```
  — auxillary info
  from β-lt have β-ne-m: β ≠ m by simp
  from mem have
    β-mem-rmv-α: β : set (remove1 α xs) and rmv-α-ne: (remove1 α xs) ≠ [] by
auto

  from α-def have α-ne-m: α ≠ m by simp

  — now show that new list is more homogeneous
  have lt-het: het new-list < het xs
  proof cases
    assume bm: b=m
    with het-def have
      het new-list = length (list-neq new-list (mean new-list))
      by simp
    also with m nl eq-mean have
      ... = length (list-neq (m#b#(left-over)) m)
      by simp
    also with bm have
      ... = length (list-neq left-over m)
      by simp
    also with lo β-def α-def have
      ... = length (list-neq (remove1 β (remove1 α xs)) m)
      by simp
    also from β-ne-m β-mem-rmv-α rmv-α-ne have
      ... < length (list-neq (remove1 α xs) m)
      apply −
      apply (rule list-neq-remove1)
      by simp
    also from αmem α-ne-m xsne have
      ... < length (list-neq xs m)
      apply −
      apply (rule list-neq-remove1)
      by simp
    also with m het-def have ... = het xs by simp
    finally show het new-list < het xs .
  next
    assume bnm: b≠m
    with het-def have
      het new-list = length (list-neq new-list (mean new-list))
      by simp
    also with m nl eq-mean have
      ... = length (list-neq (m#b#(left-over)) m)
      by simp
    also with bnm have
      ... = length (b#(list-neq left-over m))
      by simp
    also have
      ... = 1 + length (list-neq left-over m)
```

```
    by simp
  also with lo β-def α-def have
    ... = 1 + length (list-neq (remove1 β (remove1 α xs)) m)
    by simp
  also from β-ne-m β-mem-rmv-α rmv-α-ne have
    ... < 1 + length (list-neq (remove1 α xs) m)
    apply −
    apply (simp only: nat-add-left-cancel-less)
    apply (rule list-neq-remove1)
    by simp
  finally have
    het new-list ≤ length (list-neq (remove1 α xs) m)
    by simp
  also from αmem α-ne-m xsne have ... < length (list-neq xs m)
    apply −
    apply (rule list-neq-remove1)
    by simp
  also with m het-def have ... = het xs by simp
  finally show het new-list < het xs .
 qed

    — thus thesis by existence of newlist
 from γ-eq-def lt-het gt-gmean eq-mean leq nl-pos show ?thesis by auto
qed
```

Furthermore we show that for all non-homogeneous positive collections there exists another collection that is γ-equivalent, positive, has a greater geometric mean *and* is homogeneous.

```
lemma existence-of-het0 [rule-format]:
 shows ∀ x. p = het x ∧ p > 0 ∧ pos x ⟶
 (∃ y. gmean y > gmean x ∧ γ-eq (x,y) ∧ het y = 0 ∧ pos y)
 (is ?Q p is ∀ x. (?A x p ⟶ ?S x))
proof (induct p rule: nat-less-induct)
 fix n
 assume ind: ∀ m<n. ?Q m
 {
  fix x
  assume ass: ?A x n
  hence het x > 0 and pos x by auto
  with new-list-gt-gmean have
   ∃ y. gmean y > gmean x ∧ γ-eq (x,y) ∧ het y < het x ∧ pos y
   apply −
   apply (drule meta-spec [of - x])
   apply (drule meta-mp)
    apply assumption
   apply (drule meta-mp)
    apply assumption
   apply (subst(asm) γ-eq-sym)
   apply simp
```

```
    done
  then obtain β where
    βdef: gmean β > gmean x ∧ γ-eq (x,β) ∧ het β < het x ∧ pos β ..
  then obtain b where bdef: b = het β by simp
  with ass βdef have b < n by auto
  with ind have ?Q b by simp
  with βdef have
    ind2: b = het β ∧ 0 < b ∧ pos β ⟶
    (∃ y. gmean β < gmean y ∧ γ-eq (β, y) ∧ het y = 0 ∧ pos y) by simp
  {
    assume ¬(0<b)
    hence b=0 by simp
    with bdef have het β = 0 by simp
    with βdef have ?S x by auto
  }
  moreover
  {
    assume 0 < b
    with bdef ind2 βdef have ?S β by simp
    then obtain γ where
      gmean β < gmean γ ∧ γ-eq (β, γ) ∧ het γ = 0 ∧ pos γ ..
    with βdef have gmean x < gmean γ ∧ γ-eq (x,γ) ∧ het γ = 0 ∧ pos γ
      apply clarsimp
      apply (rule γ-eq-trans)
      by auto
    hence ?S x by auto
  }
  ultimately have ?S x by auto
 }
 thus ?Q n by simp
qed
```

1.2.8 Cauchy's Mean Theorem

We now present the final proof of the theorem. For any positive collection we show that its geometric mean is less than or equal to its arithmetic mean.

```
theorem CauchysMeanTheorem:
 fixes z::real list
 assumes pos z
 shows gmean z ≤ mean z
proof −
 from ‹pos z› have zne: z≠[] by (rule pos-imp-ne)
 show gmean z ≤ mean z
 proof cases
   assume het z = 0
   with ‹pos z› zne het-base have gmean z = mean z by simp
   thus ?thesis by simp
 next
   assume het z ≠ 0
```

```
    hence het z > 0 by simp
    moreover obtain k where k = het z by simp
    moreover with calculation ‹pos z› existence-of-het0 have
      ∃ y. gmean y > gmean z ∧ γ-eq (z,y) ∧ het y = 0 ∧ pos y by auto
    then obtain α where
      gmean α > gmean z ∧ γ-eq (z,α) ∧ het α = 0 ∧ pos α ..
    with het-base γ-eq-def pos-imp-ne have
      mean z = mean α and
      gmean α > gmean z and
      gmean α = mean α by auto
    hence gmean z < mean z by simp
    thus ?thesis by simp
  qed
qed

end
```

Chapter 2

The Cauchy-Schwarz Inequality

theory *CauchySchwarz*
imports *Complex-Main*
begin

2.1 Abstract

The following document presents a formalised proof of the Cauchy-Schwarz Inequality for the specific case of R^n. The system used is Isabelle/Isar.

Theorem: Take V to be some vector space possessing a norm and inner product, then for all $a, b \in V$ the following inequality holds: $|a \cdot b| \leq \|a\| * \|b\|$. Specifically, in the Real case, the norm is the Euclidean length and the inner product is the standard dot product.

2.2 Formal Proof

2.2.1 Vector, Dot and Norm definitions.

This section presents definitions for a real vector type, a dot product function and a norm function.

Vector

We now define a vector type to be a tuple of (function, length). Where the function is of type *nat* $\Rightarrow$ *real*. We also define some accessor functions and appropriate notation.

types *vector* = (*nat*$\Rightarrow$*real*) $*$ *nat*

definition
ith :: *vector* ⇒ *nat* ⇒ *real* (((-)_) [*80*,*100*] *100*) **where**
ith v i = *fst v i*

definition
vlen :: *vector* ⇒ *nat* **where**
vlen v = *snd v*

Now to access the second element of some vector v the syntax is v_2.

Dot and Norm

We now define the dot product and norm operations.

definition
dot :: *vector* ⇒ *vector* ⇒ *real* (**infixr** · *60*) **where**
dot a b = (∑ *j*∈{*1*..(*vlen a*)}. a_j∗b_j)

definition
norm :: *vector* ⇒ *real* (‖-‖ *100*) **where**
norm v = *sqrt* (∑ *j*∈{*1*..(*vlen v*)}. v_j ^*2*)

notation (*HTML* **output**)
norm (||-|| *100*)

Another definition of the norm is $\|v\| = sqrt\ (v \cdot v)$. We show that our definition leads to this one.

lemma *norm-dot*:
‖*v*‖ = *sqrt* (*v*·*v*)
proof −
have *sqrt* (*v*·*v*) = *sqrt* (∑ *j*∈{*1*..(*vlen v*)}. v_j∗v_j) **unfolding** *dot-def* **by** *simp*
also with *real-sq* **have** ... = *sqrt* (∑ *j*∈{*1*..(*vlen v*)}. v_j ^*2*) **by** *simp*
also have ... = ‖*v*‖ **unfolding** *norm-def* **by** *simp*
finally show *?thesis* **..**
qed

A further important property is that the norm is never negative.

lemma *norm-pos*:
‖*v*‖ ≥ *0*
proof −
have ∀ *j*. v_j ^*2* ≥ *0* **unfolding** *ith-def* **by** *auto*
hence ∀ *j*∈{*1*..(*vlen v*)}. v_j ^*2* ≥ *0* **by** *simp*
with *setsum-nonneg* **have** (∑ *j*∈{*1*..(*vlen v*)}. v_j ^*2*) ≥ *0* **.**
with *real-sqrt-ge-zero* **have** *sqrt* (∑ *j*∈{*1*..(*vlen v*)}. v_j ^*2*) ≥ *0* **.**
thus *?thesis* **unfolding** *norm-def* **.**
qed

We now prove an intermediary lemma regarding double summation.

lemma *double-sum-aux*:

```
  fixes f::nat ⇒ real
  shows
  (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) =
   (∑k∈{1..n}. (∑j∈{1..n}. (f k * g j + f j * g k) / 2))
proof −
  have
    2 * (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) =
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) +
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j))
    by simp
  also have
    ... =
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) +
    (∑k∈{1..n}. (∑j∈{1..n}. f j * g k))
    by (simp only: double-sum-equiv)
  also have
    ... =
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j + f j * g k))
    by (auto simp add: setsum-addf)
  finally have
    2 * (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) =
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j + f j * g k)) .
  hence
    (∑k∈{1..n}. (∑j∈{1..n}. f k * g j)) =
     (∑k∈{1..n}. (∑j∈{1..n}. (f k * g j + f j * g k)))*(1/2)
    by auto
  also have
    ... =
     (∑k∈{1..n}. (∑j∈{1..n}. (f k * g j + f j * g k)*(1/2)))
    by (simp add: setsum-right-distrib real-mult-commute)
  finally show ?thesis by (auto simp add: inverse-eq-divide)
qed
```

The final theorem can now be proven. It is a simple forward proof that uses properties of double summation and the preceding lemma.

```
theorem CauchySchwarzReal:
  fixes x::vector
  assumes vlen x = vlen y
  shows |x·y| ≤ ‖x‖*‖y‖
proof −
  have 0 ≤ |x·y| by simp
  moreover have 0 ≤ ‖x‖*‖y‖
    by (auto simp add: norm-pos intro: mult-nonneg-nonneg)
  moreover have |x·y| ^2 ≤ (‖x‖*‖y‖) ^2
  proof −
```

We can rewrite the goal in the following form ...

```
    have (‖x‖*‖y‖) ^2 − |x·y| ^2 ≥ 0
    proof −
```

obtain n **where** nx: $n = vlen\ x$ **by** *simp*
with ‹$vlen\ x = vlen\ y$› **have** ny: $n = vlen\ y$ **by** *simp*
{

Some preliminary simplification rules.

have $\forall j{\in}\{1..n\}.\ x_j\ \hat{}\ 2 \geq 0$ **by** *simp*
hence $(\sum j{\in}\{1..n\}.\ x_j\ \hat{}\ 2) \geq 0$ **by** (*rule setsum-nonneg*)
hence xp: $(sqrt\ (\sum j{\in}\{1..n\}.\ x_j\ \hat{}\ 2))\ \hat{}\ 2 = (\sum j{\in}\{1..n\}.\ x_j\ \hat{}\ 2)$
by (*rule real-sqrt-pow2*)

have $\forall j{\in}\{1..n\}.\ y_j\ \hat{}\ 2 \geq 0$ **by** *simp*
hence $(\sum j{\in}\{1..n\}.\ y_j\ \hat{}\ 2) \geq 0$ **by** (*rule setsum-nonneg*)
hence yp: $(sqrt\ (\sum j{\in}\{1..n\}.\ y_j\ \hat{}\ 2))\ \hat{}\ 2 = (\sum j{\in}\{1..n\}.\ y_j\ \hat{}\ 2)$
by (*rule real-sqrt-pow2*)

The main result of this section is that $(\|x\|{*}\|y\|)\ \hat{}\ 2$ can be written as a double sum.

have
$(\|x\|{*}\|y\|)\ \hat{}\ 2 = \|x\|\ \hat{}\ 2 * \|y\|\ \hat{}\ 2$
by (*simp add*: *real-sq-exp*)
also from *nx ny* **have**
$\ldots = (sqrt\ (\sum j{\in}\{1..n\}.\ x_j\ \hat{}\ 2))\ \hat{}\ 2 * (sqrt\ (\sum j{\in}\{1..n\}.\ y_j\ \hat{}\ 2))\ \hat{}\ 2$
unfolding *norm-def* **by** *auto*
also from *xp yp* **have**
$\ldots = (\sum j{\in}\{1..n\}.\ x_j\ \hat{}\ 2){*}(\sum j{\in}\{1..n\}.\ y_j\ \hat{}\ 2)$
by *simp*
also from *setsum-product-expand* **have**
$\ldots = (\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k\ \hat{}\ 2){*}(y_j\ \hat{}\ 2)))$ **.**
finally have
$(\|x\|{*}\|y\|)\ \hat{}\ 2 = (\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k\ \hat{}\ 2){*}(y_j\ \hat{}\ 2)))$ **.**
}
moreover
{

We also show that $|x{\cdot}y|\ \hat{}\ 2$ can be expressed as a double sum.

have
$|x{\cdot}y|\ \hat{}\ 2 = (x{\cdot}y)\ \hat{}\ 2$
by *simp*
also from *nx* **have**
$\ldots = (\sum j{\in}\{1..n\}.\ x_j{*}y_j)\ \hat{}\ 2$
unfolding *dot-def* **by** *simp*
also from *real-sq* **have**
$\ldots = (\sum j{\in}\{1..n\}.\ x_j{*}y_j){*}(\sum j{\in}\{1..n\}.\ x_j{*}y_j)$
by *simp*
also from *setsum-product-expand* **have**
$\ldots = (\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k{*}y_k){*}(x_j{*}y_j)))$
by *simp*
finally have
$|x{\cdot}y|\ \hat{}\ 2 = (\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k{*}y_k){*}(x_j{*}y_j)))$ **.**
}

We now manipulate the double sum expressions to get the required inequality.

ultimately have
$(\|x\|*\|y\|)\ \hat{}\ 2 - |x \cdot y|\ \hat{}\ 2 =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k\ \hat{}\ 2)*(y_j\ \hat{}\ 2))) -$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k*y_k)*(x_j*y_j)))$
by *simp*
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ ((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2))/2)) -$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k*y_k)*(x_j*y_j)))$
by (*simp only*: *double-sum-aux*)
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ ((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2))/2 - (x_k*y_k)*(x_j*y_j)))$
by (*auto simp add*: *setsum-subtractf*)
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (inverse\ 2)*2*$
$(((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2))*(1/2) - (x_k*y_k)*(x_j*y_j))))$
by *auto*
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (inverse\ 2)*(2*$
$(((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2))*(1/2) - (x_k*y_k)*(x_j*y_j)))))$
by (*simp only*: *real-mult-assoc*)
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (inverse\ 2)*$
$((((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2))*2*(inverse\ 2) - 2*(x_k*y_k)*(x_j*y_j)))))$
by (*auto simp add*: *real-add-mult-distrib real-mult-assoc mult-ac*)
also have
$\ldots =$
$(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (inverse\ 2)*$
$((((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2)) - 2*(x_k*y_k)*(x_j*y_j)))))$
by (*simp only*: *real-mult-assoc*, *simp*)
also have
$\ldots =$
$(inverse\ 2)*(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.$
$(((x_k\ \hat{}\ 2*y_j\ \hat{}\ 2) + (x_j\ \hat{}\ 2*y_k\ \hat{}\ 2)) - 2*(x_k*y_k)*(x_j*y_j))))$
by (*simp only*: *setsum-right-distrib*)
also have
$\ldots =$
$(inverse\ 2)*(\sum k{\in}\{1..n\}.\ (\sum j{\in}\{1..n\}.\ (x_k*y_j - x_j*y_k)\ \hat{}\ 2))$
by (*simp only*: *real-diff-exp real-sq-exp*, *auto simp add*: *mult-ac*)
also have $\ldots \geq 0$
proof −
{
fix k::*nat*
have $\forall j{\in}\{1..n\}.\ (x_k*y_j - x_j*y_k)\ \hat{}\ 2 \geq 0$ **by** *simp*

```
      hence (∑j∈{1..n}. (x_k*y_j − x_j*y_k) ^2) ≥ 0 by (rule setsum-nonneg)
    }
    hence ∀k∈{1..n}. (∑j∈{1..n}. (x_k*y_j − x_j*y_k) ^2) ≥ 0 by simp
    hence (∑k∈{1..n}. (∑j∈{1..n}. (x_k*y_j − x_j*y_k) ^2)) ≥ 0
      by (rule setsum-nonneg)
    thus ?thesis by simp
  qed
  finally show (‖x‖*‖y‖) ^2 − |x·y| ^2 ≥ 0 .
 qed
 thus ?thesis by simp
qed
ultimately show ?thesis by (rule real-sq-order)
qed

end
```